Task Oriented Programming and Service Algorithms for Smart Robotic Cells

Stefano Trapani

Academic Supervisor : **Prof. Marina Indri**
Company Supervisor : **Ing. Rosario Cassano**

Motivation and goals

- Converting standard production lines into smart factories, without changing the initial layout of the line
- **Two approaches** :
  1. Automatic offline programming methodology
  2. Advanced functionalities implemented in standard industrial manipulators

Motivation and goals

- Converting standard production lines into smart factories, without changing the initial layout of the line
- **Two approaches** :
  1. **Automatic offline programming methodology**
  2. Advanced functionalities implemented in standard industrial manipulators

Classic approach

- Sharp skilled programmers with a very high knowledge of the process
- Possible problems are solved by the experience of the programmer (e.g., how to satisfy cycle time constraints, avoid collisions, etc.)
- Time consuming and hence suitable for cells used to perform the same process for long time (low reconfigurability)

Automatic Task-Oriented approach

- More intuitive automatic programming approach
- Soft skilled programmers must specify the features of the robotic cell and a structured set of tasks defining the process (CAD softwares)
- Fast process allowing high level of cell reconfigurability

Motivation and goals

- Converting standard production lines into smart factories, without changing the initial layout of the line
- **Two approaches** :
  1. Automatic offline programming methodology
  2. **Advanced functionalities implemented in standard industrial manipulator**

• Increase the **smartness** of the robotic cell
• Increase the number of **applications**

Motivation and goals

- Converting standard production lines into smart factories, without changing the initial layout of the line
- **Two approaches** :
  1. **Automatic offline programming methodology**
  2. Advanced functionalities implemented in standard industrial manipulator

PART 1
Development of a task-based robot programming approach, that automatizes the programming of a generic robotic cell, providing as a result the work-flow defining the required process

PART 2
Development of a set of service algorithms based on the information already available in standard industrial robots

Task-Oriented Programming - Task analysis

- Definition of a generic task using a minimal set of basic actions
- Analysis of a wide range of industrial applications provided by the main robot constructors in order to find a common set of features

1
- Arc welding
- Cosmetic sealing
- Polishing and deburring
- Laser welding/cutting
- Plasma cutting and water jet
- Spot welding

→ Requiring a path

2
- Machine tending
- Handling
- Processing machining
- Press brake bending
- Interpress
- Foundry

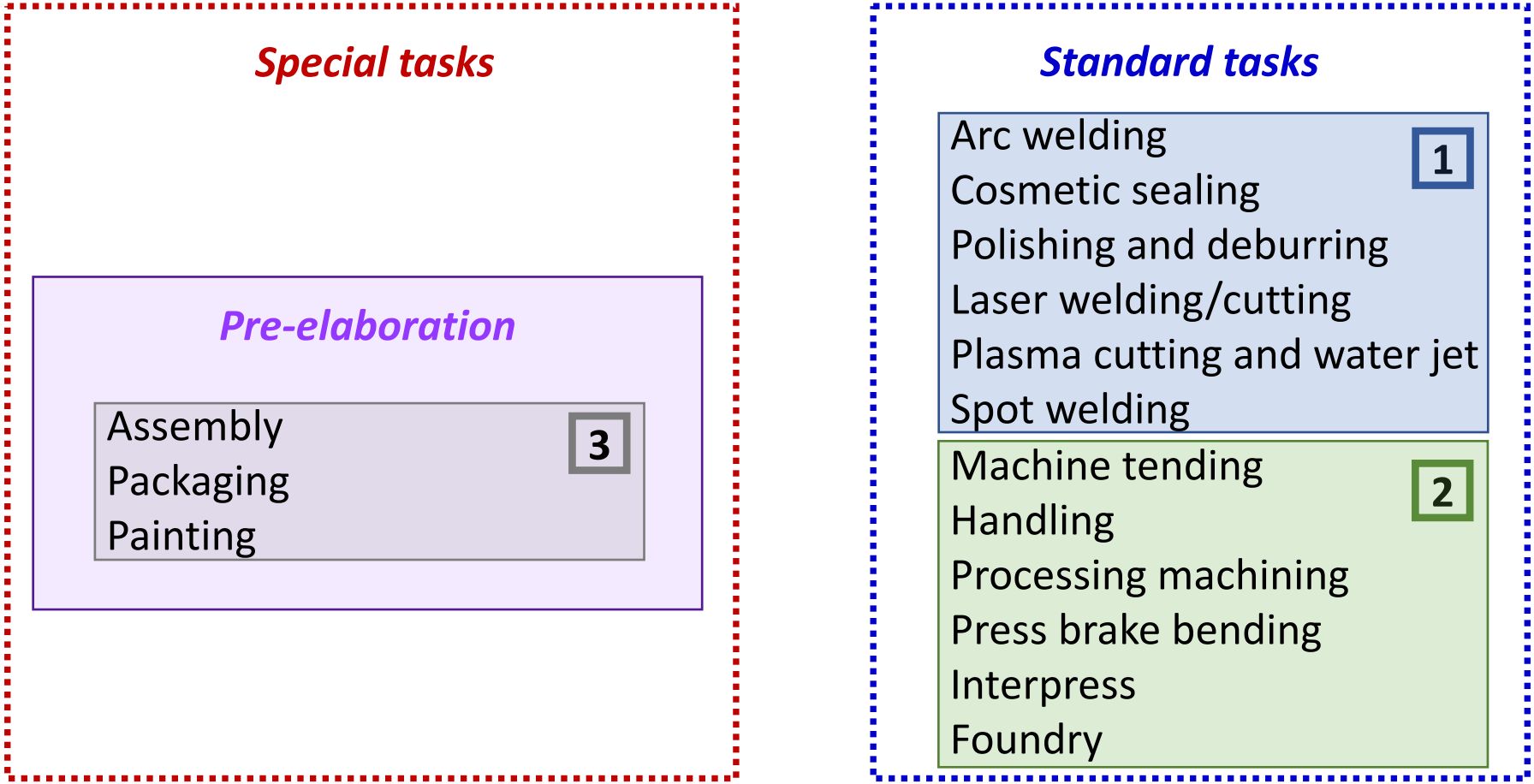
→ Involving only the motion of the work-piece

3
- Assembly
- Packaging
- Painting

→ Needing further elaborations to obtain a work-path

Task-Oriented Programming - Task analysis

- After a proper pre-elaboration the applications belonging to the third class may fall into one of the first two classes
- Two main types of tasks can be defined: Standard tasks and Special tasks



Task-Oriented Programming - Task analysis

- After a proper pre-elaboration the applications belonging to the third class may fall into one of the first two classes
- Two main types of tasks can be defined: Standard tasks and Special tasks
- Only standard tasks are managed by the task model in which
  - the real machines performing a process along a predefined path using a specific tool are denoted as *workers*
  - the real machines equipped with a proper gripper carrying the work-piece from a starting point to a final point are denoted as *positioners*

Standard tasks

Processes carried out on a specific path or devoted to carry the work-piece

Task-Oriented Programming - Approach

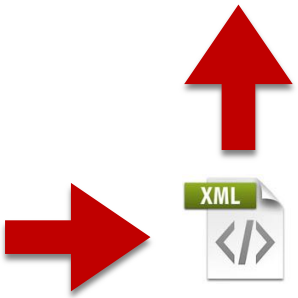
A three-step approach

CAD Design

- Physical definition of the robotic cell (robots, objects, workpieces)
- Definition of the tasks (type, paths, required tools)
- Generation of the description file (e.g., a xml file) used as input of the Task-Oriented Programming module

Task-Oriented Programming

Automatic definition of the sequence of tasks and related path planning of the required process



Deploy into the robotic line

Translation of the output of the Task-Oriented Programming into a program specific for the adopted robot controller

Task-Oriented Programming - Approach

The actually adopted three-step approach

Matlab script → **Task-Oriented Programming** → **ORL simulator**

Matlab script providing an xml file containing the information necessary to run the automatic task-oriented programming tool

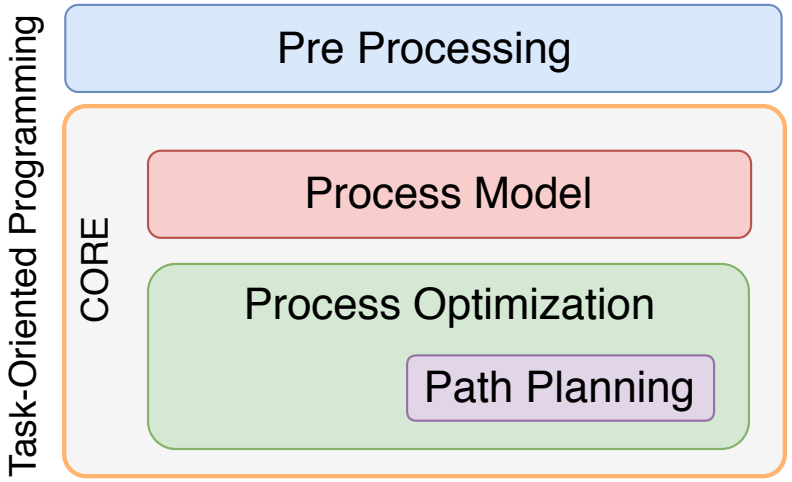
Automatic definition of the sequence of tasks and related path planning of the required process

Simulator of the Comau control system provided by Comau

Task-Oriented Programming - Approach

Task-Oriented Programming

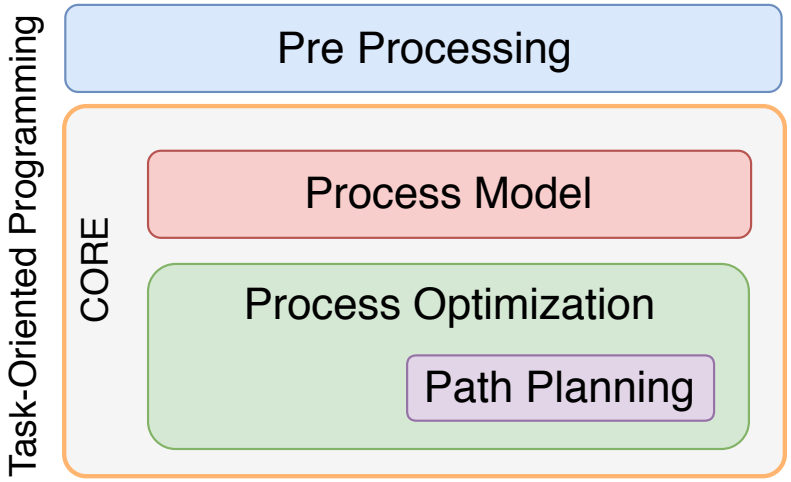
Automatic definition of the sequence of tasks and related path planning of the required process

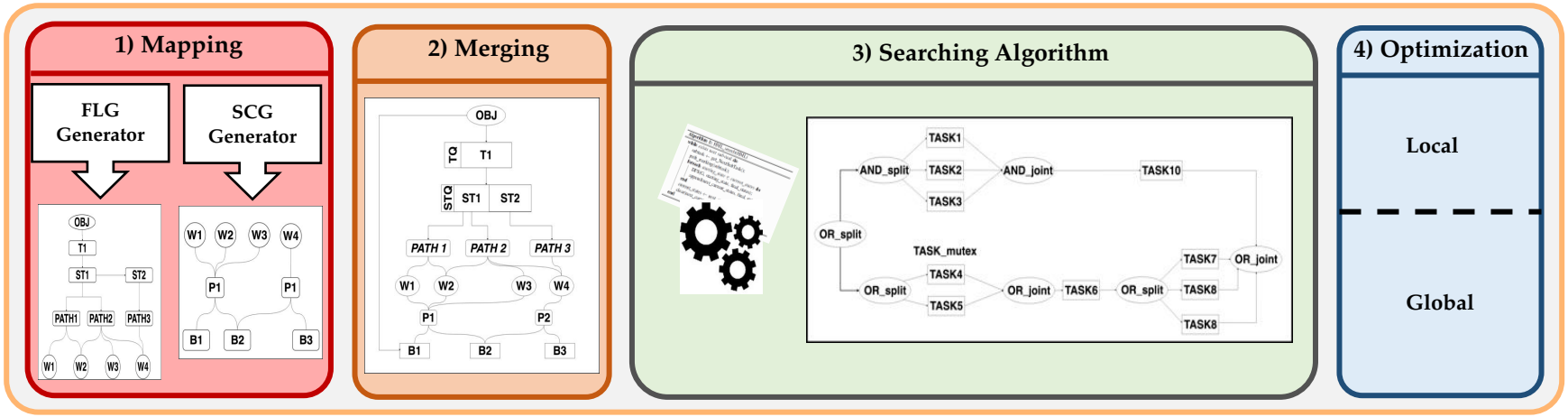


Task-Oriented Programming - Approach

Task-Oriented Programming

Automatic definition of the sequence of tasks and related path planning of the required process

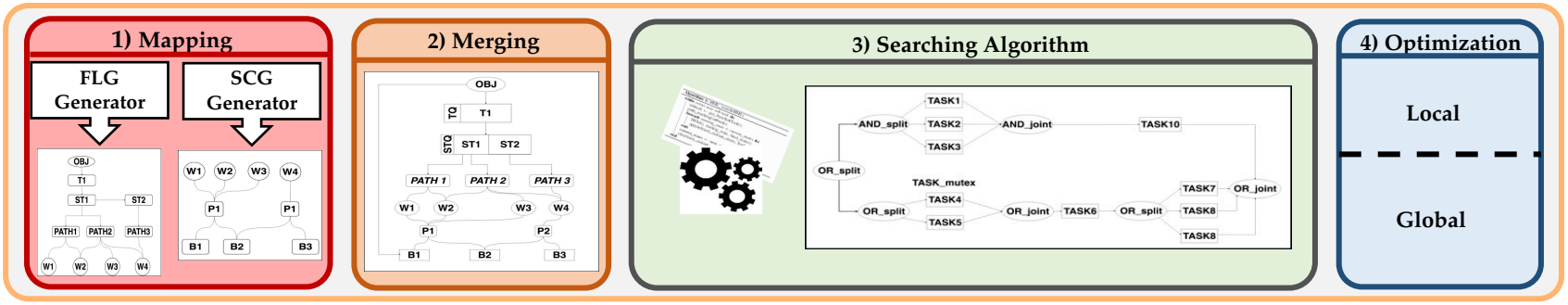




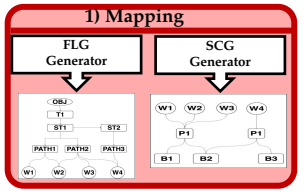
Task-Oriented Programming - Core

The task model-based approach allows to take into account both **physical** constraints and **functional** ones between machineries and tasks

