

# Interreg



EUROPEAN UNION

## Grande Région | Großregion



### Robotix-Academy

Fonds européen de développement régional | Europäischer Fonds für regionale Entwicklung



## Robotix-Academy Summer School

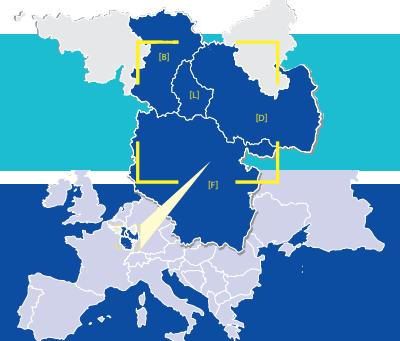
### September 12<sup>th</sup> to 14<sup>th</sup> 2017

### ZeMA, Saarbrücken

Partenaires du projet | Projektpartner:



[www.robotix.academy](http://www.robotix.academy)





---

## Vorwort

Die Robotix-Academy hat 2017 viel unternommen, um sich in der Großregion einen Namen zu machen: Zahlreiche Veranstaltungen fanden bereits statt, sei es, um eine interessierte Öffentlichkeit zu gewinnen oder um sich mit den Partnern auszutauschen und gemeinsam zu forschen. Voneinander zu lernen und dieses Wissen in die Praxis umzusetzen, grenzübergreifend und zum Nutzen der KMU in der Großregion, das ist das Anliegen aller Projektbeteigter aus Wissenschaft und Wirtschaft. Die Projektpartner freuen sich über das wachsende Interesse an der Robotix-Academy und bedanken sich an dieser Stelle bei allen Beteiligten und Förderern für die gute Zusammenarbeit. Das Projekt „Robotix-Academy“ wird im Rahmen des Programms INTERREG V A Großregion 2014-2020 gefördert und mit bis zu 60% aus EFRE-Mitteln (Europäischer Fonds für regionale Entwicklung) kofinanziert. Ziel ist es, einen dauerhaften Forschungscluster für industrielle Robotik in der Großregion zu etablieren. Die Academy dient den beteiligten Hochschulen und Forschungseinrichtungen, Transferpartnern, Anwender- und Ausrüsterunternehmen als Kooperationsplattform. Das ZeMA ist hier federführend, neben den beteiligten Projektpartnern aus Belgien, Luxembourg, Lothringen und Rheinland-Pfalz. Eine wesentliche Aufgabe der Academy besteht darin, Know-how aufzubauen und für die industrielle Praxis bereitzustellen. Hier werden vor allem KMU bei der Einführung neuer Technologien unterstützt, aber auch Großunternehmen profitieren von den außeruniversitären Bildungs-, Qualifizierungs- und Beratungsangeboten. Modernstes Equipment für praxisnahe Demonstrationen und Versuche werden von den Hochschulen gemeinsam genutzt und Forschungsergebnisse überregional verwertet. Zahlreiche grenzüberschreitende Kooperationsprojekte wird die „Robotix-Academy“ in den nächsten Jah-

## Préface

En 2017, la Robotix-Academy s'est beaucoup investie afin de se faire connaître en Grande Région : De nombreux événements ont déjà eu lieu visant à attirer l'attention d'un public intéressé ou bien à échanger des informations avec des partenaires et pour faire de la recherche en commun. Apprendre mutuellement et mettre en pratiques ces connaissances, au-delà des frontières et au bénéfice des PME en Grande Région, ce sont des objectifs principales poursuivis par tous les responsables du projet soit du domaine scientifique ou que du domaine économique. Les partenaires du projet sont très contents de l'intérêt croissant apporté à la Robotix-Academy et voudraient saisir cette occasion pour remercier tous les participants et parrains pour cette excellente collaboration. Le projet « Robotix-Academy » est subventionné dans le cadre du programme Interreg V A Grande Région 2014-2020, et il est cofinancé à hauteur de 60% par les fonds FEDER (Fonds Européen de Développement Régional). Le but est d'établir un pôle de recherche durable pour la robotique industrielle dans la Grande Région. L'Academy sert de plateforme de coopération aux universités et aux instituts de recherche ainsi qu'aux partenaires de transfert, aux entreprises utilisatrices, et aux fournisseurs. Le centre ZeMA coordonne le projet aux côtés des partenaires participants basés en Belgique, au Luxembourg, en Lorraine et en Rhénanie-Palatinat. Construire un savoir-faire applicable à la pratique industrielle est le devoir crucial de l'Academy. Ainsi, les PME bénéficient en particulier d'un soutien en matière de mise en place des nouvelles technologies. Toutefois, les grandes entreprises profitent également des offres de formation, de qualification et de préparation extra-universitaires. Les équipements les plus modernes pour faire des démonstrations et des essais pratiques sont utilisés en commun par les universités, et leurs résul-

ren initiieren und so, ganz im Sinne des INTERREG Programms, die Innovationskapazitäten und die Wettbewerbsfähigkeit der Großregion stärken.

tats sont utilisables au niveau suprarégional. La Robotix-Academy initiera de nombreux projets de coopération transfrontalière dans les années à venir, et contribue ainsi à renforcer la compétitivité et l'attractivité de la Grande Région. Ces objectifs s'inscrivent d'ailleurs parfaitement dans la logique du programme Interreg.

---

# Inhaltsverzeichnis

Vorwort	1
Préface	1
Robotix-Academy Summer School	1
Bosch APAS Workshop	4
Atelier Bosch APAS	4
Zertifizierung eines Cobots	12
Certification d'un Cobot	12
Einführung in ROS und InteraSDK	22
Introduction to ROS and InteraSDK	22
Human-Robot-Interaction	54
Pôle MecaTech, Impulsgeber für Innovationskraft im Maschinenbau	84
Le pôle MecaTech, moteur d'innovation en génie mécanique	84
Kontakt	102
Contact	102





## 1st Robotix Academy Summer School

Saarbrücken, September 12<sup>th</sup> to 14<sup>th</sup>



# Robotix-Academy Summer School

Die erste Robotix-Academy Summer School fand vom 12. bis 14. September 2017 am ZeMA in Saarbrücken statt. Wissenschaftliche Mitarbeiter aller Partnerhochschulen kamen zusammen, um in verschiedenen Workshops voneinander zu lernen. Der Einstieg in die drei Workshop-Tage erfolgte mit einem Pitch jedes Workshop-Referenten. Dabei ging es vor allem darum, den anderen Teilnehmern innerhalb von ein bis zwei Minuten zu erklären, was sie erwartet. Im ersten Workshop wurde den Teilnehmern der Bosch APAS assistant von Ali Kanso (ZeMA) vorgestellt. In einem praktischen Teil konnten die Teilnehmer die theoretisch erlernten Grundlagen am Roboter anhand einer Aufgabenstellung umsetzen. Jean Denoël vom Projektpartner Pôle MécaTech gab im Anschluss den Teilnehmern einen Einblick in den Aufbau und die Arbeit des Clusters sowie eines neuen Industrie 4.0 Projektes. Am Abend fand dann ein Teambuilding-Event statt und im Anschluss konnten sich alle Teilnehmer bei einer Stunde Bubble Soccer auspowern und ihren Teamgeist zeigen. Am zweiten Tag begann Miriam Drieß (ZeMA) mit einer Einführung in die Zertifizierung eines Cobots. Als Teil der Risikoanalyse wurde das Thema „Kraft- und Druckgrenzwerte bei Kollisionen zwischen Mensch und Roboter“ genauer betrachtet. Mit einem speziell designten Messgerät konnte in einer praktischen Übung das Messen und Analysieren solcher Grenzwerte veranschaulicht werden. Im Anschluss übernahmen Robin Pellois und Arthur Lismonde (Universität Liège) mit einem Workshop zu ROS (robot operating system). Nach einer kurzen Vorstellung ihres Sawyer Roboters

La première école d'été de la Robotix Academy s'est déroulée du 12 au 14 septembre 2017 au centre ZeMA de Sarrebruck. Les collaborateurs scientifiques de chaque université partenaire se sont réunis pour échanger lors de différents ateliers. Chacun des trois jours de workshops a été inauguré par le discours de l'intervenant concerné. L'objectif premier était d'expliquer brièvement aux participants ce qui était attendu d'eux. Lors du premier workshop, les participants ont pu découvrir l'assistant Bosch APAS grâce à la présentation tenue par Ali Kanso (ZeMA). Au cours d'une partie plus pratique, les participants ont pu mettre en application les bases théoriques apprises sur des robots, et ce par le biais d'une tâche confiée. À l'issue de cet exercice, les participants ont été introduits à la mise en place et au travail dans les pôles industriels et dans le cadre d'un nouveau projet d'industrie 4.0. Cette partie a été menée par Jean Denoël, représentant à cette occasion le partenaire de projet Pôle MécaTech. Un événement de consolidation d'équipes a été organisé au soir, à l'issue duquel l'ensemble des participants a pu mettre à l'épreuve son esprit d'équipe lors d'une partie d'une heure de Bubble Soccer. Le coup d'envoi de la deuxième journée a été donné par Miriam Drieß (ZeMA), avec une introduction à la certification d'un Cobot. Le thème “valeurs limites de puissance et de pression lors de collisions entre l'homme et le robot” a été approfondi dans le cadre d'une analyse de risques. La mesure et l'analyse de ces valeurs limites ont pu être davantage concrétisées lors d'un exercice fait à l'aide d'un appareil de mesure spécialement pensé pour ce contexte.

erlernten die Teilnehmer unter Anleitung von Pellois/Lismonde, wie der Roboter über ROS angesteuert werden kann. Der letzte Workshop wurde am dritten Tag von Sebastian Groß, Jan Jungbluth und Thomas Bartscherer zum Thema Mensch-Roboter-Interaktion durchgeführt. Ziel war es, eine Kommunikation über PC mit einem Universal Robot herzustellen. Zum Abschluss der drei Workshop-Tage ging es für die Teilnehmer nach Hambach zur Smartville. In einer Werksführung lernten die Teilnehmer die Produktion eines Smart for two kennen.

te. La journée a continué avec la prise de parole de Robin Pellois et Arthur Lismonde (Université de Liège) avec un workshop consacré au ROS (robot operating system). Après une courte présentation du robot Sawyer, et en suivant les instructions de Pellois et Lismonde, les participants ont appris comment les robots peuvent être activés par ROS. Le dernier atelier a été mené par Sebastian Groß, Jan Jungbluth et Thomas Bartscherer. Il tournait autour du thème de l'interaction homme-robot. Ces trois jours de formation se sont finis par une visite de la Smartville de Hambach. Les participants ont pu y découvrir les étapes de production d'une Smart for two.



## 1st Robotix Academy Summer School

Saarbrücken, September 12<sup>th</sup> to 14<sup>th</sup>



HOCHE SCHULE TRIER  
Umwelt-Campus Birkenfeld



UNIVERSITÉ DU  
LUXEMBOURG



UNIVERSITÉ  
DE LORRAINE



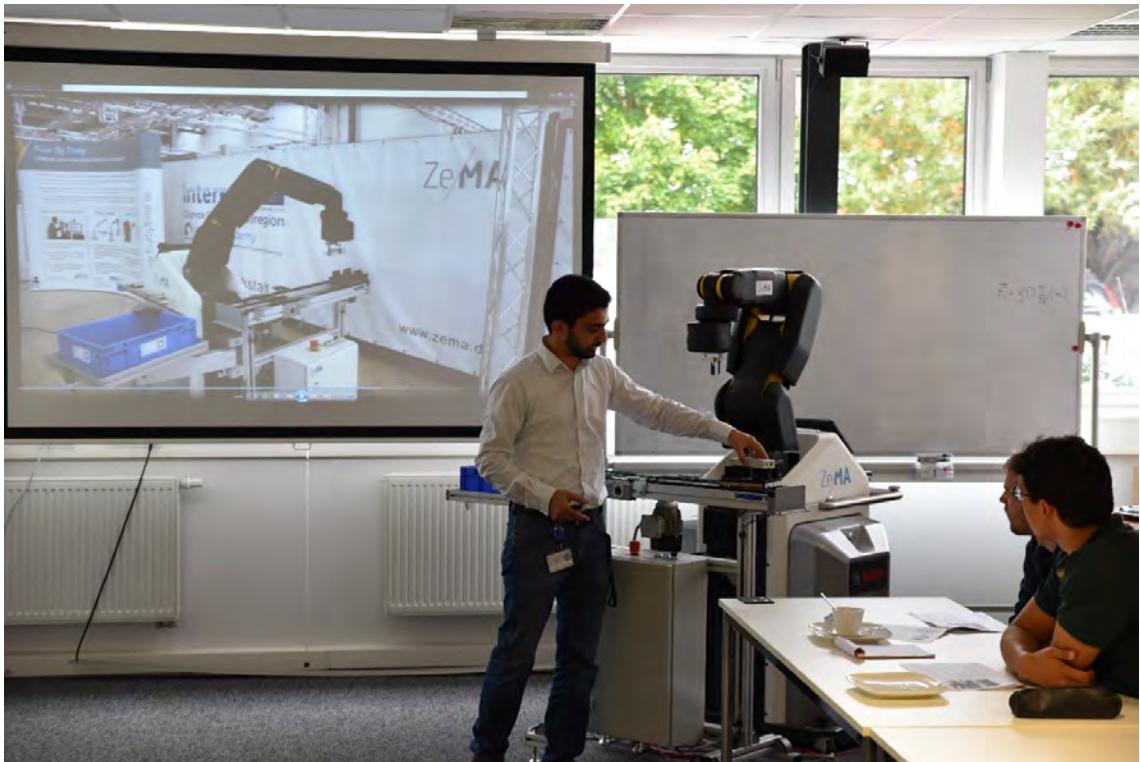
PÔLE  
MECA TECH



LIÈGE  
université

## Timetable

Monday	Tuesday	Wednesday	Thursday	Friday
	11.00 – 11.45 Welcome & Introduction ZeMA	8.30 – 10.00 Certification of a cobot Part 1	8.30 – 10.00 Human-Robot-Interaction Part 1	
	11.45 – 12.30 APAS Part 1	10.15 – 12.30 Certification of a cobot Part 2 ROS Part 1	10.15 – 12.30 Human-Robot-Interaction Part 2	
	12.30 – 13.15 Lunch	12.30 – 13.30 Lunch	12.30 – 13.15 Lunch	
	13.15 – 15.15 APAS Part 2	13.30 – 14.30 ROS Part 2	13.15 – 16.30 (ca.) Exkursion to Smart Hambach	
	15.30 – 17.00 Pôle MecaTech	14.45 – 17.00 ROS Part 3	Feedback & Goodbye	



## Bosch APAS Workshop

Mensch-Roboter-Kooperation ist in aller Munde. Der Bosch APAS assistant ist ein speziell für diesen Bereich entwickelter Roboter. Im Bosch APAS Workshop war es das Ziel, den Teilnehmern die Grundlagen des Roboters sowie die Umsetzung dieser in einem Fallbeispiel zu vermitteln. Dafür wurde in einem theoretischen Teil aufgezeigt, wie der APAS aufgebaut ist und was ihn für die sichere Mensch-Roboter-Kooperation auszeichnet. Dies ist vor allem seine Sensorhaut, die den Roboter bereits stoppt, bevor der Mensch ihn überhaupt berührt. Weiterhin wurde die benutzerfreundliche Bedienung erklärt, die die Teilnehmer im praktischen Teil selbst ausprobieren konnten. Auf einer mit Sensoren ausgestatteten Linie sollte der APAS je nach Position des Werkstückträgers eine Aktion durchführen. Die Aktion bestand daraus, ein Werkstück vom Werkstückträger in ein KLT zu legen und umgekehrt. In dieser Aufgabenstellung konnte die Anwendung verschiedener Programmierbausteine des Roboters kennengelernt werden, von Pick & Place zur

## Atelier Bosch APAS

La coopération homme-robot est sur toutes les lèvres. L'APAS assistant de Bosch est un robot spécialement conçu pour ce domaine. Le but de l'atelier Bosch était de transmettre les bases du robot ainsi que son application dans le cadre d'une étude de cas. Une partie théorique montrait comment l'APAS est construit et qu'est-ce qui le qualifie pour la coopération homme-robot sécurisée. Cela est avant tout sa peau sensorielle qui fait arrêter le robot, avant que même l'homme puisse le toucher. De plus, on a expliqué l'utilisation conviviale que les participants pouvaient essayer eux-mêmes lors de la partie pratique. L'APAS devait exécuter une action selon la position du porte-pièce sur une ligne équipée de capteurs. L'action consistait en posant une pièce du porte-pièce dans un KLT et à l'inverse. Pendant cette mission on pouvait découvrir l'application de différents éléments du programme du robot, de Pick & Place jusqu'à la détection de caractéristiques spécifiques d'une pièce grâce aux deux caméras intégrées. Les participants ont donc

Erkennung von spezifischen Merkmalen eines Werkstücks mittels der zwei integrierten Kameras. Die Teilnehmer hatten somit die Gelegenheit, innerhalb kürzester Zeit die grundlegenden Funktionen sowie die Programmierung des Bosch APAS kennenzulernen.

**Kontakt:**

ZeMA  
Zentrum für Mechatronik und Automatisierungstechnik gGmbH  
Ali Kanso  
E-Mail: [a.kanso@zema.de](mailto:a.kanso@zema.de)

eu l'occasion de découvrir les fonctions de bases ainsi que la programmation du Bosch APAS dans les plus brefs délais.

**Contact:**

ZeMA  
Zentrum für Mechatronik und Automatisierungstechnik gGmbH  
Ali Kanso  
e-mail: [a.kanso@zema.de](mailto:a.kanso@zema.de)

## Project work – APAS

Ali Kanso M.Sc.

ZeMA - Zentrum für Mechatronik und  
Automatisierungstechnik gemeinnützige GmbH

Saarbrücken, 12.09.2017 – 14.09.2017

### Task for practicing on the APAS Assistant

1. Teach-in reference coordinate system
2. Create a triangle path
3. Move an object
  - Grip an object
  - Put it in end position
  - Return to initial position
4. Reconfiguration of the robot cell
5. Grip an object via camera-based contour detection





- Moving the marker via the hand control to a distance of 20 cm from the reference marker
- Checking the detection via stereo camera (marker appears green)
- Saving the robot pose



## Basic functions



- Turn on the main switch on the switchboard
  - The power supply will switch on.
  - The start menu will appear on the touch panel.
- Push "Control on / function proceed" (1) on the control panel.
- Press the pushbutton (4) of the enabling button (center position) and hold it down during the initialization run
- Press the button "Start" to start the initialization of the sensor skin
  - The sensor skin will be initialized.
  - APAS drives in park position.
  - After successful initialization, the start window appears with the saved work lists.
- Release the pushbutton of the enabling button.
  - APAS is ready for operation.

### „APAS assistant“

### Flexible, mobile automation solution for Human-Robot-Cooperation

#### Bosch APAS assistant



- Robot system for a direct Human-Robot-Cooperation
- Maximum range 911 mm
- Maximum load capacity 5 kg
- Own weight 230 kg (with cart)
- TCP speed up to 500 mm/s in the human-robot-cooperation

#### Features:

- Contactless safety sensor
- Simple programming interface
- Stereo camera system optionally integrated in the gripper
- Certified by the German employers' liability insurance association

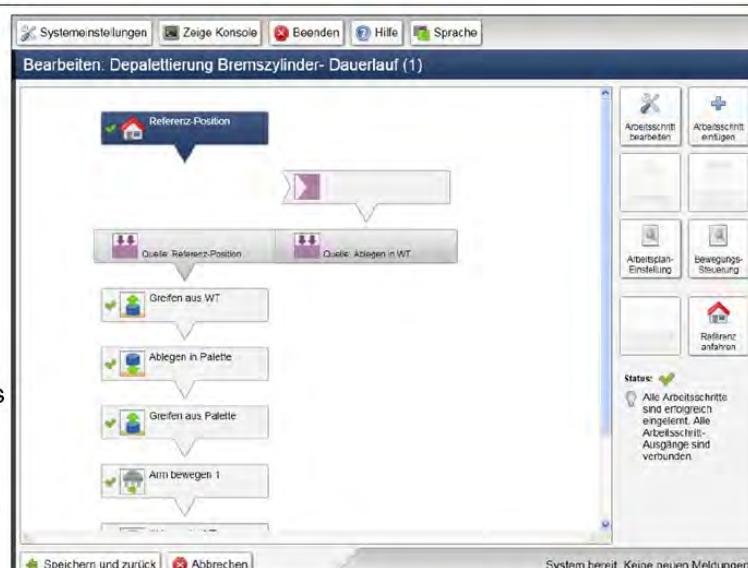
Images: Bosch

© ZeMA gGmbH

Seite 5



### Intuitive programming of the robot system with graphical user interface of a work plan



- Simple configuration of new production schedules
- “Teach-In Assistant” guides the operator through the configuration process

Images: Bosch

© ZeMA gGmbH

Seite 6



## Flexible workplace rotation of the APAS in a human-robot-cooperation



The reference positioning via markers allows the determination of the robot position in relation to its working range. This allows the process to be performed with high precision, even if the robot's location varies slightly.



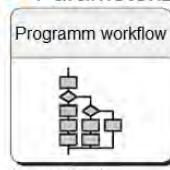
© ZeMA gGmbH

Seite 7



## Offline-Programming: Program flow chart

- Program flow charts are more clearly laid out and therefore less prone to error than textual representations.
- The programming is done graphically through icons, e.g. for
  - Removing a component
  - Inserting a component
  - Assembling of a pallet
- The programming doesn't depend on textual syntax.
- Parameterization via an entry mask



Images: Bosch

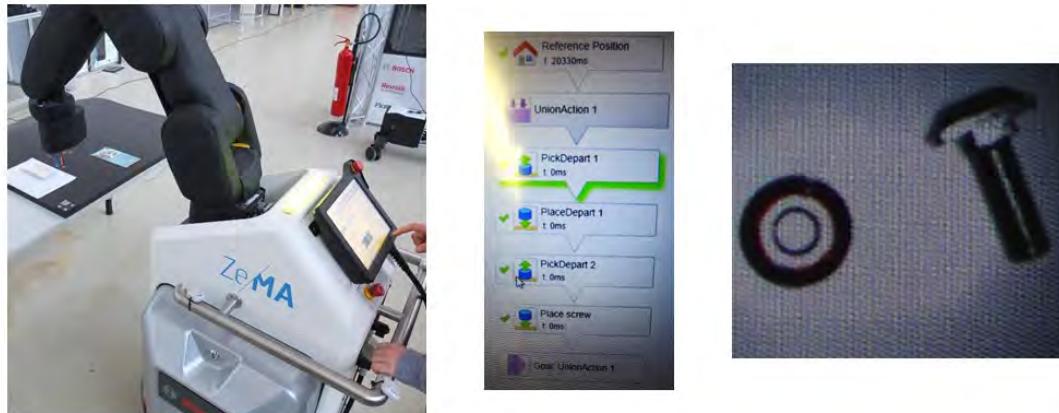
© ZeMA gGmbH



Seite 8

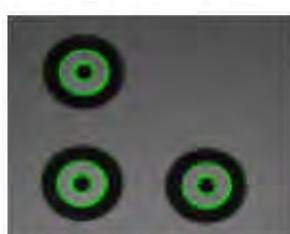


## Project Work – Bosch APAS



- Pick & Place task of bolts and nuts
  - Detect the bolts and nuts in a defined area and do a differentiation as well as a separation
  - Bolts and nuts should be palletized to two different plates

## Pick & Place Program flow chart



Calibrate the reference coordinate system

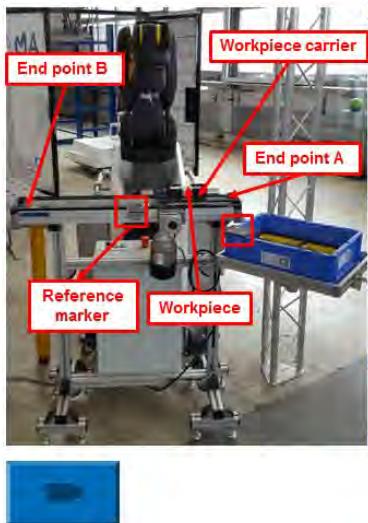


Teach-in the object

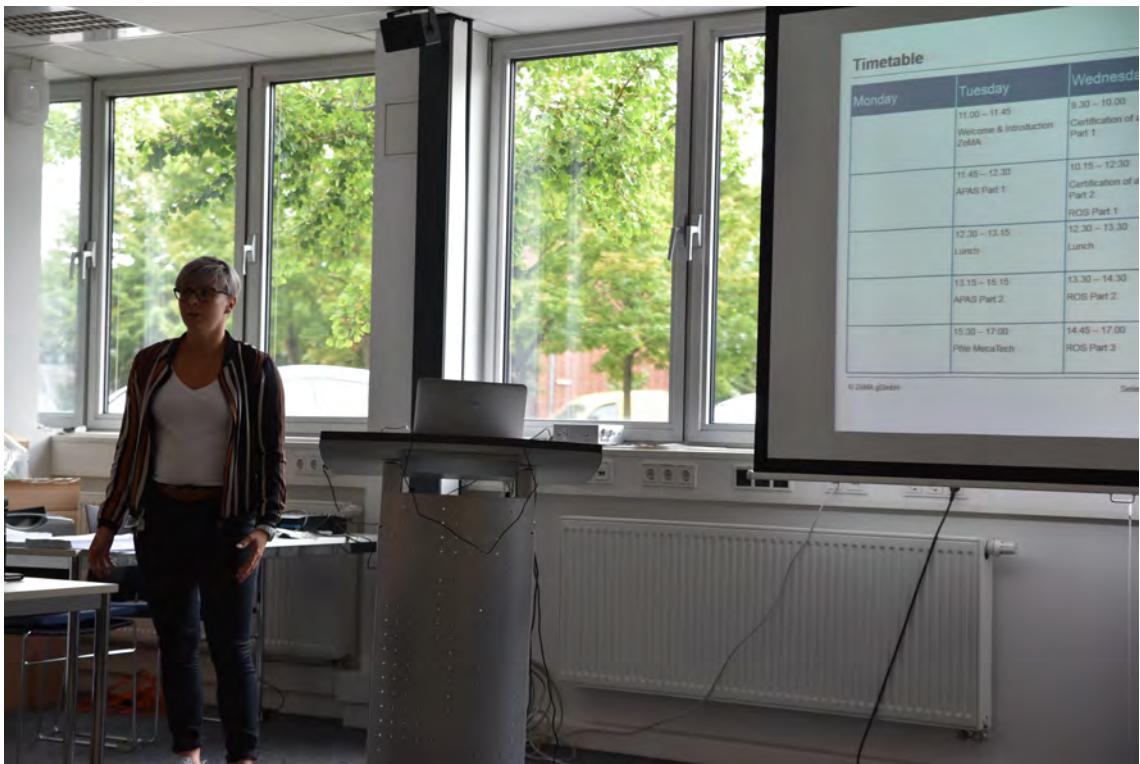
Detection and palletizing of an object:

