Mobile Manipulation with the KUKA KMR iiwa & ROS

Thomas Rühr, Senior Dev. Corporate Research, KUKA Deutschland GmbH
Milestones in Our History

1898
KUKA turns on the lights
Johann Josef Keller and Jakob Knappich open their acetylene gas plant, making residential and urban lighting economical.

1920
Expertise drives growth
On the basis of its know-how in the areas of welding and cutting, the company begins manufacturing large containers.

1964
Augsburg’s Princess
The “Princess” compact portable typewriter was a masterpiece of machine tooling.

1973
Famulus, the pioneer
With the world’s first industrial robot using six electric motor-driven axes, KUKA goes down in history as a pioneer in the field of robotics.

2010
Top-selling robot family
KR QUANTEC immediately becomes the world’s top-selling series of robots thanks to its unique range capacity.

2013
Hand-in-hand with machines
The highly sensitive LBR iiwa is the first series-produced lightweight robot capable of safely collaborating with humans.

2016
One KUKA, many solutions
KUKA is one of the world’s leading provider of intelligent robot-based automation.

2016
Robot control by PC
KUKA becomes the first manufacturer to make the paradigm shift to a PC-based robot controller.
Customers Benefit from KUKA’s Unique Smart Automation Expertise

KUKA Group

Systems
- Smart, safe & efficient production solutions

Robotics
- Robots, cells & service for intelligent automation

Swisslog Logistics Automation
- Data & robotics-driven logistics automation

Swisslog Healthcare
- Medication management solutions

China
- Automation for China – from China

Global presence

KUKA HQ
- Augsburg, Germany

>40 Countries
€ 3.3bn. Order Intake (2018)
14,200 Employees

Focus industries
- Automotive/Tier1
- Electronics
- General Industry
- E-Commerce/Retail
- Consumer Goods
- Healthcare

» The clear choice for smart automation.«
A Global Company with a Wide Range of Know-How – Some Examples

- **Augsburg, DE**
  Headquarters, R&D, Production

- **Buchs, CH**
  Headquarters, Swisslog Logistics Automation

- **Austin, USA**
  Cloud and IT Software

- **Broomfield, USA**
  Swisslog Healthcare, Technology Center

- **Toledo, USA**
  Production Operations
  Jeep® Wrangler

- **Espoo, FIN**
  3D Simulation

- **Taksony, HUN**
  R&D and Production Control cabinets and cables

- **Shanghai, CN**
  APAC-Hub and Production

- **Foshan, CN**
  Robotics Park
- WP – Waterproof
- HM – Hygienic Machine (Food processing,..)
- CR – Cleanroom
- EX – Explosive Environments
...
How to interface KUKA robots with ROS – KUKA KRC / KSS

In **KUKA KRC** (KUKA Robot Controller) / **KSS** (KUKA System Software),
→ the robot application is written in KRL (KUKA Robot Language)

Interfaces :

− **RSI (Robot Sensor Interface)** package allows control via external client at 250Hz in realtime
  − External client must be able to deliver set points at a high rate!
− **EKI (Ethernet KRL Interface)** allows to receive XML Messages in KRL → PTP, LIN, SPL etc.
  − Motion planning in ROS → Trajectory transfer and execution via EKI (Joint SPL)
  − No realtime requirements for external client

− **mxAutomation** allows to control the robot from a PLC, used e.g. in CNC machines to control a KUKA arm for loading/unloading workpiece → advantage over RSI / EKI questionable
− **EMI (External Motion Interface)** used to integrate KUKA Robots for CNC → same controller as RSI
How to interface KUKA iiwa robots with ROS – KUKA SUNRISE / RoboticsAPI

– In KUKA Sunrise / RoboticsAPI, the robot application is written in Java
– ROSJava supports Topics, Actions, Services
  – we can expose robot/controller functionality to ROS,
    – e.g. https://github.com/IFL-CAMP/iiwa_stack exposes PTP, LIN, ... + Servoing
  – Motion planning in ROS → Transfer and execution as MotionBatch of SplineJPs