- **Assumption:** task as a sequence of **four basic phases** (or by a subset of them): i) **picking** the workpiece, ii) **positioning** the work-piece within a proper sub-set of the working-area, iii) **working**, iv) **placing** the work-piece
- **Physical constraints**: the adoption of some machineries can be limited because of their location or their physical characteristics; such constraints can be modeled by a proper graph called **Spatial Constraints Graph** (SCG)
- **Functional constraints**: the set of relations between the required tasks and the available machineries (e.g., $task_1$ can be performed by $machine_1$), can be modeled by a proper graph called **Functional Link Graph** (FLG)

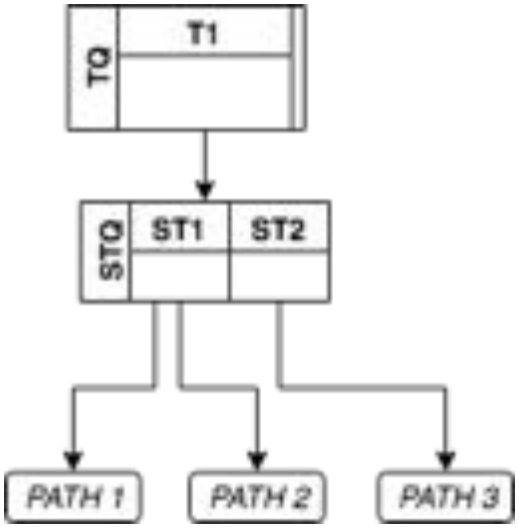


Task-Oriented Programming – Mapping (I)



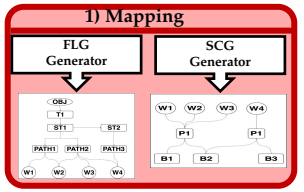
A set of **logical entities** and a three-level task descriptor (Tasks, Sub-Tasks and Paths) have been developed to build the FLG and the SCG within a **mapping** process

Entities	Role
Buffers	Real objects used to store the work-piece
Positioners	Real objects able to grip the work-piece
Workers	Real objects able to perform a specific process
Objects	Real objects that need to be processed
Virtual	Synchronization and physical connections



The **PATH** element corresponds to the **minimum action** which can be executed by a machinery. A complex task can be obtained by a proper sequence of PATHs

Task-Oriented Programming – Mapping (II)

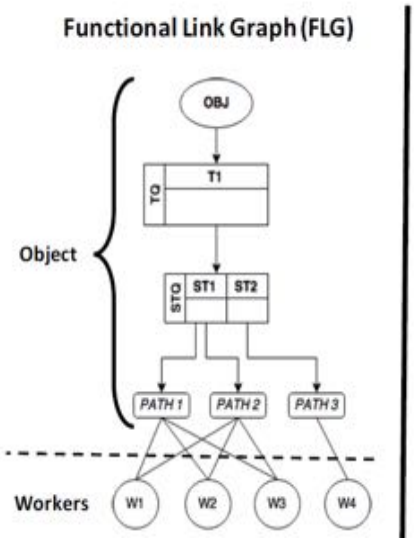


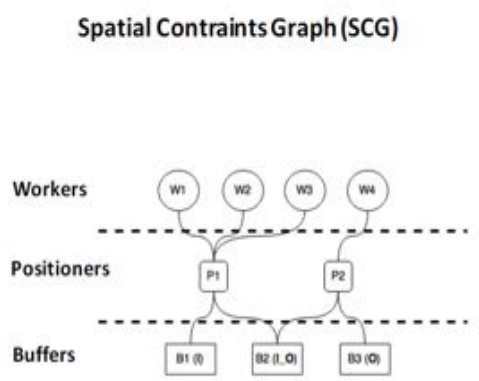
The entities can be connected according to two criteria: 1) the **type of the task** and 2) the **level of possible interaction** between entities

The two models are obtained by mapping the real objects of the robotic cell into the corresponding **logical entities**, and then applying a set of **rules** that allow to create the connections between the entities

FLG rule
A Worker and a PATH element can be linked only if the functional constraints are satisfied

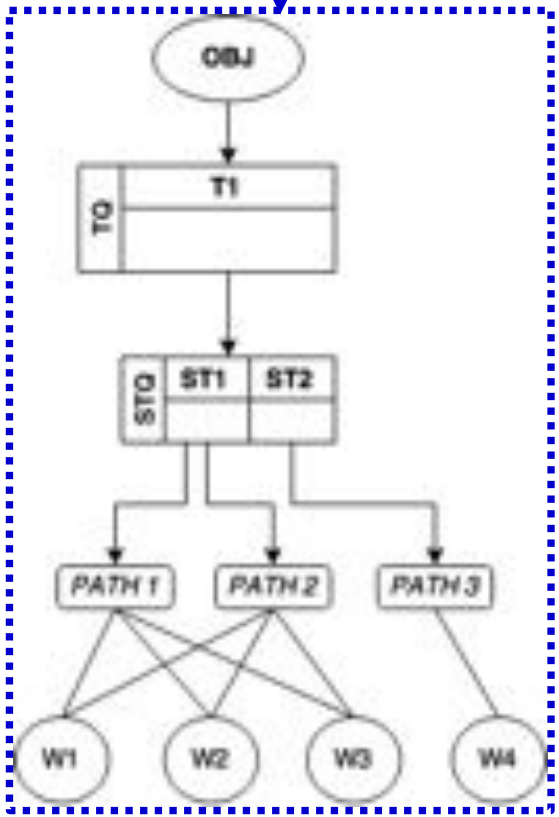
SCG rules
Two entities can be linked only if the spatial constraints are satisfied
Worker can be linked only to Positioner
Buffer can be linked only to Positioner
A pair of entities can be linked only if the spatial constraints are satisfied
Positioners can be linked with other Positioners only if explicitly required





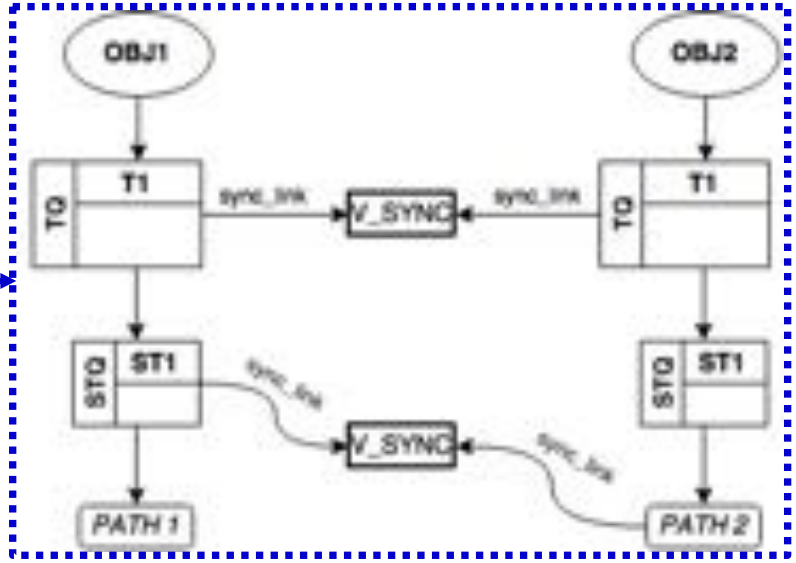
Remarks on FLG

- The **Functional Link Graph** (FLG) defines the possible relations between each PATH element and the available workers
- Different scenarios are taken into account
  1. **Several workers** can perform the **same task**
  2. Some tasks need to be synchronized
  3. The work-piece is physically modified during the process



Remarks on FLG

- The **Functional Link Graph** (FLG) defines the possible relations between each PATH element and the available workers
- Different scenarios are taken into account
  1. Several workers can perform the same task
  2. Some **tasks** need to be **synchronized**
  3. The work-piece is physically modified during the process



Remarks on FLG

- The **Functional Link Graph** (FLG) defines the possible relations between each PATH element and the available workers
- Different scenarios are taken into account
  1. Several workers can perform the same task
  2. Some tasks need to be synchronized
  3. The **work-piece is physically modified** during the process
     - Three possible cases: 1) joining, 2) splitting, 3) none

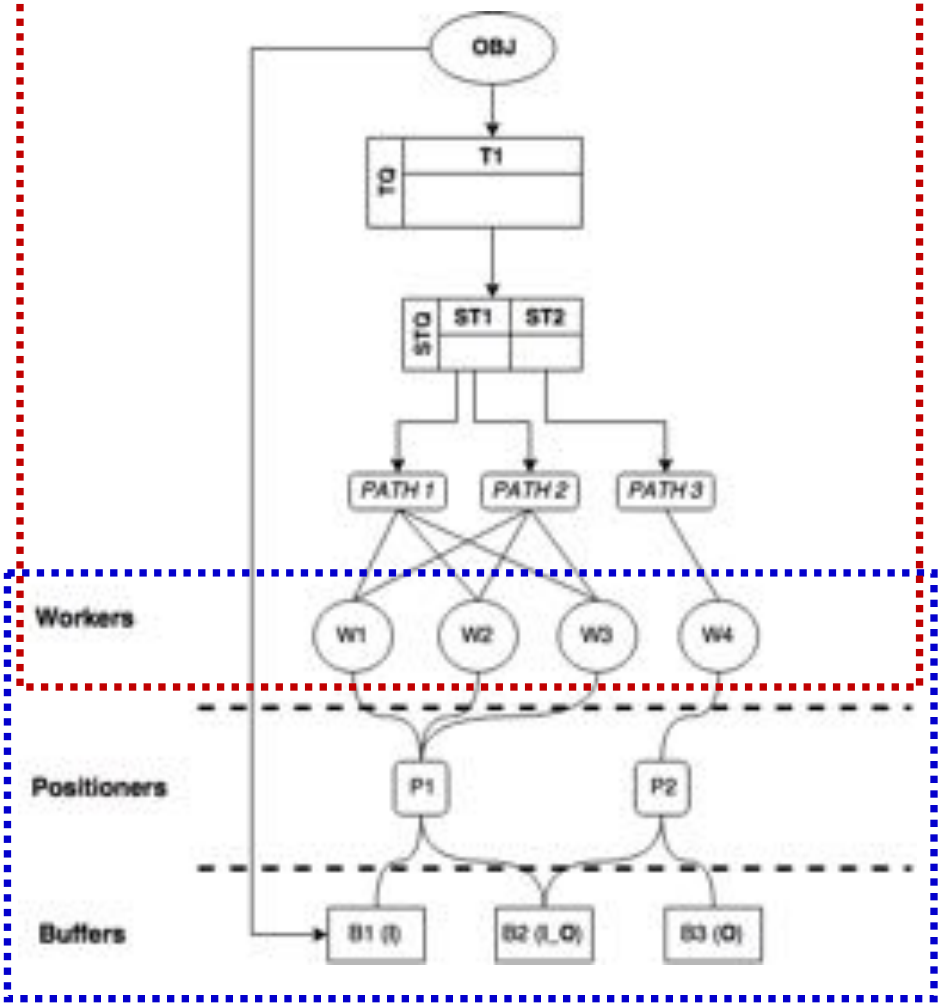
action_type	**effect on the model**
joining	Decreasing of the number of object entities
splitting (rejection)	Number of object entity unchanged
splitting (division)	Increasing of the number of object entities
none	Number of object entity unchanged

Task-Oriented Programming – Merging

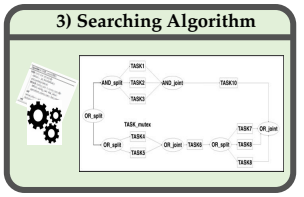
- The **High Level Model** is thus obtained joining the SCG and the n FLGs (one for each object)

FLG →

SCG →



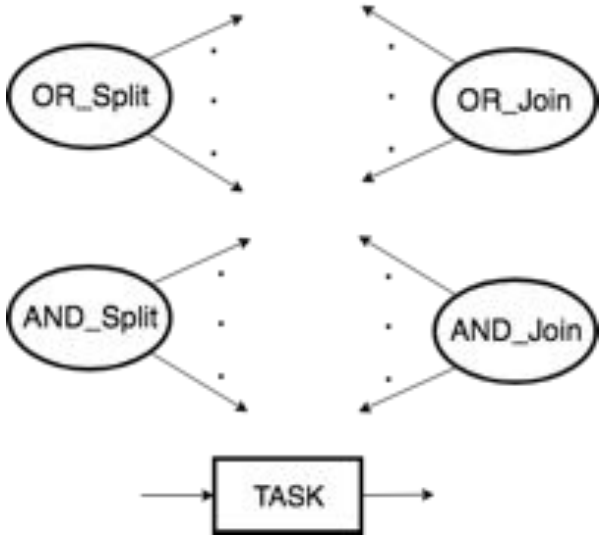
Task-Oriented Programming – Searching Algorithm



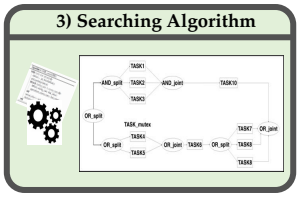
All the possible work-flows carrying out the process are represented using a specific model called **Work Flow Model** (WFL)

The WFM is a recursive model defined by **five basic blocks.** The *TASK* block can be used to combine basic blocks to create a more complex one, thanks to its recursive nature. It is also used to define the basic actions

WFM block	Role
AND_Split	Open a parallel execution
OR_Split	Open a mutual exclusion execution
AND_Join	Close a parallel execution
OR_Join	Close a mutual exclusion execution
TASK	Recursive basic blocks or a basic task



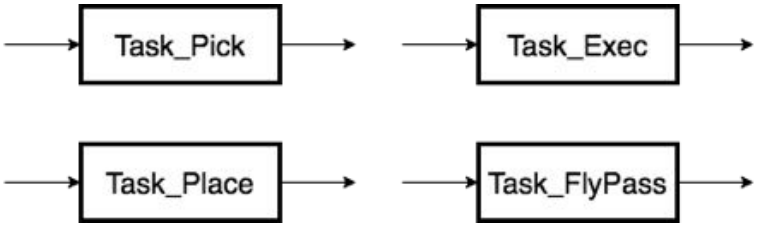
Task-Oriented Programming – Searching Algorithm