1. Start the system, initialize the contactless safety sensor skin, create a new program
2. Teach-in reference coordinate system
3. Object recognition through camera and subsequent gripping of the object
4. Palletizing the objects
5. Loop creating

## Workshop application



- Bosch APAS communicates with production line and sends workpiece carrier to end point A
- Workpiece carrier gives a signal when it reaches the end point A
  - The detection is based on the signal of induction sensors
- Robot searches for the reference marker of the production line
- Robot grasps the workpiece and searches for the reference marker of the blue box
- Robot palletizes the workpiece in the blue box and sends the workpiece carrier to end point B
- Once the workpiece carrier reaches the end point B, it will be detected and the robot grasps the workpiece from the blue box and places it in the workpiece carrier again.
- Robot sends workpiece carrier to end point A → replay



## Zertifizierung eines Cobots

Der Hersteller eines Produktes belegt im europäischen Wirtschaftsraum die Einhaltung aller rechtlichen Anforderungen mit dem CE-Kennzeichen. Auch für einen Roboter wird eine solche CE-Kennzeichnung ausgestellt. Für einen Roboter der Mensch-Roboter-Kooperation (MRK) gelten die sicherheitstechnischen Normen EN ISO 10218 Teil 1 und 2, die ISO/TS 15066 und die Maschinenrichtlinie, die eingehalten werden müssen. Auf dem Weg zur CE-Kennzeichnung werden verschiedene Schritte für die Zertifizierung durchlaufen. Einer dieser Schritte ist die Risikoanalyse des Roboters. Trotz sicherheitstechnischer Maßnahmen kann eine Kollision zwischen Roboter und dem Menschen nicht vollständig vermieden werden. Die ISO/TS 15066 gibt Grenzwerte für Kraft und Druck bei der Kollision eines MRK-Roboters mit dem menschlichen Körper für 29 spezifische Stellen an. Innerhalb der Risikoanalyse werden diese Grenzwerte validiert, um dem Roboter seine CE-Kennzeichnung ausstellen zu können. Diese Grundlagen wurden den Teilneh-

## Certification d'un Cobot

Dans l'Espace économique européen, le fabricant d'un produit prouve le respect de toutes exigences légales avec le marquage CE. Un tel marquage CE est aussi apposé sur un robot. Pour un robot de la coopération homme-robot (CHR), les normes de sûreté EN ISO 10218 partie 1 et 2, la ISO/TS 15066 et la directive machine s'appliquent et doivent être respectés. Sur le chemin vers le marquage CE on doit passer à la certification par de différentes étapes. Une de ces étapes est l'analyse des risques des robots. Malgré des mesures de sûreté une collision entre le robot et le corps humain ? La ISO/TS 15066 indique des limites pour la force et la pression pendant une collision d'un robot CHR avec le corps humain pour 29 endroits spécifiques. Ces limites sont validés au long de l'analyse des risques afin d'apposer le marquage CE sur le robot. Dans le cadre d'un petit atelier, ces bases sont expliquées aux participants de la Robotix-Academy Summer School. De plus, un des systèmes de mesure actuellement disponible sur le marché pour la validation des

mern der Robotix-Academy Summer School in einem kurzen Workshop erläutert. Weiterhin wurde eines der derzeit am Markt verfügbaren Messsysteme zur Validierung der Grenzwerte vorgestellt. Anhand eines praktischen Beispiels konnte das Messsystem sowie die zugehörige Software zur Analyse der Ergebnisse von den Teilnehmern getestet werden.

**Kontakt:**

ZeMA  
Zentrum für Mechatronik und Automatisierungstechnik gGmbH  
Miriam Drieß  
E-Mail: [m.driess@zema.de](mailto:m.driess@zema.de)

limites a été présenté. A l'aide d'un exemple pratique, le système de mesure ainsi que le logiciel associé pour l'analyse des résultats des participants pouvait être testé.

**Contact:**

ZeMA  
Zentrum für Mechatronik und Automatisierungstechnik gGmbH  
Miriam Drieß  
e-mail: [m.driess@zema.de](mailto:m.driess@zema.de)



## Certification of a cobot – CE marking

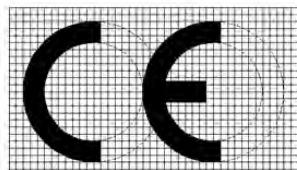
Force measuring on cobots with the KMG 500

Miriam Drieß, M.Eng.

Robotix Academy Summer School

Saarbrücken, September 12<sup>th</sup> to 14<sup>th</sup>

### What does a CE mark stands for?



- CE – Conformité Européenne
- On products traded on the single market in the European Economic Area (EEA)
- Products with CE mark meet high safety, health and environmental requirements

With the CE mark, a manufacturer declares that the product meets all the legal requirements for CE marking and can be sold throughout the EEA.

## Steps to get a CE marking



1. Product definition
2. Supplier monitoring
3. Choose the right guidelines and norms
4. Choose the possible conformity assessment procedure
5. Bring in a named authority (e.g. type examination)
6. **Conduct of the risk assessment**
7. Warranty the compliance of the guidelines
8. Provision of information
9. Author the technical documentation
10. Review and correction of the instruction manual
11. Invoice the declaration of conformity
12. Application of the CE marking

## CE marking includes risk assessment

- Basis for the risk assessment are EN ISO 10218 parts 1 and 2 and the Machinery Directive
- ISO/TS 15066 is giving additional safety requirements for cobots
- All requirements have to be verified before marking the product with the CE mark

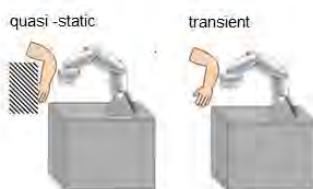
A comprehensive risk assessment is required to assess not only the robot system itself, but also the environment in which it is placed, i.e. the workplace.

## Validation on two types of impact is needed



- Four methods for collaborative operations

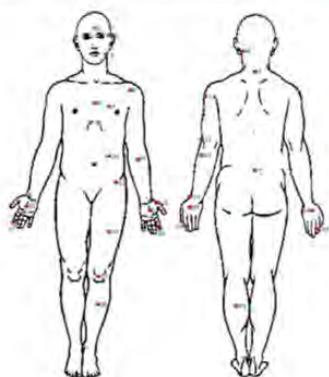
- 1 Safety-rated monitored stop
- 2 Hand guiding
- 3 Speed and separation monitoring
- 4 Power and force limiting (PFL)



- Two types of impact : "quasi-static contact" and "transient contact"

For a robot using the Power and force limiting (PFL) method the limits for quasi-static and transient contact have to be validated.

## Force/pressure measurement as a part of the risk assessment



- ISO/TS 15066 gives a maximum collision force and a local maximum pressure occurring on 29 specified parts of the human body
- Testing of an accepted level takes place in the risk assessment procedure
- Use of a mechanic humanoid measurement system

## How much pressure/force do you think is allowed to the following body parts?

Body Region	Specific Body Area	Front/ Rear
Skull and forehead	1 Middle of forehead	Front
	2 Temple	Front
Face	3 Masticatory muscle	Front
	4 Neck muscle	Rear
Neck	5 Seventh neck vertebra	Rear
	6 Shoulder joint	Front
Back and shoulders	7 Fifth lumbar vertebra	Rear
	8 Sternum	Front
Chest	9 Pectoral muscle	Front
	10 Abdominal muscle	Front
Abdomen	11 Pelvic bone	Front
	12 Deltoid muscle	Rear
Pelvis	13 Humerus	Rear
	14 Arm nerve	Front
Upper arms and elbow joints	15 Radial bone	Rear
	16 Forearm muscle	Rear
Lower arms and wrist joints	17 Forefinger pad D	Front
	18 Forefinger pad ND	Front
Hands and fingers	19 Forefinger end joint D	Rear
	20 Forefinger end joint ND	Rear
Thighs and knees	21 Thenar eminence	Front
	22 Palm D	Front
Lower legs	23 Palm ND	Front
	24 Back of the hand D	Rear
	25 Back of the hand ND	Rear
	26 Thigh muscle	Front
	27 Kneecap	Front
	28 Middle of shin	Front
	29 Calf muscle	Rear

D = dominant body side (right or left); ND = non-dominant body side

## Company GTE offers three systems for measuring force & pressure



- System A: KMG 500 KOLROBOT
- System B: KDMG KOLROBOT with Fuji® Prescale
- System C: KDMG KOLROBOT with TEKSCAN®

## System A: KMG 500 KOLROBOT – the simplified system



- Developed in cooperation with BGHM
- Force-time diagram is recorded
- Measurement data can be analyzed with the FPM vision software
- Fixed spring with spring constant 75 N/mm - leads to a higher force as with a spring constant of 25 N/mm which would be closer to the human body



- Fuji® PreScale is used for pressure distribution
- Color change is analyzed by software and converted into a pressure distribution

## System B: KDMG KOLROBOT with Fuji® PreScale



- Developed from IFA
- Linear-guided measuring mechanics and Piezo force transducer
- Replaceable compression spring
- Compression elements can be mounted on measuring plate
- Sensors are connected to prewired measuring amplifier with data logger
  
- PILZ also offers a collision measurement set with this functionality called „PROBms“

### System C: KDMG KOLROBOT with TEKSCAN®



- System with Piezo sensor exactly as System B



- TEKSCAN® sensor films take place on the compression elements on the collision surface
- Pressure distribution of the collision is recorded and shown as 3D profile

### Case: measuring a specific case of the Bosch APAS assistant

#### Bosch APAS assistant



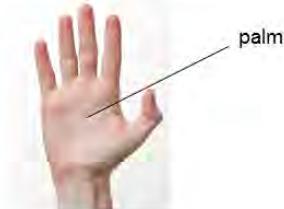
- Maximum range 911 mm
- Maximum load capacity 5 kg
- Own weight 230 kg (with cart)
- Axis speed up to 500 mm/s in the human-robot-cooperation

#### Features:

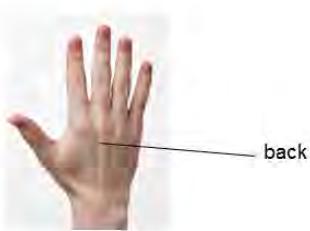


- Contactless safety sensor
- Simple programming interface
- Stereo camera system optionally integrated in the gripper
- Certified by the German employers' liability insurance association

## Measuring the force and pressure of gripper



- APAS assistant captures every risk of collision between human and robot except for the gripper
- The three fingers of the gripper deflect in case of a collision → avoids pinching and jamming



- Exercise: Finding out the force and pressure occurring on the human body when the three fingers collide with it.
- Chosen body parts:
  - Palm
  - Back of the hand

## Measuring setup and measuring process



- Place KMG 500 secured, placed and aligned
- Force effect should be rectangular to the measuring surface
- Bosch APAS should be programmed to fulfill that requirement
- Required compression element also secured on the measuring surface (use adhesive stripes to secure)
- Measuring temperature and relative humidity with device
- Fuji-Prescale on compression element in a suitable size



Start KMG 500 and then the APAS program to measure

## Software configuration for analyzing the measurement

---

- Create a new data report „File\New“
- Connect the force gauge
- In the top right area: select force sensor and pressure sensor
- Select the type of film used
- Switch to „Biomechanical configuration“ and choose the body selection and then select the suitable measuring point using „Specific location“
- „Temperature“ and „Relative humidity“ have to be filled with the data of the measuring device (temperature and relative humidity are influencing the Fuji films)



## Einführung in ROS und InteraSDK

ROS (Robotic Operating System) ist ein großes Panel verschiedener Buchhandlungen und Programme, das das Design von Roboteranwendungen vom Betriebssystem bis hin zum Userinterface oder simplen Steuerprogramm erlaubt. Auf Linux basierend und in C++ oder Python geschrieben, ist ROS ein Projekt mit freier Lizenz und zahlreichen Mitwirkenden aus der ganzen Welt. Somit sind zahlreiche Codezeilen für eine große Auswahl von Anwendungen verfügbar und ROS ist mit einer sehr großen Zahl von Robotern und anderen Softwares kompatibel. Diese Eigenschaften verleihen ihm einen gewissen Vorteil in der wissenschaftlichen Forschung. In einem ersten Schritt haben wir uns also mit den verschiedenen Teilelementen von ROS vertraut gemacht. Wie sieht die Architektur eines Workspace aus? Wie erstellt man die nodes, die den aktiven Teil des ROS bilden? Wie kommunizieren die verschiedenen Elemente untereinander, durch messages und topics oder durch ser-

## Introduction to ROS and InteraSDK

ROS, pour Robotic Operating System, est un large panel de librairies et de programmes divers permettant la conception d'application robotiques allant du système d'exploitation à l'interface utilisateurs ou au simple programme de commande. Basé sous Linux et écrit en C++ ou Python, ROS est un projet sous licence libre avec de nombreux contributeurs venant du monde entier. Ainsi de nombreuses lignes de codes sont disponibles pour un vaste choix d'applications et ROS est compatible avec de très nombreux robots ainsi que d'autres logiciels. Ces caractéristiques lui confèrent un avantage certain pour la recherche scientifique. Dans un premier temps nous avons donc apprivoisé les différents éléments constituant de ROS. Quelle est l'architecture d'un workspace? Comment créer les nodes qui constituent la partie active de ROS? Comment les différents éléments communiquent entre eux, via messages et topics ou via services? Dans un deuxième temps

vices? In einem zweiten Schritt lag das Ziel darin, eine konkrete ROS - Anwendung in der Robotik zu sehen. Dafür haben wir den Sawyer Roboter von Rethink Robotics genutzt. Dieser Roboter besitzt zwei auf ROS basierende Betriebssysteme: Intera (Haupt-OS nur graphisch) und Intera SDK (OS in der Befehlszeile und sehr offen). Mit diesem zweiten Teil möchten wir die Philosophie von ROS in Interaktion mit einem Roboter aufzeigen. Wir können also bestimmte Funktionen aktivieren oder deaktivieren und sogar neue erstellen. Das Ziel des im Rahmen der Robotix-Academy Summer School angebotenen Workshops lag darin, einen ersten Eindruck von diesem Werkzeug zu vermitteln und die Grundlagen kennenzulernen. Sind diese Grundlagen einmal erworben, werden die Teilnehmer dazu fähig sein selbstständig bei der Nutzung von ROS voranzuschreiten.

**Kontakt:**

Universität Liège  
Arthur Lismonde  
E-Mail: alismond@ulg.ac.be

Robin Pellois  
E-Mail: robin.pellois@ulg.ac.be

l'objectif était de voir une application concrète de ROS en robotique. Pour cela nous avons pris comme support le robot Sawyer de Rethink Robotics. Ce robot possède deux systèmes d'exploitation basés sur ROS : Intera (OS principal uniquement graphique) et InteraSDK (OS en ligne de commande et très ouvert). Lors de cette deuxième partie nous cherchions à montrer la philosophie de ROS en interaction avec un Robot. Nous pouvions alors activer ou désactiver certaines fonctions et même en créer de nouvelles. L'objectif du workshop proposé à la Robotix Academy Summer School visait à donner une première impression de cet outil et d'en connaître les bases. Une fois ces bases acquises, les participants seront capables de progresser seul dans l'utilisation de ROS.

**Contact:**

Université de Liège  
Arthur Lismonde  
e-mail:alismond@ulg.ac.be

Robin Pellois  
e-mail:robin.pellois@ulg.ac.be



## Robotix-Academy Summer-School 2017

September 12<sup>th</sup> to 14<sup>th</sup> 2017



# Introduction to ROS

## and InteraSDK



WorkShop presented by Arthur Lismonde and Robin Pellois



## Table of contents

1 – About ROS (Robotic Operating System).....	3
a – Introduction.....	3
b – Software installation.....	3
i – Ubuntu Installation.....	3
ii – ROS installation.....	4
c – ROS overview.....	4
2 – ROS first trial.....	6
a – Create a workspace.....	6
b – Create a package.....	6
c – What is a node?.....	7
i – Playing with turtles.....	7
ii – Writing your own node.....	9
d – Topics and messages.....	11
i – How topics and messages work ? (With turtles).....	11
ii – Create your own topic and message.....	12
Creating the message type.....	12
Creating the topic using the message type created above.....	13
To publish on a topic in a node.....	16
e – Services.....	18
i – Create your own service type.....	18
ii – Create and use a service.....	18
3 – InteraSDK : Discovering ROS and Tom (Sawyer).....	20
a – Create an adapted workspace.....	20
b – Examples.....	21
c – Topics relative to Sawyer.....	22
d – How to create a ROS package and node to command the robot.....	24
i – Create a node.....	24
ii – Program a simple node and Rosbag.....	25
iii – Program a node.....	27
e – A graphic interface : RVIZ and a trajectory planning framework : MoveIt! (For Your Information).....	28
i – Installation.....	28
ii – First step.....	28
Conclusion.....	30

# 1 – About ROS (Robotic Operating System)



## a – Introduction

**Definition :** ROS is a set of programs helping to simulate and program robots. « A library with a structure that allows you to build your own programs, functions »

### ROS features/asset :

- ROS is working only on Unix based platform (mostly Ubuntu or Debian).
- ROS can work with other softwares ( OpenCV, MoveIt...) and a lot of robots (<http://robots.ros.org/>)
- ROS is open-source and free making it a well documented software (cf : <http://wiki.ros.org/ROS/>)
  - ROS programs are written in Python or C++. ROS doesn't come with a graphic interface by default, it has to be handled using commands in a terminal.
  - ROS can work with a lot of robots. The advantage is that it uses the same language to command different robots.

### ROS distributions : (<http://wiki.ros.org/Distributions>)

- 1st distribution : ROS Box Turtle (2010)
- ...
- 8th distro : **ROS Indigo Igloo** (2014) : the one enable to program the Sawyer robot.
- 11th distro : ROS Lunar Loggerhead (2017)

## b – Software installation

### i – Ubuntu Installation

- Download and install a virtual machine :

<https://www.virtualbox.org/>  
or your favorite one

- Download Ubuntu (make sure you download the 14.04 version) :

WeTransfer link :

<https://wetransfer.com/downloads/cb0f70db76d097098424dac2b6eabd5c20170904084645/1c0d837aa95626bf1dc4d521faa2df0e20170904084645/aa7f83>

Ubuntu mirror link :

<ftp://ftp.free.fr/mirrors/ftp.ubuntu.com/releases/14.04/>  
Download the “i386” for an 32 bit processor or the “amd64” version for a 64 bit processor

Index de <ftp://ftp.free.fr/mirrors/ftp.ubuntu.com/releases/14.04/>

[Vers un rép. de plus haut niveau](#)

Nom	Taille	Dernière modification
FOOTER.html	1 KB	04/08/2016 00:00:00
HEADER.html	3 KB	04/08/2016 00:00:00
MD5SUMS	1 KB	16/02/2017 00:00:00
MD5SUMS-metalink	1 KB	04/08/2016 00:00:00
MD5SUMS-metalink.gpg	1 KB	04/08/2016 00:00:00
MD5SUMS.gpg	1 KB	16/02/2017 00:00:00
SHA1SUMS	1 KB	16/02/2017 00:00:00
SHA1SUMS.gpg	1 KB	16/02/2017 00:00:00
SHA256SUMS	1 KB	16/02/2017 00:00:00
<b>ubuntu-14.04.5-desktop-amd64.iso</b>	42 KB	04/08/2016 00:00:00
<b>ubuntu-14.04.5-desktop-amd64.iso.torrent</b>		
ubuntu-14.04.5-desktop-amd64.iso.zsync		04/08/2016 00:00:00
ubuntu-14.04.5-desktop-amd64.list		04/08/2016 00:00:00
ubuntu-14.04.5-desktop-amd64.manifest		04/08/2016 00:00:00
<b>ubuntu-14.04.5-desktop-i386.iso</b>	44 KB	04/08/2016 00:00:00
<b>ubuntu-14.04.5-desktop-i386.iso.torrent</b>		
ubuntu-14.04.5-desktop-i386.iso.zsync		04/08/2016 00:00:00
ubuntu-14.04.5-desktop-i386.list		04/08/2016 00:00:00
ubuntu-14.04.5-desktop-i386.manifest		04/08/2016 00:00:00
ubuntu-14.04.5-desktop-i386.metalink	44 KB	04/08/2016 00:00:00
ubuntu-14.04.5-server-amd64.iso		04/08/2016 00:00:00
<b>ubuntu-14.04.5-server-amd64.iso.torrent</b>	25 KB	04/08/2016 00:00:00
ubuntu-14.04.5-server-amd64.iso.zsync		04/08/2016 00:00:00

- Install Ubuntu on this virtual machine :

Mount the .iso file into your virtual machine to install it

Or install Ubuntu in Dual-Boot :

Use a bootable USB-key and boot your computer on it :  
<https://help.ubuntu.com/community/Installation/FromUSBStick>

- Download a convenient terminal, in the default terminal run the command :

```
sudo apt-get install terminator
```

or install it via the *logitech*.

## ii – ROS installation

- Follow this tutorial :

<http://wiki.ros.org/indigo/Installation/Ubuntu>

## c – ROS overview

ROS is composed of several elements each having a specific function:

**ROS-Workspace** : A specific workspace including all elements is required to run ROS software. It

is simply a folder with specific files and folders in it.

**ROS-Master** : Main program that allows all ROS-element to work together

**Packages** : Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

**Nodes** : "Node" is the ROS term for an **executable** that is connected to the ROS network. They are basically programs (C++ or Python) that anyone can write or adapt in order to control whatever one wants.

**Topics** : Nodes use topics to communicate between them. For instance, node A « write » on a topic and node B can « read » the same topic to get the information.

**Messages (msg)** : Messages are what it is written on topics. Messages can have different formats but the format has to be specified within a .msg file.

**Services (srv)** : Services are similar to messages. They have the same function (communicating between nodes via topics) but they have two parts : a request and a response. The format on each services are recorded into .srv files and the two parts are separated by « --- ». Services has the advantage to make sur that the information has been well transmited .

Illustration of the main elements of ROS :

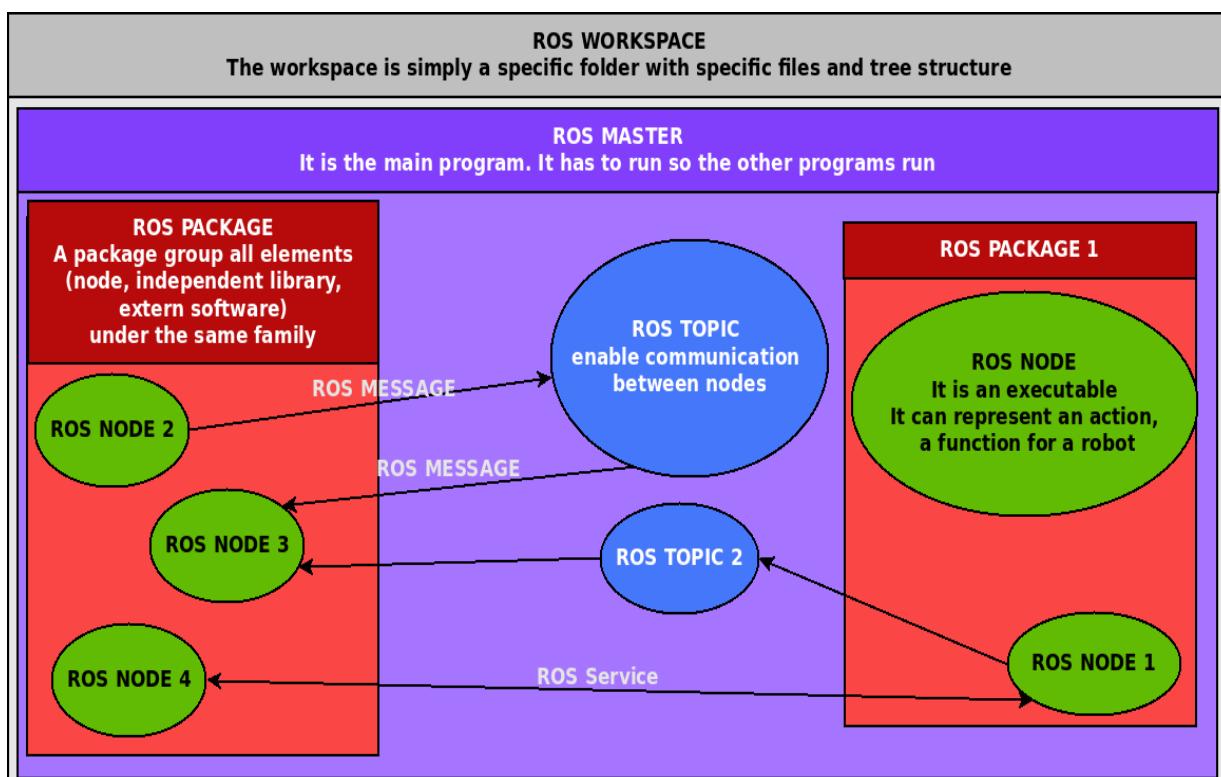


Illustration 1: Schema of the main elements which compose ROS and how they work together

First we will get familiar with ROS-elements then we will play with ROS and the robot Sawyer.

