Interreg

EUROPEAN UNION

Grande Région | Großregion

Robotix-Academy

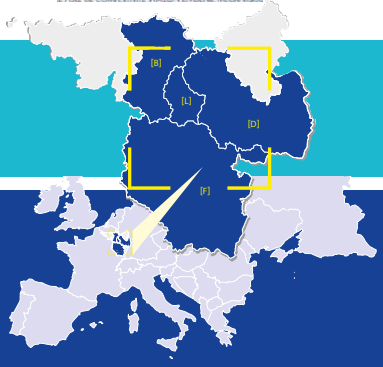
Fonds européen de développement régional | Europäischer Fonds für regionale Entwicklung

Axe prioritaire | Prioritätsachse 4
Compétitivité et attractivité
Wettbewerbsfähigkeit und Attraktivität

Robotix-Academy Summer School
September 04th to 06th 2019
Université de Metz

Partenaires du projet | Projektpartner:

ZeMA · uni.lu Université du Luxembourg · Liège université · Université de Lorraine · Umwelt-Campus Birkenfeld · Hochschule Trier · Pôle MecaTech



www.robotix.academy

Vorwort

Die Robotix-Academy konnte ihre Präsenz als internationaler Forschungscluster für industrielle Robotik und Mensch-Roboter-Kooperation weiter ausbauen. Die Projektpartner arbeiten inzwischen im vierten Jahr erfolgreich zusammen und nutzen die vorhandenen Ressourcen besonders effizient. Die Kooperation mit Verbänden und Unternehmen aus der Region hat sich sehr positiv entwickelt, so dass die Marke „Robotix-Academy“ inzwischen überregional bekannt ist und als Referenzprojekt für gute grenzüberschreitende Zusammenarbeit gilt. Die 2019 neu eingerichtete Webseite `www.robot-hub.org` für öffentlich geförderte Projekte steht auch der Robotix-Academy zur Verfügung. Die gemeinsamen Veranstaltungsformate Vorlesung, Roadshow, Summer School und wissenschaftliche Konferenz sind mittlerweile ausgereift und fester Bestandteil des Aktionsplans.
Darüber hinaus ist die Robotix-Academy 2019 wieder auf einer Vielzahl von Veranstaltungen und Messen vertreten bzw. bringt sich dort ein. Um diese Arbeit erfolgreich fortzusetzen und die Academy weiter zu verstetigen, haben sich die Partner entschieden, eine Projektverlängerung zu beantragen. Neu hinzukommen die Schwerpunkte Healthcare und Logistik sowie eine weitere Zielgruppe, die adressiert wird: Schüler zwischen 16 und 20 Jahren, die auf ein technisches Studium in der Großregion vorbereitet werden sollen. Dafür stehen weitere 18 Monate mit 30% des ursprünglichen Budgets zur Verfügung.
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.

Préface

La Robotix-Academy a pu renforcer sa présence en tant que pôle de recherche international pour la robotique industrielle et la coopération homme-robot. Depuis quatre ans maintenant, les partenaires du projet travaillent ensemble avec beaucoup de succès et utilisent les ressources disponibles de manière très efficace. La coopération avec les associations et les entreprises de la région s'est développée de manière très positive, de sorte que la marque «Robotix-Academy» est maintenant connue au niveau national et est considérée comme un projet de référence pour une bonne coopération transfrontalière. Le site web `www.robot-hub.org` qui a été nouvellement créé en 2019 pour les projets financés par des fonds publics, est également accessible à la Robotix-Academy. Les formats d'événements communs à savoir le cours magistral, la présentation itinérante, l'université d'été et la conférence scientifique sont désormais pleinement développés et font partie intégrante du plan d'action.
En outre, la Robotix-Academy est de nouveau présente en 2019 à un grand nombre d'événements et de foires commerciales ou va y contribuer. Afin de poursuivre ce travail avec succès et de consolider davantage l'Académie, les partenaires ont décidé de demander une extension du projet. Les nouveaux domaines d'intervention sont les soins de santé et la logistique ainsi qu'un groupe cible supplémentaire qui sera traité : Les élèves de 16 à 20 ans qui doivent être préparés à des études techniques dans la Grande Région. Dix-huit mois supplémentaires avec 30% du budget initial sont disponibles à cette fin.
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.

Inhaltsverzeichnis

3rd Robotix-Academy Summer School

Metz, September 04th to 06th

Robotix-Academy Summer School

Vom 04. bis 06. September 2019 fand an der Universität Lothringen die dritte Robotix-Academy Summer School statt.
Wissenschaftliche Mitarbeiter aller Partnerhochschulen der Robotix-Academy kamen zusammen, um in verschiedenen Workshops voneinander zu lernen. Es ist bereits die dritte Veranstaltung dieser Art, nachdem die Summer School 2017 in Saarbrücken am ZeMA gestartet war und 2018 von der Universität Lüttich ausgetragen wurde.
In den Workshops der Summer School werden theoretische Grundlagen anhand von konkreten Aufgaben in die Praxis umgesetzt: Die Universität Metz stellte dieses Jahr eine Roboterapplikation vor, bei der das Schweißen durch Schleifen realisiert wurde. Die wissenschaftlichen Mitarbeiter der Universität Lüttich präsentierten eine ROS-Simulation (Robot Operating System), bei der ein mobiler Roboter in die ROS-Umgebung implementiert und autonom gesteuert wurde.
Das Team der Universität Luxemburg gab eine Einführung in die Programmierung eines ABB Roboters. Hier sollte der Roboter einen vorgegebenen Schriftzug abfahren, den die wissenschaftlichen Mitarbeiter dann einlernen mussten. Das ZeMA stellte in seinem Workshop eine Methode zur Kalibrierung eines Moving-Head vor. Dieser wird auf eine Fläche eingestellt und kann über die x-, y- und z-Koordinaten angefahren werden. Darüber hinaus wurde noch ein Node-RED vorgestellt, ein webbasierter Baukasten, mit dessen Hilfe der Moving-Head programmiert werden kann.
Diese und andere Fragestellungen im Rahmen der Mensch-Roboter-Kollaboration werden behandelt und diskutiert, um daraus

Du 04 au 06 septembre 2019 s'est tenue à l'Université de Lorraine la troisième Université d'été Robotix-Academy.
Des assistants de recherche de toutes les universités partenaires de la Robotix-Academy se sont réunis pour apprendre les uns des autres dans différents ateliers. C'est déjà le troisième événement de ce type, après l'Université d'été 2017 à Sarrebruck au centre ZeMA et l'Université de Liège ayant lieu en 2018.
Dans les ateliers de l'Université d'été, les bases théoriques sont mises en pratique au moyen de tâches concrètes : Cette année, l'Université de Metz a présenté une application robotisée dans laquelle le soudage était réalisé par rectification. Les assistants de recherche de l'Université de Liège ont présenté une simulation ROS (Robot Operating System) dans laquelle un robot mobile a été implémenté dans l'environnement ROS et contrôlé de manière autonome.
L'équipe de l'Université du Luxembourg a donné une introduction à la programmation d'un robot ABB. Ici, le robot devait exécuter un script prédéfini, que les assistants de recherche devaient ensuite apprendre. Dans son atelier, le centre ZeMA a présenté une méthode d'étalonnage d'une tête mobile. Il est réglé sur une surface et peut être approché par les coordonnées x, y et z. De plus, un Node-RED a été présenté, un kit de construction basé sur le Web avec lequel la tête mobile peut être programmée.
Ces questions et d'autres dans le contexte de la collaboration homme-robot seront traitées et discutées afin d'apporter une valeur ajoutée aux travaux scientifiques futurs. L'objectif de l'Université d'été est de développer un contenu qui intéressera éga-

einen Mehrwert für die weitere wissenschaftliche Arbeit abzuleiten. Ziel der Summer School ist es, Inhalte zu erarbeiten, die künftig auch für die Unternehmen der Großregion von Interesse sind.
Ergänzt wurde das Workshop-Programm durch ein Team-Building-Event: Gemeinsam besuchte die Gruppe abends im Rahmen der „Constellations de Metz“ die Aufführung „Morphosis“ an der Kathedrale Saint-Étienne. Bei diesem Video-Mapping-Event wurden zahlreiche historische Ereignisse der Stadt Metz visualisiert und auf die Kathedrale projiziert, um sie als zeitgenössische Einrichtung im konstanten Wandel darzustellen.

lement les entreprises de la Grande Région dans l'avenir.
Le programme de l'atelier a été complété par un événement de team-building : le soir, le groupe a assisté au spectacle «Morphosis» à la cathédrale Saint-Étienne dans le cadre des «Constellations de Metz». Au cours de ce spectacle de projection vidéo, de nombreux événements historiques de la ville de Metz ont été visualisés et projetés sur la cathédrale afin de la présenter comme une institution contemporaine en constante évolution.

3rd Robotix-Academy Summer School at University of Lorraine, September 04th to 06th 2019

Wednesday	Thursday	Friday
Arrival	**Modelisation and programming of a mobile robot on ROS and Gazebo** *part 1* *Liège Université* (8:30-10h00)	**Modelling and calibration of moving head** *Zema* (8:30-10h00)
ENIM visit *Lorraine Université* (11:00-12h30)	**Modelisation and programming of a mobile robot on ROS and Gazebo** *part 2* *Liège Université* (10:15-12h30)	**Modelling and calibration of moving head** *Zema* (10:15-12h30)
Lunch	Lunch	Lunch
Presentation :Robot trajectory in FSW *Lorraine Université* (13:30-14h30)	**Modelisation and programming of a mobile robot on ROS and Gazebo** *part 3* *Liège Université* (13:30-15h15)	**LCFC robot's demonstration** *Lorraine Université* *(13:30-15h15)*
ABB programming *Luxembourg Université* (14:45-17h30)	**Modelisation and programming of a mobile robot on ROS and Gazebo** *part 4* *Liège Université* *(15:30-17h30)*	**Yumi : modeling and programming** *Lorraine Université* *(15:30-17h30)*
	Dinner	

Path correction of an industrial robot used for friction stir welding (UL)

Das Reibrührschweißen (Friction Stir Welding, FSW) ist ein neues Verfahren zum Schweißen von Metallteilen mit verschiedenen Anwendungen in der Industrie. Aufgrund der Steifigkeit der Serienroboter führt eine Abweichung der Schweißwerkzeugbahn in Position und Orientierung zu Fehlern in der Schweißnaht. Es wurden zwei Methoden zur Korrektur dieser Abweichungen vorgestellt. Die erste Methode basiert auf der Abschätzung von Positions- und Orientierungsabweichungen im kartesischen Raum von Roboter- und elastostatischen Verformungsmodellen von Körpern und Getrieben. Die zweite Methode verwendet eine Echtzeit-Positionsregelung mit einem 2D-Laserprofilsensor in der Rückkopplungsschleife.

Kontakt:
Universität Lothringen
Komlan KOLEGAIN
E-Mail:k.kolegain@outlook.fr

Le procédé de soudage par friction malaxage ou Friction Stir Welding (FSW), est un procédé récent utilisé pour le soudage de pièces métalliques avec différentes applications dans l'industrie. A cause de la rigidité des robots séries une déviation de trajectoire de l'outil de soudage en position et en orientation introduit des défauts dans le cordon de soudure. Deux méthodes de correction de ces déviations ont été prénsentées. La première méthode est basée sur l'estimation des déviations en position et en orientation dans l'espace cartésien à partir des modèles du robot et de déformation élastostatique des corps et transmissions. La deuxième méthode de correction des déviations utilise un asservissement de position en temps réel avec un capteur de profil laser 2D dans la boucle de retour.

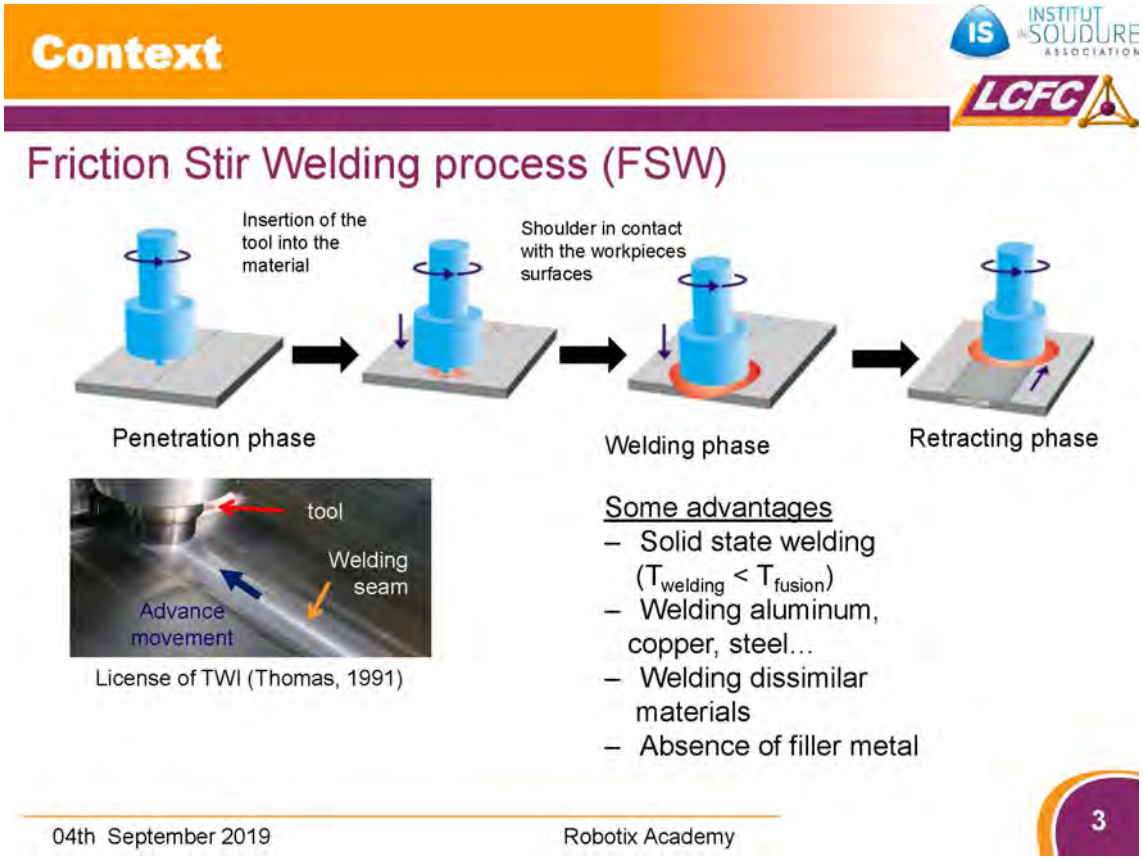
Contact:
Université de Lorraine
Komlan KOLEGAIN
e-mail: k.kolegain@outlook.fr