– External realtime control → FRI (Fast Robot Interface) can be used at 1kHz rate
  – Realtime PC connects to KUKA controller, application starts FRI → control by external PC
  – FRI client and Application can be a ROS nodes

– PLCs: generated Java API / configuration via WorkVisual → expose via ROSJava
KUKA mobility: omniMove

- Up to 30m length
- 0.4t - 90t payload
- Combinable
- Manipulators
From “omniRob” to KMR iiwa

2007 Prototype
2010 „omniRob“
2015 „KMR iiwa“
2018 „KMR diiwa“ (18DOF)

2008 Prototype
2015 „Valeri“ (12DOF)
2017 KMR iiwa
2018 Wafer Handling
Shaping Innovation Through Cooperative Research Projects

Enabling Technologies: Connectivity, IIoT / Industry 4.0
Changes in technology from prototype to product

- Lead-acid battery (2007) → **Lithium** with BMS
- Timing-belt transmission → **Backlash-free cycloid gearbox for omni-wheels**
- KSS → **Sunrise**
- LBR 4 / 4+ → **iiwa**
- „Cobbled together prototype“ → **Certified safety, cleanroom-qualified ...**
- „Researchy navigation“ → **Industry proven** navigation product (Virtual Lines, Fleet Management, Calibration, ...)

- „Closed-up“ system? → **How to enable research?**
Example project: Hybr-iT

- Hybrid and intelligent human-robot collaboration – hybrid teams in versatile, cyber-physical production environments
- One of many topics: collaborative dual-arm mobile robot
Safety in Research & Development

- **Who is responsible for safety?**
  - We as a manufacturer guarantee that our **products** comply to certain standards (single point of control, ...)
  - The **customer/integrator** is typically responsible for the application, safety engineer checks & signs
- **Now, together, we „hack“ our product → product doesn’t follow standards!**
- **Solution:** „**Prototype agreement**“
  - **Customer signs** that he is aware of the prototypical nature of the equipment
  - **Customer/integrator commits to only use the robot in laboratory setting**
    - A specially trained person will have to **supervise the robot at all times** and stop it using a **safe remote stop**
    - e.g. TYRO INDUS GEMINI ~1k€
    - People in the vicinity of the robot have to be **trained professionals** (e.g. not „shopfloor workers“)
    - HRC: use „deadman-switches“ on robot, footswitches etc. (MF touch electric)

→ **Prototyping/development yes! Production NO!**
→ Results inspire new products or product features that will then get the „full treatment“
Grey vs. Yellow safety
move_base

Navigation Stack Setup

```
move_base
    amcl
      sensor transforms
        "/tf" tf/tfMessage
      odometry source
        "odom" nav_msgs/Odometry
    global_planner
    recovery_behaviors
      global_costmap
    local_planner
    local_costmap
    "cmd_vel" geometry_msgs/Twist
  base controller
```

```
map_server
  "/map" nav_msgs/GetMap
  sensor topics
    sensor_msgs/LaserScan
    sensor_msgs/PointCloud
  sensor sources
```

- provided node
- optional provided node
- platform specific node
KUKA Nav Box

IPC External
- Localization
- Motions
- ...

IPC Internal
- Laser
- Localization
- Odometry
- Velocity
- Localizer
- Motions
- LaserDriver
- Skills
- Basedriver
- ...

RoboticsAPI / ApplicationServer
- SunriseJavaAPI
- SunriseRealtimeSystem
- miiwa-Plattform

Basedriver
Nav Box <-> ROS

- IPC External:
  - Localization
  - Motions
  - ...