## 2 – ROS first trial



### a – Create a workspace

First, activate ROS:

```
source /opt/ros/indigo/setup.bash
```

Create a *ros\_ws\_tuto* folder with a *src* folder inside :

```
mkdir -p ~/ros_ws_tuto/src
```

Move to the folder you just created and build it as a ROS-Workspace :

```
cd ~/ros_ws_tuto  
catkin_make
```

NB : Check the new folders and files created by the *catkin\_make* command within the *ros\_ws\_tuto*. This is the minimal structure of a ROS-Workspace. You can use the graphic interface or explore from a terminal using the following non exhaustive linux commands :

```
cd = move to the home directory  
cd /My_folder = move to the directory « My_folder » starting from the root  
cd My_folder (or cd ./My_folder) = move to the directory « My_folder » from the current directory  
cd ~/My_folder = move to the directory « My_folder » from the home directory  
(home directory = /home/UserName )  
cd .. = move to parent folder (add « .. » x times to move to x parent folder ex : cd ../../ = parent folder twice)  
cd - = previous directory  
ls = print the list of folders and files into the current directory ( ls /my/path = print files and folders into the /my/path directory)  
. / = current directory
```

### b – Create a package

Move to *src* folder in the workspace you just created and create the package called *package\_tuto* :

```
cd ~/ros_ws_tuto/src  
catkin_create_pkg package_tuto std_msgs rospy roscpp
```

re-build the workspace ( you might first have to go back to the *ros\_ws\_tuto* folder)

```
cd ..  
catkin_make
```

Over 2000 packages are available for download on the ROS website at this address :  
[http://www.ros.org/browse/list.php?package\\_type=package&distro=indigo](http://www.ros.org/browse/list.php?package_type=package&distro=indigo), each package including several nodes. Since ROS is an open source project, this library of packages increases constantly.

NB : The prototype of the `catkin_create_pkg` command is :

```
catkin_create_pkg <package_name> <dependency1> <dependency2> ...
```

The dependencies are external libraries required by the package regarding its purpose. For instance, the package you just created wouldn't be able to run python program without the *rospy* dependency.

Finally, to integrate the package you just created to the ROS-environnement you need to run a *setup.bash* file in the *devel* directory of your package.

```
source ~/ros_ws_tuto/devel/setup.bash
```

**WARNING : This command has to be run in every new terminal in order to use nodes from the package.**

## c – What is a node?

Start ROS-Master :

```
roscore
```

The ROS-Master is running on the terminal so to continue to work, you need to switch to a new terminal and initiate it as previously in 4.a :

```
source /opt/ros/indigo/setup.bash
```

### i – Playing with turtles

In order to understand nodes in a first time we will use an educational tool from ROS. Install the turtle simulator :

```
sudo apt-get install ros-indigo-ros-tutorials
```

You just downloaded a package called *turtlesim* with nodes in it. Let's try it by calling the node *turtlesim\_node* :

```
rosrun turtlesim turtlesim_node
```

NB : if you type the uncompleted command « rosrun turtlesim » and the <tab> <tab> , it will show you all node you can call from the package *turtlesim*.

A node is an executable file. So theoretically it doesn't necessarily require to use the *rosrun* function to run a node. Let's make appear another turtle by directly calling the node. Close the windows with the turtle and run the commands :

```
cd /opt/ros/indigo/lib/turtlesim  
ls
```

By running the above commands, you move to the *turtlesim* package and print the files that are in it, in the terminal. You can see the node you called before with the *rosrun* command. Run the file by typing the following command :

```
./turtlesim_node
```

But as you can see it's more convenient to use the *rosrun* function as it can be run from any directory.

Two nodes can be run at the same time. Another node exists to move the turtle, switch to an other terminal and initiate it :

```
source /opt/ros/indigo/setup.bash  
rosrun turtlesim turtle_teleop_key
```

You can now move the turtle using the arrow keys (make sure the terminal where the *turtle\_teleop\_key* node is running is selected). It's possible to check the node currently running. In a new terminal :

```
source /opt/ros/indigo/setup.bash  
rosnode list
```

The results show well that the two nodes *turtlesim\_node* and *turtle\_teleop\_key* are running. There is another one called *rosout* which comes from the *roscore* function.

NB : Every time you run a node, ROS will assign a name to it. By default ROS assigns the name of the executable. If you run the same node twice, ROS will be confused with 2 nodes having the same name. To remedy to this issue you can set manually the name of the 2<sup>nd</sup> node when you run it by :

```
rosrun package_name node_name __name:=new_node_name
```

## ii – Writing your own node

It's really fun to play with turtles but it's more useful to create our own node. First it is required to create the code of the node in C++ (or python) located in the *src* folder in the package you created before.

**In C++ ( for python see next page ) :**

```
cd ~/ros_ws_tuto/src/package_tuto/src  
gedit hello.cpp
```

Write the following code into the *hello.cpp* file :

```
#include "ros/ros.h"  
int main()  
{  
    printf("Hello World \n");  
}
```

Save and quit the file editor *gedit*.

Once you wrote the code for the node, it has to be linked to the package.

Open the *CMakeLists.txt* file located in the package folder :

```
gedit ~/ros_ws_tuto/src/package_tuto/CMakeLists.txt
```

And add the following lines at the end of the file :

```
add_executable(hello src/hello.cpp)  
target_link_libraries(hello ${catkin_LIBRARIES})  
add_dependencies(hello package_tuto_generate_messages_cpp)
```

Re-compile the workspace :

```
cd ~/ros_ws_tuto  
catkin_make  
source devel/setup.bash
```



Now you can run your own node :

```
rosrun package_tuto hello
```

It has been said earlier that a node is an executable. The *hello.cpp* file is not an executable. But when the workspace has been re-build, an executable file has been created and you can check at the following location :

```
cd ~/ros_ws_tuto/devel/lib/package_tuto
```

**In python :**

Creating a node in python has some differences. In the *package\_tuto* folder, create a *scripts* folder which will contain all the python codes. Then create a *.py* file named *hello.py*.

Then write in the *.py* file :

```
#! /usr/bin/env python
import rospy
print "Hello world"
```

and save. Then, the *CMakeList.txt* doesn't need to be adapted contrary to the C++ code. However the *.py* file still has to be converted into an executable. Go to the *scripts* directory if needed :

```
cd ~/ros_ws_tuto/src/package_tuto/scripts
```

Run the following command to convert the file :

```
chmod +x hello.py
```

NB : The file now appears in the terminal in green (in grey before) when you run the *ls* command into the *scripts* directory. The file is now executable.

Finally, re-build and add the workspace to the ROS-environment :

```
cd ../../..
catkin_make
source devel/setup.bash
```

You can then launch the node by using :

```
rosrun package_tuto hello.py
```

## d – Topics and messages

### i – How topics and messages work ? (With turtles)

Run the two nodes to make turtle move

```
rosrun turtlesim turtlesim_node
```

in a new terminal

```
rosrun turtlesim turtle_teleop_key
```

NB : don't forget to run the ROS\_master and to initialize the terminals

Those two nodes communicate with a topic. ROS allows to see that. In a new terminal :

```
rqt_graph
```

This graph enables to see all the currently actives ROS-element and their interaction. The node *teleop\_turtle* “writes” (so it is the *publisher*, in blue in the graph) the displacement command on the topic */turtle1/cmd\_vel* and the *turtlesim* node “reads” (it is the *subscriber*, in green in the graph) the information on the same topic

NB : The color change regarding where your mouse is pointing (in red).

Let's run a few interesting commands :

To see the current topics :

```
rostopic list
```

Run the following command :

```
rostopic echo turtle1/cmd_vel
```

Now move the turtle, you'll see the *message* displayed into the terminal.

It's possible to publish directly on a topic. For that you need to know the *message* used on the topic :

```
rostopic type turtle1/cmd_vel
```

And the format of the message :

```
rosmsg show geometry_msgs/Twist
```

In this example, the last command show you that the message is composed of 2 vectors. The first one represents the XYZ linear velocity and the other one represent the angular velocity. So to publish on the topic, the command would be :

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -- '[-5, -3, 0]' '[0, 0, 2]'
```

As the motion of the turtle can only be 2-dimensional, the angular vector can only accept a value on its third component. Here, a 2 rad/s angular velocity is applied. The first vector represent the linear velocity and, as the motion is bidirectional, the Z-component has to be zero. The value can be changed as you want.

## ii – Create your own topic and message

You can now close all the running node except for the *roscore* node. To do so use *ctrl+C* in each terminal.

### Creating the message type

Creating a message is not complicated. A message is simply a *.msg* file describing the format of the message. All messages for a package are saved into a *msg* folder.

```
roscd package_tuto  
mkdir msg  
gedit msg/word.msg
```

Within the *gedit* editor interface, write the following line :

```
string data
```

Save and quit *gedit*.

Other steps are required to integrate the *.msg* file to the package, in the *package.xml* file :

```
roscd package_tuto  
gedit package.xml
```

find the following line and uncomment them (respectively around line 35 & 39):

```
<build_depend>message_generation</build_depend>  
<run_depend>message_runtime</run_depend>
```

In the *ros\_ws\_tuto/src/package\_tuto/CMakeLists.txt* file, add the line *message\_generation* like below ( $\approx$  L.10) :

```
find_package(catkin REQUIRED COMPONENTS  
    roscpp  
    rospy  
    std_msgs  
    message_generation  
)
```

Uncomment the following block and replace *Message1.msg* and *Message2.msg* by the name of the message you created ( $\approx$  L.51) :

```
add_message_files(  
    FILES  
    word.msg  
)
```

Uncomment the following lines ( $\approx$  L.71) :

```
generate_messages(  
    DEPENDENCIES  
    std_msgs  
)
```

Uncomment the following lines and add “*message\_runtime*” as following ( $\approx$  L.105) :

```
catkin_package(  
    ...  
    CATKIN_DEPENDS message_runtime roscpp rospy std_msgs  
    ...)
```

And finally rebuild the workspace to be sure that everything is all right :

```
cd ~/ros_ws_tuto  
catkin_make
```

### **Creating the topic using the message type created above**

A topic is nothing but a name. There is no file with the source code of a topic. The topic appears through the nodes which subscribe and publish on it. To create a topic using the *word* type message, we will create, for example, a node which listen to what is published on the topic.

REMINDER : Don't forget every step of node creation :

- create a *.cpp* or *.py* file into the *src* folder of your package
- add to the *CMakeLists.txt* file the 3 lines relative to the new node you just created or convert the *.py* file into an executable one.
- re-built the workspace
- run the *setup.bash* file

**In C++ (python code following next page) :**

If you love C++, create the publisher node named *listener.cpp* in the *src* directory of the package *package\_tuto* with the following code :

```
#include "ros/ros.h"
#include "package_tuto/word.h"
void chatterCallback(const package_tuto::word::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}
int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ros::Subscriber sub =
n.subscribe("tutopic",1000,chatterCallback);
    ros::spin();
    return 0;
}
```

The header *word.h* (created by the *catkin\_make* command) has to be included to allow the node to use the message type we created ( it would be the same thing with a standard message type)

The 2 lines :

```
ros::init(argc, argv, "listener");
ros::NodeHandle n;
```

are required to initiate the node but we won't extend on it.

The line :

```
ros::Subscriber sub =
n.subscribe("tutopic",1000,chatterCallback);
```

is actually the declaration of the topic to the ROS-master. Here we, arbitrarily, defined the topic name as “*tutopic*”. *1000* is the size of the buffer. If node A publishes messages faster than *listener* can process, messages will accumulate in a buffer up to 1000 before *listener* starts “throwing” them away. The value of the message is the result of the “*ChatterCallback*” function.

### In python :

In case your favorite language is Python, create a *listener.py* file into the *scripts* directory created previously and add the following code in it :

```
#!/usr/bin/env python
import rospy
from package_tuto.msg import word
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)
def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("tutopic", word, callback)
    rospy.spin()
if __name__ == '__main__':
    listener()
```



In the python code, the main line is

```
rospy.Subscriber("tutopic", word, callback)
```

This line enables the node to “*subscribe*” to the arbitrarily named topic “*tutopic*” and to exchange “*word*” type message with a certain value. The value is the result of the “*callback*” function defined at the beginning of the code.

Now let's check our topic.

Let's see all the currently active topics. Run the command :

```
rostopic list
```

The *tutopic* topic is not in this list. It's all right, because the node you just created which uses the topic is not running yet. Now, if you run the *listener* node, it should activate the *tutopic* topic. Run the following command :

In C++ : `rosrun package_tuto listener`

In python : `rosrun package_tuto listener.py`

In a new terminal, check the list of topic currently available (`rostopic list`). You should see a new topic called *tutopic*. What a wonderful surprise !

In the same terminal, publish something on the topic :

```
rostopic pub /tutopic package_tuto/word 'put whatever you want
between the apostrophes'
```

You should see the node « heard » what you written on the topic.

**NB : Don't forget to initialize again the workspace in a new terminal by running the command :**

```
source ~/ros_ws_tuto/devel/setup.bash
```

## To publish on a topic in a node

Publishing on a topic is similar to subscribe from it. Create a node called *talker* and copy paste the following code into its .cpp file. Re-build the workspace (*catkin\_make*) and run both *talker* and *listener* at the same time (in 2 different terminals).

### C++ Code :

```
#include "ros/ros.h"
#include "package_tuto/word.h"
#include <sstream>
int main(int argc, char **argv)
{
    ros::init(argc, argv, "speaker");
    ros::NodeHandle n;
    ros::Publisher
chatter_pub=n.advertise<package_tuto::word>("tutopic",1000);
    ros::Rate loop_rate(10);
    int count = 0;
    while (ros::ok())
    {
        package_tuto::word msg;
        std::stringstream ss;
        ss << "hello world " << count;
        msg.data = ss.str();
        ROS_INFO("%s", msg.data.c_str());
        chatter_pub.publish(msg);
        ros::spinOnce();
        loop_rate.sleep();
        ++count;
    }
    return 0;
}
```

**Python Code :**

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from package_tuto.msg import word
def talker():
    pub = rospy.Publisher('tutopic', word, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

The publishing function is represented by these lines :

**In C++ :**

```
chatter_pub=n.advertise<package_tuto::word>("tutopic",1000);
    ros::Rate loop_rate(10);
and
chatter_pub.publish(msg);
```

**In python :**

```
pub = rospy.Publisher('tutopic', word, queue_size=10)
and
pub.publish(hello_str)
```

Now run the talker node while the listener is running too. You should see that *talker* publishes continuously on the *tutopic* topic while *listener* is subscribing and printing the message (“*hello world 'counter'* ”) on the terminal. The two nodes are communicating with each other via the “*tutopic*” topic.

## e – Services

Services are similar to Topic + Messages. There is a service type defined by a *.srv* file and the service is created by a node enabling to communicate between node. However, they are not used in the same way. Topic and message are used for continuous data flow as for sensor informations, while services are used for an occasional communication.

### i – Create your own service type

A service type is simply a *.srv* file into the write directory :

```
cd ~/ros_ws_tuto/src/package_tuto  
mkdir srv  
gedit srv/service_tuto.srv
```

A service is composed of two parts, a request part and a response part. The request represent the input of the service, and the response the output. In the *.srv* file, the two parts will be separated by “---”. The format of the service depends of course on the purpose of the service. As an example we will create here a service which gives the average of two integers. So let's write a service type having 2 integer as request and 1 integer as response. In the *.srv* file you just created, write :

```
int64 numA  
int64 numB  
---  
float64 moy
```

In order to integrate this new file to the system, modify the *CMakeLists.txt* file and uncomment these lines :

```
add_service_files(  
    FILES  
    service_tuto.srv  
)
```

Then, from the main folder of the workspace, run the following command :

```
catkin_make install
```

This command creates the useful files to use the new service type you just created like the header called in nodes.

### ii – Create and use a service

Creating a service is similar than creating a topic. So first create a node called *AverageNode* and write the following C++ code :

**In C++ (the python code is not following) :**

```
#include "ros/ros.h"
#include "package_tuto/service_tuto.h"

bool average(package_tuto::service_tuto::Request &req,
package_tuto::service_tuto::Response &res)
{
    res.moy = (req.numA + req.numB)/2;
    ROS_INFO("the average of %ld and %ld is %f .", (long
int)req.numA, (long int)req.numB, (float)res.moy);
    return true;
} // this is the function associated to the action of the service
itself.

int main(int argc, char **argv)
{
    ros::init(argc, argv, "Average_Server");
    ros::NodeHandle n;
    ros::ServiceServer service =
n.advertiseService("Average_service", average);
    ROS_INFO("Ready...");
    ros::spin();
    return 0;
}
```

Don't forget to re-built the workspace (*catkin\_make*).

The node create the service with the following line :

```
ros::ServiceServer service = n.advertiseService("Average_service",
average);
```

Now run the node you just created. You'll see the service *Average\_service* in the service list. You can also call this service with the following command :

```
rosservice call /Average_service 10 20
```

With 10 and 20 the two arguments of the service request.



## 3 – InteraSDK : Discovering ROS and Tom (Sawyer)



The Intera SDK provides a platform for development of custom applications for Intera Robots.

This repository contains meta-packages and files for installation/use of the Intera SDK. Additionally, this repositories contain the Python interface classes and examples for action servers and control of the Intera Robot from Rethink Robotics.

ROS can be used to develop software to program robots. Rethink Robotics did that for their robots. Their operating system *Intera* presents 2 versions, one with a graphical interface and the other one, *InteraSDK* based on ROS and open source. Obviously, the second one allows going deeper into robot programming.

The ROS-based operating system for Sawyer (and Baxter) is well documented on this website :

[http://sdk.rethinkrobotics.com/intera/Main\\_Page](http://sdk.rethinkrobotics.com/intera/Main_Page)

### a – Create an adapted workspace

Create a regular workspace as we learned in the previous part and call it *ros\_ws\_intera* : create the directory then run the *catkin\_make* command (don't forget to run the *setup.bash* file :  
`source /opt/ros/indigo/setup.bash`)

Install *Intera SDK* dependencies and softwares required.

```
sudo apt-get install git-core python-argparse python-wstool
python-vcstools python-rosdep ros-indigo-control-msgs ros-
indigo-joystick-drivers ros-indigo-xacro ros-indigo-tf2-ros
ros-indigo-rviz ros-indigo-cv-bridge ros-indigo-actionlib
ros-indigo-actionlib-msgs ros-indigo-dynamic-reconfigure ros-
indigo-trajectory-msgs
```

(if you are experiencing problems with the copy/paste of the previous line, please click on the following link :

[http://sdk.rethinkrobotics.com/intera/Workstation\\_Setup#Install\\_Intera\\_SDK\\_Dependencies](http://sdk.rethinkrobotics.com/intera/Workstation_Setup#Install_Intera_SDK_Dependencies) )

Install *Intera* robot SDK

```
cd ~/ros_ws_intera/src
wstool init .
git clone https://github.com/RethinkRobotics/sawyer_robot.git
wstool merge sawyer_robot/sawyer_robot.rosinstall
wstool update
```

Re-built the workspace

```
source /opt/ros/indigo/setup.bash  
cd ~/ros_ws_intera  
catkin_make
```

Configure the workspace accordingly to the robot Sawyer. It's only consisting on adapting the file *intera.sh*

First, copy the file to the main directory of the workspace :

```
cp ~/ros_ws_intera/src/intera_sdk/intera.sh ~/ros_ws_intera  
edit it,
```

```
cd ~/ros_ws_intera  
gedit intera.sh
```

change: - Ip : your\_ip="192.168.XXX.XXX"

you can find your IP address by typing ifconfig into a terminal.

- Or you can also change your hostname (instead of your IP, not both): your\_hostname = "<computer\_name>.local"
  - Distribution : ros\_version="indigo"
  - robot name : robot\_hostname="tom.local"

Now, to initialize the system all you have to do is launch the *intera.sh* file :

```
cd ~/ros_ws_intera  
. ./intera.sh
```

## b – Examples

*Intera SDK* offers some nodes as examples. The source code for those examples are located in the following directory : *~/ros\_ws/src/intera\_sdk/intera\_examples/scripts/*

Try to get familiar by running some examples. To run an example (which is a node) you have to adapt the following command :

```
rosrun intera_examples <example_name.py>
```

Instructions for each examples are given on the terminal.

Here is some of the available examples :

**intera\_examples** (located in *~/ros\_ws\_intera/src/intera\_sdk/intera\_examples/scripts/*) :

*gripper\_keyboard.py* = to control the gripper with the keyboard

*gripper\_cuff\_control.py* = to control the gripper with button on the robot

*head\_wobbler.py* = make the head move

*joint\_position\_keyboard.py* = enable to control the robot with the keyboard

*joint\_position\_waypoints.py -s X -a Y* = enable to record point to point

trajectory and to play it back at the speed ratio X and at the accuracy of Y rad.

```
camera_display.py -c <camera_name> = display the designated camera.  
Option : -c head_camera or right_hand_camera  
        -r raw  
        -e only edge on the image.  
  
lights_blink.py -l <light_name> = make designated light blink :  
light name: head_red_light, right_hand_blue_light, head_blue_light,  
right_hand_green_light, head_green_light, right_hand_red_light
```

## c – Topics relative to Sawyer

You can use the same commands about *topics* we learned previously.

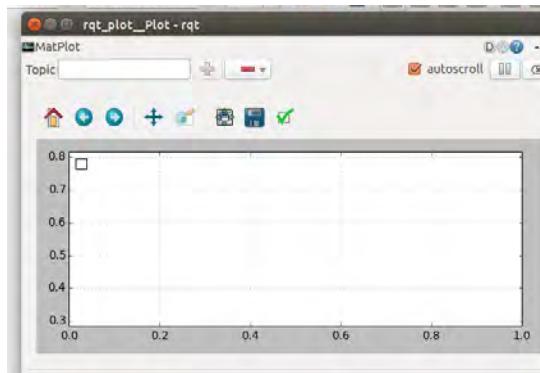
Interesting topic to listen are :

```
rostopic echo ...  
... /robot/limb/right/endpoint_state = pose, orientation, twist and wrench  
... /robot/head/head_state = state of the head  
... /robot/limb/right/gravity_compensation_torques = show  
parameters of the current gravity compensator running.
```

interesting topic to publish on :

```
rostopic pub ...  
... /robot/head/command_head_pan intera_core_msgs/HeadPanCommand  
-- 'angle' 'speed' 'panMode' = make the « head » moving  
angle = - π → π  
speed = 0,1 → 3  
panMode = 1  
  
... -r 10 /robot/limb/right/suppress_cuff_interaction  
std_msgs/Empty = disable the « zero-G » button  
  
... -r 10 /robot/limb/right/suppress_gravity_compensation  
std_msgs/Empty = enable to stop the gravity compensator controller
```

Another way to see topics available is using the command `rqtplot`. Once you run this command, a window appears as follow :



*Illustration 2: example of an rqtplot window*

Specify the topic of your choice in the right corner blank case. All the topic start with */robot*. As you can see there is a complete list of available topic. If you choose a topic, it will be plot in live in the graph part bellow.

There is also another way to visualize topics on a graphic interface: using *Rviz*. Simply type in

```
rviz
```

in the terminal and the *rviz* software will start. If you are connected to Sawyer, you'll see Sawyer in the visualization window on the right and on the left, a list of topics being displayed in the visualization window.

For example, in the *Effort* element in the display tree (if you can not see it, add it by doing Add → Effort → Ok, on the bottom left), you can choose to display the */robot/joint\_state* topic which display live the efforts currently being applied at each joint of the robot.

Similarly, in the *Camera* element, you can also choose to display the */io/internal\_camera/right\_hand\_camera/image\_raw* topic that shows you the image of the hand camera of Sawyer (it is also possible to display the image of the head camera but it must be enabled first).

The *Rviz* software is a practical Graphic User Interface, GUI, that also allows you to model external elements of the robot, visualize simulated robot trajectories, and much more. We will not extend too much on this software here.

## d – How to create a ROS package and node to command the robot

### i – Create a node

Firstly, a packages is required. It's the exact same method which has been presented above :

Move to *src* folder in the workspace you just created and create the package called *package\_tuto* :

```
cd ~/ros_ws/src  
catkin_create_pkg package_tuto_sawyer std_msgs rospy roscpp
```

re-build the workspace :

```
cd ..  
catkin_make
```

Secondly, the node can be created. As a reminder, a node is nothing else than an executable file. In this examples the node will be written in python but it could be written in C++.

Into the *src* folder, create an empty .py file :

```
cd ~/ros_ws/src  
echo "" > nodeName.py
```

For now this file is not an executable. Let's do so :

```
chmod +x nodeName.py
```

Rebuilt the workspace from the *ros\_ws* fodler :

```
catkin_make
```

## ii – Program a simple node and Rosbag

In the node you just created, type in the following python code :

```
#!/usr/bin/env python

import rospy
import intera_interface
rospy.init_node('Hello_Sawyer')
limb = intera_interface.Limb('right')
angles = limb.joint_angles()
angles['right_j0']=0.0
wave_1 = {'right_j6': -1.5126, 'right_j5': -0.3438,
'right_j4': 1.5126, 'right_j3': -1.3833, 'right_j2': 0.03726,
'right_j1': 0.3526, 'right_j0': -0.4259}
wave_2 = {'right_j6': -1.5101, 'right_j5': -0.3806,
'right_j4': 1.5103, 'right_j3': -1.4038, 'right_j2': -0.2609,
'right_j1': 0.3940, 'right_j0': -0.4281}
for _move in range(3):
    limb.move_to_joint_positions(wave_1)
    rospy.sleep(0.5)
    limb.move_to_joint_positions(wave_2)
    rospy.sleep(0.5)
```

Save it and check if it runs correctly using the command:

```
rosrun package_tuto_sawyer nodeName.py
```

As seen previously, this code should simply make the robot wave its hand in a “hello” motion. In this code, the two commands `limb.move_to_joint_positions` in the `for` loop, simply publish a message containing the joint position data on the `/robot/limb/right/joint_command` topic. The actuators of the robot then subscribe to this topic and move the joints.

Another interesting feature about ROS is the `rosbag` command to record and play back some topics. This command allows you to record the messages published on specific topics and to replay them afterwards. We will now make the robot “say hello” again, without running the node but by replaying the joint position data on a specific topic.

Open a new terminal and in your home directory (type `cd` in the terminal), or any other directory you are (just remember where), first create a folder named *bagfiles* and go into it using the following commands:

```
mkdir bagfiles  
cd bagfiles
```

This file will contain the recorded data to be replayed later on.

Now type

```
rosbag record -O jointData /robot/limb/right/joint_command
```

ROS is now recording everything that is being published on the */robot/limb/right/joint\_command* topic and storing it in the *jointData* (.bag file) file located in the folder we created earlier.

In another terminal, now run the *nodeName.py* node we created earlier:

```
rosrun package_tuto_sawyer nodeName.py
```

Once the motion is over, terminate the rosbag recording by typing `ctrl+c` in the terminal where rosbag was recording. In this same terminal and to replay the recorded motion, you can now type:

```
rosbag play jointData.bag
```

The robot should now replay the motion.

Basically all topics could be recorded and replayed using *rosbag*. The trick is to now on which topic is published the data you want. To find some topics on which there is currently message being published or subscribed, you can type

```
rostopic list -v
```

One way to record ALL topics at once is to type

```
record jointData -a
```

But be very careful when typing this: if many topic are active, you may record many unwanted messages and also a huge amount of data if you record for too long.

There are also some option for the *rosbag* replay command:

```
rosbag play -s <a number of second> jointData
```

to make the replay start “a number of second” after the beginning of the file.