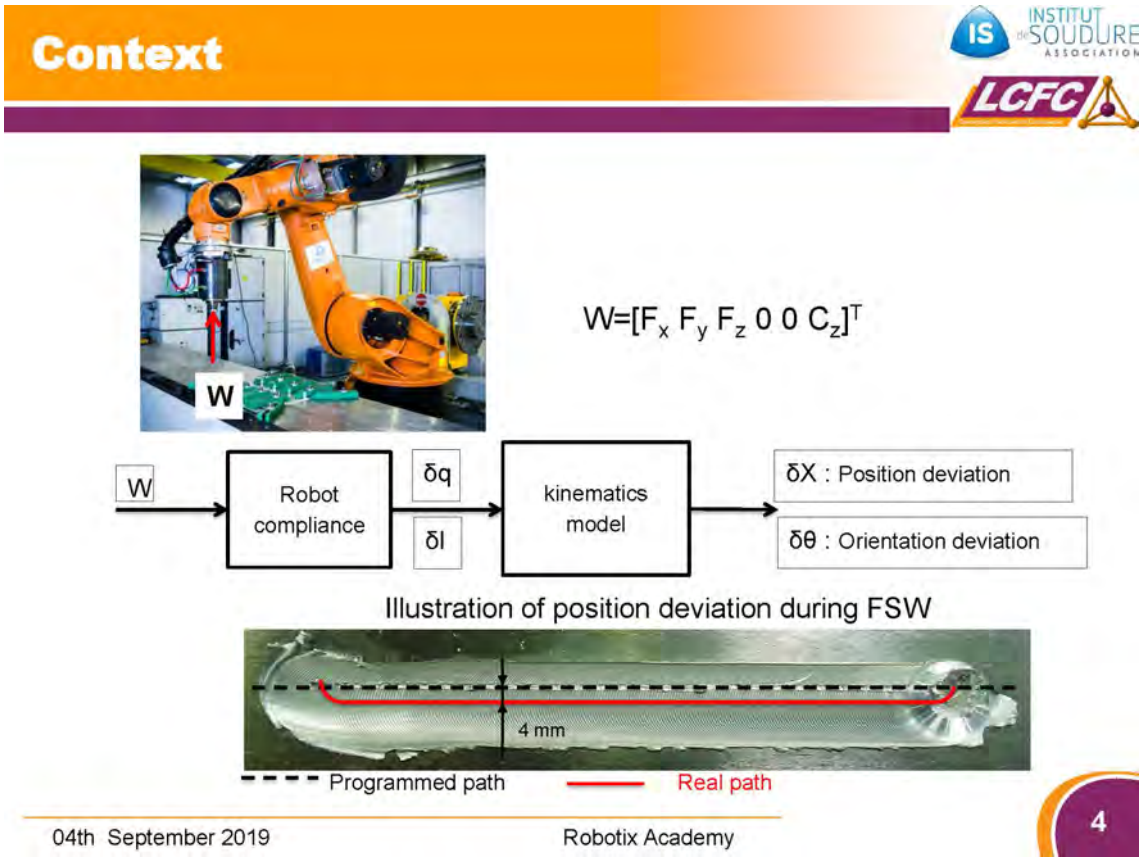
Path correction of an industrial robot used for friction stir welding

Komlan KOLEGAIN

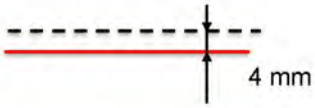
Table of Contents

- Context and problematics
- Methods, results and discussions
- Conclusions


Context
INSTITUT DE SOUDURE ASSOCIATION
LCFC
Friction Stir Welding process (FSW)
Insertion of the tool into the material
Shoulder in contact with the workpieces surfaces
Penetration phase
Welding phase
Retracting phase
tool
Welding seam
Advance movement
License of TWI (Thomas, 1991)
Some advantages
– Solid state welding ($T_{welding} < T_{fusion}$)
– Welding aluminum, copper, steel…
– Welding dissimilar materials
– Absence of filler metal
04th September 2019
Robotix Academy
3


Context
INSTITUT DE SOUDURE ASSOCIATION
LCFC
W
$W=[F_x\ F_y\ F_z\ 0\ 0\ C_z]^T$
W
Robot compliance
δq
δl
kinematics model
δX : Position deviation
δθ : Orientation deviation
Illustration of position deviation during FSW
4 mm
Programmed path
Real path
04th September 2019
Robotix Academy
4

Problematic



- - - - Desired path

—— Real path

4 mm (determined by deflection model or measured by a sensor)

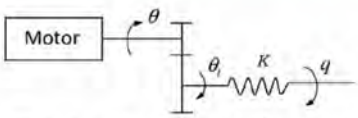
Problematic

- Offline compensation method based on deflection model
- Online compensation method based on laser sensor

Offline compensation method based on deflection model

Offline compensation model

➢ Deflection model with forces



$$\Delta q = K^{-1}\Gamma \quad (1)$$

For the six axes, the rigidity matrix K is diagonal

Torsional spring deformation model for one axis

$$\Gamma = M(q)\ddot{q} + H(\dot{q}, q) + F_f(\dot{q}) + J^T(q)\,W$$

$$\Delta q = K^{-1}(J^T(q)W + G(q) + F_f(\dot{q})) \quad (2)$$

Hypothesis: In FSW, low and constant cartesian velocities, low angular velocities, Acceleration null

$$\begin{bmatrix}\Delta P \\ \Delta\theta\end{bmatrix} = L_a J(q) K^{-1}(J^T(q)W + G(q) + F_f(\dot{q})) \quad (3)$$

$\Delta P = [\Delta X\ \Delta Y\ \Delta Z]^T$
$\Delta\theta = [\Delta A\ \Delta B\ \Delta C]^T$
L_a : Relation between roll-pitch-yaw angles variation and angular velocity in the cartesian space

Path Planning Methodology

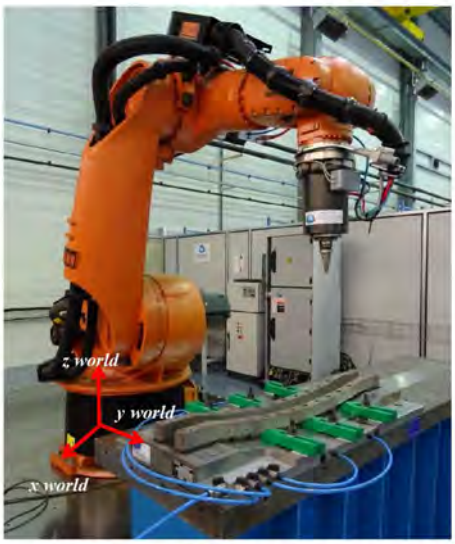
CAD model → 1 Extraction of positions and orientations X_{CAD} → 2 Estimation of deflections ΔX → 3 Adding ΔX to X_{CAD} data → 4 P_{BP} data fitting by Bsplines curves → Robot program

$$X_{CAD}=\begin{bmatrix} x \\ y \\ z \\ A \\ B \\ C \end{bmatrix} \qquad \Delta X=\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta A \\ \Delta B \\ \Delta C \end{bmatrix} \qquad X_{BP}=\begin{bmatrix} x+\Delta x \\ y+\Delta y \\ z+\Delta z \\ A+\Delta A \\ B+\Delta B \\ C+\Delta C \end{bmatrix} \qquad X_B(u)=\begin{bmatrix} x_B(u) \\ y_B(u) \\ z_B(u) \\ A_B(u) \\ B_B(u) \\ C_B(u) \end{bmatrix}$$

04th September 2019 Robotix Academy 9

Methodology implementation

➢ Determination of processing parameter



Kuka KR500-2MT (Institut de Soudure)

Test workpiece

By courtesy of Stelia Aerospace

Experimental parameters:

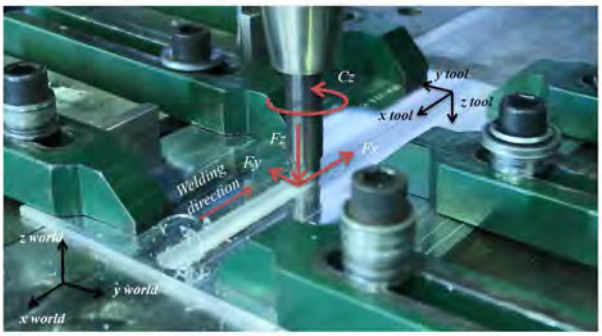
Material: AW 2024-T3 2 mm
Advance speed: 300 mm/min
Rotation speed: 800 rpm
Axial force: 8000 N
Tilt angle: 2,5°

Inside Process window

04th September 2019 Robotix Academy 10

Methodology implementation

➢ Determination of processing parameter



Measured Forces:

Axial force: F_z **= 8000 N**
Longitudinal force : F_x **= 420 N**
Transversal force : F_y **= -175 N**
Rotational torque : C_z **= 7 Nm**

Methodology implementation

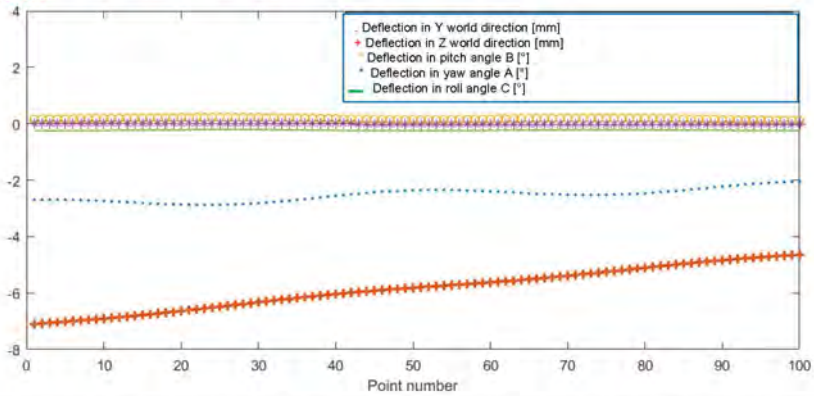
➢ Application

CAD model ⇨ Extraction in base coordinate system of positions and orientations of 100 points

Methodology implementation

➢ Deflection calculation

$$\begin{bmatrix}\Delta P\\ \Delta\theta\end{bmatrix} = L_a J(q) K^{-1}(J^T(q)W + G(q) + F_f(\dot{q}))$$

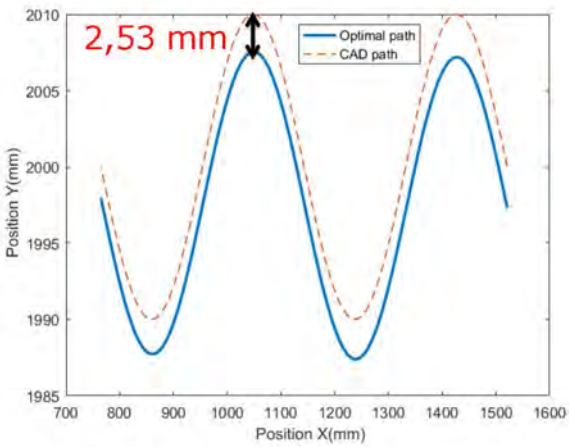


Deflections	Y direction	Z direction	Yaw angle A	Pitch angle B	Roll angle C
Rms value	2,53 mm	5,88 mm	0,013°	0,17°	0,23°

04th September 2019 Robotix Academy 13

Methodology implementation

➢ FSW path generation in position: 4th degree Bspline curve



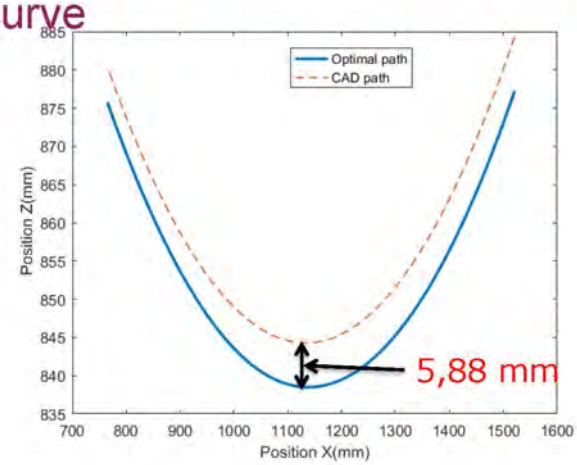
Position Y in robot world coordinate

- Deflection rms value: 2,53 mm
- Smooth path
- Desired path≠ programmed path

04th September 2019 Robotix Academy 14

Methodology implementation

➢ FSW path generation in position: 4th degree Bspline curve



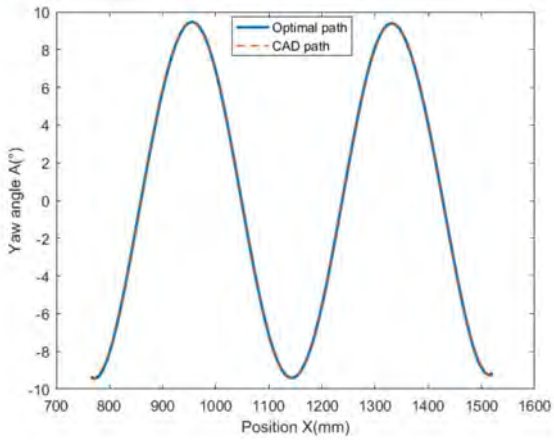
Position Z in robot world coordinate

- Deflection rms value: 5,88 mm
- Smooth path
- Desired path≠ programmed path

04th September 2019 Robotix Academy 15

Methodology implementation

➢ FSW path generation in orientation: 4th degree Bspline curve



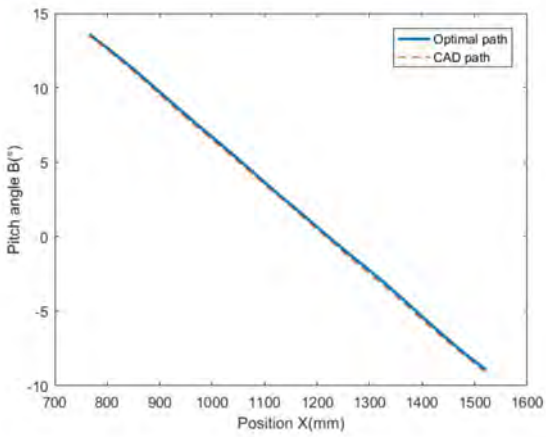
Yaw angle A

- Deflection rms value: 0,0013°
- Smooth path
- Desired path ≈ programmed path

04th September 2019 Robotix Academy 16

Methodology implementation

➢ Path generation in orientation: 4th degree Bspline curve



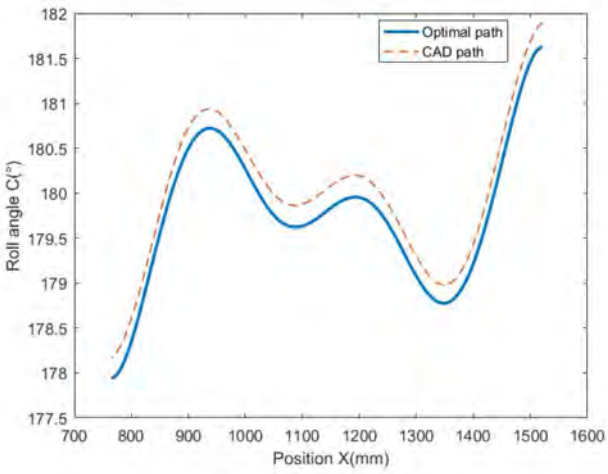
Pitch angle B

- Deflection rms value: 0,17°
- Smooth path
- Desired path ≈ programmed path

04th September 2019 Robotix Academy 17

Methodology implementation

➢ FSW path generation in orientation: 4th degree Bspline curve



Roll angle C

- Deflection rms value: 0,23°
- Smooth path
- Small angle variation around 180°
- Desired path ≈ programmed path