- IPC Internal:
  - Laser
  - Localization
  - Odometry
  - Velocity

- Basedriver

- ROSJava Adapter
  - ipjava.jar
  - Ipjava.dll / .so

- ROS

- RAPI

- RoboticsAPI / ApplicationServer
  - SunriseJavaAPI

- SunriseRealtimeSystem
  - miwa-Plattform

- Mobile Manipulation with the KUKA KMR iiwa & ROS | Thomas Rühr | Thomas.Ruehr@kuka.com | 11.12.19 | www.kuka.com
ROS vs. IPC messages

**sensor_msgs/LaserScan Message**

**File:** sensor_msgs/LaserScan.msg

**Raw Message Definition**

```java
public class IPC_kuka_laser_laser_message {

  public static String getHeader() {
    return "Kuka_Laser_Laser";
  }

  public static String getClassName() {
    return "LaserScan\r\n  ";
  }

  public static String getBody() {
    return "LaserScan\r\n  ";
  }

  // The default ctor.
  public IPC_kuka_laser_laser_message() {
    //
    this.id = msg.id;
    this.config = msg.config;
    this.numReadings = msg.numReadings;
    this.range = msg.range;
    this.numReadings = msg.numReadings;
    this.remission = msg.remission;
    this.timestamp = msg.timestamp;
  }

  //
  // The copy constructor.
  //
  // Copy the message to this.
  //
  public IPC_kuka_laser_laser_message(IPC_kuka_laser_laser_message msg) {
    //
    this.id = msg.id;
    this.config = msg.config;
    this.numReadings = msg.numReadings;
    this.range = msg.range;
    this.numReadings = msg.numReadings;
    this.remission = msg.remission;
    this.timestamp = msg.timestamp;
  }

  //
  // The message to copy.
  //
  // Copy the message to this.
  //
  public IPC_kuka_laser_laser_message(IPC_kuka_laser_laser_message msg) {
    //
    this.id = msg.id;
    this.config = msg.config;
    this.numReadings = msg.numReadings;
    this.range = msg.range;
    this.numReadings = msg.numReadings;
    this.remission = msg.remission;
    this.timestamp = msg.timestamp;
  }
}
```

Listener Thread generally shouldn’t have to wait for publisher to complete sending the message
  – New messages could arrive on the listener while publisher is still busy
• Throttle where necessary, ideally we should be able to always process all messages
  – Mind the Stop-the-world events for garbage collection in Java!
  – Length of queue can be checked → react to overloads, log an error (e.g. discard old msgs)
Communication between Threads via BlockingQueues (java.util.concurrent)

```java
public class LaserTalker implements NodeMain {
    BlockingQueue<sensor_msgs.LaserScan> outQueue = new LinkedBlockingQueue<sensor_msgs.LaserScan>();

    public synchronized double sendLaserScan(IPC_kuka_laser_laser_message msg) // → Called from IPC Listener Thread
    {
        LaserScan outScan = StaticMessageFactory.getMessageFactory().newFromType(sensor_msgs.LaserScan._TYPE);
        outScan.setAngleIncrement((float) (msg.config.angular_resolution));
        ... All the conversion of data ..
        outQueue.add(outScan); // → Put Sensor Data in the Queue and by that wake up other Thread
    }
```
Communication between Threads via BlockingQueues (java.util.concurrent)

```java
@Override
public void onStart(ConnectedNode node) {
    final Publisher<sensor_msgs.LaserScan> publisher = node.newPublisher(_topic, "sensor_msgs/LaserScan");

    // This CancellableLoop will be canceled automatically when the Node shuts down.
    node.executeCancellableLoop(new CancellableLoop()
    {
        @Override
        protected void loop() throws InterruptedException
        {
            sensor_msgs.LaserScan msg = outQueue.take();  // → Will be woken up once a message is available
            publisher.publish(msg);  // → Publish Sensor Data to ROS
            ...
        }
    });
```
Time synchronization – PTP, NTP?