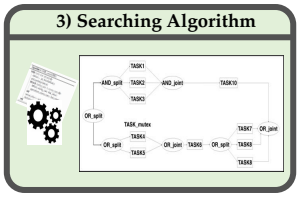
All the possible work-flows carrying out the process are represented using a specific model called **Work Flow Model** (WFL)

The WFM is a recursive model defined by **five basic blocks.** The *TASK* block can be used to combine basic blocks to create a more complex one, thanks to its recursive nature. It is also used to define the basic actions

Basic Tasks	Role
Task_Pick	Picking action
Task_Place	Placing action
Task_Exec	Working carried out by a Worker
Task_FlyPass	Passage of the work-piece



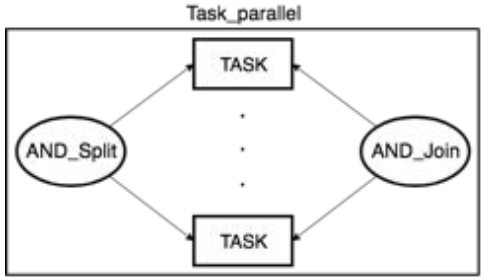
Task-Oriented Programming – Searching Algorithm

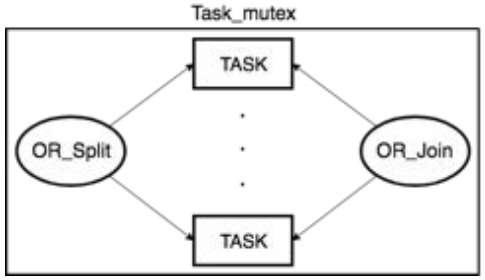


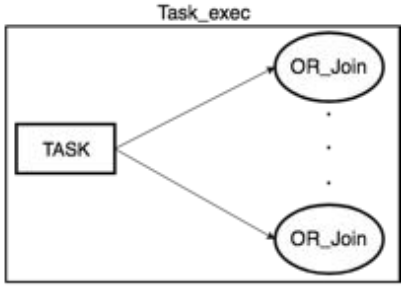
All the possible work-flows carrying out the process are represented using a specific model called **Work Flow Model** (WFL)

The WFM is a recursive model defined by **five basic blocks.** The *TASK* block can be used to combine basic blocks to create a more complex one, thanks to its recursive nature. It is also used to define the basic actions

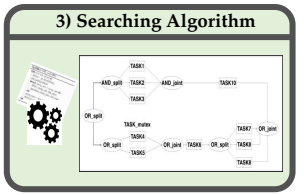
Complex block	Role
TASK_parallel	one AND Split block, one AND Join block and n TASK Blocks
TASK_mutex	one OR Split block, one OR Join block and n TASK blocks
TASK_st_exec	is defined by one TASK block followed by n OR Join blocks





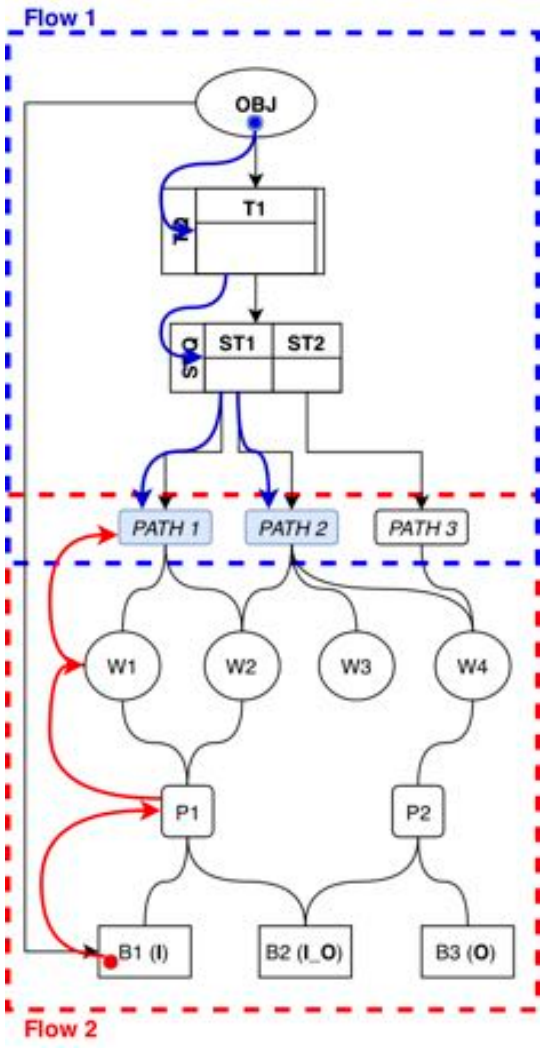


Task-Oriented Programming – Searching Algorithm

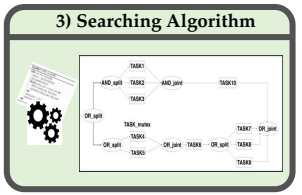


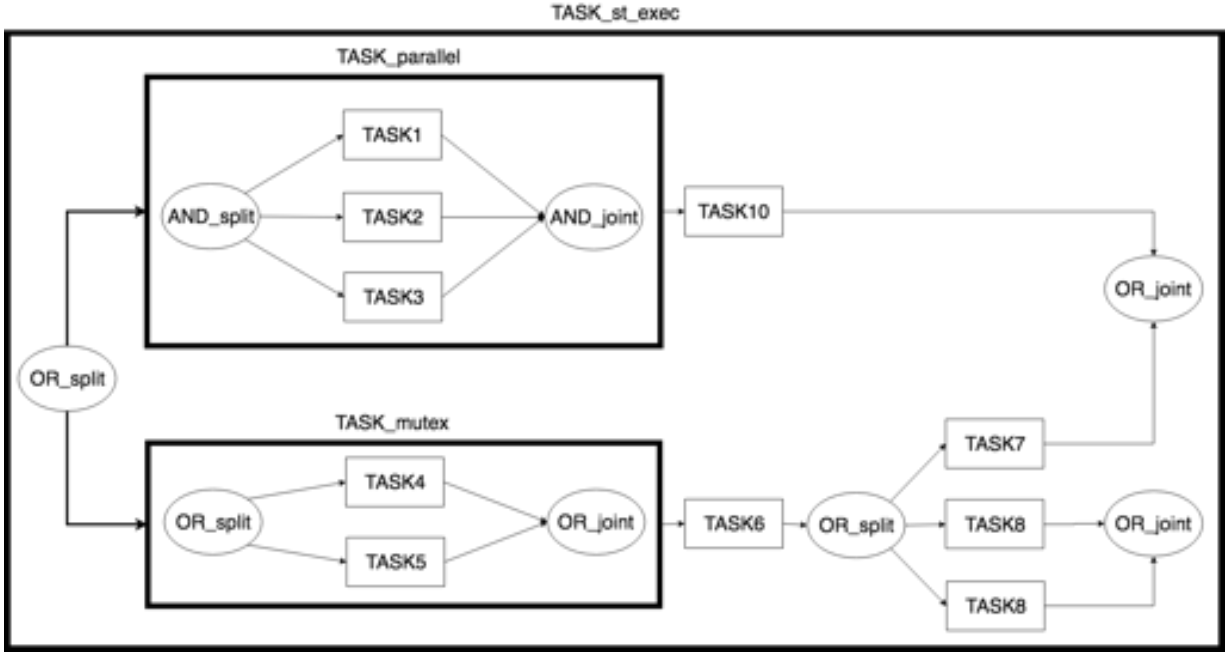
The WFM can be created automatically by applying a specific searching algorithm to the HLM. The algorithm is divided into two flows, the **blue one** that at each iteration selects the next tasks to be performed, and the **red one** that computes all the corresponding work-flows. While scrolling the HLM a set of translation rules are applied in order to build the WFM

#	Transition/condition HML	WFM block
1	Start of the procedure	AND_split
2	Entry to a Sub-Task	TASK_st_exec
3	Transition Buffer → Positioner	TASK_pick
4	Transition Positioner → Buffer	TASK_place
5	Transition Positioner → Positioner	TASK_flypass
6	Entry to a Buffer	OR_split
7	Entry to a Positioner	TASK_parallel n TASK_mutex
7.a	If not all the PATH element can be executed	OR_split
8	Transition Worker → PATH	TASK_exec

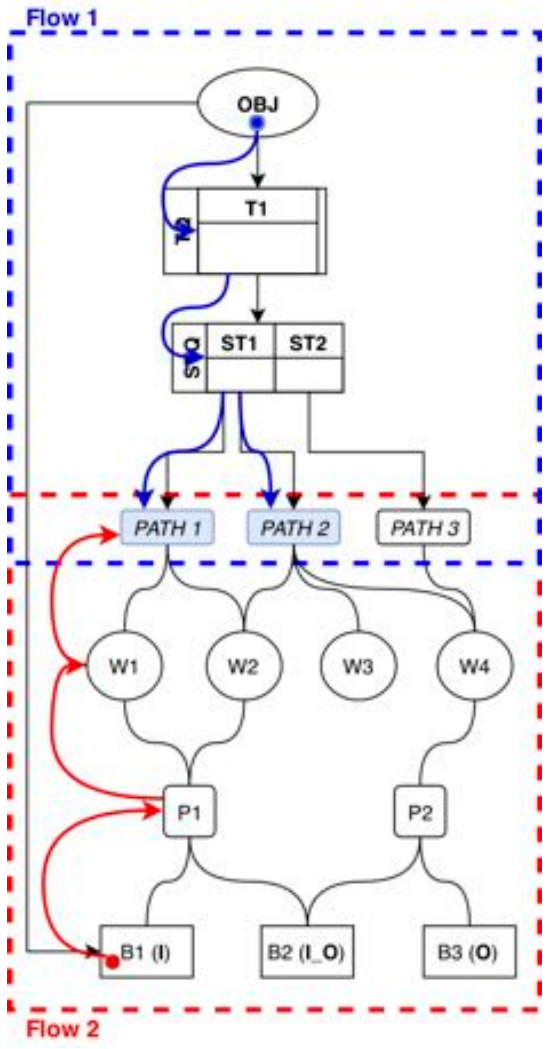


Task-Oriented Programming – Searching Algorithm

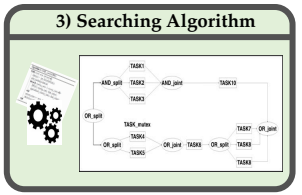


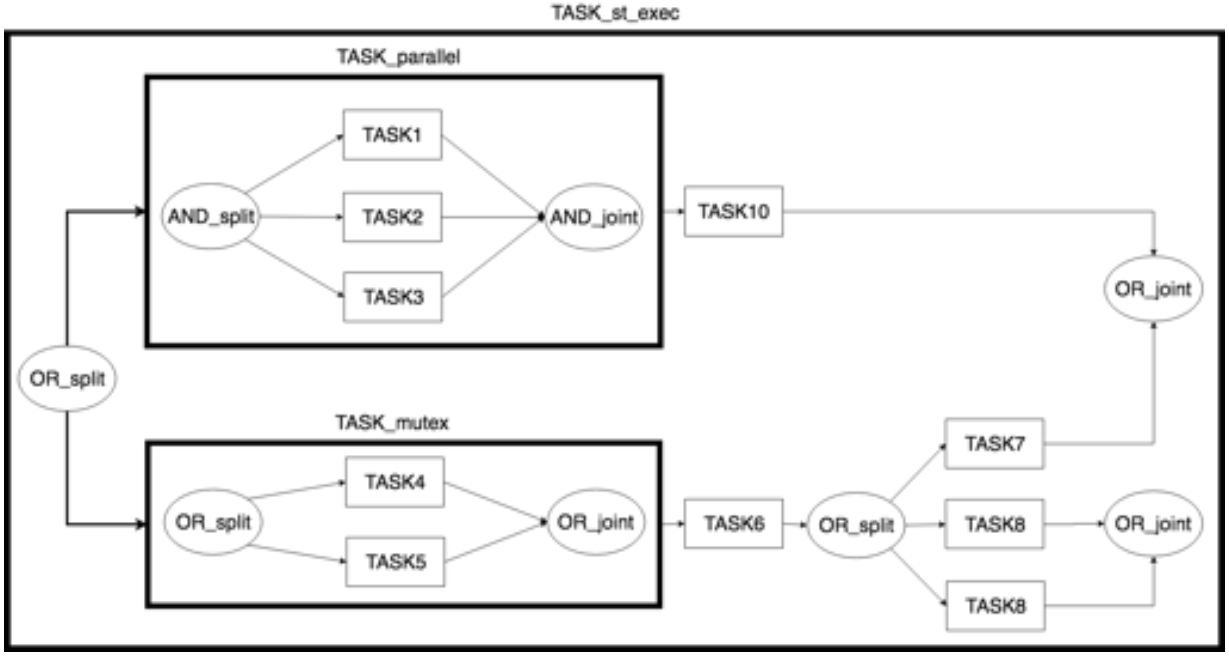


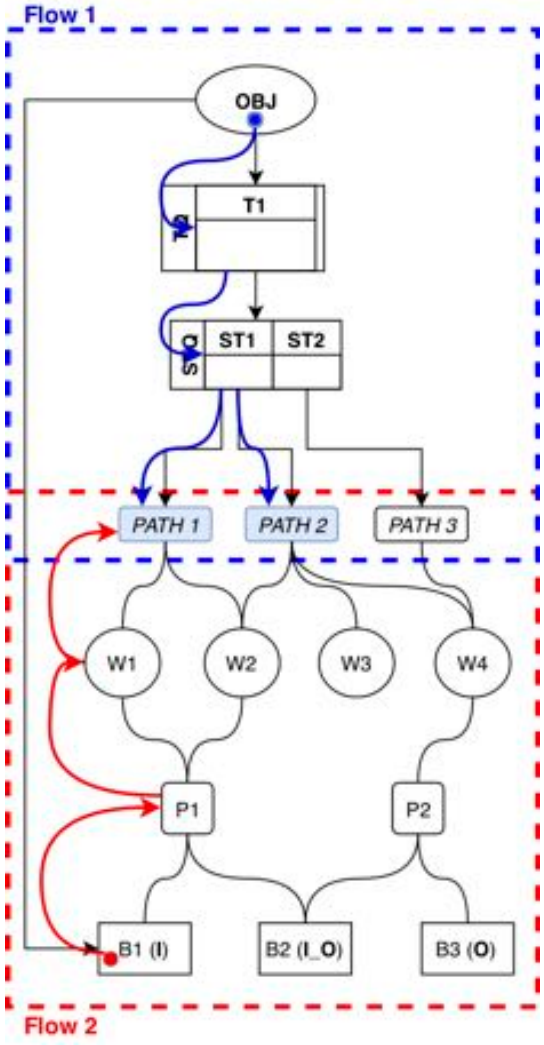
#	Transition/condition HML	WFM block
1	Start of the procedure	AND_split
2	Entry to a Sub-Task	TASK_st_exec
3	Transition Buffer → Positioner	TASK_pick
4	Transition Positioner → Buffer	TASK_place
5	Transition Positioner → Positioner	TASK_flypass
6	Entry to a Buffer	OR_split
7	Entry to a Positioner	TASK_parallel n TASK_mutex
7.a	If not all the PATH element can be executed	OR_split
8	Transition Worker → PATH	TASK_exec