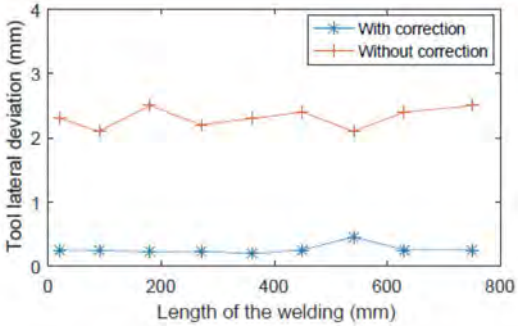
04th September 2019 Robotix Academy 18

Experiments

04th September 2019 Robotix Academy 19

Methodology validation

➢ Results



Tool deviation measured

Experiments	With compensation	Without compensation
Rms lateral deviation	0,27 mm	2,32 mm

Gain: 88 %

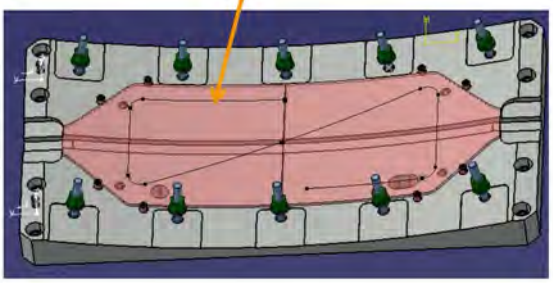
04th September 2019 Robotix Academy 20

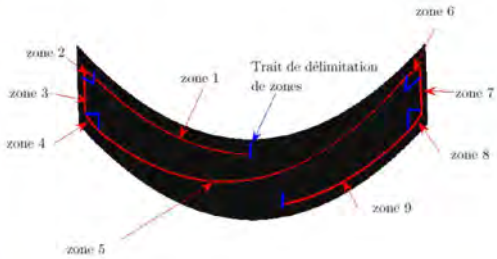
Methodology implementation

➢ Extraction of positions and orientations

CAD model → Extraction in base coordinate system of positions and orientations each 2 mm

Desired weld trajectory





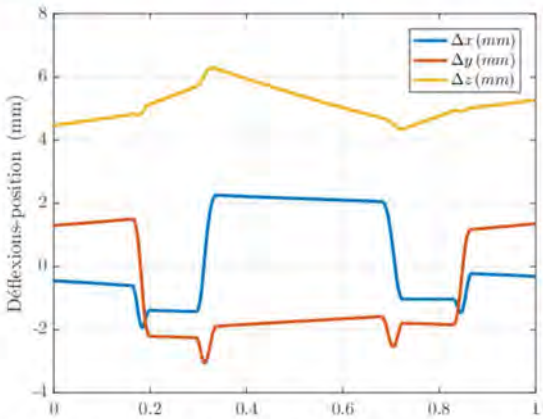
Curvilinear path defined in the test workpiece

04th September 2019 Robotix Academy 21

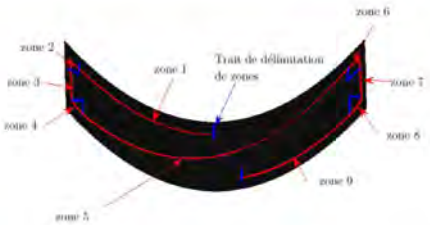
Methodology implementation

➢ Deflections calculation

$$\begin{bmatrix}\Delta P\\ \Delta\theta\end{bmatrix} = L_a J(q) K^{-1}(J^T(q)W + G(q) + F_f(\dot{q}))$$



Position deflections calculated in the tool frame

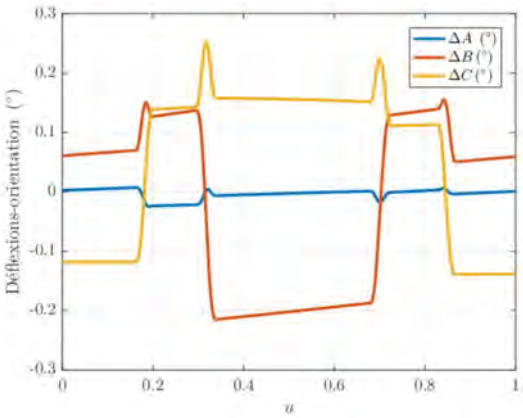


	Lateral deviation Δy (mm)	Advance deviation Δx (mm)
Zone 1	1.3	-0.6
Zone 3	-2.3	-1.6
Zone 5	-1.5	2.2
Zone 7	-1.9	-1.3
Zone 9	1.3	-0.4

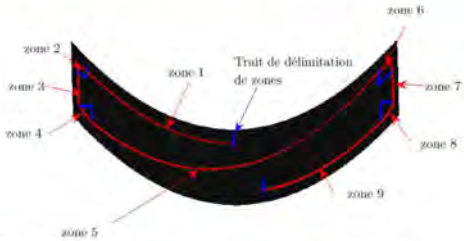
04th September 2019 Robotix Academy 22

Methodology implementation

➢ Deflections calculation



Orientation angles deflections



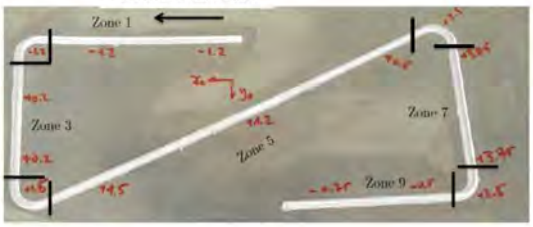
Deflections	Intervalle of variation
ΔA	[-0.01° 0.01°]
ΔB	[-0.25° 0.15°]
ΔC	[-0.12° 0.25°]

Negligible due to robustness DSO

Methodology validation

➢ Results – Deviation measurement

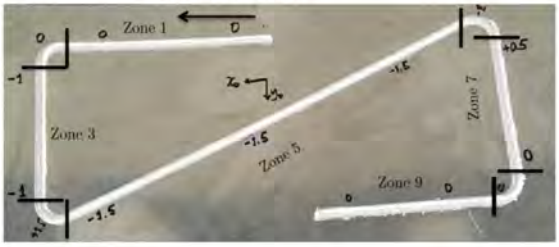
Advance direction



Without correction

Deviations: [-1.2 mm +3.75 mm]

Advance direction



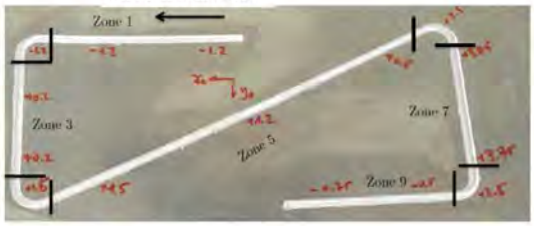
With correction

Deviations: [-1.5 mm +1.5 mm]

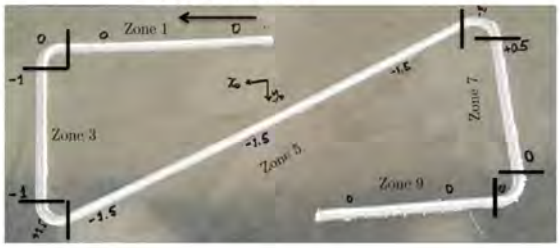
Methodology validation

➢ Results – Deviation measurement

Advance direction



Advance direction

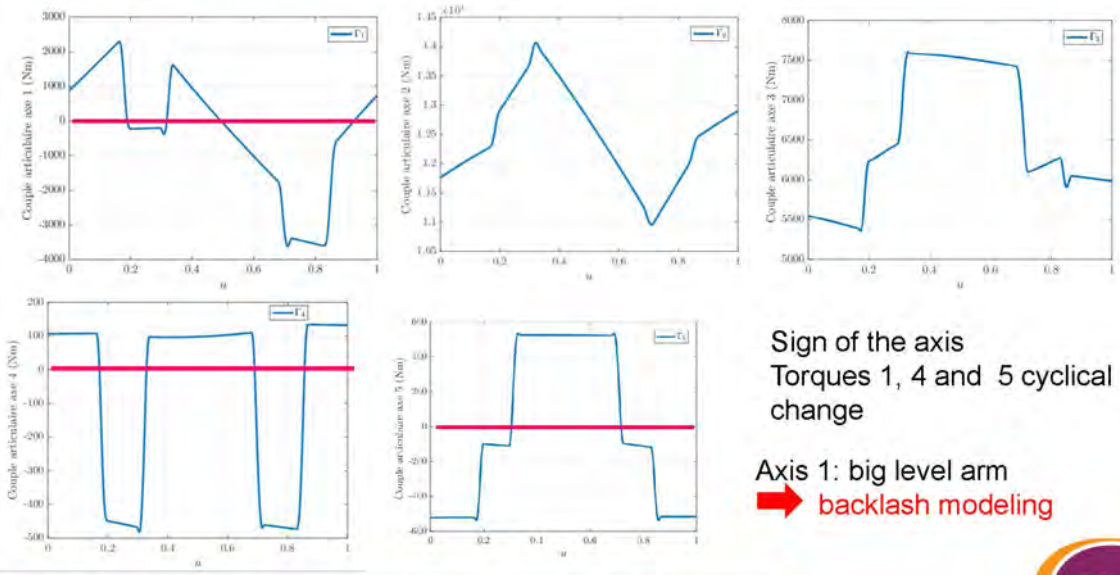


Zone 1	👍	Zone 6	👎
Zone 2	👍	Zone 7	👍
Zone 3	👎	Zone 8	👍
Zone 4	👎	Zone 9	👍
Zone 5	👎		

04th September 2019 Robotix Academy 25

Methodology validation

➢ Torque evolution



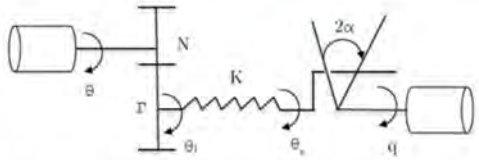
Sign of the axis
Torques 1, 4 and 5 cyclical change

Axis 1: big level arm
➡ backlash modeling

04th September 2019 Robotix Academy 26

Methodology validation

➢ Deflection model with backlash



Axis 1:

$$\Gamma_1 = \begin{cases} K_1(\Delta q_1 - \alpha_1) & \Gamma_1 > 0 \\ K_1(\Delta q_1 + \alpha_1) & \Gamma_1 < 0 \end{cases}$$

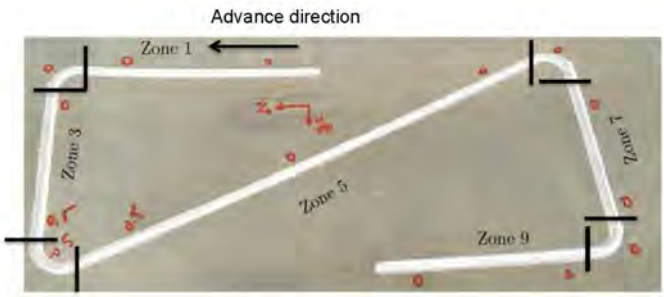
Γ_1: Torque axis 1
Δq_1: angular deflection axis 1
$2\alpha_1$: backlash axis 1

Identification $2\alpha_1$= 5.01 arcmin

04th September 2019 Robotix Academy 27

Methodology validation

➢ Results – Deviation measurement



With backlash correction on axis 1

Residual deviation:
[-0.5 mm 0 mm]

Zone 1	Zone 2	Zone 3	Zone 4	Zone 5	Zone 6	Zone 7	Zone 8	Zone 9
Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok

04th September 2019 Robotix Academy 28

Conclusions

- Conclusion
  - ✓ offline methodology developped to plan 3D path based on B-splines
  - ✓ Backlash of axis 1 identified locally
  - ✓ Good compensation of tool path deviations
  - ✓ Residual deviation contained in the process tolerance [-0.5 mm 0.5 mm]
- Future works
  - ✓ Global backlash identification for all axes

04th September 2019 Robotix Academy 29

2D profile laser sensor servoing

04th September 2019 Robotix Academy 30

Laser sensor servoing

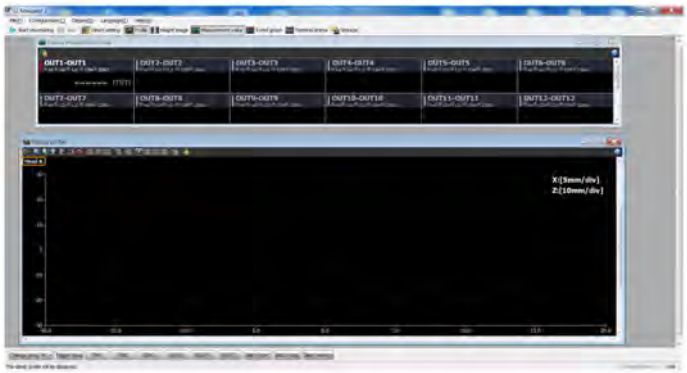
➢ 2D laser profiler sensor Keyence LJ-V7080

Laser Sensor	Keyence LJ-V7080
Reference distance	80 mm
Height measure range	80+-23 mm
Width measure range	32+-7 mm
Interval of emission profile data	0,05 mm

04th September 2019 Robotix Academy 31

Laser sensor servoing

➢ Programming Keyence sensor

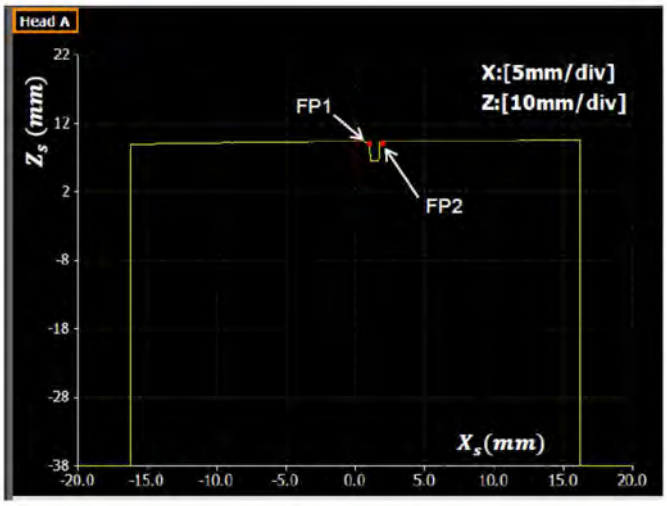


Measurement: Position and height of the extremum points, of center of circles, profile processing (filters), angles

04th September 2019 Robotix Academy 32

Laser sensor servoing

➢ Programming the laser sensor



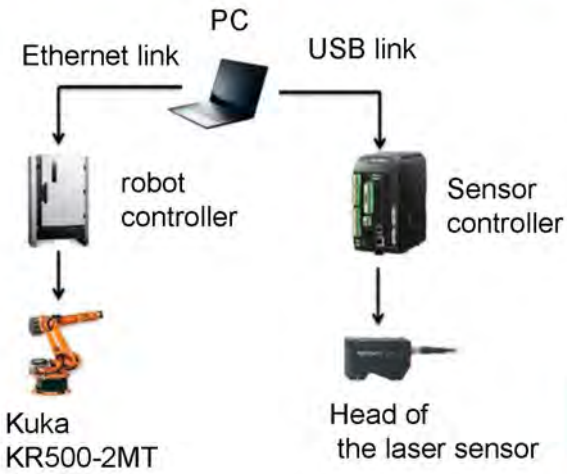
Measures:

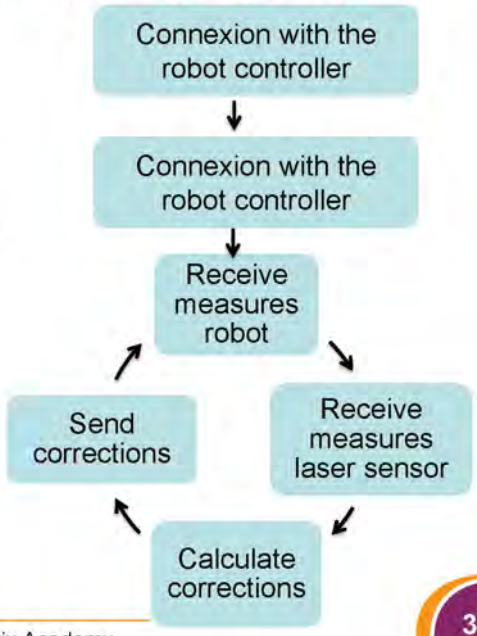
- ❑ Position x of point FP1
- ❑ Position x of point FP2
- ❑ Height z of point FP1
- ❑ Height z du point FP2
- ❑ Angle of the profil with axis x_s

04th September 2019 Robotix Academy 33

Laser sensor servoing

➢ System robot-PC-laser sensor (C++ program)

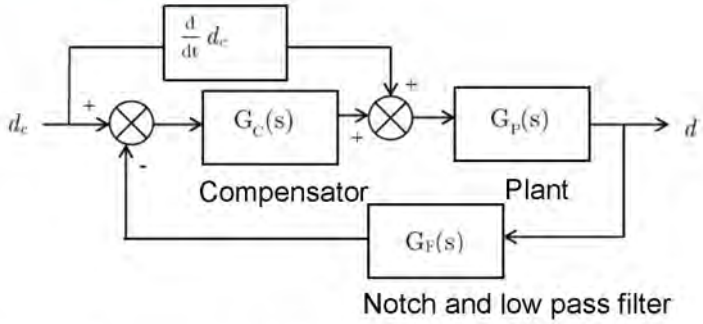




04th September 2019 Robotix Academy 34

Laser sensor servoing

➢ Feedback loop



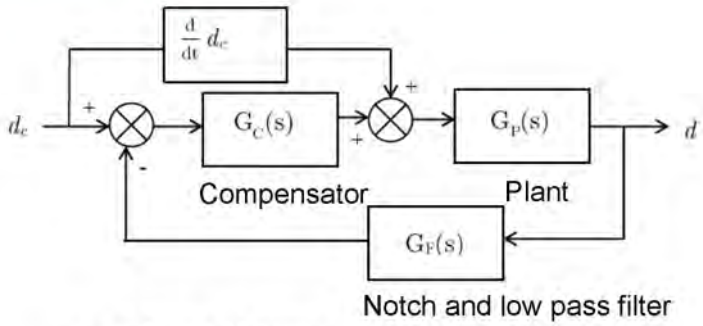
$G_p(s)=\frac{1/T_e}{s}\frac{1}{1+\frac{2\varepsilon}{\omega_n}s+\frac{1}{\omega_n^2}s^2}$ with ε=0.7 and ω_n=10 rad/s

$G_F(s)=\frac{1+\frac{s^2}{\omega_r^2}}{1+\frac{2\zeta}{\omega_0}s+\frac{1}{\omega_0^2}s^2}$ with ω_0=low pass frequency and ω_r=rotation tool frequency

04th September 2019 Robotix Academy 35

Laser sensor servoing

➢ Feedback loop

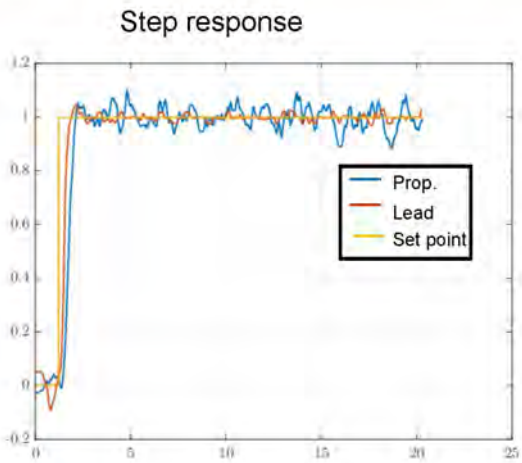


Two types of compensators

- Proportional compensator
  C(s)=0,025
- Lead compensator
  C(s)= 0,036 $\frac{1+0{,}0702s}{1+0{,}0032s}$

04th September 2019 Robotix Academy 36

Laser sensor servoing

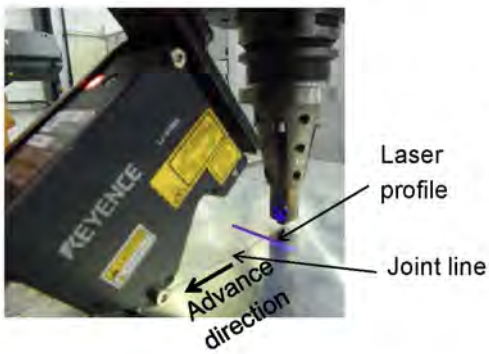


	Proportionnal	Lead
Response time	0.9 s	0.5 s
Phase margin	65.6°	65.4°
Gain margin	12.6 dB	12.4 dB
Overshoot	2.2%	2%

04th September 2019 Robotix Academy 37

Laser sensor servoing

➢ Test 1: Line trajectory

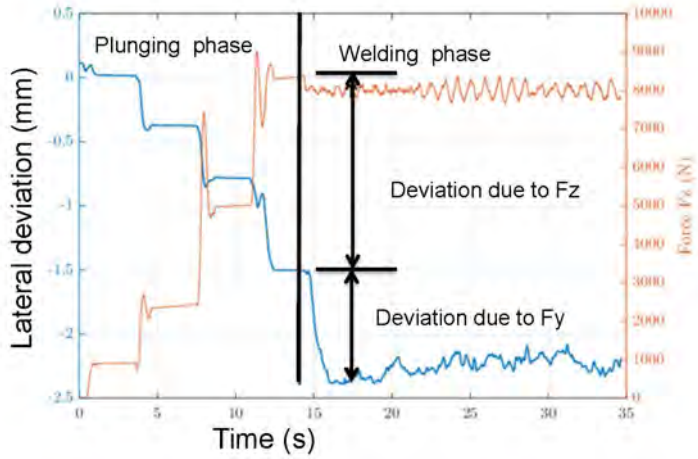


- ❑ Line trajectory of 100 mm
- ❑ Two pieces in butt joint configuration With a gap of 0.2 mm
- ❑ Laser sensor is placed ahead the trajectory

04th September 2019 Robotix Academy 38

Laser sensor servoing

➢ Test 1: without correction



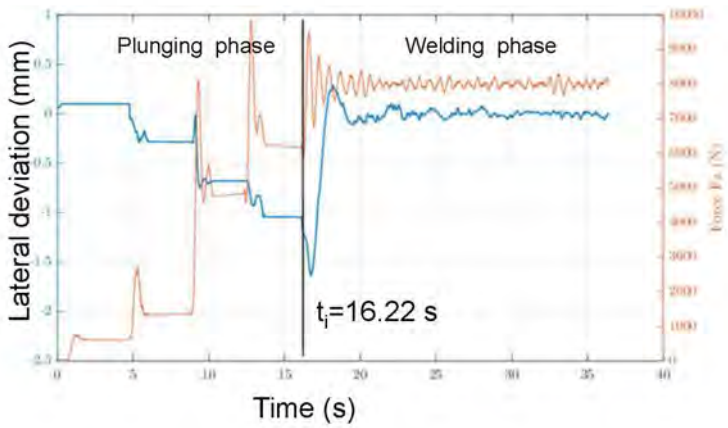
Transitional regime : 1,58 s
Error : -2,25mm

04th September 2019 Robotix Academy 39

Laser sensor servoing

➢ Test 1 with correction

✓ Proportionnal compensator

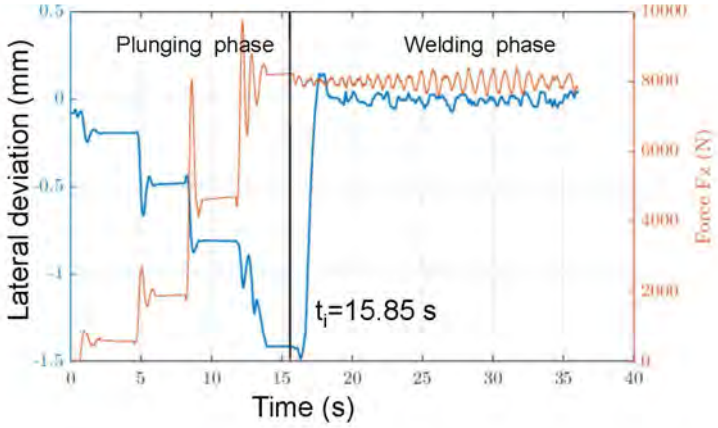


Response time: 2.72 s
Residual error: 0.05 mm
Overshoot: 0.3 mm
Stable

04th September 2019 Robotix Academy 40

Laser sensor servoing

- Test 1 with correction
  - ✓ Lead compensator



Response time: 2,45 s
Overshoot: 0.3 mm
Residual error: 0,05mm
Stable

04th September 2019 Robotix Academy 41

Laser sensor servoing

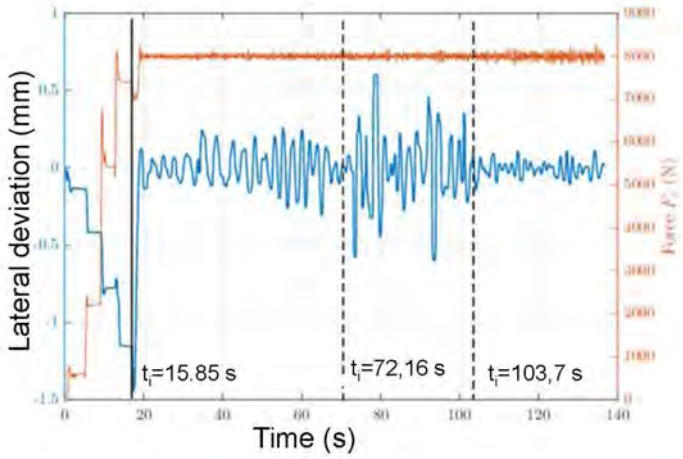
- Test 2: curvilinear trajectory

04th September 2019 Robotix Academy 42

Laser sensor servoing

➢ Test 2: curvilinear trajectory

✓ Proportionnel compensator

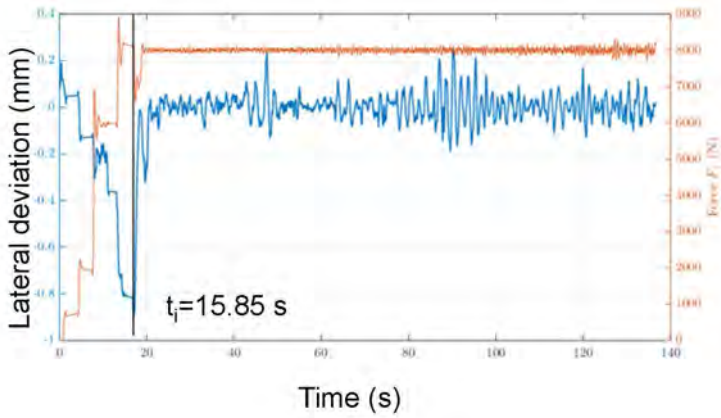


Response time: 3,7 s
Overshoot: 0.1 mm
Residual error: 0.05mm
Std deviation: 0.18 mm
Interval=[-0.6 0.6 mm]

Laser sensor servoing

➢ Test 2: curvilinear trajectory with correction

✓ Lead compensator



Response time: 3.5 s
Residual error: 0.05mm
Std deviation: 0.08 mm
Interval=[-0.2; 0.2 mm]

Laser sensor servoing

- Conclusions
  - ✓ Measures instability due to irregular borders of the workpieces
  - ✓ Good time response of the feedback loop
  - ✓ Correction precise and stable on small trajectory length
  - ✓ Correction precise and globally stable
  - ✓ Oscillation remarked in certain part of the weld seam on long trajectory
- Futurs works
  - ✓ Robust algorithm to stablize the measures
  - ✓ Process transfert function modelling
  - ✓ Advanced compensators (adaptative command)

04th September 2019 Robotix Academy 45

THANK YOU FOR YOUR ATTENTION

ABB Rapid Programming Language (Uni Lu)

In diesem Workshop wurde die RAPID-Programmierung vorgestellt. Wenn ein Roboter eine Aufgabe ausführen soll, muss ein Programm erstellt werden. RAPID ist eine Programmiersprache für die Kommunikation mit einem ABB-Roboter. Eine Anwendung bestand darin, mit einem Stift, der am Tool Center Point des Roboters angebracht war, einige Wörter zu schreiben. Eine Reihe von Punkten, die die Buchstaben beschreiben, wurde auf dem Objektrahmen ausgewählt, und jeder Student betätigte den Roboter mit Hilfe der verfügbaren Laufbewegungen, um einen bestimmten Punkt anzusteuern. Zum Schluss wurde das erstellte Programm überprüft, und der Roboter führte die Aufgabe erfolgreich aus.

Kontakt:
Universität Luxemburg
Natago Mbodj
E-Mail: natago.mbodj@uni.lu

Au cours de cette session, la programmation RAPID a été présentée. Si l'on veut qu'un robot effectue une tâche, il faut créer un programme. RAPID est un langage de programmation pour la communication avec le robot ABB. Une application consistait à écrire des mots à l'aide d'un stylo fixé au Tool Center Point du robot. Un ensemble de points décrivant les lettres a été sélectionné sur le cadre-objet et chaque élève a manipulé le robot en utilisant les mouvements de course disponibles pour guider un point donné. Une fois terminé, le programme créé a été vérifié et le robot a accompli la tâche avec succès.