```
rosbag play -r 2 jointData
```

to make the replay goes 2 times faster.

### iii – Program a node

In this part, we will try to create a node inspired from the others nodes available. The objective is to create a node which enable to move the robot along the Cartesian space using the keyboard key.

For instance the num-pad could be used as follow :

4 = increase y  
6 = decrease y  
8 = increase x  
2 = decrease x  
9 = increase z  
7 = decrease z  
1 = open gripper  
0 = close gripper

All needed should be found in 2 different nodes :

- *joint\_position\_keyboard* from the package *intera\_examples* which is basically the same thing asking here but in the joint-space of the robot.

- *ik\_service\_client* also from the *intera\_examples* package.

The general idea would be to pick up parts of each codes and adapt them in order to create the desired node. Good Luck !!!

A few tips for the python program :

- The first line of the code has to be : #! /usr/bin/env python
- Don't forget to import the required element to run function in the code, for instance :

```
import argparse
import rospy
import intera_interface
import intera_external_devices
from intera_interface import CHECK_VERSION
```

- you can use the command

```
print(« my string %d » % (my_variable))
print( « my string » )
```

to help you to see how the program works and what object are you manipulating

- in *Python* spaces and tabs **matters**

## e – A graphic interface : RVIZ and a trajectory planning framework : MoveIt! (For Your Information)

### i – Installation

“MoveIt! is state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains.

MoveIt! is the most widely used open-source software for manipulation and has been used on over 65 robots. See the Robots page for a list of some of the robots that MoveIt! is being used with.”

In practice, MoveIt! is like a big ROS package with trajectory planning, collision detection, kinematic models and many other built in codes. It can also be interfaced with the Rviz (GUI presented above). This has the advantage, for example, to visualize the trajectory planned using MoveIt! and check where the collisions happen. Of course, additional home made codes (C++ and Python) can be added.

Install MoveIt! :

```
sudo apt-get install ros-indigo-moveit
```

Incorporate the MoveIt! to your workspace. Into the ros\_ws/src folder :

```
wstool merge https://raw.githubusercontent.com/  
RethinkRobotics/sawyer_moveit/master/sawyer_moveit.rosinstall  
wstool update
```

Re-built the workspace, in the ros\_ws directory :

```
catkin_make
```

And check if the robot is enabled

```
rosrun intera_interface enable_robot.py -e
```

### ii – First step

Start the joint trajectory controller :

```
rosrun intera_interface joint_trajectory_action_server.py
```

Start the *MoveIt!* application :

```
roslaunch sawyer_moveit_config sawyer_moveit.launch  
electric_gripper:=true
```

This will launch the *Rviz* plug-in of *MoveIt!* and you should see Sawyer appear in its current state.

NB: In the Motion planning window (lower left side), if there is “No planning library loaded” written under “Planning Library”, it means that some additional steps must be performed.

First, terminate all nodes running on your terminals (ctrl+c). Then go into your */ros\_ws\_intera* directory and delete the *build* and *devel* folders (either manually or using the *rmdir <name\_folder>* command). Re-build your */ros\_ws\_intera* workspace using *catkin\_make*. You should now be able to relaunch the *MoveIt!* visualization in *Rviz*.

If everything went well, you should see the *OMPL* library in green in the *Planning Library*. You can now click and drag the robot's end-effector and position it as you wish. In the Motion planning window, you can then go in the *Planning* tab and click on the *Plan* button under *Commands*. You will then see the robot performing the trajectory virtually. The trajectory can then be executed in real life using the *Execute* button. Congratulation! You now managed to move the robot using *MoveIt!*.

If you want to discover much more features on the *MoveIt!* Framework, here are some useful links:

General presentation of *MoveIt!* : <http://moveit.ros.org/>

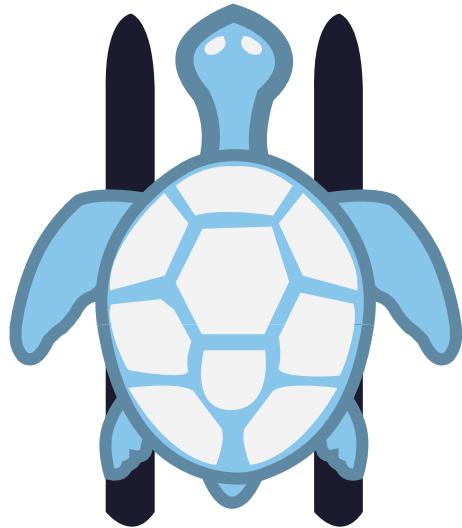
Tutorials: [http://docs.ros.org/kinetic/api/moveit\\_tutorials/html/index.html](http://docs.ros.org/kinetic/api/moveit_tutorials/html/index.html)

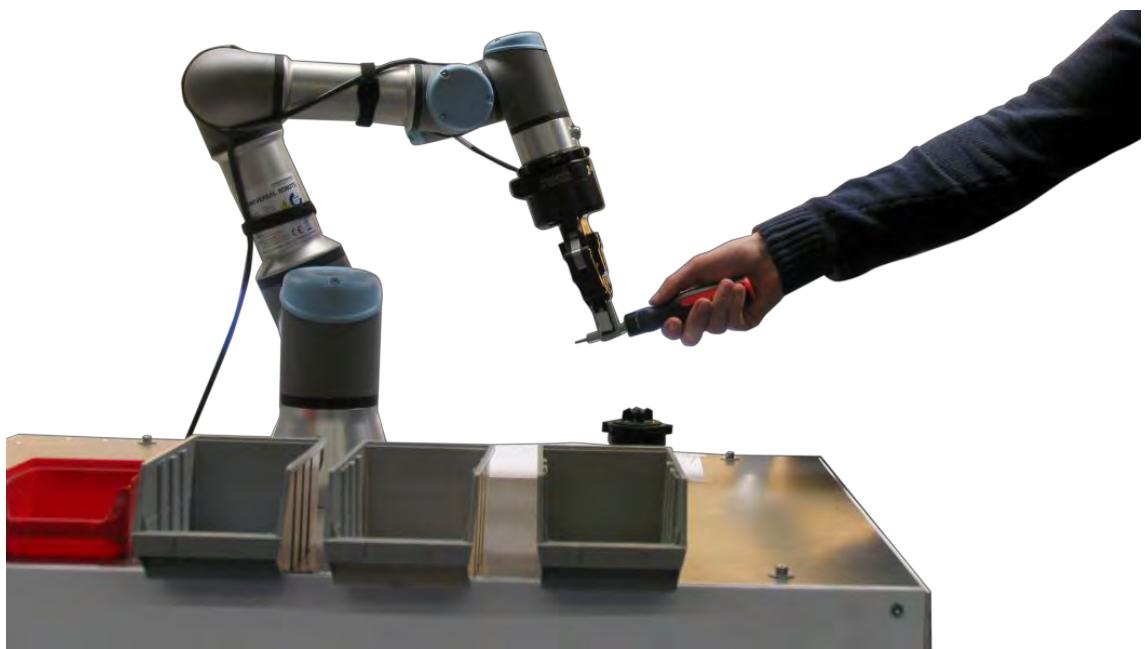
## Conclusion

This ends this short overview of ROS.

As you can now see ROS is a complex and powerful tool that enables you to do a lot with robot : from controlling the robot motion and every element of it, to making a complete robot controller with an user interface...

As an open-source software, ROS has an active community getting bigger and bigger. Any question should find an answer on on-line forums. This makes it a great tool for research!





## Human-Robot-Interaction

Im Rahmen der Robotix-Academy Summer School vom 12.-14.09. am ZeMA in Saarbrücken präsentierte der Umwelt-Campus Birkenfeld einen Workshop zum Thema „Kommunikation zwischen einem Roboter und einem externen Computer“. Das Thema ist für die Mitglieder der Robotix-Academy besonders interessant, da der Aufbau einer Kommunikationsschnittstelle zwischen Roboter und externem Computer ein essentieller Baustein in vielen Robotik-Anwendungen ist. Sie wird benötigt, um dem Roboter durch Sensorik erhaltene Informationen bereitstellen zu können. Geraade bei Mensch-Roboter-Kollaborationen ist dies für die Interaktion der beiden Parteien notwendig, wenn diese beispielsweise über Sprach- oder Gestensteuerung erfolgen soll. Ziel des Workshops war es, die Teilnehmer so zu schulen, dass sie über eine jeweils eigens programmierte Schnittstelle von ihrem Computer aus Befehle und Informationen auf einen Roboter übertragen können. Als Roboter diente hierzu der Universal Robot UR3, dessen Steuerung in einer virtuellen Maschine auf den Computern der Teilnehmer lief. Im ersten Schritt wurde eine kurze Einführung in die Programmierung des Universal Robots UR3 gegeben. Die Teilnehmer erstellten im Anschluss

Dans le cadre de la Robotix-Academy Summer School du 12 au 14 septembre au ZeMA à Sarrebruck, le Campus environnemental de Birkenfeld a présenté un atelier autour du sujet « Communication entre un robot et un ordinateur externe ». Le sujet est particulièrement intéressant pour les membres de la Robotix-Academy, car la construction d'une interface de communication entre un robot et un ordinateur externe est un élément essentiel dans de nombreuses applications de la robotique. Elle est essentielle afin de fournir au robot des informations obtenues par les capteurs. En effet, ceci est nécessaire pour l'interaction des deux parties dans la collaboration homme-robot, lorsqu'elle doit par exemple s'effectuer par commande vocale ou gestuelle. Le but de l'atelier était de former les participants ainsi qu'ils pouvaient, chacun par une propre interface programmée, transmettre des commandes et des informations par leurs ordinateurs au robot. En tant que robot servait le Universal Robot UR3 dont la commande se faisait dans une machine virtuelle sur l'ordinateur des participants. Dans un premier temps, on a donné une courte introduction à la programmation du Universal Robot UR3. Les participants ont ensuite conçu un programme de robot, qui devait être lancé par

ein Roboter-Programm, welches im späteren Verlauf des Workshops vom Computer aus gestartet werden sollte. Danach wurde eine TCP/IP-Verbindung zwischen dem virtuellen Roboter und dem Computer aufgebaut. Zum Senden von Befehlen über Sockets mittels der aufgebauten TCP/IP-Verbindung wurde zusätzlich eine graphische Benutzeroberfläche programmiert. Die Teilnehmer konnten in der Benutzeroberfläche dann Befehle eintragen und an die Robotersteuerung übermitteln, wodurch im Roboter beispielsweise Programme geladen, gestartet oder auch gestoppt werden konnten. Abschließend programmierten die Teilnehmer eine Anwendung, bei der die Übermittlung der Daten auf die Robotersteuerung per MQTT erfolgt, welches sich für den Transport ebenfalls dem TCP/IP-Protokoll bedient. Durch die Verwendung von MQTT war es den Teilnehmern möglich, dem UR3 Roboter Daten zu übermitteln, ohne zwischen diesem und dem Computer eine physische Verbindung zu besitzen. Alle Teilnehmer konnten somit ihre Befehle an einen Computer senden, der als Broker fungiert und eine physische Verbindung mit dem Roboter besitzt. Dieser Broker leitete dann die Daten an den Roboter weiter.

**Kontakt:**

Umwelt-Campus Birkenfeld  
Sebastian Groß  
E-Mail: s.gross@umwelt-campus.de

Jan Jungbluth  
E-Mail: jan.jungbluth@umwelt-campus.de

Thomas Bartscherer  
E-Mail: t.bartscherer@umwelt-campus.de

l'ordinateur à un autre moment de l'atelier. Après, une connexion TCP/IP entre le robot virtuel et l'ordinateur a été construite. Pour pouvoir envoyer des commandes par Sockets grâce à la connexion TCP/IP, une interface graphique a été programmée en plus. Les participants pouvaient alors enregistrer des commandes dans l'interface et les transmettre à la commande du robot, permettant par exemple de télécharger, lancer ou arrêter des programmes dans le robot. Enfin, les participants ont programmé une application où la transmission de données sur la commande du robot s'effectue par MQTT, qui se sert également du protocole TCP/IP pour le transport. Grâce à l'utilisation de MQTT il était possible pour les participants de transmettre des données au robot UR3 sans avoir une connexion physique entre celui-ci et l'ordinateur. Tous les participants pouvaient ainsi envoyer leurs commandes à un ordinateur qui faisait fonction de courtier et possédait d'une connexion physique avec le robot. Ce courtier a ensuite transmis les données au robot.

**Contact:**

Umwelt-Campus Birkenfeld  
Sebastian Groß  
e-mail: s.gross@umwelt-campus.de

Jan Jungbluth  
e-mail: jan.jungbluth@umwelt-campus.de

Thomas Bartscherer  
e-mail: t.bartscherer@umwelt-campus.de



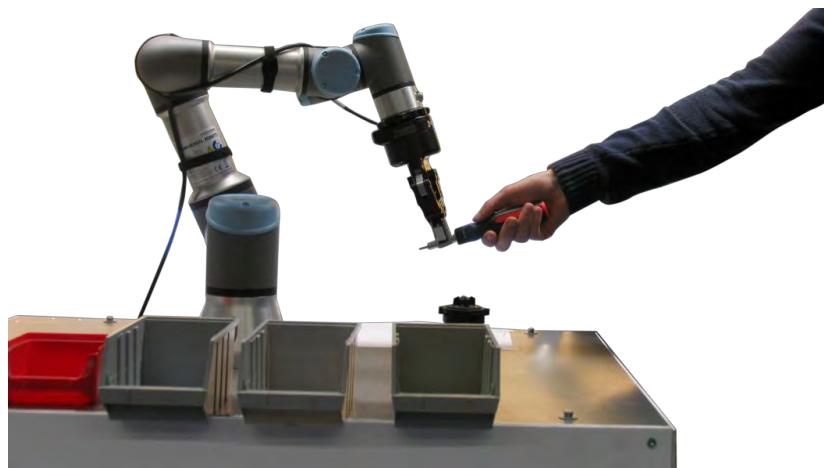
Fonds européen de développement régional | Europäischer Fonds für regionale Entwicklung

## Robotix-Academy Summer-School 2017

September 12<sup>th</sup> to 14<sup>th</sup> 2017

# Human-Robot-Interaction

Workshop presented by Sebastian Groß, Jan Jungbluth and Thomas Bartscherer



## Content

1.	What will we do during this workshop?.....	3
2.	Installing the required software .....	3
2.1	Install URSim and VMWare .....	3
2.2	Installing Microsoft Visual Studio and Mosquitto Broker .....	4
	Installing Microsofts Visual Studio 2017 .....	4
	The Hello World Program.....	5
	Install the Mosquitto MQTT Broker .....	8
	Using Mosquitto .....	8
3.	Introduction to human-robot-interaction.....	11
3.1	Roles of the human in a HRI .....	11
3.2	Communication channels.....	12
4.	Communication-interface between UR3 and a PC.....	13
4.1	Universal Robots UR3 .....	13
4.2	Creating a mechanical communication with the UR3 .....	16
5.	Communication between the virtual machine and a program .....	18
5.1	Programming a tool to communicate with the Universal Robots.....	19
5.2	Communicate with the UR via MQTT .....	22
	Introduction to MQTT .....	22
	Let's start programming .....	23

## 1. What will we do during this workshop?

The goal of this workshop is to give you an introduction and an overview about human-robot-interaction (HRI). To reach this goal theoretical and practical tasks will be executed:

- Getting a short overview what human-robot-interaction is and why you need it for a human-robot-collaboration
- Looking at different communication channels like speech and gesture recognition
- Realizing a haptic interaction between human and robot
- Implementing a communication interface between an Universal Robots UR3 and a PC

## 2. Installing the required software

For exercises, during this workshop, you need to download and install different software-tools. Which software is required and how you install them are explained in this chapter.

### 2.1 Install URSim and VMWare

URSim is a simulation software that is used for offline programming and simulation of robot programs. There are some limitations to the simulator since no real robot arm is connected. Especially the force control will be limited in use. If Simulation Mode is selected in the bottom left corner, it is possible to simulate digital inputs on the I/O page.

URSim is made for the Linux operating system. For running the simulator in another operating system, a virtual machine is needed. A virtual machine is basically just a program, where multiple operating systems can be installed, including Linux.

There are various options available, but we will use VMWare Player (free for non-commercial use, small fee for commercial use). You can download the software through the following link:

<https://www.vmware.com/products/player/playerpro-evaluation.html>

After the download is finished you have to install the VMWare player (admin-rights required).

For easy convenience for non-Linux users, a virtual machine has been created for UR users. This contains a simulator for UR3, UR5, and UR10. It is not possible to run more than one simulator simultaneously. You can download the required URSim Software through the following link:

<https://www.universal-robots.com/download/>

Select: Software → Offline Simulator → Non Linux → URSim 3.4.1

After the download is finished you have to unpack the zipped files.

To now run the URSim you just have to execute the following steps:

1. Start '**VMWare Player**'
2. Press '**Open a virtual machine**' and find the path where the zipped files were unpacked
3. Select URSim in the list and press '**Play virtual machine**'
4. Press '**OK**' when incompatibility dialog is shown
5. Press '**Download and Install**' if VMWare Tools needs to be installed
6. Press '**OK**' if keyboard hook timeout needs to be updated

7. Press '**OK**' when the Removable Devices dialog is shown
8. The Virtual machine is now started!

The documentation of the URScript Programming Language can be downloaded through the following link:

<https://s3-eu-west-1.amazonaws.com/ur-support-site/28901/scriptManual3-4-3.pdf>

You will need some of the URScript commands to successfully complete the tasks in this workshop

#### **URCaps:**

To use URCaps within the virtual machine you need to download the URCaps installation file for your Robotiq device (download: <http://support.robotiq.com/pages/viewpage.action?pageId=5963876>) and unzip the content on a USB stick. Now you need to connect the USB stick to your virtual machine, and therefore disconnect it from the host - your computer (upper right corner in the VMWare Player). Once the USB stick is available to the virtual machine, you can copy the .urcap file from your USB key to the "programs" folder (home/ursim-current/programs) of the Polyscope you want to use (UR3, UR5 or UR10).

Once this is done you have to execute the following steps:

1. Tap **Setup Robot**
2. Tap **URCaps Setup**
3. Tap the + sign
4. Open **Robotiq\_2-Finger\_Adaptive\_Gripper-X.X.X.urcap**.
5. Tap the **Restart** button to restart Polyscope and activate the URCaps

## **2.2 Installing Microsoft Visual Studio and Mosquitto Broker**

In this chapter, we will download and install Microsofts Visual Studio® 2017. With a short "Hello World" program we will check if the Visual Studio installation was successful and give you a very brief introduction to the usage of Visual Studio. Next, we will download and install the mosquito broker for our MQTT based communication program. After the installation, we will also check its functionality.

### Installing Microsofts Visual Studio 2017

Go to the Visual Studio Web page at <https://www.visualstudio.com/> .

Download the Visual Studio IDE Community 2017 version and start the installation.

You will enter the window in Figure 1, for our purpose we only need to select the DotNet-Desktop Environment, so select the check box and continue the installation. The installation will take some minutes and will require a system restart.

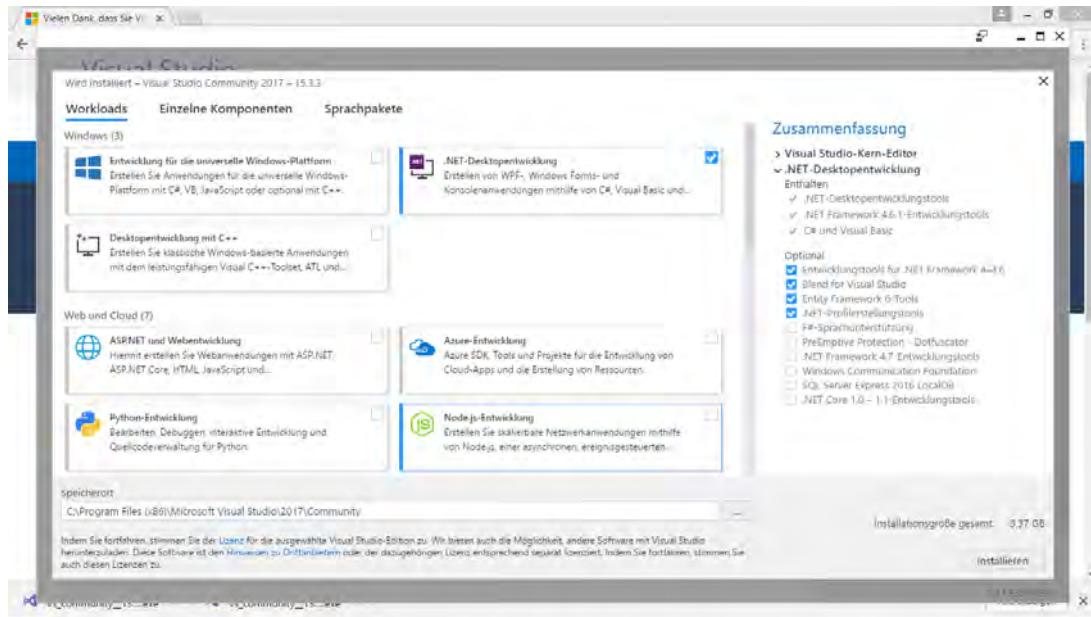


Figure 1 Customization of your Visual Studio installation

### The Hello World Program

The first time you start Visual Studio, you will be pleased to sign up or to register. If don't want to register just click the "remind me later" link. On the next screen, will ask you for your color settings and start preferences. Choose your color setting but don't change the start preferences.

You will enter the start page of the Visual Studio IDE, which looks similar to Figure 2.

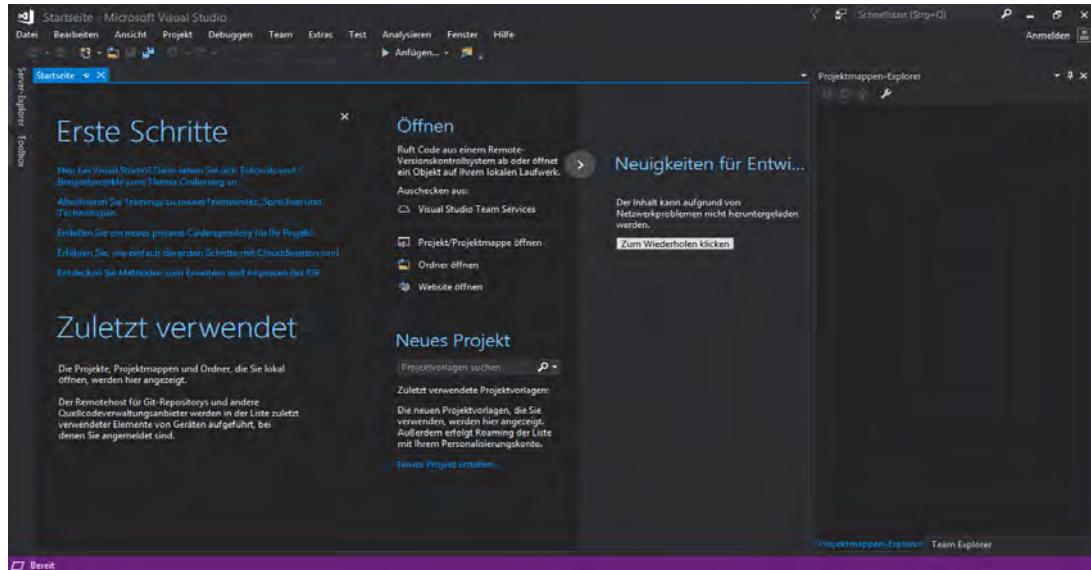


Figure 2 Start page of the Visual Studio 2017 (in dark color style)

We will now:

1. Create a new Windows Forms Projekt
2. Add a Button to the program window
3. Create a button-click-event to open a message box displaying “Hello World”
4. Run and test the program
5. Be proud of our achievement

Let's start:

1. To create a new project we will select “File” -> “New”-> “Project...” in the menu bar. In the appearing window, select the “Window Forms-Application”. Also, name the project “HelloWorld” and click “OK”.

We will see the empty program window like in Figure 3.

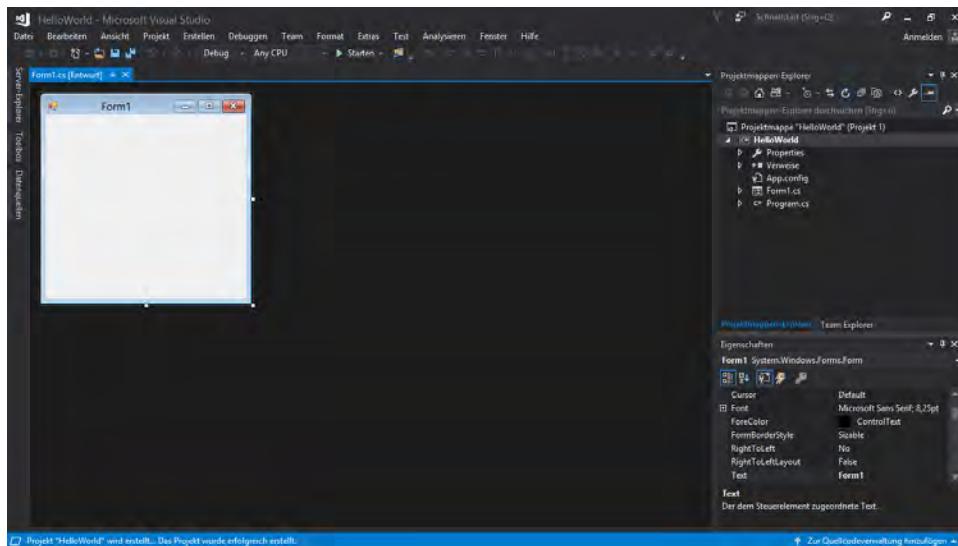
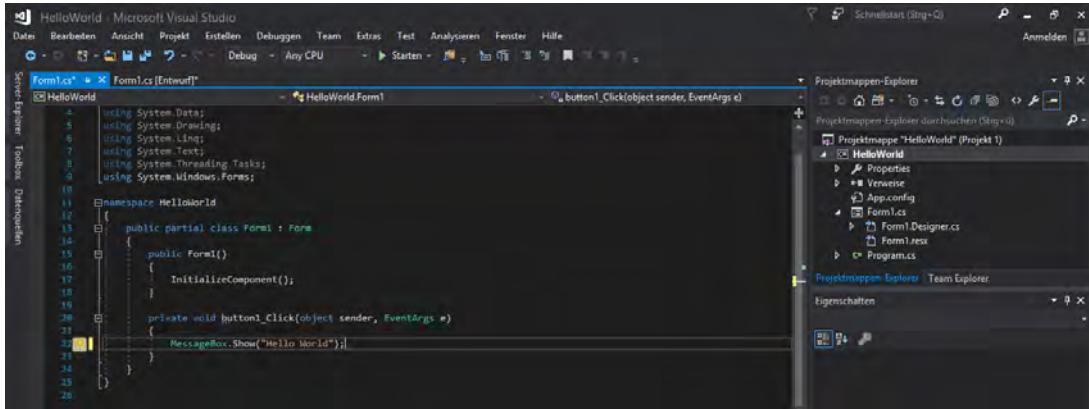


Figure 3 The empty program window of our HelloWorld application

2. To add a button element to the window we have to open the “Toolbox” side menu. In the menu, you can search for the “Button” element. With a double click, or by drag and drop, we can add the button to the program window. You should now resize the button to your preferences by dragging the edges of the button. To change the button text to “push me”, you need to find the “Text”-Field in the button “Properties” (in the lower left corner).
3. The next step is to create the button-click-event. To do so, select the lightning symbol in the button “Properties”. Find the Field called “Click” and double click on it. You will be redirected to the source code of the “Form1.cs” file. In the created “button1\_Click” method we will add our code: `MessageBox.Show("Hello World");`

Your code should look like Figure 4.



The screenshot shows the Microsoft Visual Studio interface with the 'HelloWorld' project open. The 'Form1.cs [Entwurf]' tab is active, displaying the following C# code:

```

4  using System;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 
11  namespace HelloWorld
12  {
13      public partial class Form1 : Form
14      {
15          public Form1()
16          {
17              InitializeComponent();
18          }
19 
20          private void button1_Click(object sender, EventArgs e)
21          {
22              MessageBox.Show("Hello World");
23          }
24      }
25  }

```

The 'Projekt-Explorer' and 'Toolbox' panes are visible on the left, and the 'Solution Explorer', 'Properties', and 'Task List' panes are visible on the right.

Figure 4 The C# source code of the HelloWorld program

- Now we are ready to run and test your program. Press the green play button in the menu bar. After a second the program window will appear. Click the “push me” button, you should see the message box opening and the “Hello World” text.

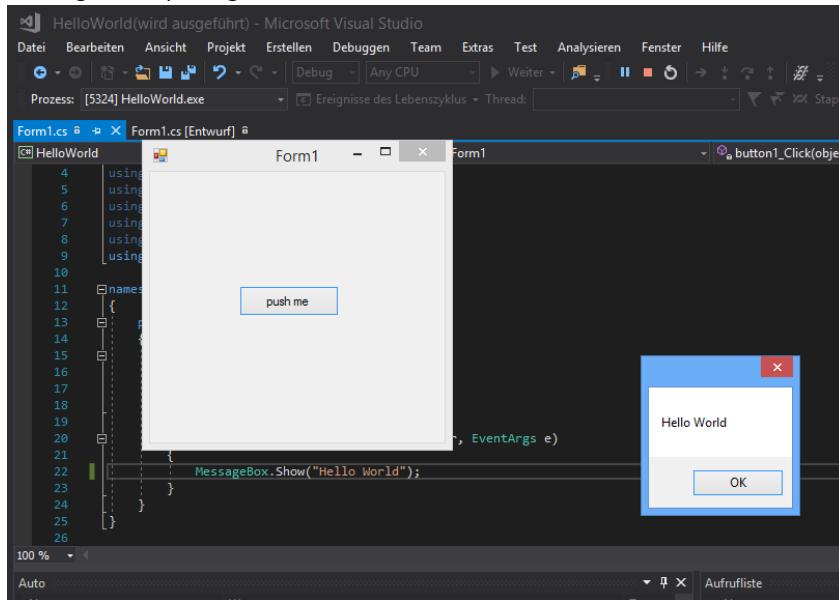


Figure 5 The Hello World program

- Celebrate your victory

We can really recommend the C# language, its very powerful object-oriented language and easy to learn. Also, the DotNet Framework provides you with tons of useful libraries. But with higher DotNet Framework versions you are limited to Windows based operating systems. For lower versions of DotNet, the Mono framework supports C# under Linux and macOS. If you want to know more about the language just let us know, we can provide video courses and more.

## Install the Mosquitto MQTT Broker

MQTT is a lightweight communication protocol based on the publish/subscribe-mechanism. In the MQTT protocol, all messages are handled and distributed by the broker. Clients are publishing messages in a topic and all other clients will receive the messages in an event based manner if they subscribed to the topic. Mosquitto is an Open Source MQTT broker written in C. After the mosquito download, we have to install some additional software, before we can use the mosquito MQTT Broker.

We will now explain the installation for a Windows based OS.

### *Download the Broker and additional Software*

Go to the web page <http://mosquitto.org/download/> and download the “mosquitto-1.4.14-install-win32.exe” file.

We need also OpenSSL. Download the “Win32 OpenSSL v1.0.2L Light” version (another version might not work) from <https://slproweb.com/products/Win32OpenSSL.html> .

We also need the “pthreadVC2.dll” file, download it from  
<http://www.sourceforge.org/pub/pthreads-win32/dll-latest/dll/x86/> .

### *Installation*

First, you should install OpenSSL and remember the installation path (default is C:\OpenSSL-Win32).

Then, we install Mosquitto. Also remember the installation path (default is C:\Program Files(x86)\mosquitto). After the installation, a command prompt will appear and an error message will be thrown, that's ok.

Now have to copy and move all the “.dll” –files from “C:\OpenSSL-Win32” and the “pthreadVC2.dll” to the “C:\Program Files(x86)\mosquitto” directory. It should look like Figure 6.

You can check the installation by double click on “mosquito.exe”. If an empty command prompt appears and stays, you are fine. Most of the time, the windows firewall will ask you to assign network privileges to the program. Let the program use private, public and domain networks to avoid any firewall based connection problems. The MQTT Broker is now running and, if you don't close it, we can test its functionality.

## Using Mosquitto

In the Mosquitto directory are two more programs we will use. With the first program “mosquitto\_sub.exe” we can subscribe to a topic and receive messages from other clients publishing on that topic. With the second program “mosquito\_pub.exe” we can publish messages. We will start two clients which subscribe to the topic “Robotix-Academy” and one client which publishes messages on that topic. To start these programs we have to use the command prompt (cmd.exe).

First, change to the mosquito directory with “cd C:\Program Files (x86)\mosquitto” . Don't forget the “” for the spaces in your path! To get an overview of the accepted parameters of the “mosquitto\_sub.exe” program use “mosquitto\_sub.exe --help” .

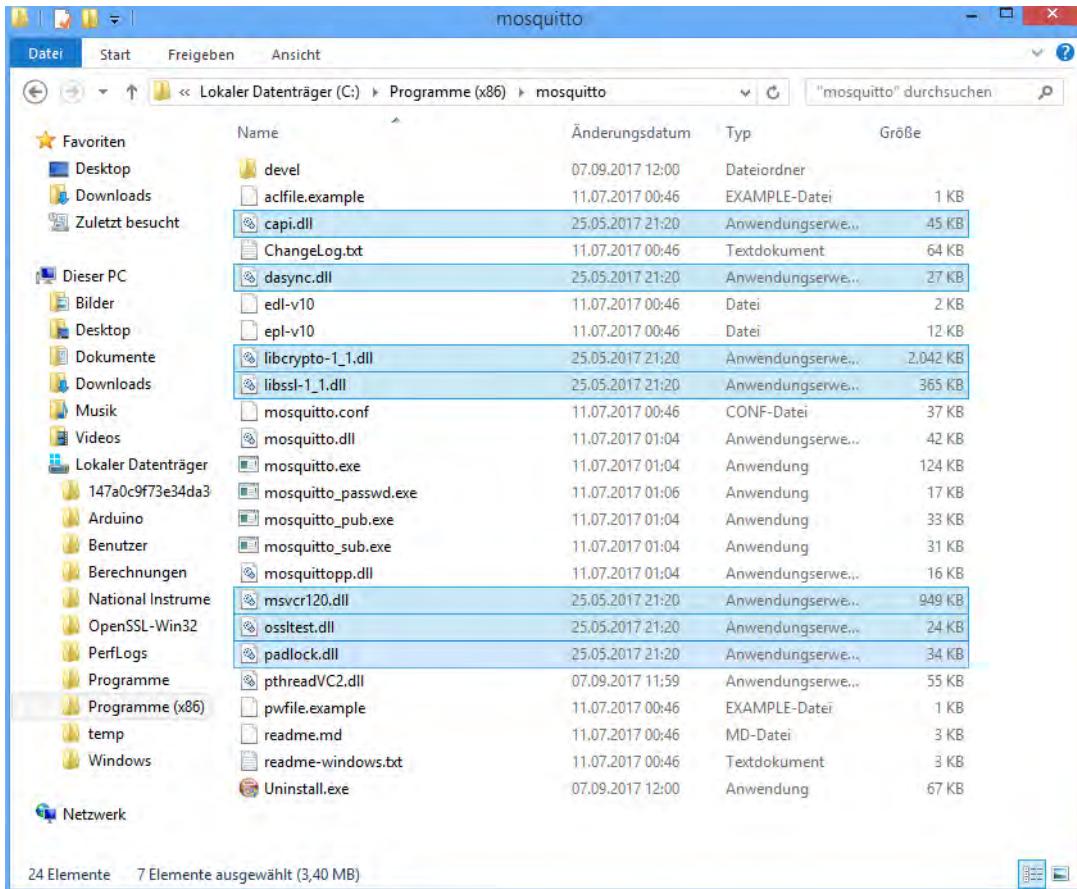


Figure 6 The mosquito directory after placing the necessary DLLs.

All parameters in brackets (see Figure 7), like [-h host], are optional, only parameters without brackets are necessary, like -t topic.

To subscribe to the topic just type “mosquitto\_sub.exe –t Robotix-Academy” and hit enter, don’t be surprised - nothing will happen. Start another command prompt and repeat the procedure to start a second client listing to the topic.

To start a publishing client open a new command prompt, change the directory again to the mosquito directory and enter “mosquitto\_pub.exe –t Robotix-Academy –m rocks” to send the message “rocks” to the topic “Robotix-Academy”. Now all listing Clients should receive the message, see Figure 8.

In the workshop, we will not use the “mosquitto\_sub.exe” and “mosquitto\_pub.exe”. Instead, we will use libraries to implement the messaging service into our C# program. The MQTT protocol is easy to use, light and powerful. You can run it on microcontrollers, like Arduino or an ESP8266. You can use your mobile phone or tablet to send and receive MQTT messages (you will find the necessary Apps in your App-Store). Amazons assistant Alexa speaks MQTT, the facebook messenger uses the protocol and it is, de facto, the standard protocol of the Internet of Things. Take some minutes and have a look at the MQTT capabilities on <http://www.hivemq.com/mqtt-essentials/>, it will affect your career!

```

Eingabeaufforderung
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\JanJungbluth>cd C:\Program Files (x86)\mosquitto

C:\Program Files (x86)\mosquitto>mosquitto_sub.exe --help
mosquitto_sub is a simple mqtt client that will subscribe to a single topic and print
mosquitto_sub version 1.4.10 running on libmosquitto 1.4.10.

Usage: mosquitto_sub [-c] [-h host] [-k keepalive] [-p port] [-q qos] [-R] -t topic .
                  [-C msg_count] [-T filter_out]
                  [-A bind_address] [-S]
                  [-i id] [-I id_prefix]
                  [-d] [-N] [-quiet] [-v]
                  [-u username [-P password]]
                  [--will-topic [--will-payload payload] [--will-qos qos] [--will-
                  [--cafile file | --capath dir} [--cert file] [--key file]
                  [--ciphers ciphers] [--insecure]]
                  [--psk hex-key --psk-identity identity [--ciphers ciphers]]
                  [--proxy socks-url]
mosquitto_sub --help

-A : bind the outgoing socket to this host/ip address. Use to control which interface
the client communicates over.
-c : disable 'clean session' (store subscription and pending messages when client disconnects).
-C : disconnect and exit after receiving the 'msg_count' messages.
-d : enable debug messages.
-h : mqtt host to connect to. Defaults to localhost.
-i : id to use for this client. Defaults to mosquitto_sub_ appended with the process id.
-I : define the client id as id_prefix appended with the process id. Useful for when
broker is using the clientid_prefixes option.
-k : keep alive in seconds for this client. Defaults to 60.
-N : do not add an end of line character when printing the payload.
-p : network port to connect to. Defaults to 1883.
-P : provide a password (requires MQTT 3.1 broker)
-q : quality of service level to use for the subscription. Defaults to 0.
-R : do not print stale messages (those with retain set).
-S : use SRV lookups to determine which host to connect to.
-t : mqtt topic to subscribe to. May be repeated multiple times.

```

Figure 7 Picture of the command prompt while using mosquitto\_sub.exe –help command

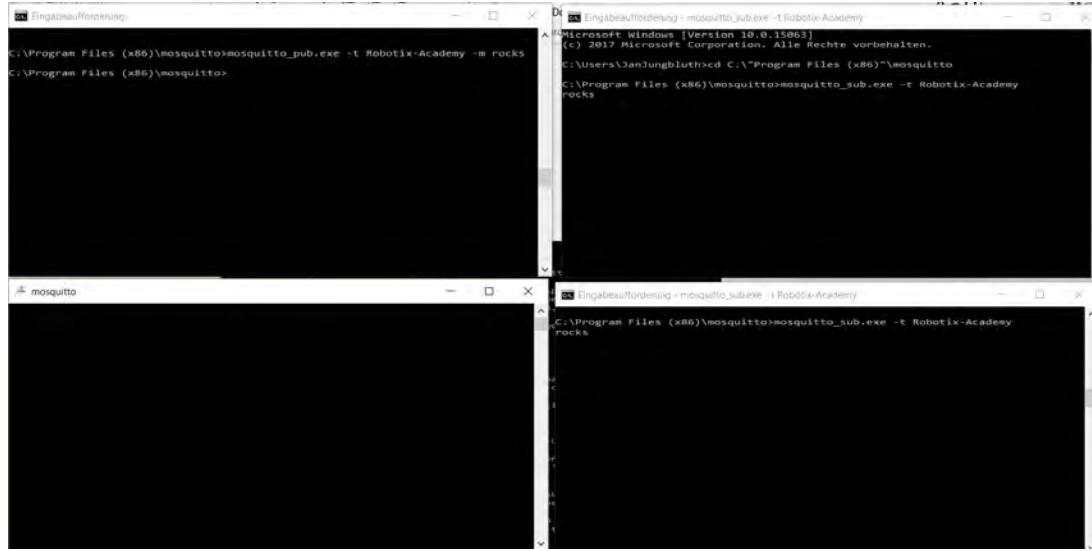


Figure 8 On the right side we see two clients receiving messages from the publishing client on the upper left command prompt. On the lower left, you see the running mosquitto MQTT broker.

### 3. Introduction to human-robot-interaction

Human-robot-interaction is the transfer of information between human and robot. The kind of information is diverse. Examples are:

- Instructions to the robot like moving the robot to a certain position
- The robot can inform the human about the current process-status
- Stopping the robot if something is going wrong

The transfer of information is bi-directional, which means that the human can provide information to the robot and vice versa.

The ability to transfer information between human and robot is often necessary to realize a human-robot-collaboration. During a collaboration, human and robot are working together on a common goal, which means that they have to coordinate the respective tasks during the whole process. To do so the human has to know what the robot does and what's the current process status and vice versa. It's also possible that the human acts as a supervisor during the collaboration and therefore has to give the robot its instructions. The transfer of these instructions is an interaction between human and robot.

One of the main goals during the development of the interaction-interface is to create an interface which is intuitively usable for humans. In the context of human-robot-collaboration, the robot is often called a co-worker. In order to see the robot as a co-worker, the human has to be able to communicate with him in the same way as with a normal (human) co-worker. This leads to communication via gestures, speech, and haptic interaction.

Besides the different mentioned forms of communication, there are also different roles for the human during a human-robot-interaction. These different roles will be explained in the following chapter.

#### 3.1 Roles of the human in an HRI

There are different roles for the human in a human-robot interaction (HRI):

Supervisor	Monitors the robot and gives him instructions
Operator	Controls the robot
Collaborator	Works dependent with the robot on a common goal
Cooperator	Works independently with the robot on a common goal

As collaborator or cooperator he works in both cases together with the robot on an overall goal. The difference between these two interaction-roles is, that as a collaborator, human and robot are also working together on sub goals, which isn't the case during a cooperation. To understand this difference, we will look at a real use case, the disassembly of a cooling water pump, which is shown in the picture below. To disassemble this pump different tasks have to be done. For example, the unscrewing of bolts or the removal of different parts, which are both sub goals. The overall goal is the complete disassembly of the cooling water pump. If human and robot working together at the unscrewing, the process is a human-robot collaboration. If the human does the unscrewing and the robot removes all other parts it is a human-robot cooperation, because human and robot don't work together on a sub goal.



Fig. 1 Cooling water pump which has different parts and links between them.

Which of these interaction-roles the human takes over depends on the task at hand. During the human-robot interaction of a certain process, the human can take over one or more of this interaction roles.

Besides the different roles for the human in an HRI there are also different communication channels which can be used to transfer information between human and robot.

### 3.2 Communication channels

To realize such an information exchange different communication channels can be used. The different channels are listed below:

- Electrical channel (GUI, AR, VR)

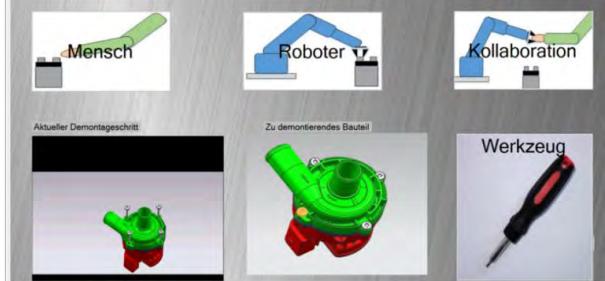


Fig. 1: Graphical-User-Interface (GUI)

- Mechanical communication

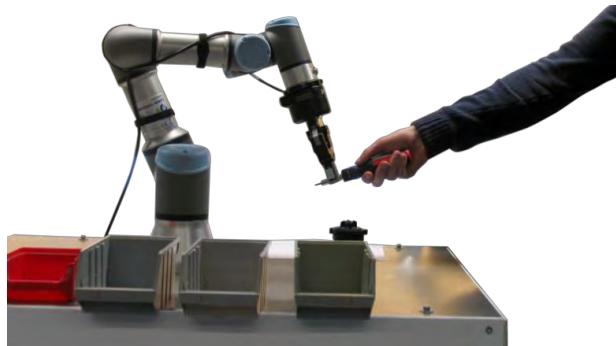


Fig. 2: Releasing the tool through force detection.

- Optical communication (Gesture recognition)

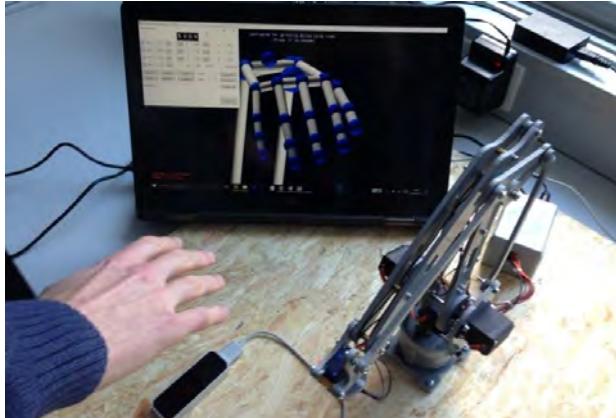


Fig. 3: Controlling the robot through hand detection and gesture recognition.

- Acoustical communication (Speech recognition)

It is possible to use one or more of this channels in one application. Which of these channels are most qualified in the present case depends on the information that needs to be communicated. To instruct the robot to execute a certain task all the listed communication-channels can be used.

Often, the first step to realize the mentioned communication, is the construction of a communication between the robot and an external PC. On the external PC, the devices and the software required to execute for example speech or gesture recognition are running. The communication between the robot and the PC is then used to exchange the data between the devices.

In the following chapter, we will program a communication interface between a Universal Robots UR3 and an external PC using a TCP/IP protocol.

## 4. Communication-interface between UR3 and a PC

UR robot can communicate with outside equipment through TCP/IP protocol. In this chapter, we will introduce you to the UR3 robot and show you how a haptic communication between the UR3 and the human can be realized. Therefore we will use the offline programming tool URSim and later test the developed programs on the real robot. We will also explain how a robot can communicate with a PC. The Robot will be the Client using URScript and the PC will be used as the server.

### 4.1 Universal Robots UR3

The UR3 is a small collaborative robot with a maximum payload of 3kg and a range of around 500mm. The tool-side joint is able to rotate infinite times which can be used for example to realize screwing tasks without attaching an additional electrical screwdriver onto the robot. Another feature of the UR3 is the ability to measure the force and torque in the individual joint. With this measurement, it's possible to detect collisions and avoid harming the human and the robot. The force measurement is based on a current measurement at the electric drive of every joint. Because of this approach, the accuracy of the force measurement isn't good.



Fig. 4: Universal Robots UR3 (Quelle: <https://www.universal-robots.com/de/produkte/ur3-roboter/>)

Some technical data to the Universal Robots UR3:

- Weight: 11kg
- Payload: 3kg
- Range: 500mm
- Degrees of freedom: 6 rotating joints
- Rotation of the joint: +/- 360°; infinite rotation of joint 6
- Speed: up to around 1m/sec
- Repeatability: +/- 0,1mm
- Collaboration operation: 15 advanced adjustable safety functions
- Programming: Polyscope graphical user interface (Fig. 5)

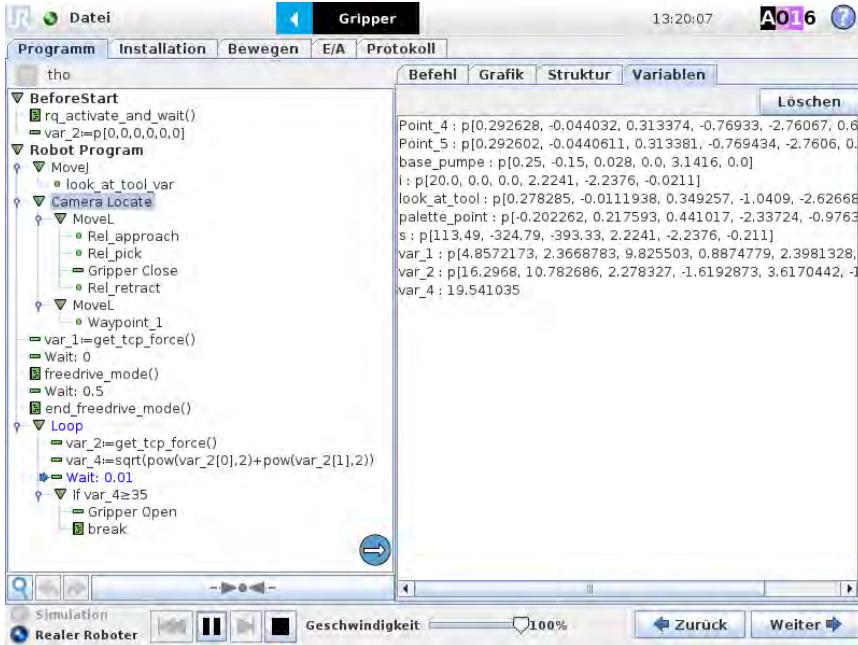


Fig. 5: Polyscope user-interface of the Universal Robots UR3.

Besides using the Polyscope user-interface to program the robot it's also possible to program the robot via the URScript Programming language (Fig. 6).