Task-Oriented Programming – Searching Algorithm



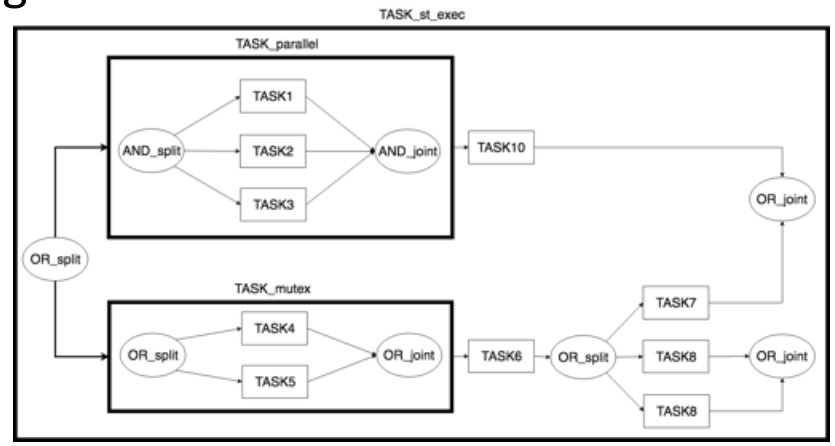




The algorithm is based on a DFS search algorithm, applied on a not oriented connected graph (the SCG corresponds to $g(V,E)$) for k times, where k is the number of PATHs required by the process. The complexity can be approximated as $O(k \cdot n \cdot m)$ where $n = |V|$ and $\mathrm{m} = |E|$

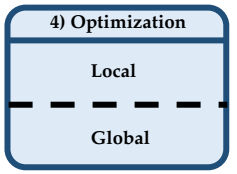
Task-Oriented Programming – Optimization

- **Local Optimization:** the optimization process is not aware to be possibly included in a wider production line
  - Two levels of optimization must be managed:
    - at path planning level (low level)
    - at work-flow level (high level)
  - **Low Level Optimization:** the **path planning** process is carried out for each task of the WFM; such process can include **optimality constraints**



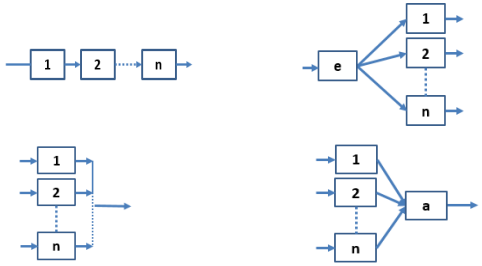
  - **High Level Optimization:** the work-flow at minimal cost can be selected using the output of the Low Level Optimization process to weight the WFM nodes. An AO* algorithm can be used.
- **Global Optimization:** the optimization process takes into account the whole production line in order to achieve a global optimal result. A high level manager is required

Integration with OTE methodology



Integration of the proposed methodology in a general optimization framework based on the usage of Key Performance Indicators (KPI) called **OTE/OEE**, in collaboration with the Università Politecnica delle Marche (UNIVPM)

Fundamental configurations for OTE
series, parallel,
assembly, expansion

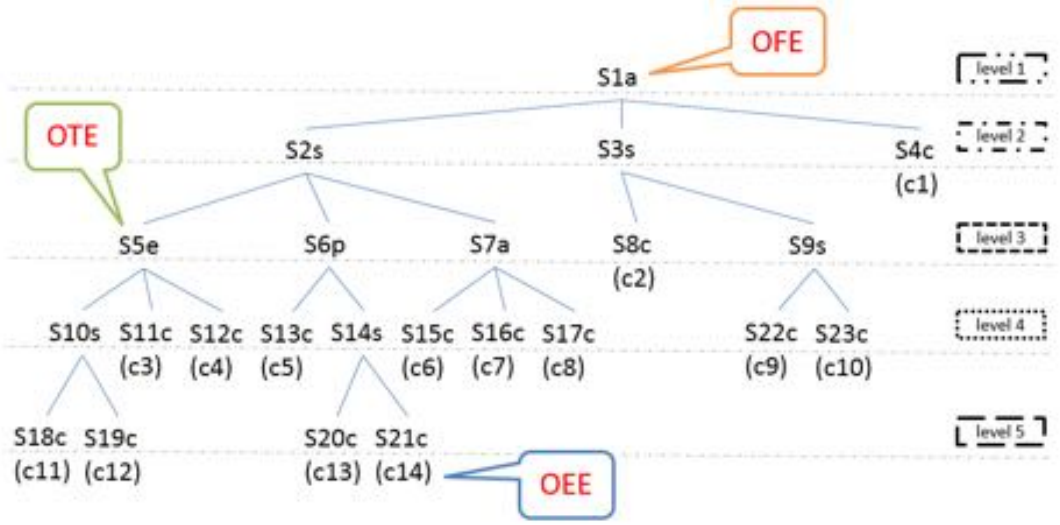


Interpretation of the cells (work unit, job)

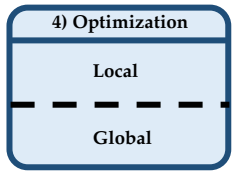
$OEE = A_{eff} \times P_{eff} \times Q_{eff}$

we construct from workflow

Tree structure
having
self-similar nodes

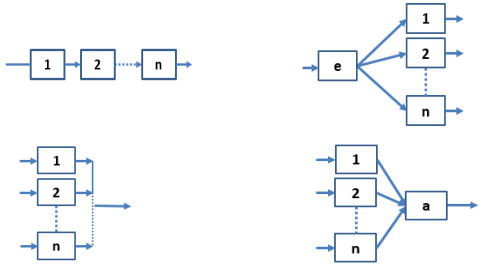


Integration with OTE methodology



Integration of the proposed methodology in a general optimization framework based on the usage of Key Performance Indicators (KPI) called **OTE/OEE**, in collaboration with the Università Politecnica delle Marche (UNIVPM)

Fundamental configurations for OTE
series, parallel,
assembly, expansion



Interpretation of the cells (work unit, job)
$OEE = A_{eff} \times P_{eff} \times Q_{eff}$

Original Interpretation

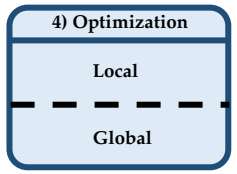
$OEE: A_{eff} \times P_{eff} \times Q_{eff} \rightarrow [0,1]$

***A_{eff}* : Availability**
breakdowns, setup,
adjustmenst, maintenance

***P_{eff}* : Performance**
reduced speed,
idling, stoppages

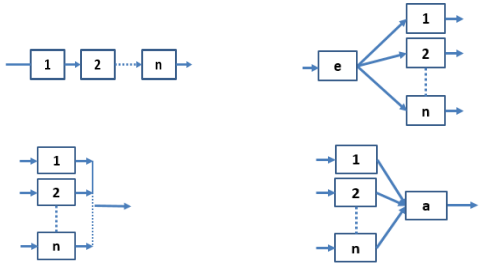
***Q_{eff}* : Quality**
defects, rework, and yield

Integration with OTE methodology



Integration of the proposed methodology in a general optimization framework based on the usage of Key Performance Indicators (KPI) called **OTE/OEE**, in collaboration with the Università Politecnica delle Marche (UNIVPM)

Fundamental configurations for OTE
series, parallel,
assembly, expansion



Interpretation of the cells (work unit, job)
$OEE = A_{eff} \times P_{eff} \times Q_{eff}$

Adopted Interpretation

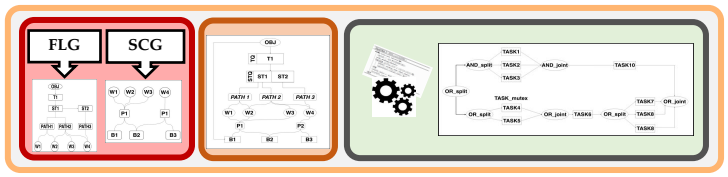
$OEE: A_{eff} \times P_{eff} \times Q_{eff} \rightarrow [0,1]$

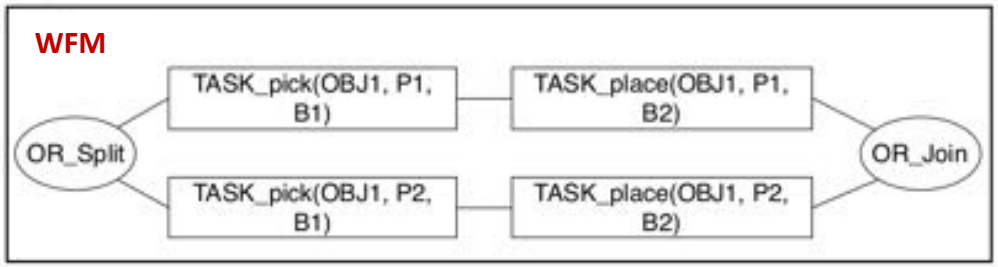
***A_{eff}* : Availability**
faults or stoppings ratio, from probability distributions (MTBF, programmed maintenance, etc.)

***P_{eff}* : Performance**
related to the cycle time required to execute the given task

***Q_{eff}* : Quality**
to take into account the energy consumption

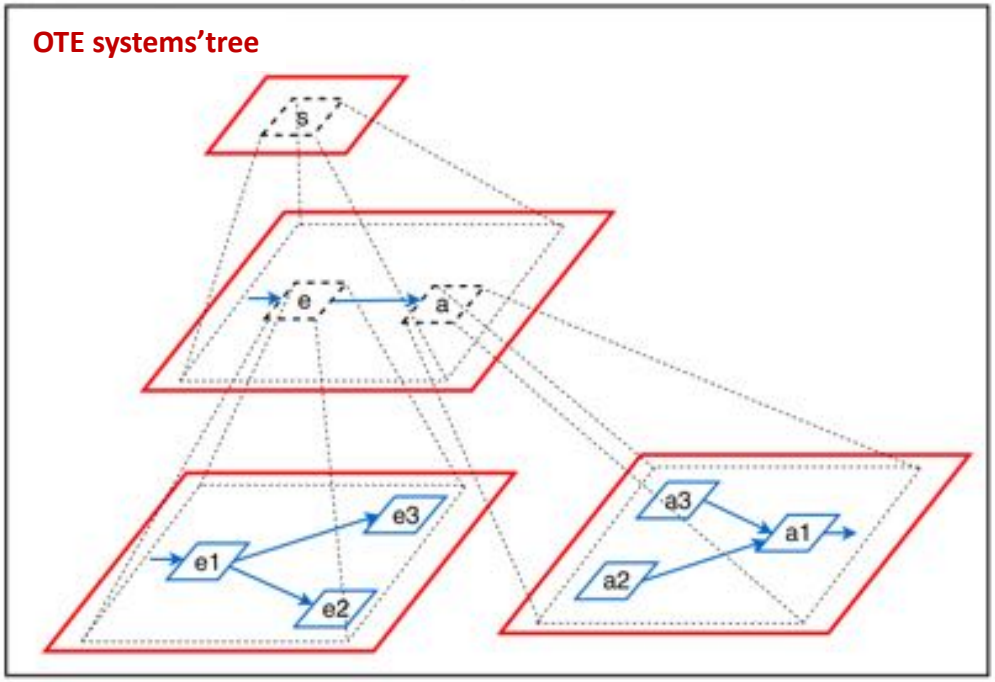
Integration with OTE methodology





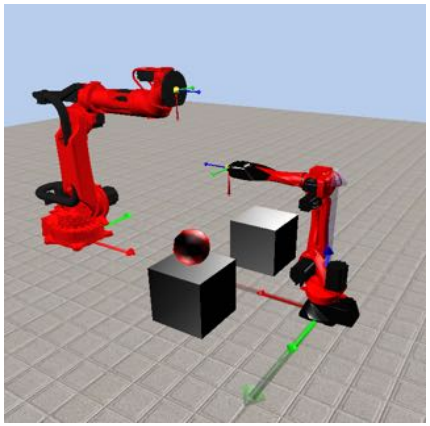
Automatic Conversion Algorithm

MATLAB



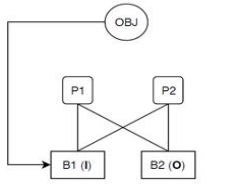
Results on Task-Oriented programming

❑ OTE Methodology

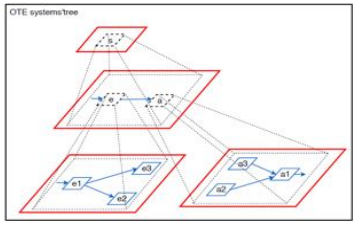


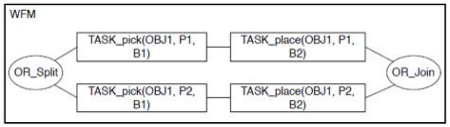
Cell objects	HML entity	HML symbol
ball	Object1	OBJ1
box1	Buffer1 (input)	B1
box2	Buffer1 (output)	B2
Racer 7 - 1.4	Positioner1	P1
NJ4 110 - 2.2	Positioner2	P2