Contact:
Université de Luxembourg
Natago Mbodj
e-Mail: natago.mbodj@uni.lu

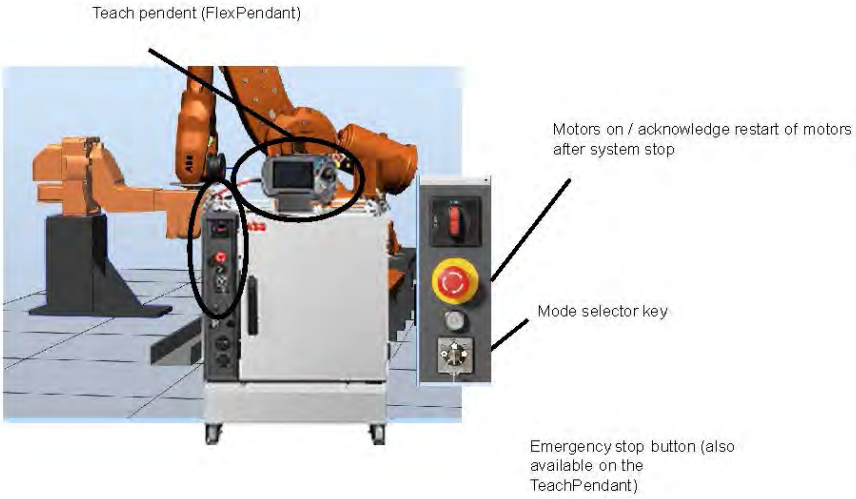
ABB Rapid Programming Language

Robot Control

The robot control system can be in three distinct modes

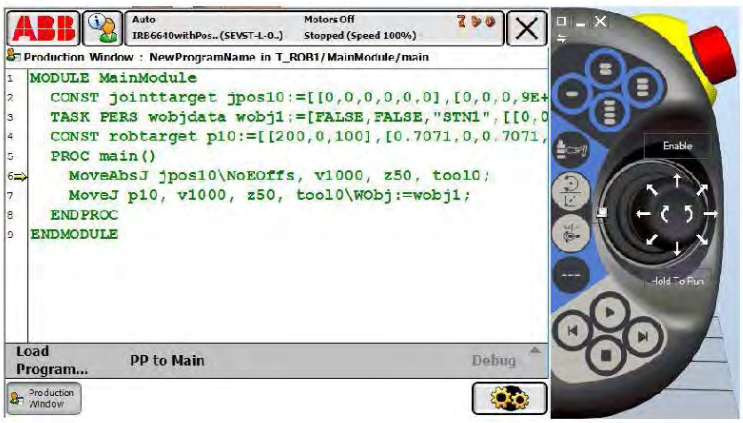
- Manual, the speed is reduced to a maximum 250 mm/s
- Manual 100%, full performance is reached but the user has to keep pushed an extra button on the flex pendent
- Auto, the robot moves at full speed completely "autonomous"

Robot hardware

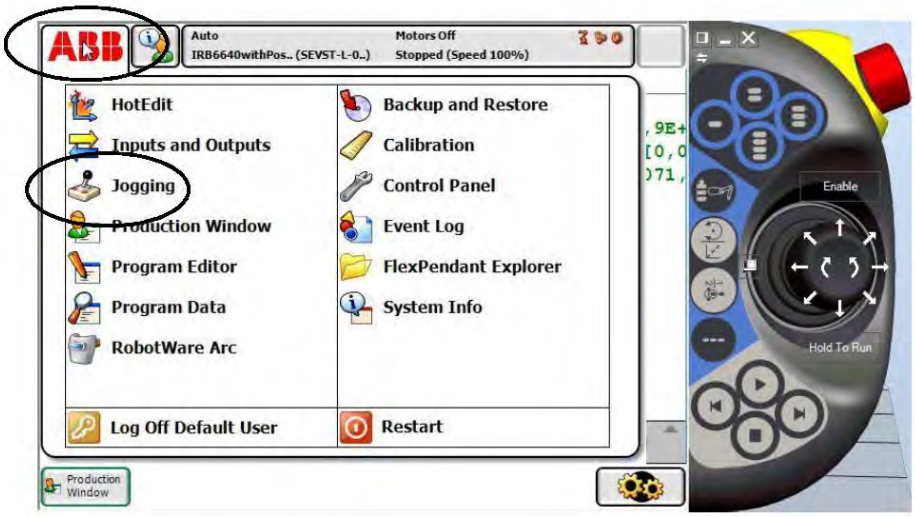


Robot hardware

Teach pendant



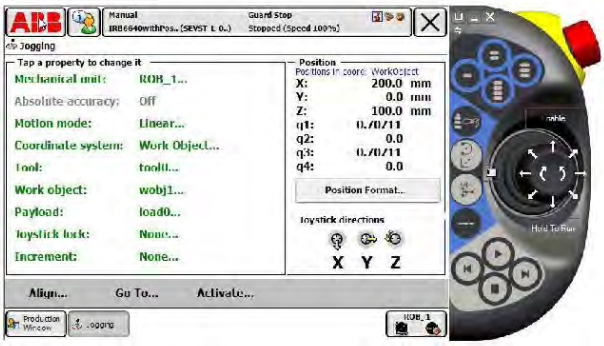
Robot hardware



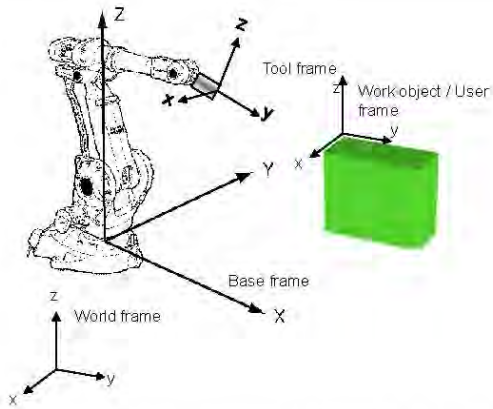
Moving the robot with the joystick (jogging)

Choose jog-mode

- *Motion mode* decides how to use the joystick
- *Coordinate system* affects the jogging directions
- *Tool* affects the position and also the jogging directions if Coordinate system is tool
- *Work object* affects the position and also the jogging directions if Coord is wobj



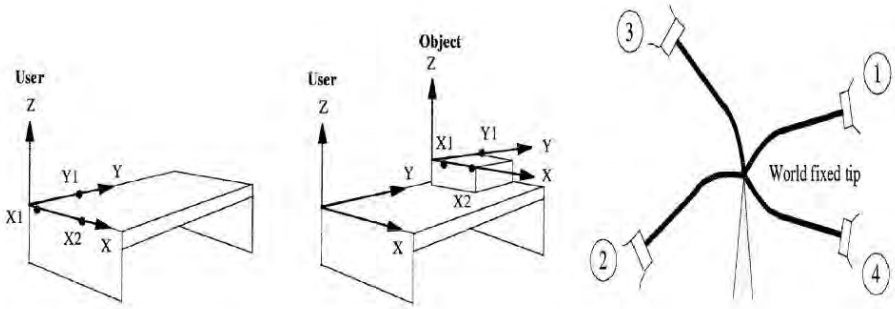
Coordinate systems



To help in the programming, a number of coordinate systems can e assigned (Calibration)

Define a Coordinate system

- The work object coordinate frame can be defined using a three point method
- Approach a world fixed tip from (at least) four directions



- MoveL is an instruction that moves the robot linearly (in a straight line) from its current position to the specified position.
- MoveJ is used to move the robot quickly from one point to another when that movement does not have to be in a straight line.
- MoveC is used to move the robot circularly in an arc.

```
Example!!!
MoveL p10, v1000, fine, tool0;
```

- p10 specifies the position that the robot shall move to.
- v1000 specifies that the speed of the robot shall be 1000 mm/s.
- fine specifies that the robot shall go exactly to the specified position and not cut any corners on its way to the next position.
- tool0 specifies that it is the mounting flange at the tip of the robot that should move to the specified position.

```
MoveC p10, p20, v1000,
fine, tool0; MoveJ
p10, v1000, fine,
tool0; MoveAbsJ j10,
v50, z50, tool0;
```

Exercise:

- Based on the information given above, create a program that draw on a sheet of paper the word ABB.
- Simulate that the robot uses a pen by moving to a specified fixed position before and after the drawing.

Modelisation and programing of a mobile robot on ROS and Gazebo (ULg)

Die Universität Lüttich hat einen ROS-Workshop angeboten. ROS (Robotic Operating System) ist eine Computerumgebung zur Roboterprogrammierung und Sensorintegration. Diese Open-Source-Umgebung ist in Forschung und Industrie weit verbreitet. Zunächst ging es darum, die grundlegenden Elemente von ROS sowie dessen Kommunikationssysteme zu verstehen. Arbeitsbereiche, Knoten, Packages und Themen wurden angesprochen. Die Teilnehmer konnten sich mit ROS vertraut machen bzw. ihre Kenntnisse auffrischen. Dann wurde der Roboter turtlebot3 modelliert und auf einer virtuellen Karte mit Hindernissen simuliert (Software Gazebo). Schließlich sollte turtlebot3 programmiert und autonom gemacht werden. Dazu wurden Packages für Kartierung, Lokalisierung und Bahnplanung verwendet.

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

L'Université de Liège a proposé un atelier ROS. ROS (Robotic Operating System) est un environnement informatique pour la programmation de robots et l'intégration de capteurs. Cet environnement open source est largement utilisé dans la recherche et l'industrie. La première partie consistait à comprendre les éléments de base du ROS et de ses systèmes de communication. Les «workspaces», les «nodes», les «packages» et les «topics» y sont abordés. Les participants pourraient se familiariser avec les ROS ou rafraîchir leurs connaissances. Ensuite, le robot turtlebot3 a été modélisé et simulé sur une carte virtuelle avec des obstacles (logiciel Gazebo). Enfin, turtlebot3 devait être programmé et rendu autonome. Pour ce faire, on a utilisé des «packages» de cartographie, de localisation et de planification de trajectoire.

Contact:
Univérsité de Liège
Robin Pellois
e-mail:robin.pellois@ulg.ac.be

Robotix-Academy Summer School 2019

September 4th to 6th 2019

Modelisation and programing of a mobile robot on ROS and Gazebo

WorkShop presented by Robin Pellois

1

Table of contents

Introduction

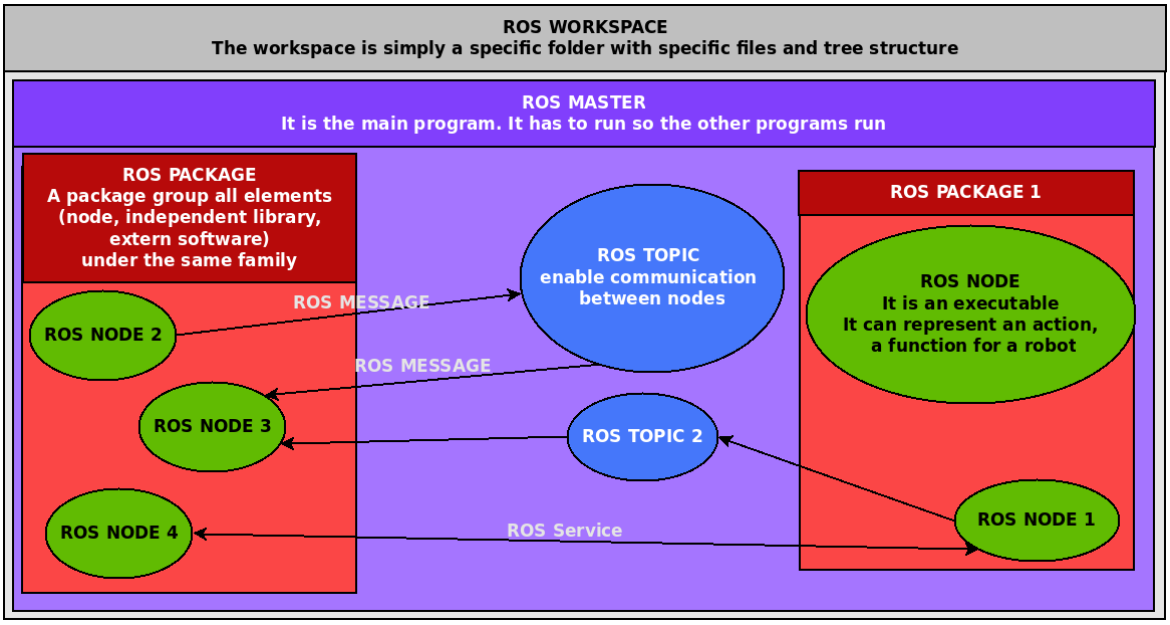
What is ROS ?:

ROS stand for Robotic Operating System. It's "a library with structure that allows you to build your own programs, functions...".
ROS is describe in the following article:
Quigley, M., B., G., K., C., J., F., T., F., J., L., … Ng, A. (2009). ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, (3.2), 5.

What are its features and assets

- Linux based system.
- Open source system under BSD licenses (reuse in commercial and closed source products).
- Compatible with a lot of robots and miscellaneous devices (ABB, Fanuc, UR, Rethink Robotics, Husky, PR, Shadow, Nao, Raspberry Pi, IMU, Cameras, scanners…).
- Large community (1500 participants, 3300 contributors on ROS-wiki, 30 wiki a day …).
- Integration with other library (Gazebo, Open CV, MoveIt …).
- Different version called Distro (Indigo (2014), Jade (2015), **Kinetic (2016)**, Lunar (2017), Melodic (2018)).
- Source code in C++ and Python.
- ROS-industrial make it compatible for any industrial robot.

The architecture:



3

A – Getting started with ROS

The objective of this first tutorial is to understand the basics elements of ROS and its communication layer. After this tutorial you will be able to create your own package, node and topic. Not all elements of ROS are presented in this tutorial. If you want to go further you will find more detailed tutorials on http://wiki.ros.org/.

On the virtual box ROS kinetic has already been installed as well as some useful packages or library. Open a terminal and follow. The password is ros.

1 – Workspace

Before anything else, a ROS workspace is necessary to host anything from ROS. A workspace a basically a folder with a special architecture. So first create a folder called “ros_ws” which include another folder call “src”. Go into the “ros_ws” folder and make it a ROS workspace with the “catkin_make” command

```
mkdir -p ~/ros_ws/src
cd ~/ros_ws
catkin_make
```

2 – Package

A package is a substructure in a workspace containing nodes, launch files, customs messages for topics, other useful files. For instance we could create a package containing everything require to make a scanner works. A package is, once again, a folder with a special architecture. Every packages of a workspace as to be in the “src” folder of a workspace. Once in this folder create a package with the command “catkin_create_pkg”. This command takes as argument the name of the package and the dependencies. Here we will need the standard messages dependencies (“std_msgs”) as well as the “rospy” dependency in order to use Python.

```
cd ~/ros_ws/src
catkin_create_pkg a_pkg std_msgs rospy
```

3 – Sourcing

Every time you open a new terminal you would need to source the workspace. In order to avoid to do it every time, we will modify the “bashrc” file. In a terminal type the following line:

```
echo 'source ~/ros_ws/devel/setup.bash' >> ~/.bashrc
```

4 – Node

i – Create a node

A node is the essential element in ROS. This is where you will write your code. We will create a python node into the “a_pkg” you just created. Go to the “src” folder in your “a_pkg” and create a “hello.py” file:

```
cd ~/ros_ws/src/a_pkg/src
gedit hello.py
```

Inside the window that just open, write the following python code and save the file:

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

This node will only print “hello word” when executed. In Python you need to make this file executable. Still at your first node location, execute the following command:

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

In order to integrate the changes you made in your workspace, re-build it from the workspace folder “ros_ws”:

```
cd ~/ros_ws
catkin_make
```

ii – Run a node

In order to run anything in ROS a “rosmaster” as to run. So just tape the following command line to start it:

```
roscore
```

Do not close the terminal where “roscore” is running and open another one. Normally you would need to source your new terminal, but it’s not necessary if you modify your “bashrc” file as explained previously. This new terminal will be used to run the node you just wrote with the following command (“rosrun package_name node_name”):

```
rosrun a_pkg hello.py
```

5

You should see **`Hello world`** appearing in your terminal. That would means the node has been correctly executed.