```
def My_first_program():
    set_analog_inputrange(0, 0)
    set_analog_inputrange(1, 0)
    set_analog_outputdomain(0, 0)
    set_analog_outputdomain(1, 0)
    set_tool_voltage(24)
    set_runstate_outputs([])
    set_payload(0.0)
    set_gravity([0.0, 0.0, 9.82])
    while True:
        $ 0 "Robot Program"
        $ 1 "MoveJ"
        $ 2 "Waypoint_1"
        movej([-0.7601482324296471, -1.9284112483400442, 2.4200850009312065,
-2.13148960204731, -1.562351390833685, -0.9523963238633675], a=1.3962634015954636,
v=1.0471975511965976)
        $ 3 "Waypoint_2"
        movej([-0.7601145807261123, -1.925313457229536, 1.4271208291636501,
-1.1406326407517442, -1.5621569587688118, -0.9518539657810257], a=1.3962634015954636,
v=1.0471975511965976)
    end
end
```

Fig. 6: Example of a program written with URScript Programming Language.

In the next chapter, we will start to program the Universal Robots UR3. The goal is to create a haptic interaction between the UR3 robot and human.

## 4.2 Creating a mechanical communication with the UR3

The mechanical communication we will realize during this chapter should include the following process steps:

- UR3 robot grips a screwdriver on a certain position
- UR3 robot hands over the tool to the human
- UR3 robot monitors the force at the TCP
- Human grips the handle of the screwdriver and pushes or pulls the screwdriver
- UR3 robot detects the force peak and opens the gripper

To grip the tool, we use the electrical gripper “2-finger-robot gripper 85” from Robotiq. The advantage of this gripper is that he can be automatically integrated into the controller of the UR3. He also comes with a GUI (Fig. 7) for the Polyscope user-interface.



Fig. 7: GUI of the Robotiq gripper

All the commands required to complete the tasks are available in the Polyscope interface.

To start with the exercise you have to start the VMWare and then open the URSim for the UR3 robot. After that, you have to install the URCap for the Robotiq-gripper like explained in chapter 2.

Now go to **Program Robot** and open an **Empty Program**. The programming interface will open.

**The first step** is to install a Tool-Center-Point (TCP) for the Robotiq gripper. To do so open the tap **Installation** and tip the figures shown in the picture below. We will use this TCP for the move commands.

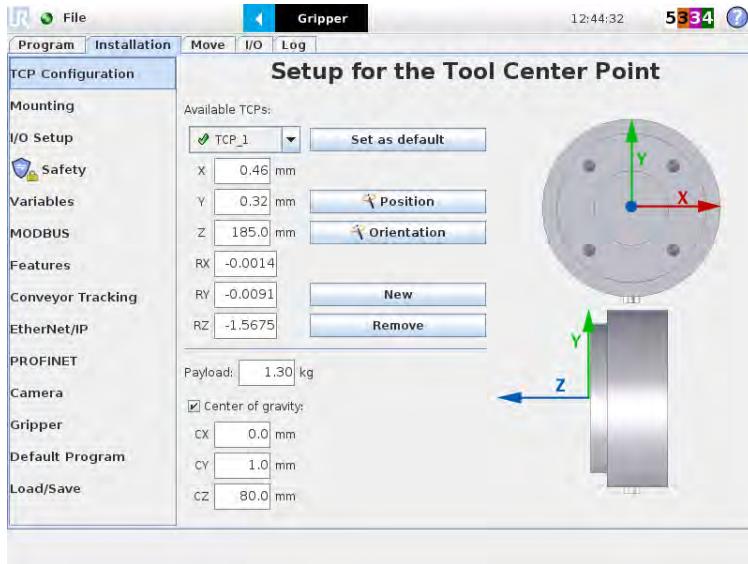


Fig. 8: Polyscope, TCP Configuration

The next step is the programming of the required move commands. We need:

- A home-position
- Points for the gripping of the tool
- A point to hand the tool to the human like shown in Fig. 2

To program a move command, go to the tab **Structure** and then choose **Move** (Fig. 9).

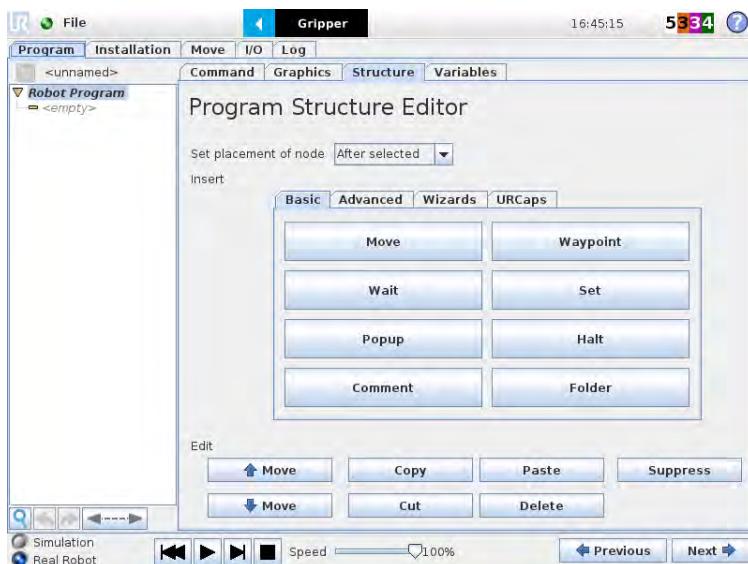


Fig. 9: Polyscope interface Structure

If you then go back to the tab **Command** you will be able to configure the move command. It isn't possible to choose the exact points for the tasks in the URSim because there is no option to include a CAD model of the application into the URSim. This means that we only roughly teach the points in the URSim and adjust them later on the real robot.

**The third step** is the adding of the gripping command to the program. Therefore we will use the UR-Caps of the Robotiq gripper. To use URCaps in a program you have to go to: Structure → URCaps and then choose the URCaps you want, in our case the **Gripper**.

The last step in the offline simulation is the adding of the loop which monitors the force at the TCP and opens the gripper if a force peak is detected. For the realization of this task we need to monitor the force in X- and Y-direction relevant to the TCP. To get the actual force you have to create an **Assignment** and then use the function `get_tcp_force()` (Fig. 10). Then you have to build the magnitude between the X- and Y-direction. If the force peak reaches 35N or more the gripper should open and release the tool.

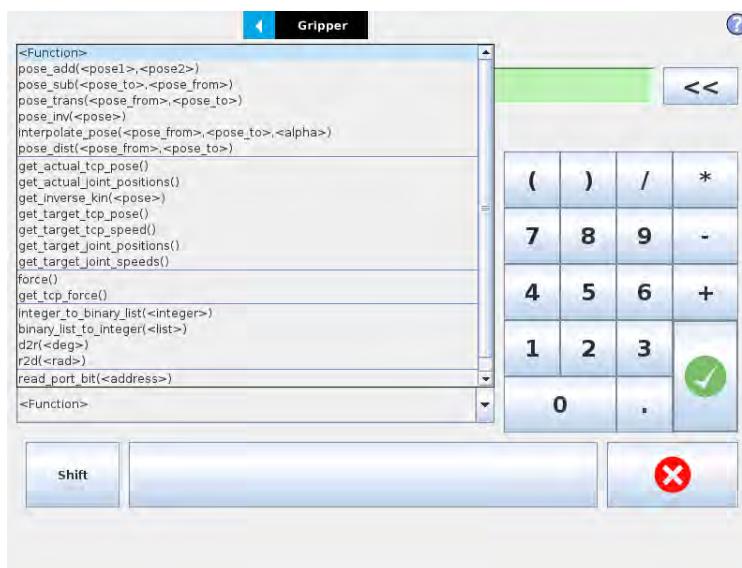


Fig. 10: Functions within the Polyscope GUI

After the program is finished we can adapt it to the real world and test it with the UR3. Therefore you have to save your program and then copy the .script file from the folder (home/ursim-current/programs) to the USB-stick. Afterwards, we can plug the USB-stick to the UR3 robot and copy the file to the robot-controller.

Unfortunately, it isn't possible to test the whole program in the URSim. The reason behind is that the functions behind the URCaps don't get any signal back from the Robotiq gripper if he isn't connected to the PC. Another reason is that the robot can't detect any forces within the simulation. It's only possible to test the waypoints if you suppress the URCaps and Force commands.

## 5. Communication between the virtual machine and a program

To be able to communicate with the virtual machine from a locally running program follow these steps:

- Start the URSim VM
- In VMware select *Player -> Manage ->"virtual machine settings" -> network adapter -> custom* select VMnet8 (NAT)

- Look for your IP within the VM by right clicking on the network button in the right corner of the taskbar in Ubuntu; It might be something like 192.168.95.128

## 5.1 Programming a tool to communicate with the Universal Robots

At the beginning, create a new **Windows Forms-Project**, as shown before.

Now build a graphical user interface (GUI), like the one shown in Figure 1. Just click on the toolbox on the left side on the window and drag and drop the single elements into the **Form1** window.

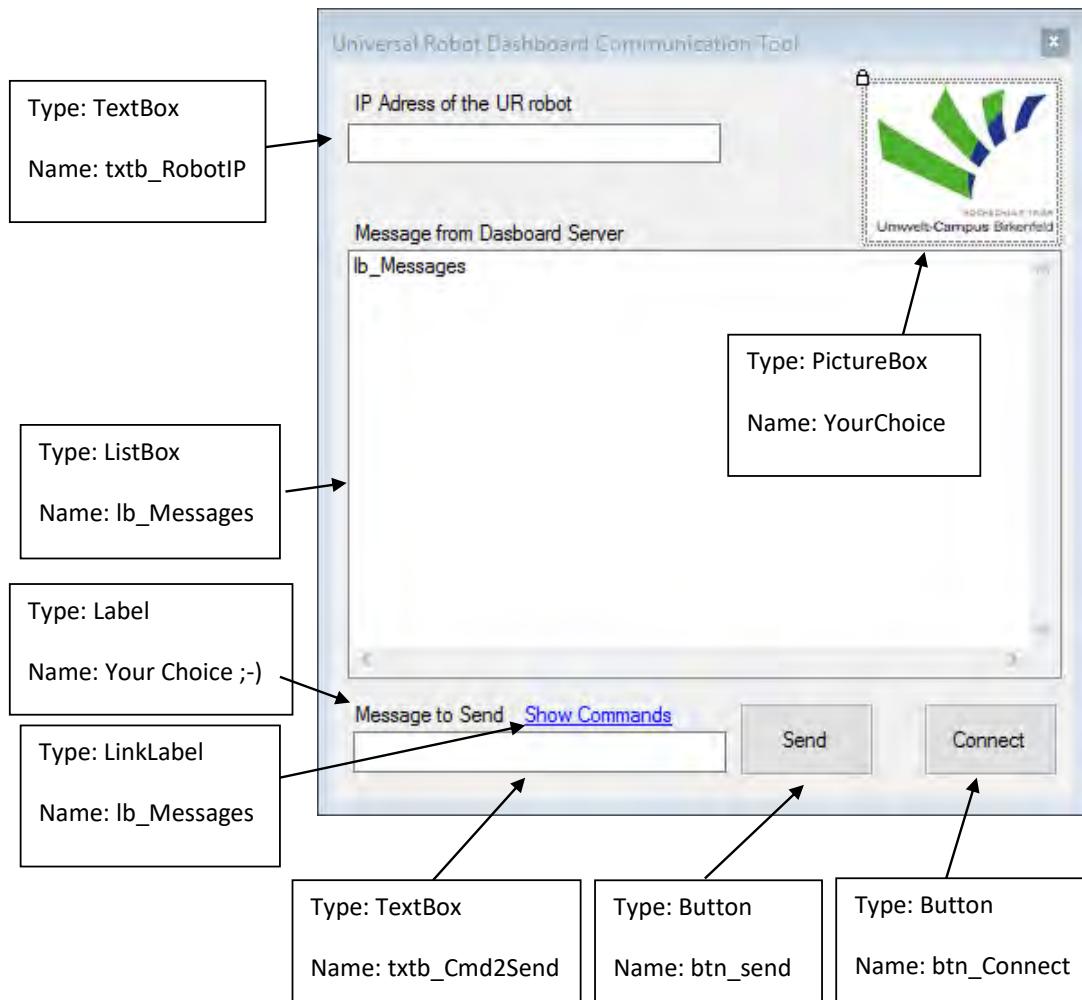


Figure 9: The GUI we want to build

After you finished building the GUI, let's fill our little program with life.

Go to the properties tab of the **txtb\_Cmd2Send** TextBox. Within the **other** section, look for the **AutoCompletionCustomSource** entry. There you can fill in the following statements, which will be automatically completed when you begin to type them in:

- load <name>.urp
- play
- stop

- pause
- quit
- shutdown
- running
- robotmode
- get loaded program
- popup <text>
- close popup
- addToLog <log\_msg>
- isProgramSaved
- programState
- PolyscopeVersion
- setUserRole programmer
- setUserRole operator
- setUserRole none
- setUserRole locked
- setUserRole restricted
- power on
- power off
- brake release
- safemode
- unlock protective stop
- close safety popup
- load installation <inst\_file>

Within **AutoCompletionMode** choose **SuggestAppend** and in **AutoCompletionSource** choose **CustomSource**.

Now switch to the **Form1.cs**. Here we have to add some objects and functions to initialize our communication. Let's begin from top to bottom. Complete the **using** statements at the top of the file. These statements allow us to use functions from other namespaces than the one we are currently using. The list should look as follows:

```
using System; //used by many functions
using System.Diagnostics; //used for starting the Internet Explorer
using System.Text; //used to encode the text into the proper format
using System.Windows.Forms; //we are writing our code in Windows Forms?
using System.Net; //used for network communication
using System.Net.Sockets; //used for network communication
```

Right under the line

```
public partial class Form1 : Form
complement

// Creation of a TCP-Client and a suitable Networkstream
private TcpClient client = new TcpClient();
private NetworkStream stream;

// Data buffer for incoming and outgoing data.
```

```
private byte[] bytes_received = new byte[1024];
private byte[] bytes_send = new byte[1024];
```

Within the constructor of Form 1 (public Form1()), you can append these lines

```
this.txtb_RobotIP.Text = "192.168.2.2"; //to prefill the IP-Address TextBox
this.AcceptButton = btn_send; //to connect the button with the Enter-Key
```

Now switch to the Design Tab of the program and click on the LinkLabel and switch there to the event-tab within the properties. Doubleclick on the **LinkClicked** event. Visual Studio will automatically create a new function for this event. Within this function write the following instruction:

```
Process.Start("iexplore.exe", "https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/dashboard-server-port-29999-15690/");
```

This instruction calls the Internet Explorer and passes the URL of the suitable “Universal Robots How-To” Page for using the dashboard server.

Now switch back to the Design Tab of your program. Double click there onto the **Connect** Button. Visual Studio creates the next function to handle the clicked-event on that button. In that newly created function, add the part to connect the program with either the VM or the Universal Robot:

```
private void btn_Connect_Click(object sender, EventArgs e)
{
    if (!this.client.Connected)
    {
        try
        {
            // Parse the string to an IPAdress instance
            IPAddress ipAddress = IPAddress.Parse(this.txtb_RobotIP.Text);
            // Connect to the Dashboard server
            this.client.Connect(ipAddress, 29999);
            // invoke the GUI Thread to give connection feedback
            this.Invoke(
                MethodInvoker
                (() => this.lb_Messages.Items.Add(String.Format("Connected to {0}", this.txtb_RobotIP.Text))));
            // initialise the networkreader
            this.stream = this.client.GetStream();
            // read out the byte response
            this.stream.Read(this.bytes_received, 0, this.bytes_received.Length);
            // convert the byte message to readable string
            string msg_received = System.Text.Encoding.UTF8.GetString(this.bytes_received);
            // invoke the GUI Thread to give display the response
            this.Invoke((MethodInvoker)((() => this.lb_Messages.Items.Add(msg_received))));
            // enable the send button
            this.btn_send.Enabled = true;
            // change the text of the connect button
            this.btn_Connect.Text = "Disconnect";
        }
        catch (Exception er)
        {
            // catch and display the throw exceptions
        }
    }
}
```

```

        this.Invoke((MethodInvoker)((() => this.lb_Messages.Items.Add(er.ToString()))));
    }
}
else
{
    // We will disconnect from the Dashboard by closing the connection
    this.client.Close();
    this.client = null;
    // disable the send button
    this.btn_send.Enabled = false;
    // change the connect button to connect
    this.btn_Connect.Text = "Connect";
}
// empty the byte storage
this.bytes_send = new byte[this.bytes_send.Length];
this.bytes_received = new byte[this.bytes_received.Length];
}

```

Switch back to the Design tab and double click on the **Send** button. Unsurprisingly Visual Studio will create a function then as well.

```

private void btn_send_Click(object sender, EventArgs e)
{
    // get the message from the text box and convert it to a byte array
    this.bytes_send = Encoding.UTF8.GetBytes(this.txtb_Cmd2Send.Text + '\n');
    // remove the text from the text box
    this.txtb_Cmd2Send.Text = "";
    // write the message into the stream
    this.stream.Write(this.bytes_send, 0, this.bytes_send.Length);
    // read the response bytestream
    int bytesRec = this.stream.Read(this.bytes_received, 0, this.bytes_received.Length);
    // encode the response
    string msg_received = System.Text.Encoding.UTF8.GetString(this.bytes_received);
    // display the response
    this.Invoke((MethodInvoker)((() => this.lb_Messages.Items.Add(msg_received))));
    // empty the byte buffer
    this.bytes_send = new byte[this.bytes_send.Length];
    this.bytes_received = new byte[this.bytes_received.Length];
}

```

## 5.2 Communicate with the UR via MQTT

In this chapter, we will program a small MQTT client and thereby give you a short introduction to the MQTT protocol. At the end of the workshop, we will connect the client to the dashboard communication program from the last chapter. We will then be able to control the robot through any device or program that uses the MQTT protocol.

### Introduction to MQTT

MQTT is a communication protocol based on TCP/IP and the publish/subscribe mechanism. In this architecture, a server, called a broker, is in charge of organizing the message flow between all the clients. Different communication channels are realized by topics. Each topic is a communication channel where interested clients can communicate with each other in a specific context. If a client

connects to the broker, he has to specify which topics he wants to subscribe and what additional communication settings are used. Clients can publish messages on a topic by sending the message to the broker. The broker will then take care that all topic subscribers will receive the message, see Figure 10.

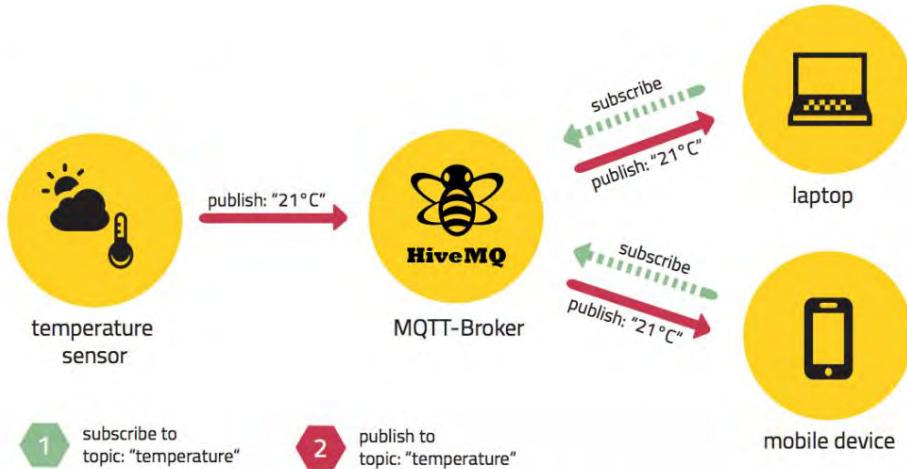


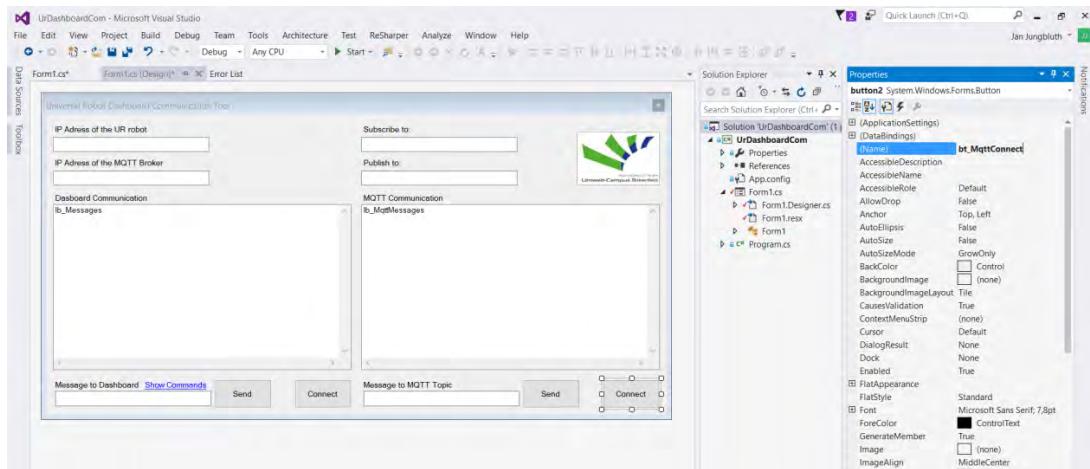
Figure 10 The publish/subscribe principle of MQTT.

### Let's start programming

We will use the project you already created for the dashboard communication with the UR Robot. We will add some controls to the GUI before we start coding. We will add four text boxes with labels, two buttons and one list box with a label, see Table 1 and Figure 11. The easiest way to do this is to select and copy the controls. Thereafter you have to remember to rename the controls according to Table 1.

Table 1 Overview of the control elements

Control Type	Name	Function
TextBox	txtb_BrokerIP	We will read from this text box the IP address of the Broker.
TextBox	txtb_SubscribeTo	We will read from this text box the topics which we will subscribe to.
TextBox	txtb_PublishTo	We will read from this text box the topic we want to publish messages.
TextBox	txtb_MqttMessage	In this text box, we will enter the messages we want to publish in the publishing-topic.
Button	bt_MqttSend	This button we will publish the message.
Button	bt_MqttConnect	This button will connect our client to the MQTT broker.
ListBox	lb_MqttCommunication	We will display all incoming and outgoing MQTT messages in this list box.



**Figure 11** The extended GUI

After you added the controls, we have to load the MQTT libraries into our project. We will use therefore the NuGet Package Manager in Visual Studio.

You find it under “Tools”->“NuGet Package Manager”-> “Manage NuGet Packages for Solution...”.

In the new window select the “Browse” label and search for “M2Mqtt”. You will find an entry from Paolo Patierno, select it and also select the check box next to “Projekt” and “Your Project Name”. Then hit install and close the window.

After the installation of the MQTT library, we have to add the code line

```
using uPLibrary.Networking.M2Mqtt;
```

on the top of our program. The using directive will give us access to the namespace of the library so that we can use its functionality.

The next thing is to declare the MQTT client instance as a field of the Form1 class.