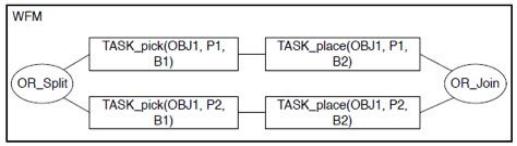
MAPPING BETWEEN REAL OBJECT AND HLM ENTITIES



HLM corresponding to the proposed case study





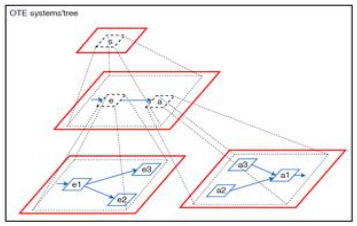


ORL Simulator

A_{eff}, P_{eff}, Q_{eff}

Mapping Table
Motion condition
OEE

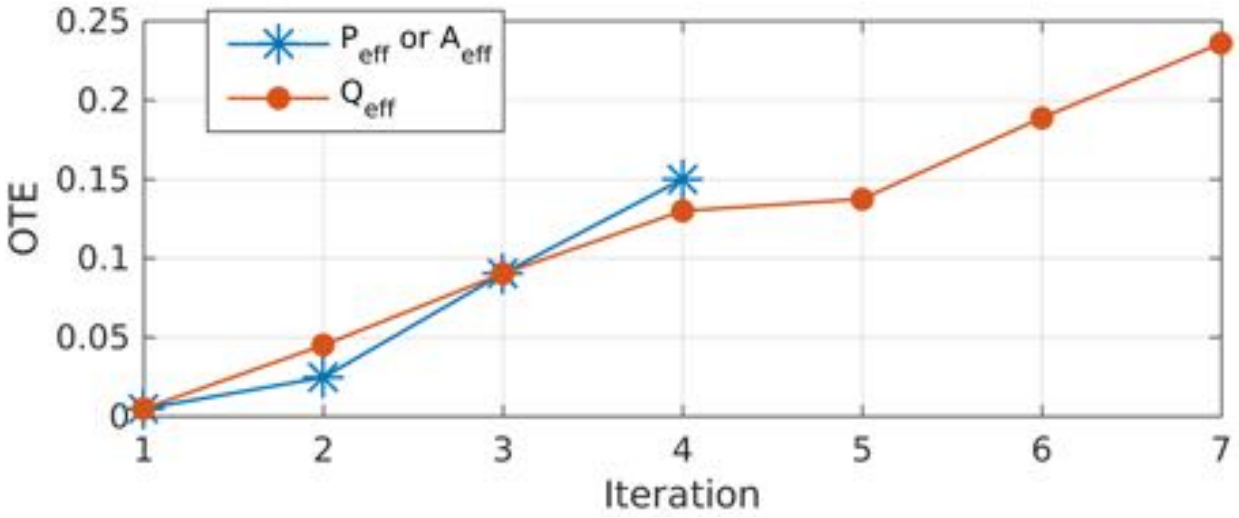
Different motion conditions



Recursive OTE Methodology

Results on Task-Oriented programming

- The improvements are communicated at each iteration
- On each iteration the appropriate suggested OTE is implemented
- Once by favouring solutions for P_{eff} and A_{eff}
- The other by favouring Q_{eff}
- The choice depends on the overall policy in the process
- Both the policies lead to improvements of the process



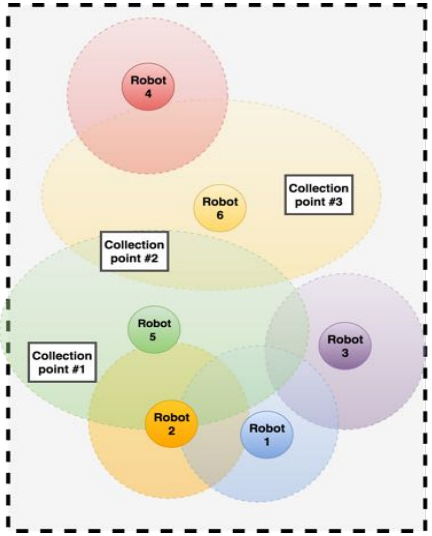
Results on Task-Oriented programming

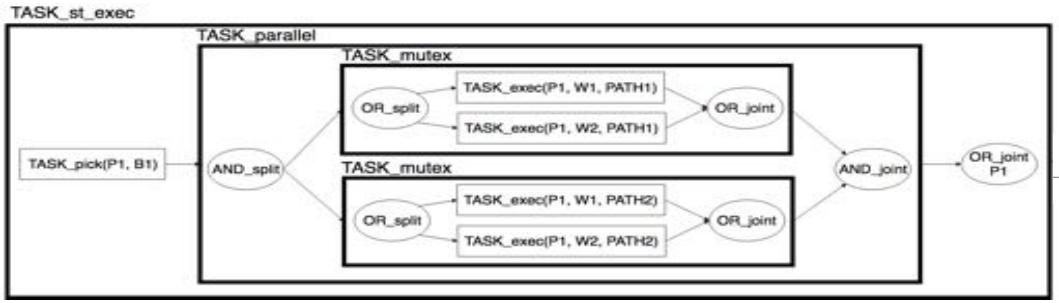
- Software implementation of the Task-Oriented Programming approach (without the optimization step)
- Building of the WFM for some simple but realistic robotic cells (e.g., Pick&Place and welding applications).
- Simulation tests using the Comau robot simulator ORL

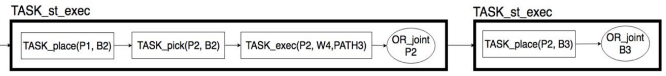
Case Study:

Application - phase1: spot welding process in PATH1 and PATH2; phase2: arc welding process in PATH3; Starting point: Collection point#1; End point: Collection point#3

Robotic cell: i) four robots equipped with a tool compatible with the required tasks, ii) two robots with a gripper suitable to pick the adopted work-piece and iii) three collection points used to place the work-piece when necessary







Motivation and goals

- Converting standard production lines into smart factories, without changing the initial layout of the line
- **Two approaches** :
  1. Automatic offline programming methodology
  2. **Advanced functionalities to be implemented in standard industrial manipulator**

PART 1
Development of a task-based robot programming approach, that automatizes the programming of a generic robotic cell, providing as a result the work-flow defining the required process

PART 2
Development of a set of service algorithms based on the information already available in standard industrial robots

Improve robot dynamic model

- Improve the accuracy of the robot dynamic model

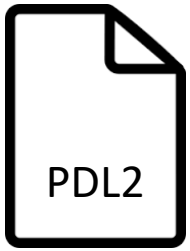
$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + \tau_f(\dot{q}) + g(q) = \tau$$

$$\tau_{res} = \tau - \hat{\tau}$$

- Friction **modelling** and **identification** is a well known problem in robotics, especially at low velocity; its solution allows
  - Improvement of the control performances
  - Enhancement of the robustness of all the applications based on the comparison between the actual motor currents and the estimated ones
- **General framework** for friction identification is proposed
  - Applicable to different manipulators
  - Handling the data acquisition and parameters identification phases
- The friction model is based on a previous **static model** with further improvements to roughly approximate some **dynamic features** of friction
- Extension of the framework for **dynamic friction** model
  - Based on the **LuGre** friction model
  - Improvements of the model for the **industrial context**

Friction Identification Framework – Data Acquisition

User program written using the COMAU programming language



Standard C5G **controller** for COMAU manipulators

Output file containing all the acquired data

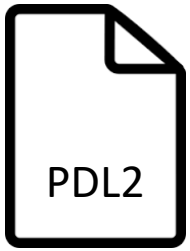
Dynamic model : $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + \tau_f(\dot{q}) + g(q) = \tau$

Rules:

1. Only one joint per time must be moved
2. Movements must be performed in the wider possible position range in order to increase the part of the motion executed at constant velocity
3. The number of measurements at **low velocity** must be a good trade-off between acquisition time and model accuracy

Friction Identification Framework – Data Acquisition

User program written using the COMAU programming language



Standard C5G **controller** for COMAU manipulators

Output file containing all the acquired data

Dynamic model : $\cancel{M(q)\ddot{q}} + \cancel{C(q,\dot{q})\dot{q}} + \tau_f(\dot{q}) + \cancel{g(q)} = \tau$

Rules:

1. Only one joint per time must be moved
2. Movements must be performed in the wider possible position range in order to increase the part of the motion executed at constant velocity
3. The number of measurements at **low velocity** must be a good trade-off between acquisition time and model accuracy

Friction Identification Framework – Data Acquisition

User program written using the COMAU programming language

PDL2

Standard C5G **controller** for COMAU manipulators

Output file containing all the acquired data

Moni

Dynamic model : $\underset{②}{\cancel{M(q)\ddot{q}}} + \underset{①}{\cancel{C(q,\dot{q})\dot{q}}} + \tau_f(\dot{q}) + \cancel{g(q)} = \tau$

A subsequent **cleaning** phase allows to remove the gravity component

Rules:

① Only one joint per time must be moved

② Movements must be performed in the wider possible position range in order to increase the part of the motion executed at constant velocity

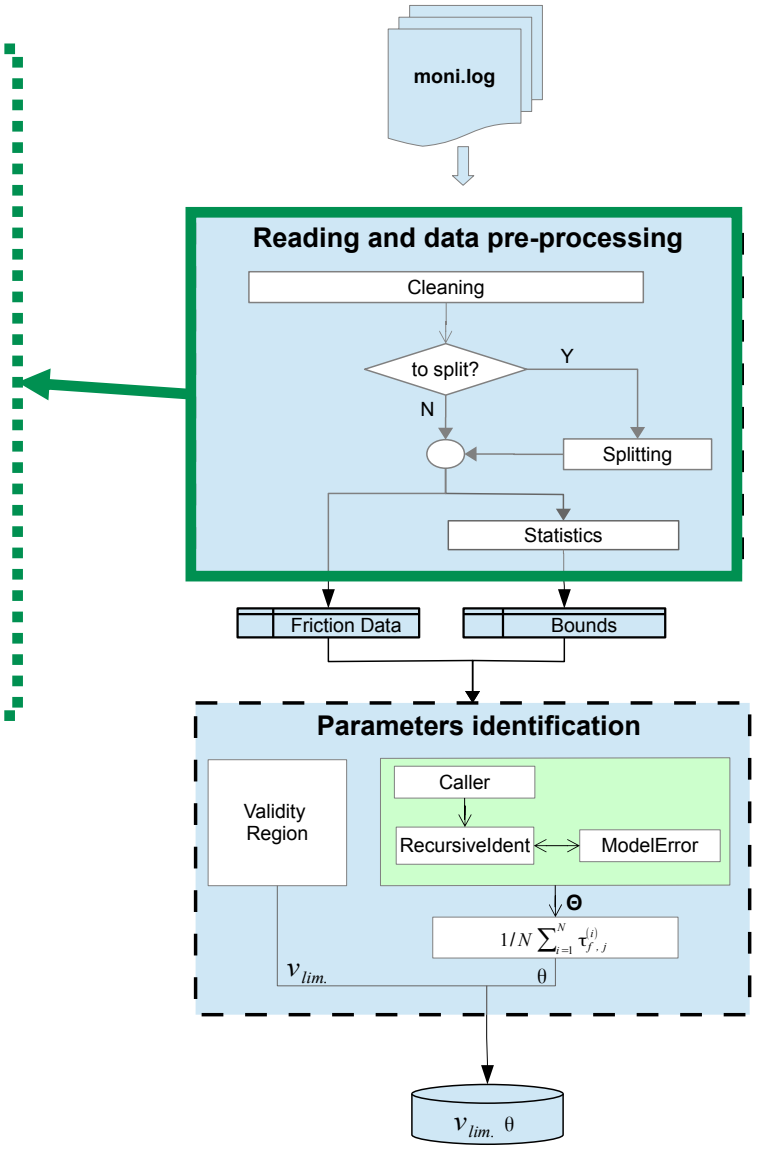
③ The number of measurements at **low velocity** must be a good trade-off between acquisition time and model accuracy

Friction Identification Framework - Identification

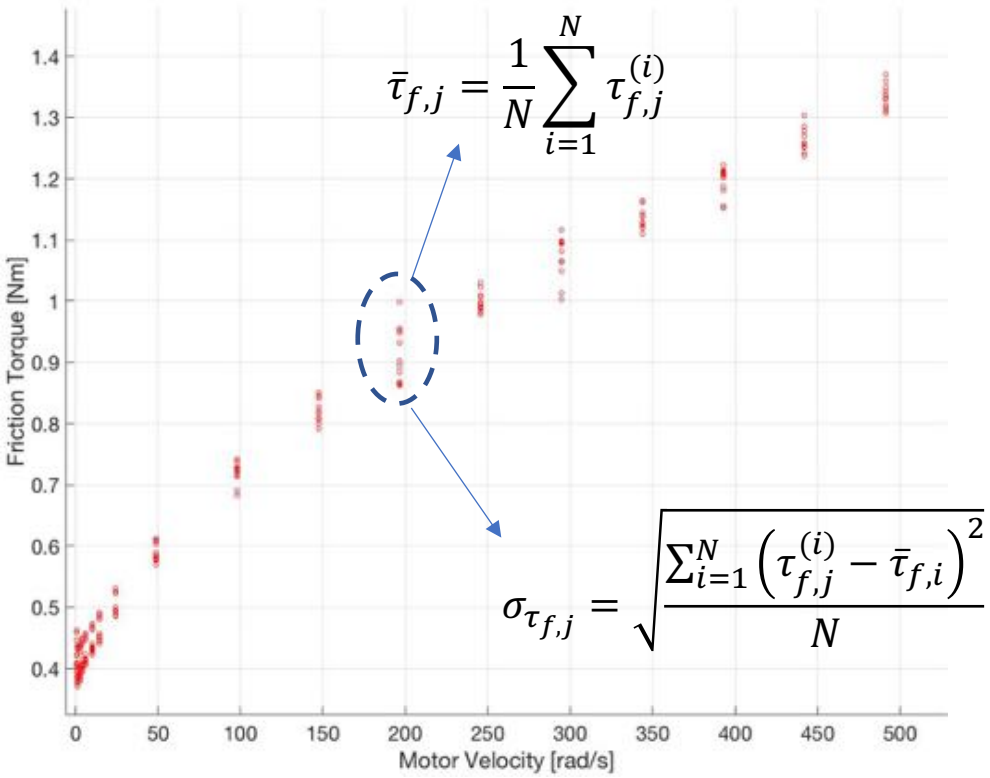
Cleaning the data in order to eliminate **gravity** contribution and to select only data acquired at **constant velocity**