5 – Topics

During robot programming you would need to exchange information between nodes. The most common tool for that are topics. A topics is not an executable code as a node. A topic is created inside node. To exchange data through topics, a node has to publish and another one must subscribe to the topic to receive the information. So in this part we will need to create 2 nodes. The objective is to make a node that count from 1 to 9 and spin. A second node will receive each number and print it with extra information. You should now know how to create python nodes. So first create the publisher we will call "a_pub.py" and write inside the following code (don't forget to make it executable afterwards):

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

import rospy
from std_msgs.msg import Int8

rospy.init_node('a_pub',anonymous = True)
rate = rospy.Rate(1)
pub =  rospy.Publisher('a_topic', Int8, queue_size = 1)

number = 0

while not rospy.is_shutdown():
    number = number + 1
    if number == 10:
        number = 1
    pub.publish(number)
    rate.sleep()
```

NB: Be careful when copy paste, Pyhton is tab sensitive.

Let's detail a bit this code:

`rospy.init_node('a_pub',anonymous = True)`
is used to initialize the node.

`rate = rospy.Rate(1)`
enable to set up the frequencies (inHz) the node will spin at. It comes along the rate.sleep(). The latter line makes the program wait that rate if the computation time to reach this line was under the rate.

6

`pub = rospy.Publisher('a_topic', Int8, queue_size = 1)`
This is the important line regarding the topic. This line "creates" the topic. The arguments of the function "Publisher" describe it. The name of the topic is "a_topic", the messages that will be exchanged through this topic are of the type "Int8". This message type is part of the "std_msgs.msg" library. That explain the 3rd line **`from std_msgs.msg import Int8`**.The last argument represents the buffer. Here, only 1 message will be stored is nothing "take" it (by "take" understand another node that subscribes to the same topic).

```
while not rospy.is_shutdown():
    number = number + 1
    if number == 10:
        number = 1
    pub.publish(number)
    rate.sleep()
```

This part is the heart of the node. It "computes" a number and publishes it on the topic previously created.
Now create the subscriber that we'll call "a_sub.py" with the following code:

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

from std_msgs.msg import Int8

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + " I heard %d
",data.data)

rospy.init_node('a_sub', anonymous=True)
rospy.Subscriber("a_topic",Int8,callback)
rospy.spin()
```

The main line is the following one:
`rospy.Subscriber("a_topic",Int8,callback)`

It makes the node subscribe to the topic "a_topic" which will receive messages of type "Int8" and process the function callback each time it receives something.

The callback function:
```
def callback(data):
    rospy.loginfo(rospy.get_caller_id() + " I heard %d", data.data)
```
simply prints the received number with additional information.

Now run both nodes. You'll need a terminal to run "roscore", another one to run the publisher and a third one for the subscriber. You should see something printed into the subscriber terminal every second.

7

You can also check a topic with some useful commands. With the previous node running, in a new terminal, try the following commands and see what they do.

```
rostopic list
```

```
rostopic info /a_topic
```

```
rostopic echo /a_topic
```

6 – Launch file

As you probably noticed, it's a bit annoying to use a terminal to run a node. The launch file is used to facilitate the execution of programs. It's basically a list of things to start simultaneously. In this tutorial we will create a launch to start both "a_pub" and "a_sub" nodes. Create a launch folder into your "a-pkg" folder with a launch file in it called "a_launch.launch".

```
mkdir ~/ros_ws/src/a_pkg launch
cd launch
gedit a_launch.launch
```

In this file write the following code:

```
<launch>
  <node name="a_pub" pkg="a_pkg" type="a_pub.py" output=
"screen" />
  <node name="a_sub" pkg="a_pkg" type="a_sub.py" output=
"screen" />
 </launch>
```

Re-build your workspace from the "ros_ws" folder:

```
cd ~/ros_ws
catkin_make
```

To launch the launch file, use the following command:

```
roslaunch a_pkg a_launch.launch
```

B – Mobile robot programming

This part is highly inspired from the ROS summer school from fh-Aachen.

This tutorial consists in modeling a mobile robot and simulate it in a virtual map with obstacles. The simulation is based on the Gazebo software. According to their website, “Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community“. Once the robot is simulated we will try to command it.

This tutorial is split into 5 steps:

1 / Creation a robot model

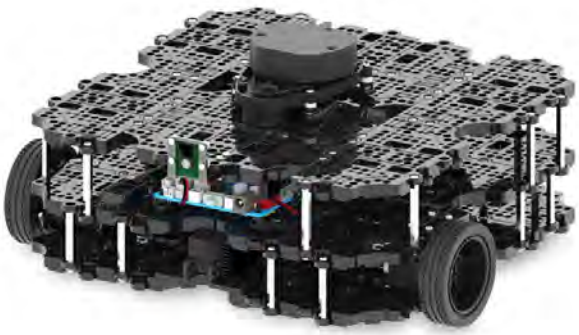
2 / Simulation of the robot

3 / Exploration and mapping

4 / Localization

5 / Path planning

The robot used in this exercise is a turtlebot3 waffle as illustrated by the picture below. The waffles are the plastic parts that can be assembled in many ways. A scanner is represented at the top of the robot. The robot can move thanks to two wheels driven by two servo-motors. An IMU and a camera has been added to the robot.



For convenience reason this tutorial is proposed on a virtual machine. Such a machine is obviously a bit slower than a regular real machine. So don’t put to much pressure on it, let it compute, it’s working..; probably. You’ll need some file during this tutorial, you’ll find them into the “Documents_ROS” directory.

Be careful, some mistakes could slide in during the copy/paste of code from this document to the useful files.

First, If you did not follow the first tutorial, please create a workspace with the name “ros_ws” and modify the “bashrc” file in order to source this workspace at the opening of the terminal.

Have fun!

1 – Model the robot

i – Describing the robot with “.urdf” files

URDF stands for **U**nified **R**obot **D**escription **F**ormat. It is an XML based file used to describe robots mechanical design and physical properties. Usually the robot builder provides the “.urdf” file for its robot.

- First create a package called “tb3_description” with the “std_msgs”, “rospy”, “tf2_ros” and “urdf” dependencies.

```
cd ~/ros_ws/src
catkin_create_pkg tb3_description std_msgs rospy tf2_ros urdf
```

- In this package add the “meshes” folder that you’ll find within the “Documents_ROS” folder in the “home” directory. This folder contain the “.stl” files necessary to visualize the robot.
-

- Now, create a “urdf” folder and add into it the file “turtlebot3_waffle_base.urdf.xacro”. This file will be the base of the robot model. Others elements will be added to it.
  Xacros(XML Macros) are commonly used to organize “.urdf” files and include among other things, constant values, macros and other “xacro” files. This file demonstrates the use of one such “xacro” to include another file containing common constants.

- Add also the “wheel.urdf.xacro” file to the “urdf” folder. Complete the “turtlebot3_waffle_base.urdf.xacro” at the beginning with the following lines:

```
<!-- Turtlebot3 platform include files -->
<xacro:include filename="$(find
tb3_description)/urdf/wheel.urdf.xacro"/>
```

- Then we use macros to replicate the code for each wheel with **alignment** and **origin** values as the variables. Paste the following line at the end of the file:

```
<!-- Turtlebot3 wheel macros -->

<xacro:turtlebot3_wheel alignment="left">
  <origin xyz="0.0 0.144 0.023" rpy="${-M_PI*0.5} 0 0"/>
</xacro:turtlebot3_wheel>

<xacro:turtlebot3_wheel alignment="right">
  <origin xyz="0.0 -0.144 0.023" rpy="${-M_PI*0.5} 0 0"/>
</xacro:turtlebot3_wheel>
```

- Add also the common properties file to the “urdf” folder and add the following line to the “turtlebot3_waffle_base.urdf.xacro”:

```
<!-- common constants -->
  <xacro:include filename="$(find
tb3_description)/urdf/common_properties.xacro"/>
```

- Similarly to the wheels, add the “caster.urdf.xacro” to the folder and complete the file “turtlebot3_waffle_base.urdf.xacro” with:

```
<!-- Turtlebot3 platform include files -->
<xacro:include filename="$(find
tb3_description)/urdf/caster.urdf.xacro"/>
```

and

```
<!-- Turtlebot3 caster macros -->
<xacro:turtlebot3_caster alignment="left">
  <origin xyz="-0.21 0.064 -0.004" rpy="0 0 0"/>
</xacro:turtlebot3_caster>
<xacro:turtlebot3_caster alignment="right">
  <origin xyz="-0.21 -0.064 -0.004" rpy="0 0 0"/>
</xacro:turtlebot3_caster>
```

- The sensors now need to be added. Add the corresponding file from the “Documents_ROS” folder to your “urdf” folder. Then add the following lines to the “turtlebot3_waffle_base.urdf.xacro”:

```
      <!-- sensors -->
<xacro:include filename="$(find tb3_description)/urdf/imu.urdf.xacro"/>
<xacro:include filename="$(find tb3_description)/urdf/laser.urdf.xacro"/>
```

11

- Finally we need to create a launch file to load the robot model. Create a launch folder and a launch file in it called “turtlebot3_model.launch”. Write in it the following code:

```
<?xml version="1.0"?>
<launch>
  <param name="robot_description" command="$(find xacro)/xacro --inorder $
(find tb3_description)/urdf/turtlebot3_waffle_base.urdf.xacro" />
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
    <param name="publish_frequency" value="30.0"/>
  </node>
</launch>
```

Now, if you want to visualize your robot start Rviz with this command line:

```
rosrun rviz rviz
```

In the right menu add an element of type “RobotModel”. You should see your robot partially and you will have an error regarding the wheels. This is because some transformation haven’t been created yet. They will be create in the next steps no worries.

ii – Gazebo plugins

The “urdf” description need an extra plugin in order to work in Gazebo. So go back to the “urdf” folder with the robot description in the “tb3_description” package. Create a “turtlebot3_waffle_base.gazebo.xacro” file and copy/paste the following code in it.

```
<?xml  version= "1.0" ?>
<robot name= "turtlebot3_waffle_sim"
xmlns:xacro="http://ros.org/wiki/xacro" >
  <gazebo>
    <plugin name= "turtlebot3_waffle_controller" filename=
"libgazebo_ros_diff_drive.so" >
      <commandTopic> cmd_vel </commandTopic>
      <odometryTopic> odom </odometryTopic>
      <odometryFrame> odom </odometryFrame>
      <odometrySource> world </odometrySource>
      <publishOdomTF> true </publishOdomTF>
      <robotBaseFrame> base_footprint </robotBaseFrame>
      <publishWheelTF> false </publishWheelTF>
      <publishTf> true </publishTf>
      <publishWheelJointState> true </publishWheelJointState>
      <legacyMode> false </legacyMode>
      <updateRate> 100 </updateRate>
      <leftJoint> wheel_left_joint </leftJoint>
      <rightJoint> wheel_right_joint </rightJoint>
      <wheelSeparation> 0.287 </wheelSeparation>
      <wheelDiameter> 0.066 </wheelDiameter>
      <wheelAcceleration> 1  </wheelAcceleration>
      <wheelTorque> 10 </wheelTorque>
      <rosDebugLevel> na </rosDebugLevel>
    </plugin>
  </gazebo>
</robot>
```

This file need to be included into the ".urdf.xacro" file of the robot. So open "turtlebot3_waffle_base.urdf.xacro" and add the following line at the right place.

```
<!-- Gazebo simulation plugins -->
<xacro:include filename="$(find
tb3_description)/urdf/turtlebot3_waffle_base.gazebo.xacro"/>
```

A plugin for the sensors is also necessary for the gazebo simulation. This time we won't use a macro and add directly the plugins into the ".urdf.xacro" files. Each plugin has to be paste inside the **`<robot>`** section.

13

IMU plugin:

```
<!-- Gazebo sensor plugin -->
  <gazebo>
    <plugin name= "imu_plugin" filename= "libgazebo_ros_imu.so" >
      <always_on> true </always_on>
      <bodyName> imu_link </bodyName>
      <frameName> imu_link </frameName>
      <topicName> imu </topicName>
      <serviceName> imu_service </serviceName>
      <gaussianNoise> 0.0 </gaussianNoise>
      <updateRate> 200 </updateRate>
      <imu>
        <noise>
          <type> gaussian </type>
          <rate>
            <mean> 0.0 </mean>
            <stddev> 2e-4 </stddev>
            <bias_mean> 0.0000075 </bias_mean>
            <bias_stddev> 0.0000008 </bias_stddev>
          </rate>
          <accel>
            <mean> 0.0 </mean>
            <stddev> 1.7e-2 </stddev>
            <bias_mean> 0.1 </bias_mean>
            <bias_stddev> 0.001 </bias_stddev>
          </accel>
        </noise>
      </imu>
    </plugin>
  </gazebo>
```

Laser plugin:

```
<!-- Gazebo sensor plugin -->
  <gazebo reference= "laser" >
    <material> Gazebo/DarkGrey </material>
    <sensor type= "ray" name= "SICK_laser" >
      <always_on> 1 </always_on>
      <update_rate> 15.0 </update_rate>
      <visualize> false </visualize>
      <ray>
        <scan>
          <horizontal>
            <samples> 820 </samples>
            <resolution> 0.5 </resolution>
            <min_angle> ${-3.0/4.0 * M_PI} </min_angle>
            <max_angle> ${ 3.0/4.0 * M_PI} </max_angle>
          </horizontal>
        </scan>
        <range>
          <min> 0.05 </min>
          <max> 25 </max>
          <resolution> 0.01 </resolution>
        </range>
      </ray>
      <plugin name= "SICK_laser_plugin" filename=
"libgazebo_ros_laser.so" >
        <update_rate> 15.0 </update_rate>
        <frameName> laser </frameName>
        <topicName> scan </topicName>
        <gaussianNoise> 0.0 </gaussianNoise>
        <SICK_laserMinIntensity> 101 </SICK_laserMinIntensity>
      </plugin>
    </sensor>
  </gazebo>
```

2 – Simulation of the robot

i – Gazebo

This part consists in simulating the robot we just model and its environment. Create a new package called “tb3_gazebo”. Add the folder “worlds” in it that you’ll find in the “Documents_ROS” folder. This folder describes the map where the robot will evolve.

```
cd ~/ros_ws/src
catkin_create_pkg tb3_gazebo rospy std_msgs sensor_msgs
geometry_msgs nav_msgs tf2_ros gazebo_ros
```

16

In a launch folder, create a launch file called “world.launch” with the following code:

```
<launch>
 <arg name="world_name" default="$(find
tb3_gazebo)/worlds/arena_2018.world"/>
 <arg name="x" default="0.0"/>
 <arg name="y" default="0.0"/>
 <arg name="z" default="0.2"/>
 <arg name="roll" default="0.0"/>
 <arg name="pitch" default="0.0"/>
 <arg name="yaw" default="1.57"/>
 <include file="$(find gazebo_ros)/launch/empty_world.launch">
 <arg name="world_name" value="$(arg world_name)"/>
 <arg name="paused" value="false"/>
 <arg name="use_sim_time" value="true"/>
 <arg name="gui" value="true"/>
 <arg name="headless" value="false"/>
 <arg name="debug" value="false"/>
 </include>
 <!-- robot description -->
 <param name="robot_description" command="$(find xacro)/xacro --
inorder $(find
tb3_description)/urdf/turtlebot3_waffle_base.urdf.xacro" />
 <!-- robot state publisher -->
 <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
 <param name="publish_frequency" value="30.0"/>
 </node>
 <!-- joint state publisher -->
 <node pkg="joint_state_publisher" type="joint_state_publisher"
name="joint_state_publisher">
 <param name="rate" value="30.0"/>
 <param name="use_gui" value="false"/>
 </node>
<node pkg="gazebo_ros" type="spawn_model" name="spawn_tb3" args="-urdf
-param robot_description -model turtlebot3_waffle -x $(arg x) -y $(arg
y) -z $(arg z) -R $(arg roll) -P $(arg pitch) -Y $(arg yaw)"/>
</launch>
```

Launch this file in order to see your robot in its world in gazebo.