```
private MqttClient mqttClient;
```

To connect to the MQTT broker we use the Click Event of the “bt\_MqttConnect” button, therefore change to the Form1.cs[Design] view and double click on the button. You will be redirected to the Form1.cs source code and see the empty “bt\_MqttConnect\_Click” method. We will now add the necessary code - to connect to the broker – in the body of the method, see Figure 12.

```

1 reference | 0 changes | 0 authors, 0 changes
private void bt_MqttConnect_Click(object sender, EventArgs e)
{
    // here we initialise the mqtt client instance with the broker ip address
    this.mqttClient = new MqttClient(this.txtb_BrokerIP.Text);

    // Next we specify what topics we want to subscribe
    this.mqttClient.Subscribe(new string[] {this.txtb_SubscribeTo.Text}, new byte[] {MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE});

    // we couple the message received event with the method MqttMsgReceived
    this.mqttClient.MqttMsgPublishReceived += MqttMsgReceived;

    // we try to connect to the broker using a globally unique identifier (Guid) as client id
    try
    {
        this.mqttClient.Connect(Guid.NewGuid().ToString());
    }
    catch (Exception)
    {
        MessageBox.Show("Can't connect to broker");
    }
}

1 reference | 0 changes | 0 authors, 0 changes
private void MqttMsgReceived(object sender, MqttMsgPublishEventArgs mqttMsgPublishEventArgs)
{
    throw new NotImplementedException();
}

```

**Figure 12 Source code to connect to the MQTT broker**

We create the MQTT client instance with the IP address, the port is by default 1883. Then we specify what topics we want to subscribe. We can subscribe to multiple topics just by adding the topics to the string array. For each topic, we have to specify the Quality of Service (QoS) in the byte array. There are three QoS levels defined by the standard:

- QoS 0 will deliver the message once, with no confirmation.
- QoS 1 will deliver the message at least once, with confirmation required.
- QoS 2 will deliver the message exactly once by using a four step handshake.

To make sure you really get the message exactly once, you have to publish and to subscribe with QoS 2! If the MQTT client receives a message, a “MqttMsgPublsihReceived” event is fired. In order to get informed when this event occurred, we have to register the “MqttMsgReceived” method to the event handler. In the body of the “MqttMsgReceived” method, we describe then how to react to this event. We will implement that at the end of this chapter.

Now we want to publish messages. Therefore we go again to the Form1.cs[Design] window and double click on the “Send” button. It will create a method called “bt\_MqttSend\_Click”. In the body of the method, we will place the code from Figure 13.

```

1 reference | 0 changes | 0 authors, 0 changes
private void bt_MqttSend_Click(object sender, EventArgs e)
{
    string msg = this.txtb_MqttCmd.Text;
    string topic = this.txtb_PublishTo.Text;

    this.Invoke((MethodInvoker)((() =>
    {
        this.lb_MqttMessages.Items.Add("Send " + topic + " " + msg);
        this.Refresh();
    }));
}

this.mqttClient.Publish(topic, Encoding.UTF8.GetBytes(msg), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
}

```

Figure 13 The source code of the "Send" button.

First, we read the msg and the topic from the text boxes. Then we use the MethodInvoker to add the message to the list box and refresh the GUI. After that, we publish our message in the topic. As you can see we have to convert the message string into a byte array with the UTF8 encoding. We also have to specify the QoS level for publishing. The last boolean parameter indicates if we want to leave a retained message (true) on the broker or not (false). If we want to leave a message on the broker, the message will be saved for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing.

Now we should edit the “MqttMsgReceived” method. Add the code in Figure 14.

```

1 reference | 0 changes | 0 authors, 0 changes
private void MqttMsgReceived(object sender, MqttMsgPublishEventArgs mqttMsgPublishEventArgs)
{
    string topic = mqttMsgPublishEventArgs.Topic;
    string msg = Encoding.UTF8.GetString(mqttMsgPublishEventArgs.Message);

    this.Invoke((MethodInvoker)((() =>
    {
        this.lb_MqttMessages.Items.Add("Received " + topic + " " + msg);
        this.Refresh();
    }));
}

```

Figure 14 Source code of the MqttMsgReceived event

From the MqttMsgPublishEventArgs we can get the topic, the message in the byte format, the QoS level of the message and much more information. We use again the MehtodInvoker to add the message to our MQTT list box.

Now we can test our program. Therefore start the mosquitto broker on your computer and start the program. In the text box for the broker IP address, enter “127.0.01”. This is an internal IP address of your computer. In the subscribe text box write the MQTT wild card “#”. With the wild card, you will receive on all topics on that level messages. In the next text box write the topic within you want to publish. Click on the “Connect” button, add a text to send and press the “Send” button. It should look like Figure 15. As you will notice the program will run even you close the window. This is because the MQTT client uses threads we don’t kill. To kill the threads by closing the window, go to the “Form-Closing” method. There add the code from Figure 16.

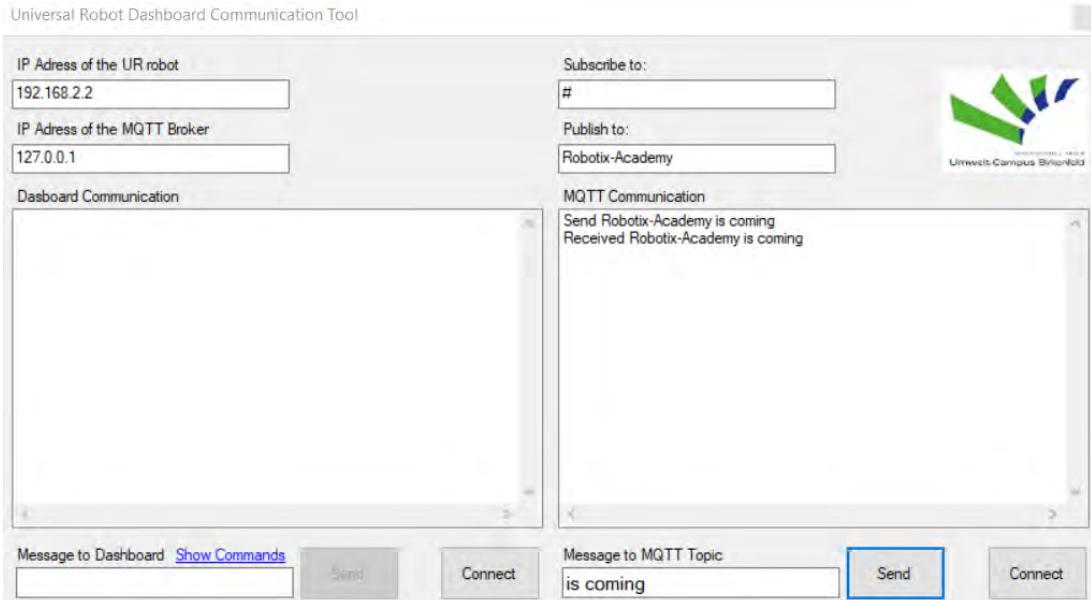


Figure 15 First test of our program

The next step is to send a robot command through MQTT to the universal robot. We will do this by a dirty but fast approach. In the “MqttMsgReceived” method we will wait on a special topic to indicate the robot command, then we will write the robot command in the “txtb\_Cmd2Send” text box and create a button click event. You see the adapted code in Figure 17.

```
1 reference | 0 changes | 0 authors, 0 changes
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    this.stream?.Close();
    this.stream = null;
    ...
    this.tcpClient?.Close();
    this.tcpClient = null;
    ...
    this.mqttClient?.Disconnect();
    this.mqttClient = null;
}
```

Figure 16 Killing all the MQTT client threads.