Splitting the data in order to standardize input format

Computation of **statistical information** which will be used to define feasible **bounds** of the friction torques

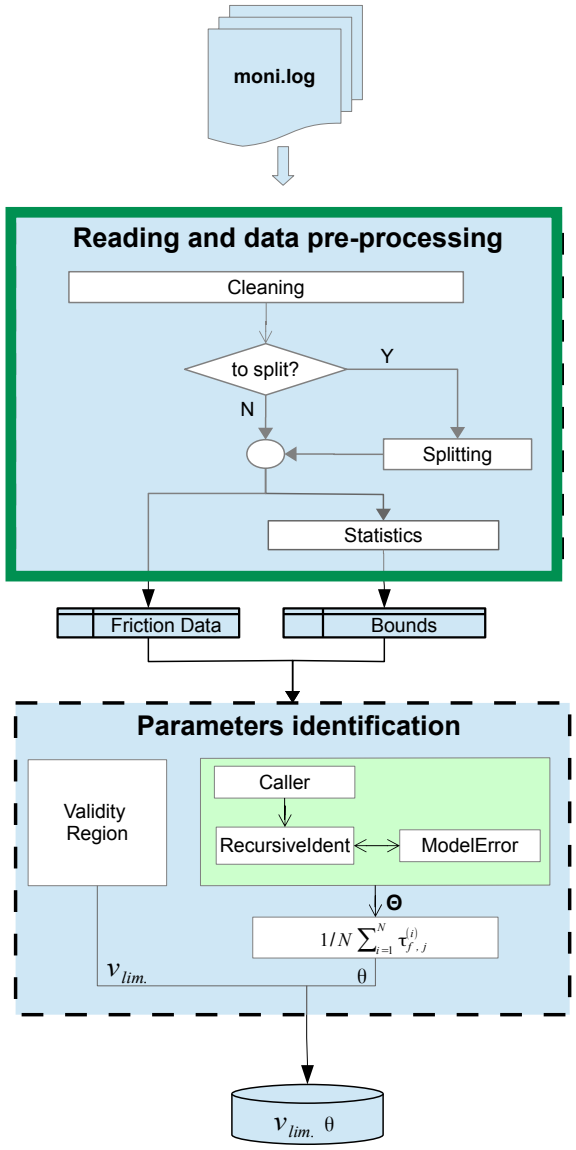


Friction Identification Framework - Identification



$$\bar{\tau}_{f,j} = \frac{1}{N}\sum_{i=1}^{N} \tau_{f,j}^{(i)}$$

$$\sigma_{\tau_{f,j}} = \sqrt{\frac{\sum_{i=1}^{N}\left(\tau_{f,j}^{(i)} - \bar{\tau}_{f,i}\right)^2}{N}}$$

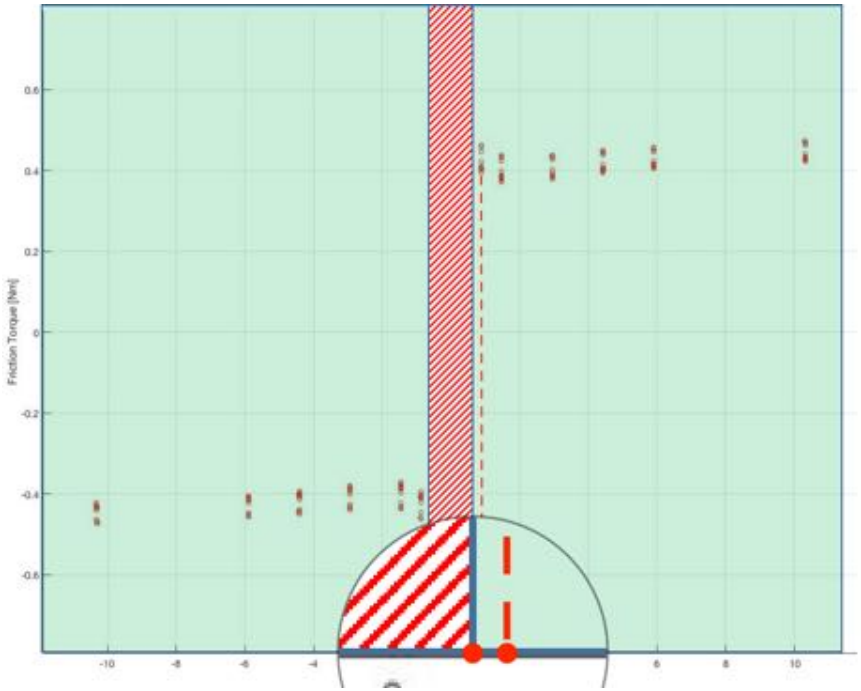


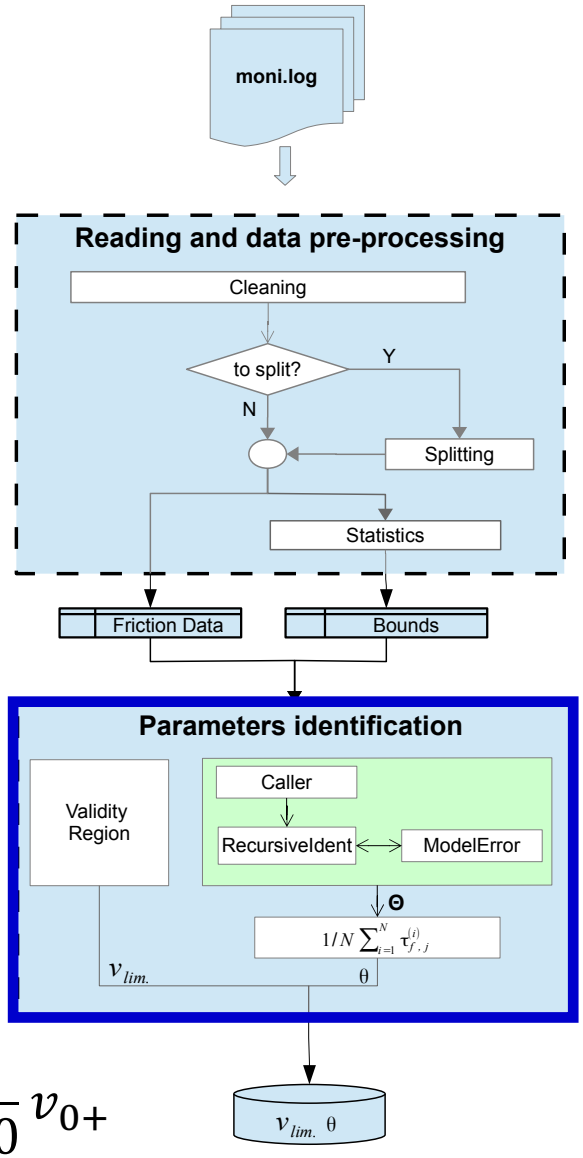
$$b(v_j) = \bar{\tau}_{f,j} \pm k \cdot max\left(\left[\sigma_{\tau_{f,1}}, \sigma_{\tau_{f,2}}, \ldots, \sigma_{\tau_{f,m}}\right]\right)$$

Feasibility of friction values by means of a pair of **bounds** for each velocity v_j

Friction Identification Framework - Identification

Definition of the set of velocities in which the **validity** of the model is guaranteed





$$VR \in \{v: v_{lim} \leq v \leq v_{max+}, v > 0\} \cup \{v: v_{max-} \leq v \leq -v_{lim}, v < 0\}$$

$$v_{lim} = \frac{p}{100} v_{0+}$$

Friction Identification Framework - Identification

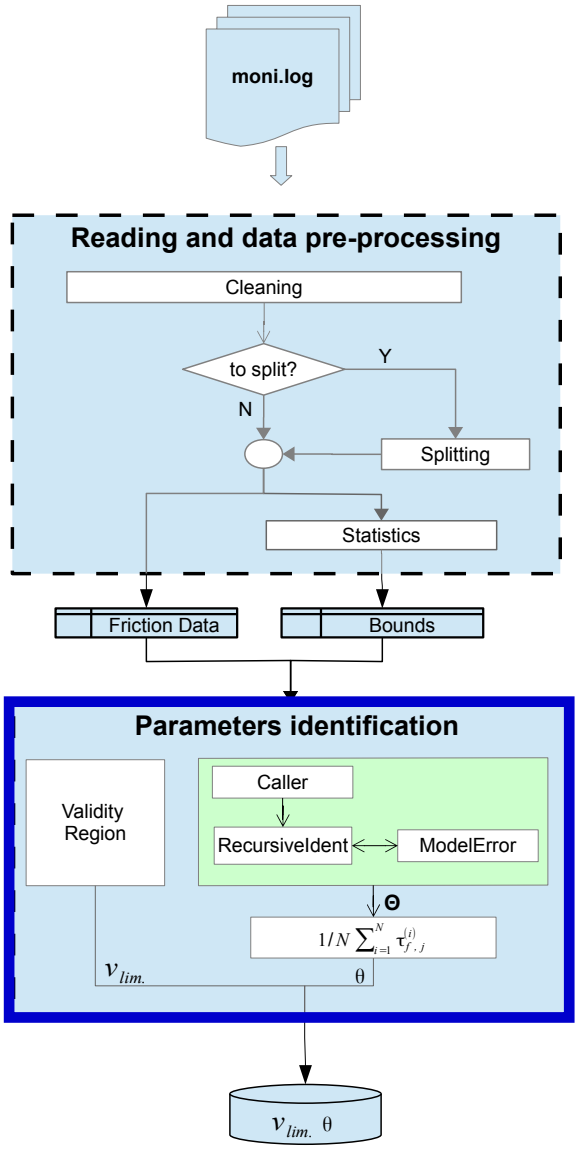
Identification of the friction parameters

$$\tau_f(v) = \boldsymbol{\tau_s}\frac{\pi}{2}\arctan(v\,K_v) + \boldsymbol{\tau_{sc}}\frac{\pi}{2}\arctan(v\,\boldsymbol{\delta}) + \boldsymbol{\tau_v}\,v + \boldsymbol{\tau_{nlv}}v^2\frac{\pi}{2}\arctan(v\,K_v)$$

Assigning a **known** value to δ the identification problem turns into a Linear In Parameters (LIP) problem, which can be solved using **Least Squares** (LS) method

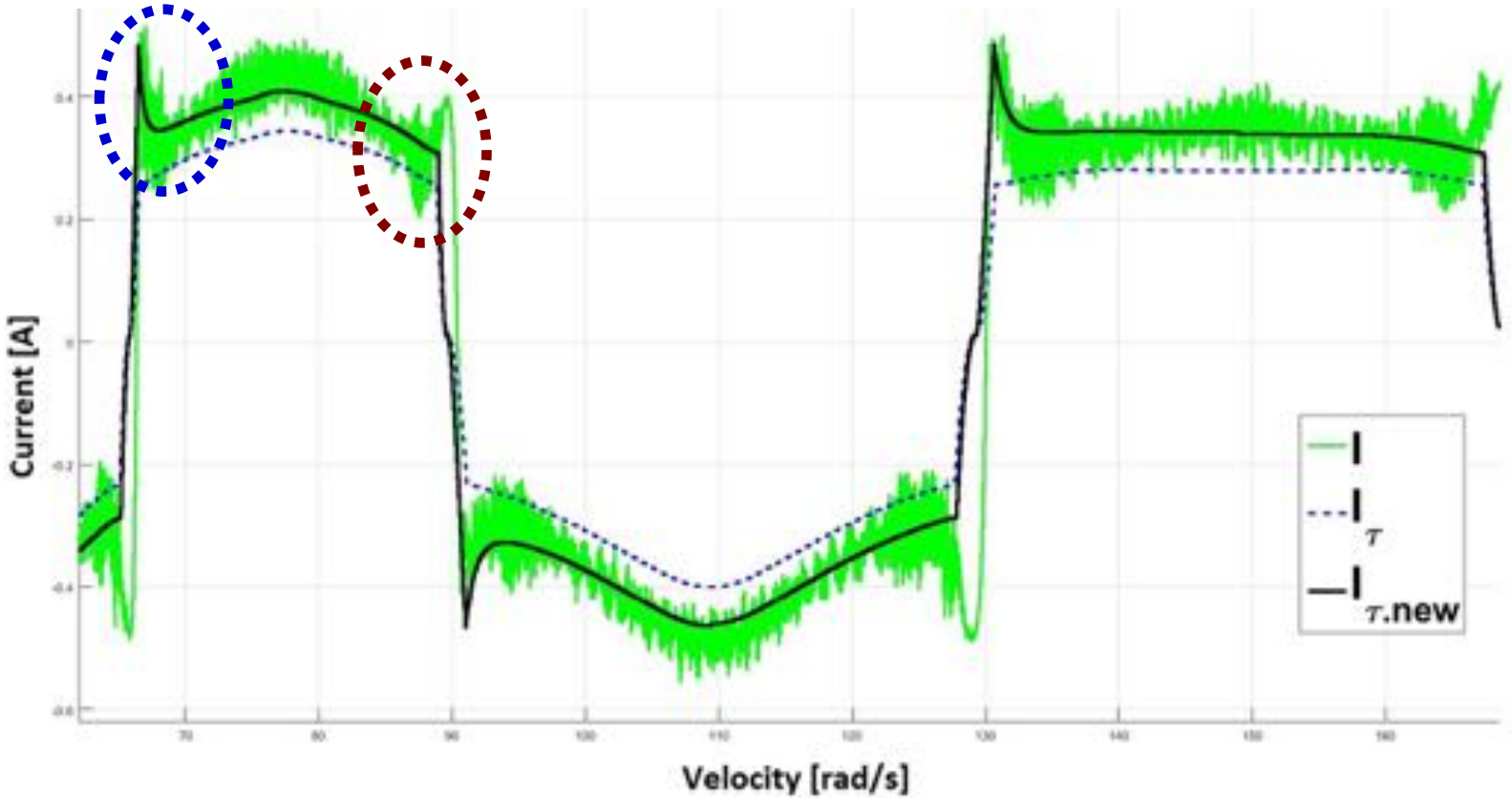
$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\boldsymbol{T}}\right)^{-\mathbf{1}}\boldsymbol{\Phi}\,\boldsymbol{T_f}$$