17

ii – Teleoperation

We would like now to be able to move the robot. First, we will use the keyboard arrow to command the robot. Our robot model is made to receive information on the topic "/cmd_vel" (it can be visualized with the command **`rostopic list`** and **`rostopic echo /cmd_vel`**). This topic takes linear velocity and angular velocity. So we need a node that publishes on this topic. Fortunately such a thing already exists. Install the package "key_teleop" with this command.

```
sudo apt-get install ros-kinetic-key-teleop
```

The package contains only one node named "key_teleop.py" that makes the job. But we need to be sure that it publishes on the right topic. For that, find the executable file in your computer and open it. It should be located here: "/opt/ros/kinetic/lib/key_teleop/key_teleop.py". Find the line where the publisher is initialized and modify it to publish on the right topic. Be careful, the file can be modified only as "sudo" so open it from a terminal with "gedit". The line to modify should be 181.

Launch the simulation and make the robot moves (the "key_teleop" node could also be added to the launch file if you want):

```
roslaunch tb3_gazebo world.launch
rosrun key_teleop key_teleop.py
```

In order to move the robot, you need to have your terminal with key_teleop running in highlight.

If you run "Rviz" you would notice that the missing transformation are present now. No error should be shown any more.

3 – Exploration and mapping

Now that we have a perfectly functioning robot that follow the arrow command, we would like to make it a bit more intelligent. The aim of the 3 lasts parts is to make the robot able to navigate into its world. Stop the simulation and create a new package called "tb3_navigation" with the following dependencies.

```
cd ~/ros_ws/src
catkin_create_pkg tb3_navigation rospy std_msgs move_base
global_planner teb_local_planner
```

The first step is to make the robot create its own map of the environment. For that we'll use the "hector_slam" package. So first install it :

```
sudo apt-get install ros-kinetic-hector-slam
```

Another package will be useful:

```
sudo apt-get install ros-kinetic-map-server
```

The hector_slam package allows to create a map according to the scanner data and the odometry of the robot. Create a launch file called “hector_slam.launch”. The launch file is used only to launch the node “hector_slam” with some parameters.

```
<?xml version="1.0"?>
<launch>
    <node pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">
        <!-- Frame names -->
        <param name="map_frame" value="map"/>
        <param name="base_frame" value="base_link"/>
        <param name="odom_frame" value="odom"/>

        <!-- Topic names -->
        <param name="scan_topic" value="scan"/>

        <!-- Tf use -->
        <param name="use_tf_scan_transformation" value="true"/>
        <param name="use_tf_pose_start_estimate" value="false"/>
        <param name="pub_map_odom_transform" value="true"/>

        <!-- Map size / start point -->
        <param name="map_resolution" value="0.025"/>
        <param name="map_size" value="1024"/>
        <param name="map_start_x" value="0.5"/>
        <param name="map_start_y" value="0.5"/>
        <param name="map_multi_res_levels" value="4"/>

        <!-- Map update parameters -->
        <param name="update_factor_free" value="0.4"/>
        <param name="update_factor_occupied" value="0.9"/>
        <param name="map_update_distance_thresh" value="0.4"/>
        <param name="map_update_angle_thresh" value="0.06"/>
        <param name="map_pub_period" value="2.0"/>

        <!-- Advertising config -->
        <param name="advertise_map_service" value="false"/>
    </node>
</launch>
```

We will drive the robot manually in its environment to make it discover it. So first we need to launch the gazebo launch file for the simulated robot and environment and then the teleoperation node to move the simulated robot:

```
roslaunch tb3_gazebo world.launch
rosrun key_teleop key_teleop.py
```

Also launch the “hector_slam.launch” file you just created to make it record its environment. You can start moving the robot around the map. But to be sure that the map is well done, it would be great to see it under construction. For that we will use “Rviz” again.

```
rosrun rviz rviz
```

In “Rviz” add a “map” element and choose the specific topic. You should then see the robot moving according to your action with the keyboard arrow and the map building step by step. Once the map seems good to you, it necessary to save it. The “map_saver” node from the “map_server” package will be used to do so. We will also choose the name “arena” for the saved file. In a new terminal run the following node:

```
rosrun map_server map_saver -f arena
```

This node has created a map as a “.pgm” file and a “.yaml” file. Put these two files in a “map” folder in your “tb3_navigation” package. The last step consist on publishing this map to make it available for other node thanks to the node “map_server” from the package “map_server”. Create a launch file “map.launch” in your package “tb3_navigation” with the following code:

```
<?xml version="1.0"?>
<launch>
   <node name="map_server" pkg="map_server" type="map_server"
args="$(find tb3_navigation)/map/arena.yaml"/>
</launch>
```

The launch file starts the “map_server” node, which will publish a map under the topic “/map”. The argument includes the path to your map file. For further information look at http://wiki.ros.org/map_server.

NB: To visualize the in Rviz, change the fixed frame to “map”. It’s not possible to see both the map and the robot at the same time. A transformation between the robot and the map is required and is computed thanks to the next step.

4 – Localization

The localization part consists in the robot to recognizing its position in the map according to what its sees (or what its sensors measure). For that we will use the AMCL package. AMCL stand for **A**daptive **M**onte **C**arlo **L**ocalization. It is an algorithm developed by Dieter Fox which uses a particle filter to track the position of a robot against a known map. The algorithm needs a first guest that could be given by the odometry of the robot. Install this package:

```
sudo apt-get install ros-kinetic-amcl
```

Create the launch file “amcl.launch” in your tb3_navigation package. The launch file will run the “amcl” node with some parameters.

```
<launch>
 <node pkg="amcl" type="amcl" name="amcl" output="screen">
  <!-- Publish scans from best pose at a max of 10 Hz -->
  <param name="odom_model_type" value="diff-corrected"/>
  <param name="odom_alpha5" value="0.1"/>
  <param name="transform_tolerance" value="0.2" />
  <param name="gui_publish_rate" value="10.0"/>
  <param name="laser_max_beams" value="30"/>
  <param name="min_particles" value="200"/>
  <param name="max_particles" value="2000"/>
  <param name="kld_err" value="0.05"/>
  <param name="kld_z" value="0.99"/>
  <param name="odom_alpha1" value="0.2"/>
  <param name="odom_alpha2" value="0.2"/>
  <!-- translation std dev, m -->
  <param name="odom_alpha3" value="0.8"/>
  <param name="odom_alpha4" value="0.2"/>
  <param name="laser_z_hit" value="0.5"/>
  <param name="laser_z_short" value="0.05"/>
  <param name="laser_z_max" value="0.05"/>
  <param name="laser_z_rand" value="0.5"/>
  <param name="laser_sigma_hit" value="0.2"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_model_type" value="likelihood_field"/>
<!-- <param name="laser_model_type" value="beam"/> -->
  <param name="odom_frame_id" value="odom"/>
  <param name="base_frame_id" value="base_footprint"/>
  <param name="global_frame_id" value="map"/>
  <param name="tf_broadcast" value="true"/>
  <param name="laser_likelihood_max_dist" value="2.0"/>
  <param name="update_min_d" value="0.002"/>
  <param name="update_min_a" value="0.005"/>
  <param name="resample_interval" value="1"/>
  <param name="transform_tolerance" value="0.1"/>
  <param name="recovery_alpha_slow" value="0.002"/>
  <param name="recovery_alpha_fast" value="0.1"/>
 </node>
</launch>
```

22

Now start the complete simulation and Rviz:

```
rosrun rviz rviz
```

```
roslaunch tb3_gazebo world.launch
```

```
rosrun key_teleop key_teleop.py
```

```
roslaunch tb3_navigation map.launch
```

```
roslaunch tb3_navigation amcl.launch
```

Put the Rviz window and the Gazebo window side by side. In Rviz, change the fixed frame to “map” and add a “PoseArray” element subscribing to the topic “/particlecloud”. This topic represent the estimation of the position through the “amcl” algorithm. Now, in Gazebo, move manually the robot. Select it and change its position and/or orientation from the left menu. See what happen in the Rviz windows. You should see a cloud of red arrow converging step by step to the right position of the robot. This is the effect of the AMCL algorithm.

Now turn off just the AMCL algorithm (ctrl+c in the terminal which run the “amcl.launch” file). Once it’s turned off, move again the robot in another position (make sure the robot is not in an obstacle). The robot in Rviz didn’t follow the displacement. Now start again the “amcl.launch” file. The robot is still lost because the AMCL algorithm need a first guess. You can give it with the “2D Pose Estimate” button on the top of the Rviz window. Put on purpose the first guess a bit aside of the actual robot position and visualize the AMCL process. The real position of the robot should finally be found.

5 – Path planning

At this moment, the robot has its own map of the environment. It also knows where it is in this map according to what it “sees”. We’ll try now to make it move into this map autonomously. For that we will use the “move_base” package.

```
sudo apt-get install ros-kinetic-move-base
```

Create the “move_base.launch” file with the following code:

```
<launch>
    <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
        <param name="base_local_planner"
value="teb_local_planner/TebLocalPlannerROS"/>
        <param name="base_global_planner"
value="global_planner/GlobalPlanner"/>
        <rosparam file="$(find
tb3_navigation)/cfg/costmap_common_params.yaml" command="load"
ns="global_costmap" />
        <rosparam file="$(find
tb3_navigation)/cfg/costmap_common_params.yaml" command="load"
ns="local_costmap" />
        <rosparam file="$(find
tb3_navigation)/cfg/global_costmap_params.yaml" command="load" />
        <rosparam file="$(find
tb3_navigation)/cfg/local_costmap_params.yaml" command="load" />
        <rosparam file="$(find
tb3_navigation)/cfg/global_planner_params.yaml" command="load" />
        <rosparam file="$(find
tb3_navigation)/cfg/teb_local_planner_params.yaml" command="load" /
>
        <rosparam file="$(find
tb3_navigation)/cfg/costmap_converter_params.yaml" command="load" /
>
        <rosparam file="$(find
tb3_navigation)/cfg/move_base_params.yaml" command="load"/>
    </node>
</launch>
```

As you can see the launch file ask for many parameters that are contained into several “.yaml” files. Copy/paste these files from “Documents_ROS” folder into a “cfg” folder in your package.
You can now test you complete simulation of the robot. Launch the following:

```
roslaunch tb3_gazebo world.launch
```

```
roslaunch tb3_navigation map.launch
```

```
roslaunch tb3_navigation amcl.launch
```

```
roslaunch tb3_navigation move_base.launch
```

Try the navigation node by giving it a goal. To do so click on the “2D Nav Goal” button on the top of the window. Give a goal by clicking on the map. You can visualize the path with “Rviz”. Add a path element. Three topics are available corresponding to the path planning system from the “move_base” node.

Annexe A: Configuration required for this tutorial

Ubuntu 16.04
terminator
ros kinetic : http://wiki.ros.org/kinetic/Installation/Ubuntu

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $
(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-
latest.list'

sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
--recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full