```
1 reference | 0 changes | 0 authors, 0 changes
private void MqttMsgReceived(object sender, MqttMsgPublishEventArgs mqttMsgPublishEventArgs)
{
    string topic = mqttMsgPublishEventArgs.Topic;
    string msg = Encoding.UTF8.GetString(mqttMsgPublishEventArgs.Message);
    this.Invoke((MethodInvoker)((() =>
    {
        this.lb_MqttMessages.Items.Add("Received " + topic + ":" + msg);
        this.Refresh();
    }));
    if (topic.Equals("URCmd"))
    {
        this.Invoke((MethodInvoker)((() =>
        {
            this.txtb_Cmd2Send.Text = msg;
            this.Refresh();
            this.btn_send_Click(null, null);
        }));
    }
}
```

Figure 17 The adapted source code of the MqttMsgReceived method.

If you connect to the universal robot Dashboard server you can now send robot commands through the MQTT protocol, see Figure 18. If you use this code to control real robots please be careful!

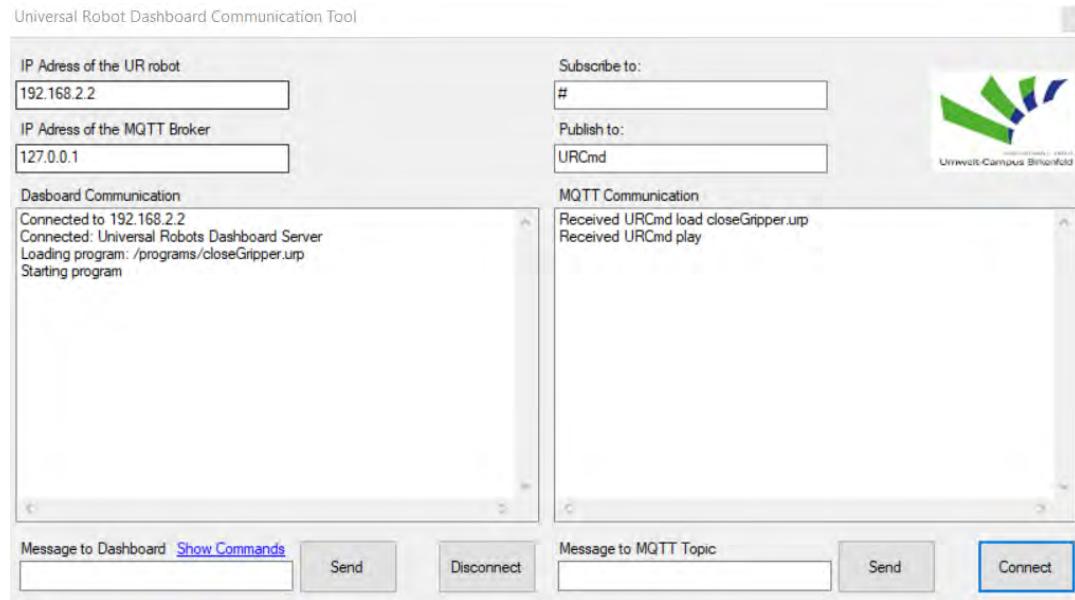


Figure 18 The last test of our program was successful.



## Pôle MecaTech, Impulsgeber für Innovationskraft im Maschinenbau

Der Pôle MecaTech hat sich die Entwicklung von Berufen und Beschäftigung durch den Aufbau und die Umsetzung von innovativen Projekten mit internationaler Ausrichtung zur Aufgabe gemacht. Zu diesem Zweck vernetzt der Pôle Großunternehmen, KMU, Universitäten sowie Forschungs- und Kompetenzzentren. Das Ziel ist es für den Pôle eine treibende Kraft des gesamten Maschinenbausektors zu sein.

### 6 wesentliche Märkte

Im Laufe der Jahre hat sich eine Konzentration von Fachwissen rund um die 6 „wesentlichen Märkte“ entwickelt, für die die Unternehmen des Pôle MecaTech Produkte, Dienstleistungen und industrielles Equipment bereitstellen:

- Gesundheit/ medizinische Geräte und Einrichtungen
- Bauen und Wohnen/ Baumaschinen und -systeme

## Le pôle MecaTech, moteur d'innovation en génie mécanique

Le Pôle MecaTech a pour mission de développer de l'activité et de l'emploi par le montage et la réalisation de projets innovants à vocation internationale. A ces fins, le Pôle met en réseau des grandes entreprises, des PME, des universités, ainsi que des centres de recherche et de compétences. L'objectif du Pôle est d'être une force d'entraînement de l'ensemble du secteur génie mécanique.

### 6 marchés prioritaires

Au fil des ans, une concentration des compétences s'est créée autour de 6 « marchés prioritaires » pour lesquels les entreprises du Pôle MecaTech fournissent des produits, services et équipements industriels:

- Santé & bien-être / équipements & dispositifs médicaux
- Habitat & construction / systèmes & équipements de construction
- Energie & environnement / systèmes &

- Energie und Umwelt/ Energie- und Umweltsysteme und –ausrüstung
- Transport und Mobilität/ Transportsysteme und –ausrüstung
- Industrie/ Industriesysteme und –ausrüstung
- Verteidigung und Sicherheit

### **Die Digitalisierung**

Der Stellenwert der Digitalisierung in der Strategie des Pôle MecaTech nimmt immer weiter zu und nimmt rund um drei Aktionsebenen Form an:

- Eine Politik, die die Integration der Digitalisierung in die von den Mitgliedern des Pôle entwickelten und vermarkteten Produkte fördert.
- Digitalisierung industrieller Prozesse, mit Blick auf die Verbesserung der Qualität, der Produktivität und der Wettbewerbsfähigkeit
- Entwicklung neuer Kompetenzen in Unternehmen, die auf Digitalisierung und Automatisierung von Unternehmen und vor allem von KMU spezialisiert sind.

Dieser Prozess der „Digitalisierung der Industriestruktur“ soll sich auf eine Verstärkung der Kooperation zwischen „traditionellen“ und spezialisierten Unternehmen stützen, um so die Nachfrage bei der Digitalisierung besser mit den Kompetenzangeboten von Wallonien zu verbinden.

### **Kontakt:**

Pôle MecaTech  
Jean Denoël  
E-Mail: [Jean.denoel@polemecatech.be](mailto:Jean.denoel@polemecatech.be)

équipements d'énergie et d'environnement

- Mobilité & transport / systèmes & équipements de transport
- Industrie / systèmes & équipements industriels

- Défense & sécurité

### **Le numériquerie**

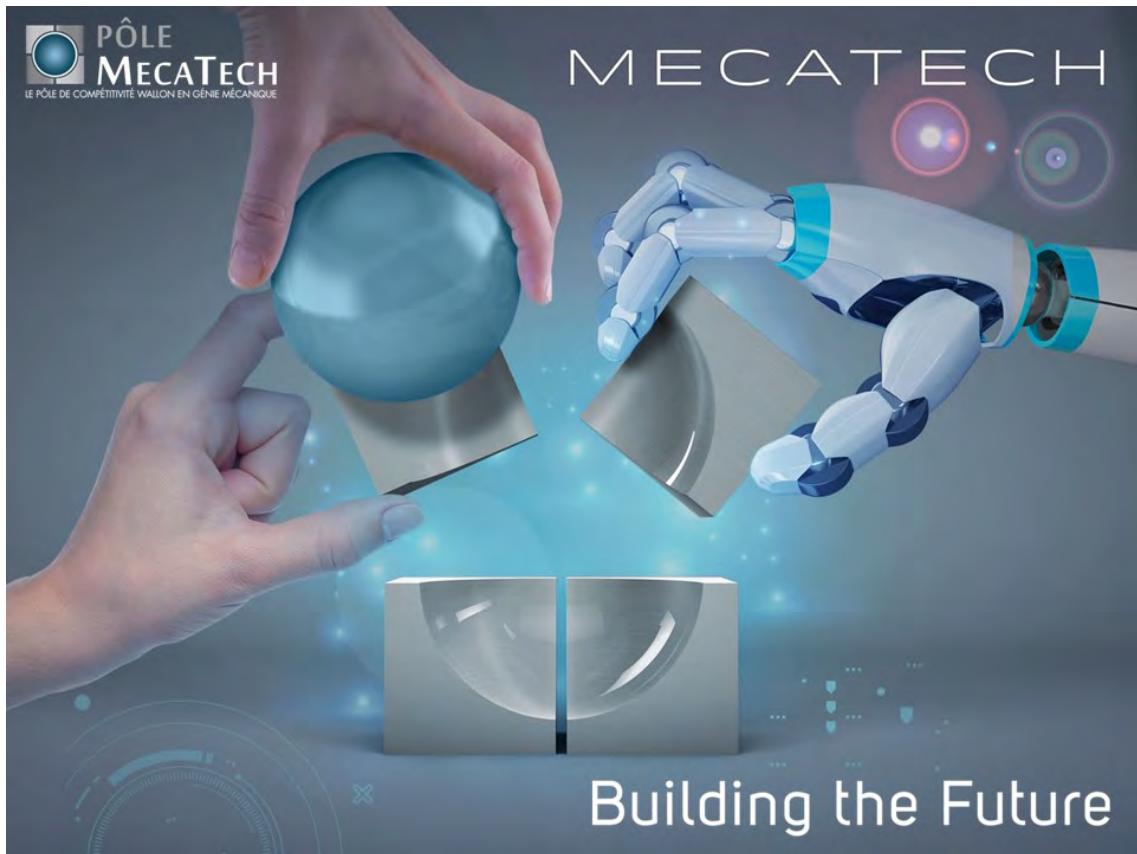
L'importance du Numérique dans la stratégie du Pôle MecaTech continue de s'accélérer et prend forme autour de trois niveaux d'actions :

- politique visant à favoriser l'intégration du numérique dans les produits développés et commercialisés par les membres du Pôle
- numérisation des processus industriels, en vue d'améliorer la qualité, la productivité et la compétitivité
- développement de nouvelles compétences au sein d'entreprises spécialisées dans la numérisation et l'automatisation des entreprises et particulièrement des PME.

Ce processus de « numérisation du tissu industriel » doit s'appuyer sur un renforcement de coopération entre les entreprises « traditionnelles » et les entreprises spécialisées afin de mieux croiser les demandes de numérisation avec l'offre de compétences en Wallonie.

### **Contact:**

Pôle MecaTech  
Jean Denoël  
e-mail: [Jean.denoel@polemecatech.be](mailto:Jean.denoel@polemecatech.be)



1. The Clusters' assigned objectives
2. MecaTech Competitiveness Cluster's scope
3. Network effects and creativity
4. Results
5. Acceleration of development
6. Industry 4.0
7. MecaTech CC' strategy
8. Organization and financing

## MECATECH

# 1. The clusters' assigned objectives

### OBJECTIVE :

BOOST ACTIVITY AND EMPLOYMENT BY SETTING UP AND CONDUCTING INNOVATIVE PROJECTS WITH AN INTERNATIONAL DIMENSION BASED ON NETWORKS THAT COMBINE LARGE COMPANIES, SMEs, UNIVERSITIES, AND RESEARCH AND SKILLS CENTERS.

### 6 COMPETITIVENESS CLUSTERS IN WALLONIA:



## MECATECH

# 1. The clusters' assigned objectives

- **Create activity and jobs**
- **Drive Wallonia's mechanical engineering sector and contribute to re-industrialization in the region**

by setting up and carrying out innovative projects with international ambitions, based on networks that combine large companies, SMEs, universities, and research and skills centers

 **MecaTech CC is project-oriented**

- **Contextual elements:** Economy of 3.5 million people
- **The Government's choices:**
  - Projects steered by businesses
  - International panel of experts
  - Principles of governance of the clusters



## 2. MecaTech CC's scope

MecaTech CC's scope: Mechanical Engineering

- Mechanical Engineering develops **innovative “functional systems”** (machines, industrial plant, consumer goods, etc.). **Science of movement...**
- Mechanical Engineering makes use of an **increasingly varied foundation of scientific and technological knowledge and know-how** (from pure mechanics to mechatronics, chemistry, materials, functionalized surfaces, nanomaterials, organic compounds, biomimicry, and much more)
- It **covers many applied fields** (construction, power, automotive sector, medicine, machines and equipment, etc.)
  - **TECHNOLOGICAL HYBRIDIZATION, INCLUDING SOME THOUGHT IMPROBABLE**
  - **BREAK-AWAY INNOVATIONS**

**SO, NETWORKING IS NOT A FASHION, BUT AN ABSOLUTE NECESSITY**



## PÔLE MECATECH

THE POLE MECATECH IS THE COMPETITIVENESS CLUSTER IN MECHANICAL ENGINEERING :



MEDICAL DEVICES, DEFENSE, CONSTRUCTION, TRANSPORTS,  
ENVIRONNEMENT & ENERGY, INDUSTRY,...



### MECATECH

## 3. Network effects and creativity

#### UNLIKELY SYNERGISM



PROTON THERAPY SYSTEM  
FOR FIGHTING CANCER



GIANT TELESCOPES

ATELIERS DE LA MEUSE

→ MECATECH WIN-GTR PROJECT :

→ A NEW PRODUCT – PROTEUS – AND 4 PATENTS



### MECATECH

## 3. Network effects and creativity (continued)

#### UNLIKELY SYNERGISM:



INTRAOCULAR LENSES FOR  
CATARACTS

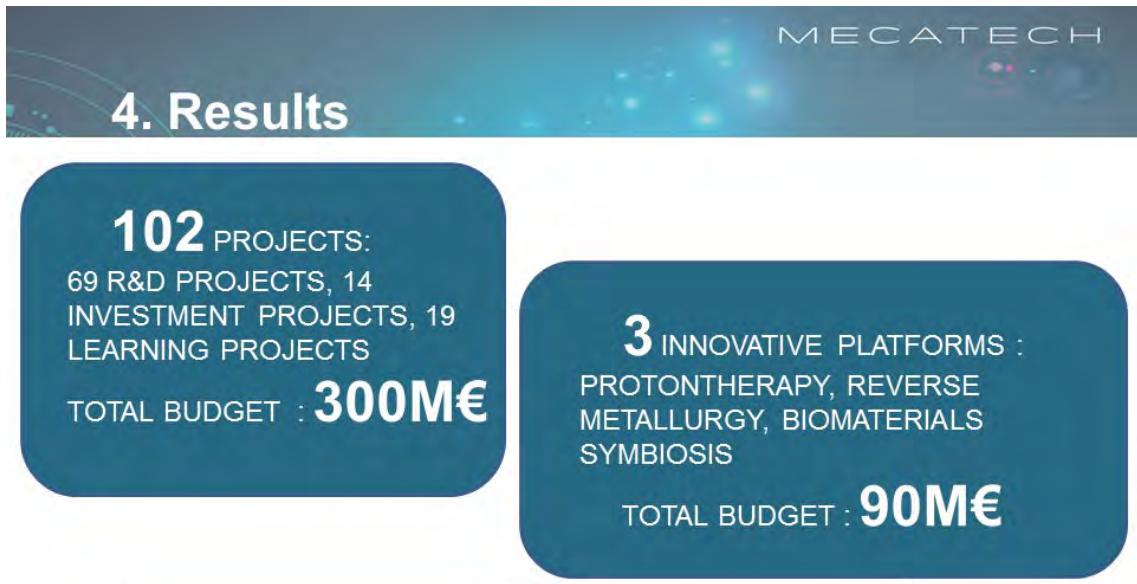


MIRRORS FOR TELESCOPES

MIRAGE PROJECT

→ MECATECH LIONEL PROJECT: 3 NEW INTRAOCULAR LENSES AND 4 PATENTS





**EVOLUTION VALEUR AJOUTÉE** SUR UNE PERIODE DE 10 ANS :  
ENTREPRISES MEMBRES

670 MIO€ => 1.231 MIO€ (+84% VS SECTEUR +11%)

**EVOLUTION CRÉATION EMPLOI** SUR UNE PERIODE DE DIX ANS :  
ENTREPRISES MEMBRES

8.384 EMPLOIS => 11.078 EMPLOIS (+32% VS SECTEUR -5%)

**LA CRÉATION D'EMPLOI S'EST ACCOMPAGNÉE  
D'UN ACCROISSEMENT DE LA PRODUCTIVITÉ**



## MECATECH

### 5. Acceleration of development

#### ■ PHASE 1

- Broad strategic framework: top-down
  - Final products and integrated services
  - Four broad, nonexclusive technological areas (break-away innovations).

NECESSARY TO REACH A CRITICAL MASS AND MULTIPLY THE PROBABILITIES OF SYNERGISTIC PROJECTS

CONSISTENT WITH EUROPE: ACCENT ON END PRODUCTS AND CHOICE OF KEY ENABLING TECHNOLOGIES (4 AREAS IN THE KETs)

- MAINLY BOTTOM-UP INITIATIVES TAKEN BY THE PLAYERS THEMSELVES



## MECATECH

### 5. Acceleration of development

- PHASE 2: GROWING ROLE OF MECATECH CC AS A FORCE FOR MAKING PROPOSALS AND “MATCH-MAKING”
- PHASE 3: PROFESSIONALIZATION OF MATCH-MAKING ACTIVITY:  
MAPPING (800 ENTERPRISES CLASSIFIED BY FIELD OF TECHNOLOGY (16) AND APPLICATION (18))  
EACH FIELD OF TECHNOLOGY OR APPLICATION CALLS FOR **STRATEGIC STUDY.**



## MECATECH

# 5. Acceleration of development

- **16 value chains at this stage (others being studied with Idea Consult and DG06)**

- Nanomaterials
- Functionalized surfaces
- Additive manufacturing and 3D printing
- Microtechnologies ( $\mu$ mechanics,  $\mu$ electronics, and  $\mu$ fluidics, notably in medical devices)
- Power electronics and microelectronics (Integration of Grépès)
- Optimal use of sensors
- Steel/glass buildings and special techniques
- Equipment for urban agriculture
- Large and extremely high-precision equipment



## MECATECH

# 5. Acceleration of development

- **16 value chains at this stage**

- Smart manufacturing and maintenance
- Renewable energy sources (photovoltaics, smart grid, etc.)
- Circular economy and reverse metallurgy: Phoenix-Comet; Carmat – Recoval ; and Solarcycle - Comet – Recma projects
- Foundries
- Supraconductivity (MIT)
- Automation, robotics (Jtekt, Procopalst (Meusinvest), etc.
- Support for SMEs: “Factories of the Future” (Agoria), etc.



## MECATECH

### 5. Acceleration of development

- **PHASE 4:** emergence of hubs of expertise:

Materials, nanomaterials, functionalization of surfaces, waste recovery, sustainable power, smart manufacturing and maintenance, medical devices, transport/automotive engineering, construction, machines and equipment, etc.

- **PHASE 5:** development of fine-linked value chains (16 so far); in conjunction with *IDEA CONSULT* (Minister J.C Marcourt and DG06).

Support based on analysis of fine-linked value chains: additive manufacturing, automation, etc.



## MECATECH

### 6. Industry 4.0

#### Factory 4.0 REVOLUTION

- ⇒ Big changes in Technology (automatisation, robot, cobot)
- ⇒ Big changes in company evolution/big data, MIS, Data captures, planning, ERP, etc.
- ⇒ Big changes in HR/skills, know how

#### **ACTION NECESSITY**

- ⇒ Priorities in short term actions
- ⇒ Agility



## MECATECH

### 5. Acceleration of development

- **PHASE 4:** emergence of hubs of expertise:

Materials, nanomaterials, functionalization of surfaces, waste recovery, sustainable power, smart manufacturing and maintenance, medical devices, transport/automotive engineering, construction, machines and equipment, etc.

- **PHASE 5:** development of fine-linked value chains (16 so far); in conjunction with IDEA CONSULT (Minister J.C Marcourt and DG06).

Support based on analysis of fine-linked value chains: additive manufacturing, automation, etc.



## MECATECH

### 6. Industry 4.0

#### Factory 4.0 REVOLUTION

- ⇒ Big changes in Technology (automatisation, robot, cobot)
- ⇒ Big changes in company evolution/big data, MIS, Data captures, planning, ERP, etc.
- ⇒ Big changes in HR/skills, know how

#### **ACTION NECESSITY**

- ⇒ Priorities in short term actions
- ⇒ Agility



## MECATECH

# 7. Stratégie MecaTech dans le numérique

3 directions:

1. DEMAND. IOT in products (auto, medical devices, machines, PM)  
Examples: IBA, maintenance intelligente, réseaux,...

2. DEMAND. IOT in process to increase productivity, competitiveness.  
Examples JTEKT, Procoplast, Legomedic, ...



## MECATECH

# 7. Stratégie MecaTech dans le numérique

3. Business side

- opportunities to develop skills and jobs
- high computing, (Cenaero,.... )
- platform big data (Plateforme big data du gouvernement wallon: NRB, CETIC,)
- sub products for more « clever » products
- companies provides services at each stage of added value

From the conception to maintenance and recycling through material, design, marketing and supply chain



MECATECH

## LE PÔLE MECATECH EST LE LEADER POUR LA WALLONIE D'UN INTERREG « FACTORY 4.0 »

**OBJECTIF : DOPER LA COMPÉTITIVITÉ DES TPE/PME DES RÉGIONS EN TRAVAILLANT SUR PLUSIEURS DOMAINES**

- Les nouvelles technologies au service de votre productivité**  
Internet des objets, big data, réalité augmentée... Anticipez les évolutions technologiques !
- Vos marchés changent : ré-inventez vos process !**  
Les besoins et attentes de vos clients évoluent : vous aussi. Il s'agit d'être réactif, flexible et adaptable, en un mot agile !
- Améliorez votre efficacité énergétique et faites des économies**  
Repensez vos modes d'approvisionnement, testez l'éco-conception, réduisez vos déchets... l'efficacité énergétique et le respect de l'environnement sont au cœur de l'optimisation de vos process et de la réduction de votre facture !
- Mettez à profit les compétences et le potentiel de vos salariés**  
Mobilisez vos salariés et rendez-les acteurs du renouveau de votre entreprise : systèmes d'information participatifs et fluides, formation aux nouveaux outils...

**PÔLE MECATECH**  
LE PÔLE DE COMPÉTITIVITÉ WALLON EN GÉNIE MÉCANIQUE

**Interreg** France-Wallonie-Vlaanderen  
GoToSS Factory 4.0

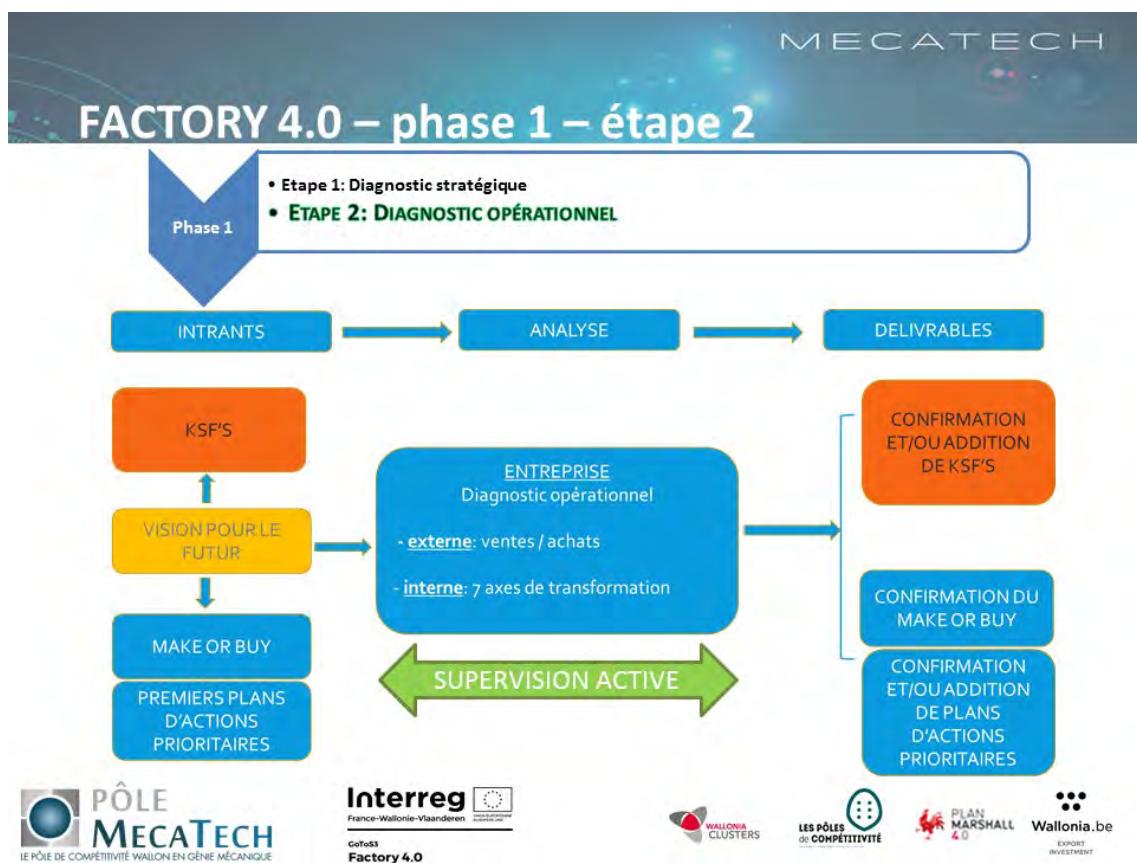
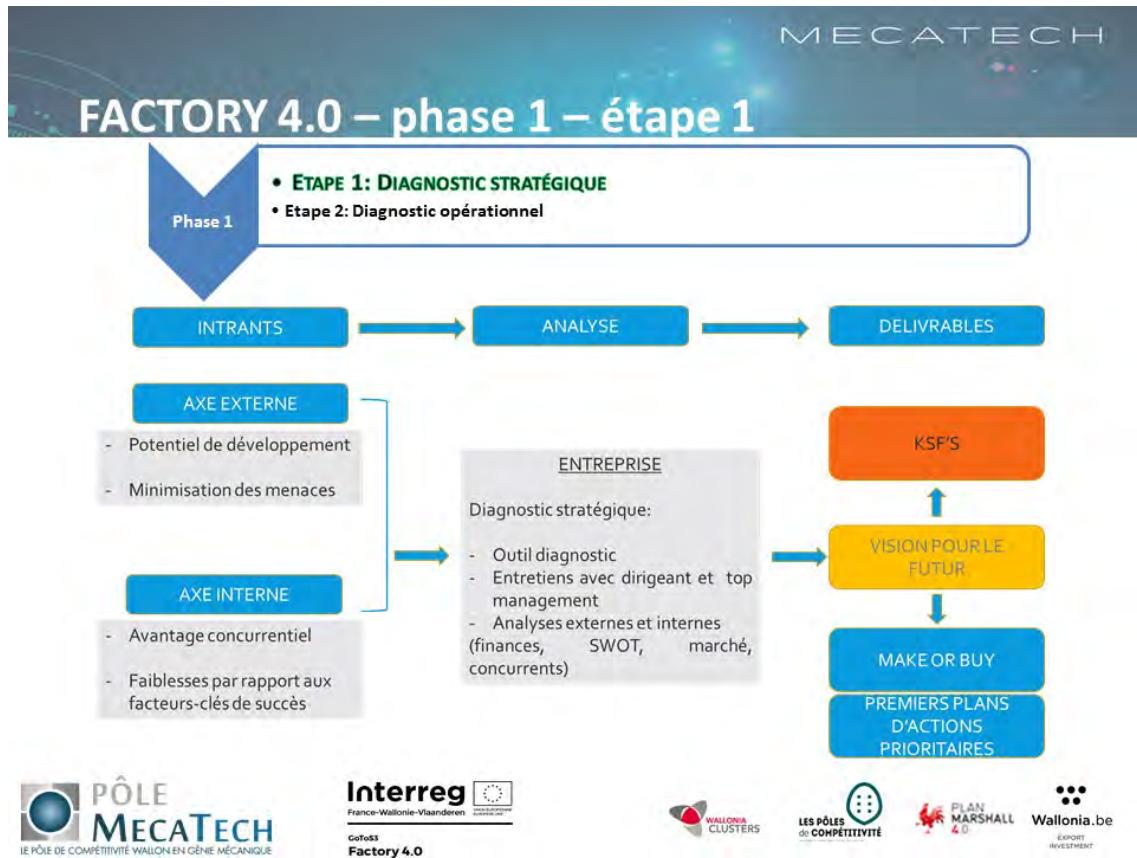
**WALLONIA CLUSTERS** **LES PÔLES de COMPÉTITIVITÉ** **PLAN MARSHALL 4.0** **Wallonia.be**  
EXPORT INVESTMENT

MECATECH

## FACTORY 4.0 - programme

Réalisation d'un diagnostic en 3 phases, d'une durée de 10-12 jours  
**100% financé par l'Interreg – pour 4 ans**

Phase	Activités	Durée	Objectif
Phase 0	• Prospection	0,5 jour	Objectif: 50/an
Phase 1	• Etape 1: Diagnostic stratégique • Etape 2: Diagnostic opérationnel	3+2 jours	Objectif: 30/an
Phase 2	• Diagnostic ciblé	5 jours	Objectif: 15/an
Phase 3	• Suivi des plans d'action	1 jour	Objectif: 15/an



**MECATECH**

## FACTORY 4.0 – phase 1 – étape 2

Phase 1

- Etape 1: Diagnostic stratégique
- **ETAPPE 2: DIAGNOSTIC OPÉRATIONNEL**

Exceller dans 7 Transformations

MADE DIFFERENT  
TECHNOLOGY OF THE FUTURE

- TOP technologies
- Simultaneous Development
- Digital Factory
- Human Centered Production
- Networked Factory
- Eco-Production
- Smart Production

sirris AGORIA

PÔLE MECATECH

Interreg France-Wallonia-Vlaanderen GoToSS Factory 4.0

WALLONIA CLUSTERS LES PÔLES de COMPÉTITIVITÉ PLAN MARSHALL 4.0 Wallonia.be

**MECATECH**

## FACTORY 4.0 – phase 2

Phase 2

- **DIAGNOSTIC CIBLÉ**

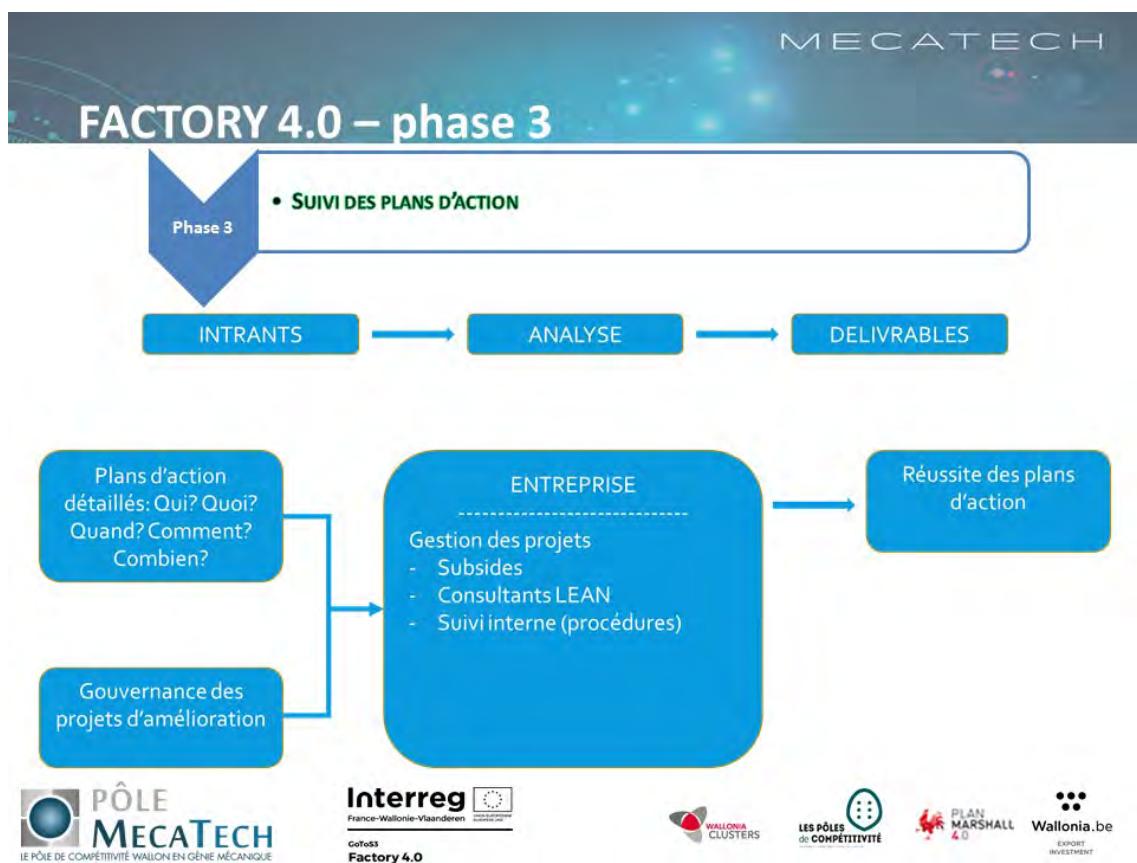
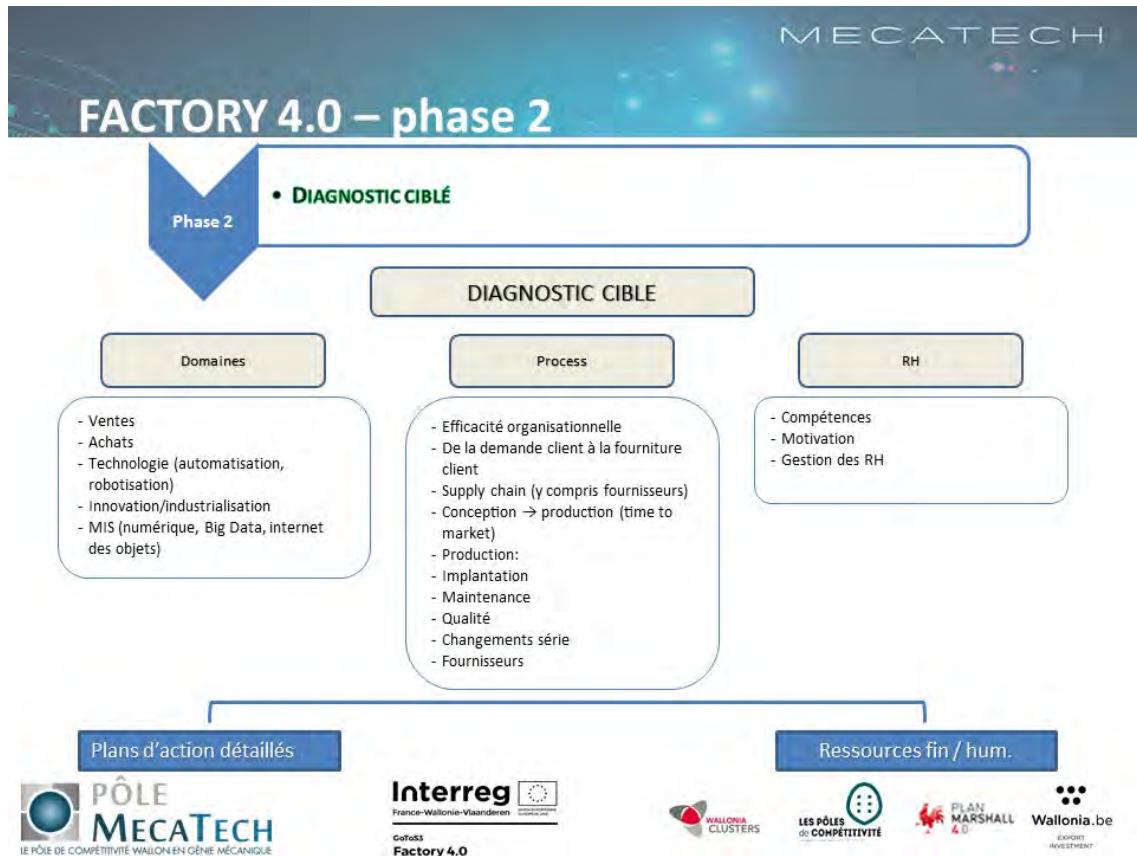
INTRANTS → ANALYSE → DELIVRABLES

```

    graph LR
        A["KSF'S venant du diagnostic opérationnel"] --> B["Confirmation du make or buy"]
        A --> C["Plans d'action prioritaires venant du diagnostic opérationnel"]
        B --> D["ENTREPRISE  
Diagnostic ciblé:  
- Revue Make or Buy / supervision active  
- Analyse domaines: ventes/achats/technologies  
- Analyse processus: administratif et production  
- Analyse management RH"]
        C --> D
        D --> E["Plans d'action détaillés  
Qui? Quoi? Quand?  
Comment? Combien?  
  
Gouvernance des projets d'amélioration"]
        E --> F["SUPERVISION ACTIVE"]
    
```

INTERREG France-Wallonia-Vlaanderen GoToSS Factory 4.0

WALLONIA CLUSTERS LES PÔLES de COMPÉTITIVITÉ PLAN MARSHALL 4.0 Wallonia.be



## MECATECH

# 8. Organization and financing.

### Agile organization.

- Small team of autonomous people (7 FTE)
- Backed up by an ecosystem
- No “support” functions
- Teleworking
- Small offices
- Little overhead aside from payroll costs



## MECATECH

# 8. Organization and financing

**Financing goal: 50% private; 50% public (Europe)**

**MecaTech CC can reach it in 2018 base on the following:**

- maintained value of private services provided (used in Europe's analyses for the silver and gold labels)
- increase in revenue
  - ✓ Increase in the number of members (more than 30 enterprises a year)
  - ✓ Changes in membership rates and success fees
  - ✓ Success fees for universities and CRA
  - ✓ Cap on cash expenditures other than wages
  - ✓ Financing of part of the “public” expenses by the cash expenditures/cash income surplus



## CONTACTS

- Jacques Germay – Administrator  
[Jacques.germay@polemecatech.be](mailto:Jacques.germay@polemecatech.be) | +32 475 767617
- Anthony Van Putte – CEO  
[Anthony.vanputte@polemecatech.be](mailto:Anthony.vanputte@polemecatech.be) | +32 476 972609
- Danielle AUBRY – Project Manager  
[Danielle.aubry@polemecatech.be](mailto:Danielle.aubry@polemecatech.be)
- Alice Szostak – Communication Manager  
[Alice.szostak@polemecatech.be](mailto:Alice.szostak@polemecatech.be) | +32 485 058195
- Jean Denoël – Training Executive  
[Jean.denoel@polemecatech.be](mailto:Jean.denoel@polemecatech.be) | +32 477 722090
- Sébastien Pinoy – Project Manager  
[Sebastien.pinoy@polemecatech.be](mailto:Sebastien.pinoy@polemecatech.be) | +32 470 217132
- Thibaud Van Rooden – International Affairs Executive  
[Thibaud.vanrooden@polemecatech.be](mailto:Thibaud.vanrooden@polemecatech.be) | +32 487 525334
- Natalie Boever – Office Manager  
[Natalie.boever@polemecatech.be](mailto:Natalie.boever@polemecatech.be) | +32 81 20 68 50



THANK YOU!



---

# Kontakt

## Contact

### Projektleitung

### Direction du projet



Rainer Müller, Prof. Dr.-Ing.  
Zentrum für Mechatronik und  
Automatisierungstechnik gGmbH  
Telefon: +49 (0)681 857 87 15  
E-Mail: rainer.mueller@zema.de  
Webseite: [www.zema.de](http://www.zema.de)

Matthias Vette-Steinkamp, Dipl.  
Wirt.-Ing. (FH), M.Eng.  
Zentrum für Mechatronik und  
Automatisierungstechnik gGmbH  
Telefon: +49 (0)681 857 87 531  
E-Mail: matthias.vette@zema.de  
Webseite: [www.zema.de](http://www.zema.de)

### Projektpartner

### Operateurs du projet



Gabriel Abba, Prof. Dr.  
Université de Lorraine  
Telefon: +33(0)387 375 430  
E-Mail: gabriel.abba@univ-lorraine.fr  
Webseite: [www.univ-lorraine.fr](http://www.univ-lorraine.fr)



Olivier Bruls, Prof.  
Université de Liège  
Telefon: +32 (0)4366-9184  
E-Mail: o.bruls@ulg.ac.be  
Webseite: [www.ulg.ac.be](http://www.ulg.ac.be)



Jean Denoël  
Pôle MecaTech  
Telefon: +32 (0)488 833 464  
E-Mail: jean.denoel@polemecatech.be  
Webseite: [www.polemecatech.be](http://www.polemecatech.be)



Wolfgang Gerke, Prof. Dr.-Ing.  
Hochschule Trier, Umwelt-Campus  
Birkenfeld  
Telefon: +49 (0)6782 17-1113  
E-Mail: w.gerke@umwelt-campus.de  
Webseite: [www.umwelt-campus.de](http://www.umwelt-campus.de)



UNIVERSITÉ DU  
LUXEMBOURG

Peter Plapper, Prof. Dr.-Ing.  
Université du Luxembourg  
Telefon : +352 (0)466644-5804  
E-mail: peter.plapper@uni.lu  
Webseite: wwwde.uni.lu

## Strategische Partner Opérateurs méthodologiques



Régis Bigot  
Manoir Industries  
Telefon: +33 (0)3 87 39 78  
Webseite: www.manoir-industries.com



Frédéric Cambier  
Technifutur  
Webseite: www.technifutur.be



Julie Corouge  
Universität der Großregion  
Telefon: +49 (0)681 301 40801  
E-Mail: julie.corouge@uni-gr.eu  
Webseite: www.uni-gr.eu



Laurent Federspiel  
LuxInnovation – National Agency  
for innovation and research  
Telefon: +352 (0)43 62 63 – 858  
E-Mail: laurent.federspiel@luxinnovation.lu  
Webseite: en.luxinnovation.lu



Amarily Ben Attar  
Institut de Soudure  
Telefon: +33 (0)3 87 55 60 76  
E-Mail: a.benattar@isgroupe.com  
Webseite: www.isgroupe.com

Christian Laurent  
Automation & Robotics  
Telefon: +32 (0)87 322 330  
E-Mail: c.laurent@ar.be  
Webseite: www.ar.be



Nigel Ramsden  
FANUC Europe Corporation  
Telefon: +352 (0)72 77 77 450  
E-Mail: nigel.ramsden@fanuc.eu  
Webseite: www.fanuc.eu



Sakina Seghir  
MATERALIA – Pôle de Compétitivité  
MatériauxMaterial, Verfahren,  
Energie  
Telefon: +33 (0)3 55 00 40 35  
E-Mail: sakina.seghir@materalia.fr  
Webseite: www.materalia.fr



Abdel Tazibt  
Critt TJFU  
Telefon: +33 (0)3 29 79 96 72  
E-Mail: a.tazibt@critt-tjfu.com  
Webseite: www.critt-tjfu.com



Grégory Reichling  
Citius Engineering  
Telefon: +32 (0)4 240 14 25  
E-Mail: gregory.reichling@citius-  
engineering.com  
Webseite: www.citius-engineering.com



[www.robotix.academy](http://www.robotix.academy)