The optimization is based on the computation of the LS algorithm for each value of δ between δ_{min} and δ_{max} with step ε



Friction Identification Framework – Modified Model

$$\tau_f(v) = \tau_s \frac{\pi}{2} \arctan(v\, K_v) + \tau_{sc} \frac{\pi}{2} arctan(v\, \delta) + \tau_v\, v + \tau_{nlv} v^2 \frac{\pi}{2} arctan(v\, K_v)$$



Friction Identification Framework – Modified Model

Two different **models** are computed for **acceleration** and **deceleration** phases
A proper **filtering action** manages the values provided by the two models

$$\tau_{fa}(v) = f_{lim}\left(\tau_s \frac{\pi}{2}\arctan(v\,K_v) + \tau_{sc}\frac{\pi}{2}\arctan(v\,\delta) + \tau_v\,v + \tau_{nlv}v^2\frac{\pi}{2}\arctan(v\,K_v)\right)$$

$$\tau_{fb}(v) = f_{lim}\left(\tau_s \frac{\pi}{2}\arctan(v\,K_v) + \tau_{sc}\frac{\pi}{2}\arctan(v\,1000\delta) + \tau_v\,v + \tau_{nlv}v^2\frac{\pi}{2}\arctan(v\,K_v)\right)$$

A **limiting function** is defined using v_{lim}
The limitation is active only for velocities outside VR
The typical **torque peak** at low velocity is cut during decelerations, in order to roughly reproduce the hysteretic behavior of friction.

$$f_{lim}(v) = \begin{cases} 1, & |v| \geq v_{lim} \\ \left|\frac{v}{v_{lim}}\right|, & |v| < v_{lim} \end{cases}$$

Friction Identification Framework - Results

Experimental tests were performed using a standard **COMAU Smart NS12**
Comparison are carried out using two performance indices: the Root Mean Square Error (*RMSE*) and the Mean Value (*MV*)

Current modality vs **Residue** modality

Joint	$RMSE_{Res}$	$RMSE_{Cur}$	MV_{Res}	MV_{Cur}
1	13.2228	18.9079	0.0666	0.0952
2	31.1739	40.3065	0.1593	0.2181
3	22.3581	24.7534	0.1155	0.1514
4	17.7277	12.6770	0.0968	0.0775
5	8.62904	11.5295	0.0470	0.0655
6	24.2024	29.3014	0.1215	0.1789

The best results are obtained for the third joint in the first test , with a reduction of 39% for RMSE and 28% for MV

Comparision between **original** model and **new** one

Joints individually moved at increasing velocity

joint	$RMSE\ I_\tau$	$RMSE\ I_{\tau,new}$	$MV\ I_\tau$	$MV\ I_{\tau,new}$
1	67.8740	35.9962	0.1892	0.0703
2	228.3464	128.6238	0.5864	0.3149
3	117.5564	46.3295	0.3438	0.0998
4	27.1074	28.4001	0.0671	0.0698
5	59.6237	54.4892	0.1138	0.0780
6	90.3507	64.3548	0.2376	0.0846

All joint are simultaneously moved at low velocity

joint	$RMSE\ I_\tau$	$RMSE\ I_{\tau,new}$	$MV\ I_\tau$	$MV\ I_{\tau,new}$
1	30.3246	13.2228	0.1758	0.0666
2	70.5386	31.1739	0.4269	0.1593
3	32.6806	22.3581	0.2052	0.1155
4	17.0120	17.7277	0.1106	0.0968
5	16.1850	8.6290	0.1081	0.0470
6	53.9987	24.2024	0.3638	0.1215

Payload Check - Approach

- The forces due to a wrong definition of the robot dynamic model parameters like the payload are computed

Inverse static equation

$$F = \left(\mathbf{J}^{T}(q)\right)^{-1} \tau_{res}$$

Residual torque

$$\boldsymbol{\tau}_{res} = \boldsymbol{\tau} - \hat{\boldsymbol{\tau}}$$

Check of the matrix **condition number**

Robot must be far from **singularities**

Payload Check - Approach

- The forces due to a wrong definition of the robot dynamic model parameters like the payload are computed

Inverse static equation

$$\boldsymbol{F} = \left(\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{q})\right)^{-1} \boldsymbol{\tau_{res}}$$

$$\boldsymbol{F} = [f_x \quad f_y \quad \boldsymbol{f_z} \quad N_x \quad N_y \quad N_z]$$

Check of the matrix **condition number**

Robot must be far from **singularities**

Residual torque

$$\boldsymbol{\tau}_{\boldsymbol{res}} = \boldsymbol{\tau} - \hat{\boldsymbol{\tau}}$$

$$Payload\_error = \frac{\boldsymbol{f_z}}{g}$$

Gravity acceleration

Payload Check - Approach

- The forces due to a wrong definition of the robot dynamic model parameters like the payload are computed

Inverse static equation

$$\boldsymbol{F} = \left(\mathbf{J}^{T}(\boldsymbol{q})\right)^{-1} \boldsymbol{\tau_{res}}$$

$$\boldsymbol{F} = [f_x \quad f_y \quad \boldsymbol{f_z} \quad N_x \quad N_y \quad N_z]$$

Residual torque

$$\boldsymbol{\tau}_{res} = \boldsymbol{\tau} - \hat{\boldsymbol{\tau}}$$

$$Payload\_error = \frac{\boldsymbol{f_z}}{g}$$

Check of the matrix **condition number**

Robot must be far from **singularities**

Ideal conditions f_z is constant

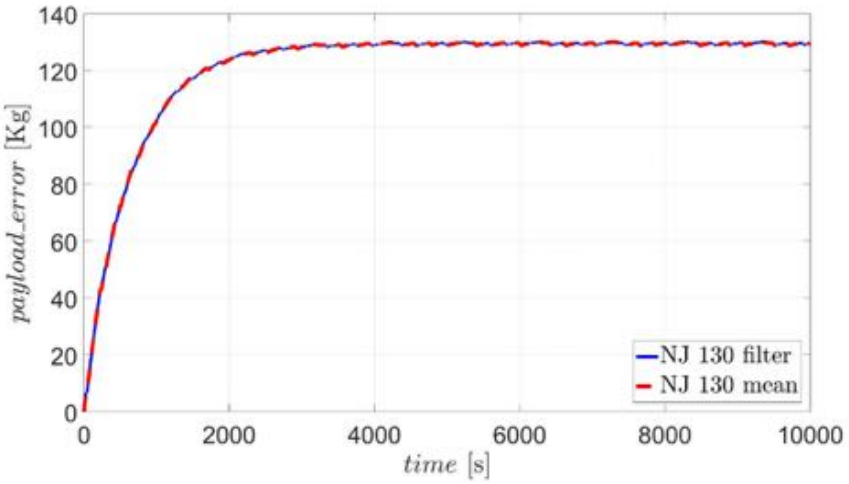
Real conditions f_z is varying with $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$

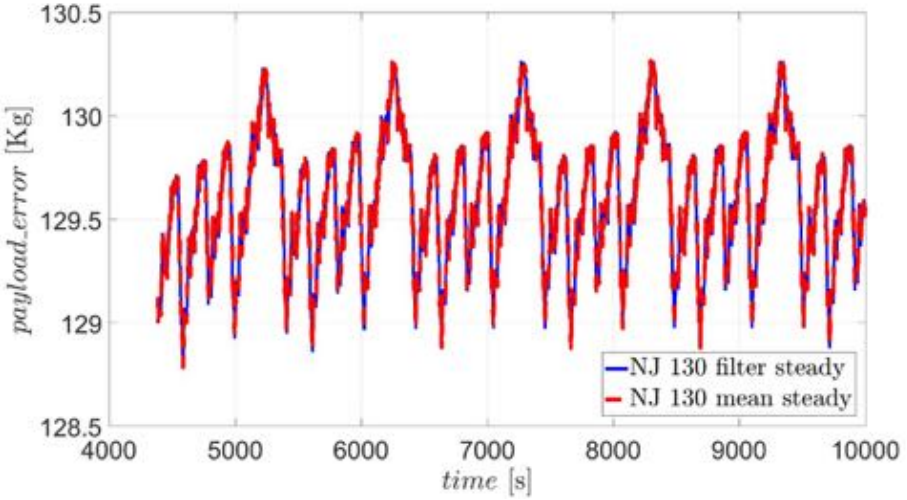
Extracting DC component of f_z to separate the payload error from model errors

$$\overline{P(i)} = \frac{\overline{P(i-1)} \cdot N + P(i)}{N+1}$$

Payload Check - Results

- Good results are obtained for a COMAU NJ 130
  - Real payload 130 kg declared payload 0 kg
  - Mean value 129.6 kg
  - Standard deviation 0.31 kg
  - Relative mean error 0.31%





Collision Detection - Motivations

- A rapid and robust collision detection is a fundamental issue for the safety of a robotic cell in any industrial environment, not only in the next future when a high presence of collaborative robots is expected, but also in current, standard production lines
- Goals and benefits:
  - Preservation of the robot mechanical parts in case of impact
  - Monitoring of the correct execution of the programmed task
    - Detection of failures whose effects are similar to those of a collision
- Industrial requirements:
  - Avoidance of false collision alarms
  - Wide applicability and portability of the SW implementation
  - Avoiding specific customizations
  - Using only the sensors that usually equip an industrial manipulator

Collision Detection - Approach

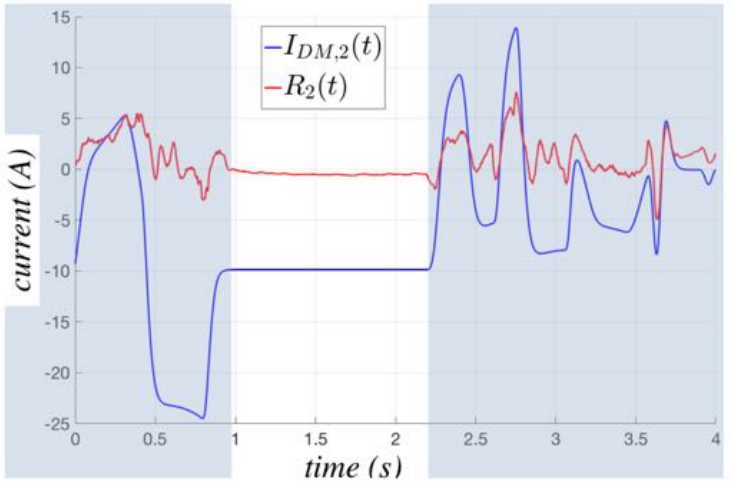
- Approach based on the residual current
- Detection based on a time varying threshold function
- The threshold is given by the sum of two terms:
  - An estimate of the absolute value of the model error in absence of collisions $\hat{m}_{err}(t)$
  - The sensitivity of the virtual sensor $Coll_{bias}(t)$

$$\mathbf{R}(t) = \mathbf{I}(t) - \mathbf{I}_{DM}(t)$$

$$|R_i(t)| > S_i(t)$$

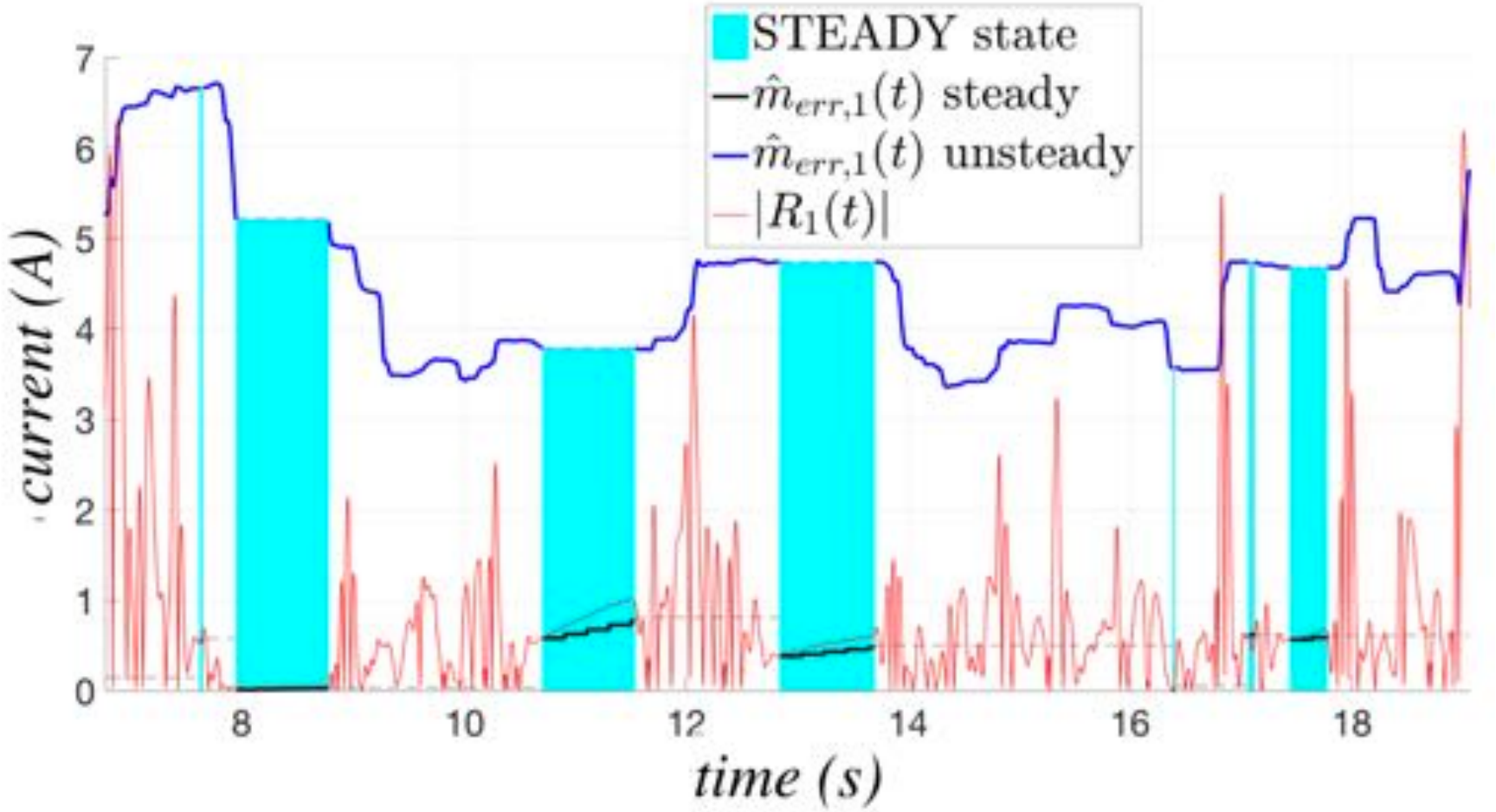
$$\mathbf{S}(t) = \hat{\mathbf{m}}_{err}(t) + \mathbf{Coll}_{bias}(t)$$