sudo rosdep init
rosdep update

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc

sudo apt install python-rosinstall python-rosinstall-
generator python-wstool build-essential
```

Packages already installed:

ros-kinetic-turtlebot3
screen
ros-kinetic-global-planner
teb-local-planner

Packages installed during the tutorial:

map-server
hector_slam
amcl
move-base

Modelling and calibration of moving head (ZeMA)

Das ZeMA bietet bei der diesjährigen Robotix-Academy Summer School einen Workshop an zur Kalibrierung, Modellierung und Steuerung von Assistenzsystemen am Beispiel eines Moving Head.
Ein Moving Head wird üblicherweise in der Theater- und Veranstaltungstechnik eingesetzt und verfügt somit nicht über ein Standardkalibrierungsverfahren. Im Rahmen dieses Workshops wird erklärt, wie der Moving Head in die industrielle Umgebung integriert werden kann, um das Ziel von Robotern oder Menschen zu visualisieren.
Zuerst wird eine Kalibrierungsmethode für den Arbeitsraum vorgestellt und dann eine Methode zur Integration des beweglichen Kopfes IoT-Ready in bestehende Systeme mittels MQTT. Der bewegliche Kopf hat eine offene serielle Kinematik, so dass das Kinematik- und Modellierungsproblem des beweglichen Kopfes analog zu einem Knickarmroboter gelöst warden kann.
Anschließend wird die Kinematik des beweglichen Kopfes mit Hilfe der Denavit-Hartenberg-Konvention modelliert. Ein neuartiges Kalibrierverfahren, das auf vier vor-

Lors de l'université d'été de la Robotix-Academy de cette année, le centre ZeMA propose un atelier sur l'étalonnage, la modélisation et le contrôle des systèmes d'assistance en utilisant l'exemple d'un moving head (projecteur asservi).
Un moving head est généralement utilisé dans la technologie du théâtre et de l'événementiel et n'a donc pas de procédure de calibrage standard. Cet atelier explique comment le moving head peut être intégré dans l'environnement industriel pour visualiser la cible des robots ou des humains.
D'abord une méthode de calibration de l'espace de travail est présentée, suivie d'une méthode d'intégration du moving head IoT-Ready dans les systèmes existants en utilisant le MQTT. Le moving head a une cinématique en série ouverte, de sorte que le problème de cinématique et de modélisation du moving head peut être résolu de manière analogue à un robot à bras articulé.
Par la suite, la cinématique du moving head est modélisé selon la convention Denavit-Hartenberg. Une nouvelle procédure d'étalonnage basée sur quatre points

definierten Punkten basiert, wird eingeführt, um die Ausrichtung des beweglichen Kopfes im dreidimensionalen Raum zu bestimmen.
Der bewegliche Kopf ist kein cyberphysikalisches System und kann nicht mit seiner Umgebung kommunizieren. Um den Anforderungen eines IoT-Szenarios gerecht zu werden, muss er mit einem Raspberry Pi ausgestattet sein. Mit Hilfe eines Node-RED-Flows wird die MQTT-Nachricht in gerätespezifische Befehle übertragen. Dies erfolgt mit anderen Geräten in vergleichbarer Weise.

Kontakt:
ZeMA
Ali Kanso
E-mail: a.kanso@zema.de

Fabian Adler
E-mail: f.adler@zema.de

prédéfinis est introduite pour déterminer l'orientation de la tête mobile dans l'espace tridimensionnel.
Le projecteur asservi n'est pas un système cyberphysique et ne peut pas communiquer avec son environnement. Pour répondre aux exigences d'un scénario IdO, il doit être équipé d'un Pi Raspberry. À l'aide d'un flux Node-RED, le message MQTT est transféré dans des commandes spécifiques à l'appareil. Cela se fait à l'aide d'autres dispositifs de manière comparable.

Contact:
ZeMA
Ali Kanso
e-mail: a.kanso@zema.de

Fabian Adler
e-mail: f.adler@zema.de

Robotix-Academy Summer School 2019

Ali Kanso, M.Sc.

Fabian Adler, M.Sc.

Lorraine Université, 06.09.2019

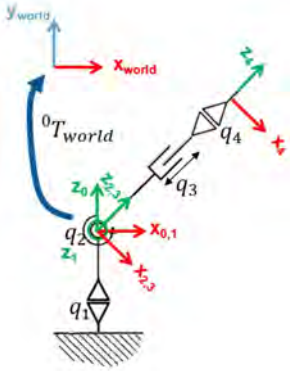
Introduction

The Moving-head calibration and the control via MQTT and Node-RED is part of a control concept called "Assist-by-X".

The Assist-By-X (ABX) system is one implementation of a modular assistance system combining the advantages of modular engineering as well as easily programmable cyber-physical systems communicating with their environment via MQTT. The idea for this project is to create a modular system that provides visual assistance for workers which can be adapted to different production environments with individual assistance tasks. In an assembly system it represents an assistance module, whereby the skills of the module can be defined by exchanging the projection device. For this reason, the system is called ABX, since the X is representative for the different devices that can be implemented. So far, a laser projector, a moving head spot and a smart video projector are implemented. Depending on the use-case it is expected to decrease processing times, probability of errors or create a safer working environment on the production line, thereby increasing worker performance. Existing commercial systems, such as "Light Guide Systems" by OPC Solutions, "Der Schlaue Klaus" by Optimum (https://www.optimum-gmbh.de/der-schlaue-klaus.html) or "Human Interface Mate" by Arkite (https://arkite.nl/) focus on using one projection system for the assistance task with commissioning and work processes aligned to the specific device. The ABX distinguishes from these systems as the requirements of assistance tasks are used as a basis. Suitable projection devices which offer the right skills to fulfill the task can be integrated into the system using a generalized work flow which is independent from the specific device.

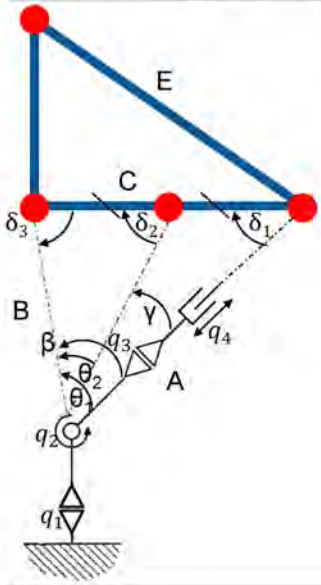
Kinematic structure of a moving head

- CBZ structure
- 4 degree of freedom



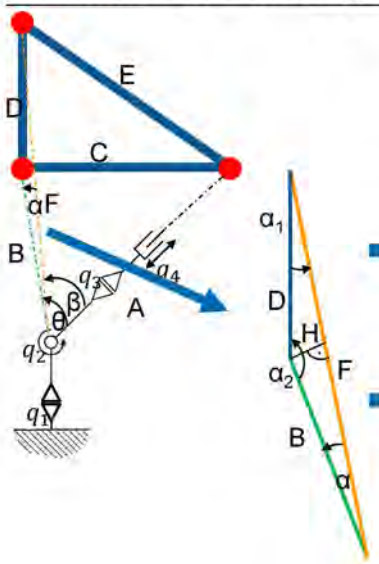
Link	δ_{i-1} [rad]	d_{i-1} [m]	l_{i-1} [m]	λ_{i-1} [rad]
1	q_1	0	0	$\frac{\pi}{2}$
2	q_2	0	0	$-\frac{\pi}{2}$
3	0	q_3	0	0

Calibration of a moving head



- Law of sines:
  - $\frac{B}{\sin(\delta_1)} = \frac{C}{\sin(\theta_1)} = c1$
  - $\frac{B}{\sin(\delta_2)} = \frac{C/2}{\sin(\theta_2)} = \frac{A2}{\sin(\delta_3)} = c2$
  - $180 = \theta_1 + \delta_1 + \delta_3$
  - $180 = \theta_2 + \delta_2 + \delta_3$
  - $\theta_1 + \delta_1 = \theta_2 + \delta_2 \rightarrow \delta_1 = \theta_2 + \delta_2 - \theta_1 = \theta + \delta_2$
  - $B = c1 \cdot \sin(\delta_1) = c1 \cdot \sin(\theta + \delta_2) = c2 \cdot \sin(\delta_2)$
  - $c1 \cdot \sin(\theta + \delta_2) = c1 \cdot \sin(\theta)\cos(\delta_2) + c1 \cdot \cos(\theta)\sin(\delta_2)$
  - $\rightarrow c2 \cdot \sin(\delta_2) = c1 \cdot \sin(\theta)\cos(\delta_2) + c1 \cdot \cos(\theta)\sin(\delta_2)$
  - $(c2 - c1 \cdot \cos(\theta))\sin(\delta 2) = c1 \cdot \sin(\theta)\cos(\delta_2)$
  - $\frac{\sin(\delta 2)}{\cos(\delta 2)} = \frac{c1 \cdot \sin(\theta)}{(c2 - c1 \cdot \cos(\theta))} \rightarrow \delta 2 = \arctan\left(\frac{c1 \cdot \sin(\theta)}{(c2 - c1 \cdot \cos(\theta))}\right)$
  - $0 \leq \delta 2 \leq 180$
  - $B = c2 \cdot \sin(\delta_2)$

Calibration of a moving head



- $\delta_3 = 180 - \delta_2 - \theta_2$
- $A = \sqrt{(B^2 + C^2) - 2BC \cdot cos(\delta_3)}$
- $H = B \cdot \sin(\alpha) \rightarrow \sin(\alpha_1) = \frac{H}{D}; cos(\alpha 1) = \frac{F - B \cdot cos(\alpha)}{D}$
- $\rightarrow \alpha_1 = arctan2\left(\frac{H}{D}, \frac{F - B \cdot cos(\alpha)}{D}\right)$
- $\alpha_2 = 180 - \alpha_1 - \alpha$

- Law of sines:

$$\frac{B}{\sin(\alpha_1)} = \frac{F}{\sin(\alpha_2)} = \frac{D}{\sin(\alpha)} = c3$$

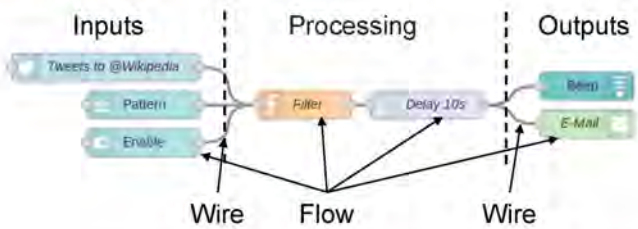
$$\rightarrow F = c3 \cdot \sin(\alpha_2)$$

- Rotation angles are known:
  - Substitute A, B and F in the direct kinematic model
  - → position of p1, p2 und p3 are known with respect Base-frame
    - →Calibration

General Facts to Node-RED

Node-RED

- Developed from IBM
- Built on Node.js and written in Javascript
- Visual programming in a web browser-based editor
- Flow based development tool




```
npm install -g --unsafe-perm node-red
```


Install Node-RED

1. Install Node.js
   - https://nodejs.org/dist/v10.16.3/node-v10.16.3-x64.msi
   - node --version && npm –version (in cmd)
2. Install Node-RED with cmd:
   - npm install -g --unsafe-perm node-red
   - start node-red (dont close the opened cmd-window)
3. Access Node-RED in browser
   - http://127.0.0.1:1880/
   - http://localhost:1880/
   - http://{YOUR IP}:1880/
4. Install package **node-red-dashboard**
5. Login to the demonstrator wifi
   - Name: Robotix_Summer_School
   - Password: Robotix123$

 Seite 7 ZeMA

Optional important packages for controlling a moving-head

- node-red-contrib-mqtt-broker
- For Moving-Head communication:
  - node-red-contrib-artnet

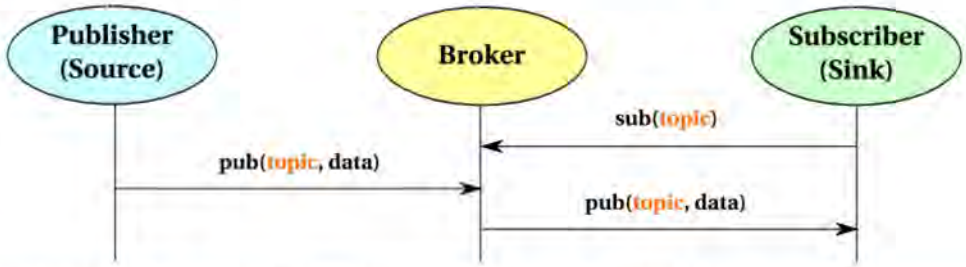
 Seite 8 ZeMA

Implementation of the software concept

MQTT – Message Queuing Telemetry Transport

- OASIS Standard
- Machine-to-machine connectivity protocol
- Publish and subscribe pattern

MQTT output-format is a:
- **String**
- Buffer
- **JSON-Object**



Urs Hunkeler & Hong Linh Truong, Andy Stanford-Clark: MQTT-S – A Publish/Subscribe Protocol For Wireless Sensor Networks, https://sites.cs.ucsb.edu/~richburroughs/cs290b-cloudpapers/mqt-s.pdf

Decoupling of all communication participants, simplicity of use and resource-saving features

Node-RED Message Object

- You will see that is has the following properties
  - payload
  - topic
  - _msgid
- A msg object that originates from an **MQTT** input node has the following properties:
  - payload
  - topic
  - qos
  - _msgid
  - retain

Getting started

Structure of the MQTT-topic:

- In general: {device}/{property}
- Unlimited herachy levels are possible
- Moving:
  - Moving-head/point
- Setting property:
  - Moving-head/dashboard

Publish to a topic:

- Moving-head/point expects following structure:
  - Array in form of [x,y,z]
- Moving-head/dashboard expects following structure:
  - { buckets: [{channel: 6, value: value},]};

- Connect to the MQTT-Broker:

Name: MQTT Broker: {IP-adress} or {name of the computer}

Server: {IP-adress} or {name of the computer}

Default-Port: 1883

For Saving = Press **deploy** at the top right corner

 Seite 11 ZeMA

13-Channel dmx mode Moving-Head (manufacturer-dependent)

Channel	Value	Function
1	0…255	Pan movement
2	0…255	Pan movement fine adjustment
3	0…255	Tilt movement
4	0…255	Tilt movement fine adjustment
5	0…255	Pan and tilt speed
6	0…255	Shutter (0 … 100 %)
7	0…9	Light is blacked out
	10…127	Strobe effect, increasing speed
	128…249	Strobe effect with random control, increasing speed
	250…255	Light is constantly on

 Seite 12 ZeMA

13-Channel dmx mode Moving-Head (manufacturer-dependent)

8	Colour wheel	
	0…4	White
	5…9	White + red
	10…14	Red
	15…19	Red + Orange
	20…24	Orange
	25…29	Orange + Cyan
	30…34	Cyan
	35…39	Cyan + Green
	40…44	Green
	45…49	Green + Light green
	50…54	Bright green
	55…59	Light green + Lavender
	60…64	Lavender
	65…69	Lavender + Pink
	70…74	Pink
	75…79	Pink + Yellow
	80…84	Yellow
	85…89	Yellow + Magenta
	90…94	Magenta

 Seite 13 ZeMA

Global variables

- Context object
  - Stores data for a node

x =context.get(name)

context.set(name)=x

- Flow object
  - Stores data for a flow

var count=flow.get('count')

flow.set('count',count)

- Global object
  - Stores data for the canvas

var count=global.get('count')

global.set('count',gcount);

Context Variables stored in function 1 are not available to function 2 and vice versa

 Seite 14 ZeMA

LCFC robot's demonstration (UL)

Während der von der Universität Lothringen organisierten Summer School 2019 führten Vianney PAPOT und Mohamed DIDI CHAOUI zwei robotergestützte Schleifversuche mit zwei verschiedenen Schleifsystemen durch.
Der erste Test fand an der ENIM (Nationale Ingenieurschule von Metz) statt. Ziel dieses Tests war, überschüssiges Material von der Oberfläche einer in der Ölindustrie verwendeten Klemme zu entfernen. Dieses Schleifsystem besteht aus einem Roboter und einem Winkelschleifer.
Der zweite Test fand im LCFC (Design, Manufacturing and Control Laboratory) statt. Bei diesem Test handelte es sich um eine Oberflächenbearbeitung eines Metallteils mit einer starren Schleifscheibe.
Das letztgenannte System besteht aus einem ABB IRB 8700 Roboter, einem pneumatischen 1DDL-Stellantrieb und einem Winkelschleifer. Der pneumatische Antrieb ermöglicht die Steuerung der Schleifkraft

Durant le Summer school 2019 Organisé par l'université de Lorraine, Vianney PAPOT et Mohamed DIDI CHAOUI ont réalisé deux essais de meulage robotisé à l'aide de deux systèmes de meulage différents.
Le premier essai a eu lieu à l'ENIM (Ecole nationale d'ingénieurs de Metz). L'objectif de cet essai est d'enlever l'excès de matière sur la surface d'un clamp utilisé dans le domaine pétrolier. Ce système de meulage se compose d'un robot et d'une meuleuse angulaire.
Le deuxième essai a eu lieu au LCFC (Laboratoire de conception, fabrication et commande). L'essai présenté au participant était un surfaçage d'une pièce métallique à l'aide d'un disque de meulage rigide.
Ce dernier système est composé d'un robot ABB IRB 8700, d'un actionneur pneumatique à 1DDL et d'une meuleuse angulaire. L'actionneur pneumatique permet la commande en force du meulage et diminue les vibrations.

und reduziert Vibrationen.
Schleifkraft- und Zylinderpositionsmessungen wurden mit den am Endeffektor des Roboters montierten Kraftsensoren und dem Zylinderpositionssensor durchgeführt.

Kontakt:
Universität Lothringen
Vianny Papot
E-Mail: vianney.papot@univ-lorraine.fr

Mohamed Didi Chaoui
E-Mail: didichaouimohamed@gmail.com

Des mesures de forces de meulage et de positions du vérin ont été réalisé grâce aux capteurs de forces montés à l'effecteur du robot et au capteur de position du vérin .

Contact:
Université de Lorraine
Vianny Papot
e-Mail: vianney.papot@univ-lorraine.fr

Mohamed Didi Chaoui
e-Mail: didichaouimohamed@gmail.com

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

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

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

Thibaud van Rooden
Pôle MecaTech
Telefon: +32 (0)81 20 68 50
E-Mail:
thibaud.vanrooden@polemecatech.be
Webseite: 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

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

Frédérik Cambier
Technifutur
Telefon: + 32 (0)4 382 44 56
E-Mail: frederik.cambier@technifutur.be
Webseite: www.technifutur.be

Ramona Ventimiglia
Universität der Großregion
Telefon: +49 (0)681 301 40801
E-Mail: ramona.ventimiglia@uni-gr.eu
Webseite: www.uni-gr.eu

Anja Höthker
Luxinnovation – National Agency for innovation and research
Telefon: +352 (0)43 62 63 – 858
E-Mail: anja.hoethker@luxinnovation.lu
Webseite: en.luxinnovation.lu

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

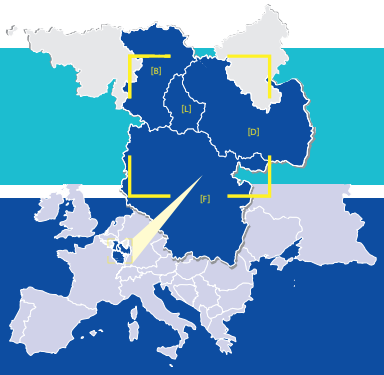
Christian Laurent
Automation & Robotics
Telefon: +32 (0)87 322 330
E-Mail: c.laurent@ar.be
Webseite: www.ar.be

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 Matériaux ; Material, 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