- **vxWorks Sunrise real-time** has its own clock, can’t (easily) be synchronized to an external clock
  - Can’t allow **time shifts during control** of the robot
- Sunrise real-time can provide a PTP (**Precision Time Protocol**) server
  - But: Over Time, controller time will deviate from UTC
  - Windows PTP clients cost $ (e.g. Meinberg ~300€)
  - Can be an option in a closed cell without „outside connections“
- Pragmatic solution:
  - Synchronize **NAV PC** (running IPC↔ROS) to external **NTP**
  - Set all timestamps to `ros::Time::now()` when receiving the IPC msgs
  - Good enough for laser-reprojection used in Localizer in our experience
Time synchronization – stamping „now“

Pragmatic:

```java
public class LaserTalker implements NodeMain {
    private static org.ros.time.WallTimeProvider clock = null;

    public synchronized double sendLaserScan(IPC_kuka_laser_laser_message msg) // → Called from IPC Listener Thread
    {
        LaserScan outScan = StaticMessageFactory.getMessageFactory().newFromType(sensor_msgs.LaserScan._TYPE);
        if (clock == null)
            clock = new org.ros.time.WallTimeProvider();
        outScan.getHeader().setStamp(clock.getCurrentTime());
    }
```

Precise: Calibrate – depending on application
Get current real-time controller time from Java

```java
public static org.ros.message.Time getControllerRosTime(Controller controller) {
    SSR getControllerTimeSSR = new SSR();
    getControllerTimeSSR.setRequestType("GetTimeStamp");
    SunriseController sc = (SunriseController)controller;
    Message responseMsgTimestamp = sc.sendSynchronousSSR(getControllerTimeSSR);

    //1st param seconds 2nd nano seconds
    int seconds = responseMsgTimestamp.getParamIntArray(0)[0];
    int nanoseconds = responseMsgTimestamp.getParamIntArray(0)[1];
    return new Time(seconds, nanoseconds);
}
```

- Caveat: Odometry is stamped by controller, Laser by Nav PC
Hitting CPU Time limitations on the Controller

- „VxWin“ → VxWorks shares CPU with Windows, run on different cores
- ApplicationServer runs the Java application in windows, commands real-time via TCP/IP

→ Move CPU-heavy load to another PC
  - we’re doing ROS → distributed system → should be easy

→ Run the robot application on another PC, maybe even Linux
  - Possible but not recommended (single point of control)
Java Performance & Garbage Collection – a problem?

Typically not the problem but often the first suspect...

- **https://gceasy.io/** is a free website for analysis of Java performance logs
  - C:\KRC\ApplicationServer\AppServer.ini has **JVM parameters**
  - E.g. „XX:+PrintGCDetails -XX:+PrintGCDateStamps -Xloggc:C:\gclog.txt -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCApplicationConcurrentTime“
  - **Log**: 2019-10-22T15:30:32.397+0200: 0.699: [GC2019-10-22T15:30:32.397+0200: 0.699: [DefNew: 4928K->467K(4928K), 0.0034140 secs] 8359K->4401K(15872K), 0.0035066 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]
  - Can also help to see if your application has **memory leaks** that might crash it after X hours

- String operations such as serialization, logging will keep GC busy
- Java comes with different garbage collectors, best free one: Garbage-first GC (-XX:+UseG1GC)
- Azul Zing is a commercial JVM optimized for max ~2ms length Stop-the-world events, also makes jHiccup
TCP Performance between Windows and vxWorks (or Linux,..)

- **Windows** is by default optimized for data throughput instead of small **TCP packet latency**
- Fix:
  - **Registry**: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces\n    - Look for the interface with the IP you’re using to talk to Controller and ROS
  - Create two DWORDs:
    - **TcpAckFrequency** = 1 (acknowledge received TCP packets immediately one by one instead of waiting for a number of them to pile up)
    - **TCPNoDelay** = 1 (send small packets immediately instead of bundling them together with Nagle’s algorithm)

→ Test on diiwa: Java program running on the NavPC polls odometry from controller PC/vxWorks, publishes on ROS topic, „rostopic hz odom“ on third (Linux) PC
→ Improvement from 20Hz to >600Hz
Conclusions

• Many KUKA robots are used with ROS today
• We are working on a more canonical offering
  – Catalog of URDF models
  – Official drivers for KSS and Sunrise robots + kmp mobile platform
• You can help us
  – Let us know you’re interested either as a user or contributor
  – What do you wish for and what issues do you see with the current implementations?
• Let our sales know you need ROS → best lever to help us help you
We’d like you to be happy with our products – let us know how to get there!

Thomas.Ruehr@kuka.com