- Different approaches to compute the model error ($\hat{m}_{err}(t)$) are adopted when:
  - The current is in the steady state
  - The current is not in the steady state

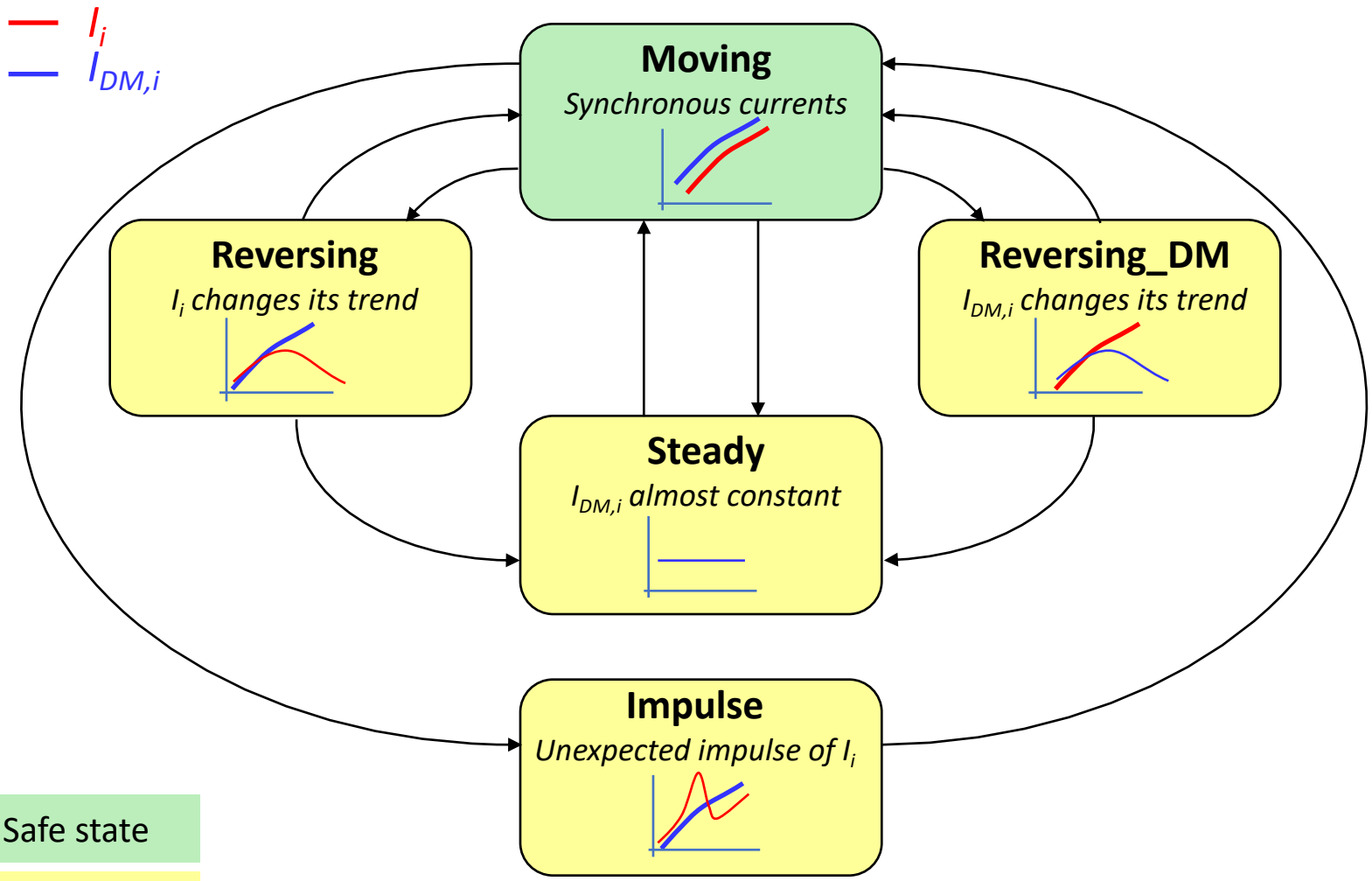


Collision Detection - Approach

Behaviour of the estimate of the model error for the first joint of a COMAU NJ4 170



Collision Detection - FSM



Collision Detection - Adaptivity

- The best threshold is used by a proper identification of the collision sensitivity $\boldsymbol{Coll_{ident}(t)}$

$$\mathbf{S}(t) = \hat{\mathbf{m}}_{err}(t) + \mathbf{Coll}_{bias}(t)$$

- The following adaptation law is applied for the i-th joint after the user request

$$Coll_{bias,i}(t_{adapt}) = \overline{Coll}^{(k)}_{Ident,i} + e^{(-t_{adapt}/\tau_a)} \left(Coll_{0,i} - \overline{Coll}^{(k)}_{Ident,i} \right)$$

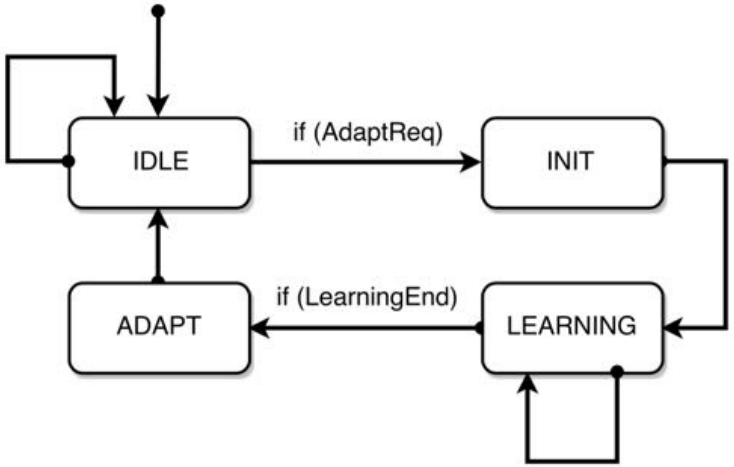
Collision Detection - Adaptivity

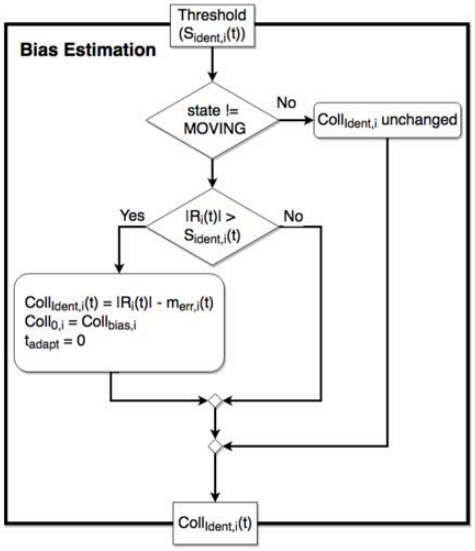
- The best threshold is used by a proper identification of the collision sensitivity $\boldsymbol{Coll_{ident}(t)}$

$$\mathbf{S}(t) = \hat{\mathbf{m}}_{err}(t) + \mathbf{Coll}_{bias}(t)$$

- The following adaptation law is applied for the i-th joint after the user request

$$Coll_{bias,i}(t_{adapt}) = \overline{Coll}^{(k)}_{Ident,i} + e^{(-t_{adapt}/\tau_a)}\left(Coll_{0,i} - \overline{Coll}^{(k)}_{Ident,i}\right)$$





$$Coll_{Ident,i}(t) = |R_i(t)| - \hat{m}_{err,i}(t)$$
$$Coll_{0,i} = Coll_{bias,i}$$
$$t_{adapt} = 0$$

Collision Detection - Results

Collision Detection - Results

Performed Tests	DT Adapt (s)	DT Basic (s)	Average Reduction %
top → bottom EOS	0.026	0.106	75.5
top → bottom NR	0.024	0.186	87.1
left → right EOS	0.010	0.044	77.3
left → right NR	0.010	0.046	78.3

		Ax					
	Performed Tests	1	2	3	4	5	6
Basic	top → bottom EOS	_	•	_	_	_	_
	top → bottom NR	_	_	_	_	•	•
	left → right EOS	•	_	_	_	_	_
	left → right NR	•	_	_	•	_	_
Adaptive	top → bottom EOS	_	•	•	•	•	•
	top → bottom NR	_	•	•	_	•	•
	left → right EOS	•	_	•	•	•	•
	left → right NR	•	•	•	•	•	•

Post-collision reaction and Manual Guidance

- Manage both collision reaction and manual guidance
- Sensor-less approach
- Distinguish accidental collisions from intended human-robot contacts

Post-collision reaction and Manual Guidance

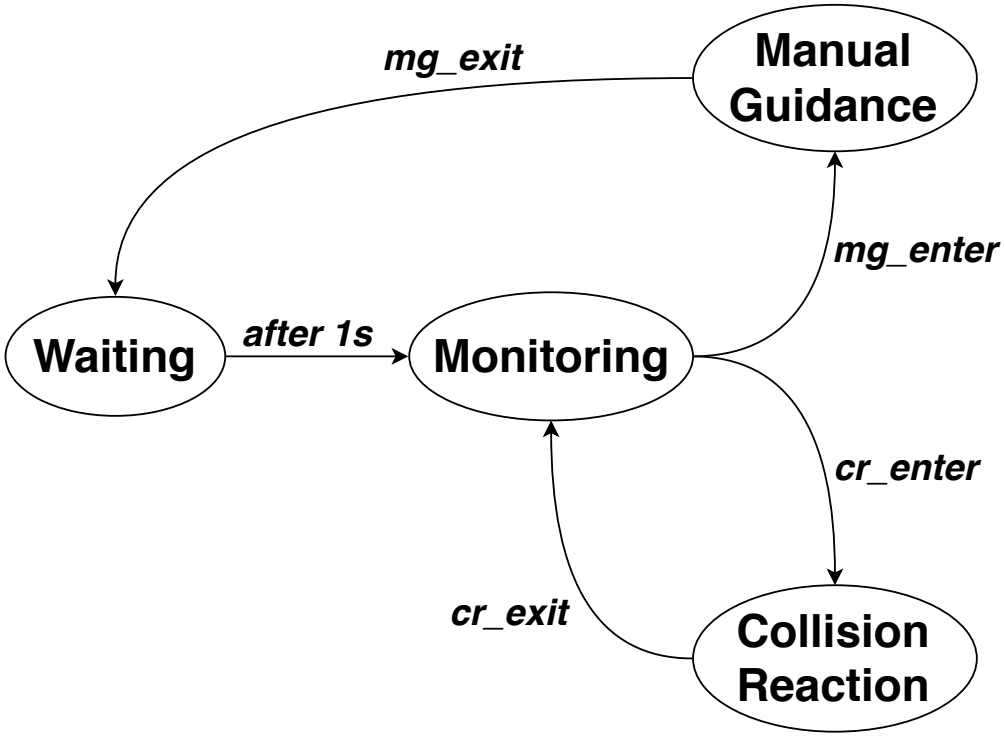
- Manage both collision reaction and manual guidance
- Sensor-less approach
- Distinguish accidental collisions from intended human-robot contacts

- Stop the robot as fast as possible
- Reduction of the impact force

- Robot compliant to the applied forces

Post-collision reaction and Manual Guidance

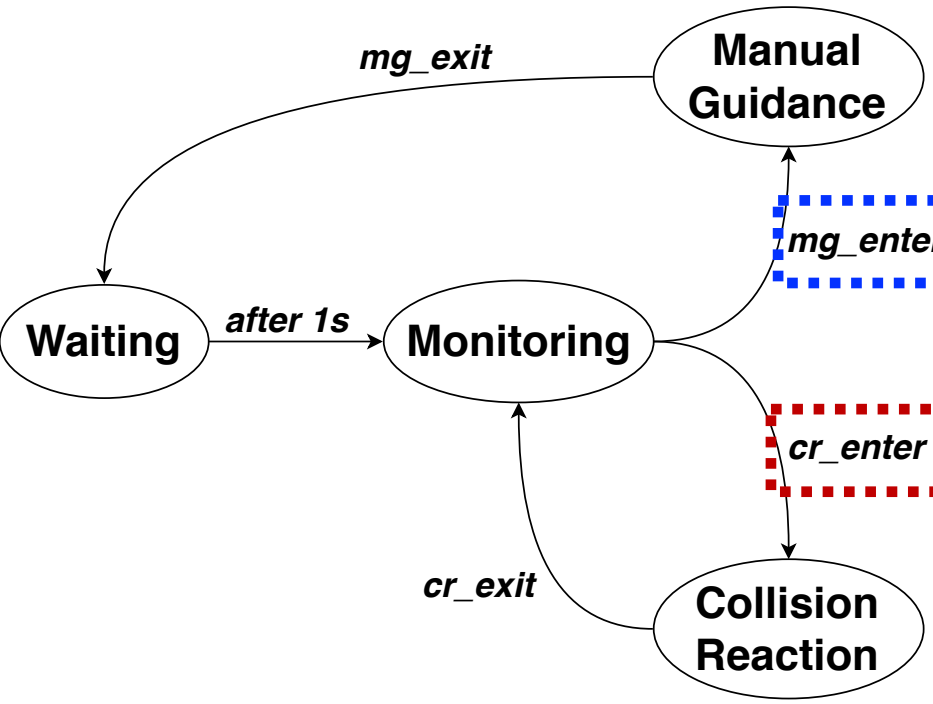
- Manage both collision reaction and manual guidance
- Sensor-less approach
- Distinguish accidental collisions from intended human-robot contacts



Post-collision reaction and Manual Guidance

2Hz Low-Pass filter

$$I_{mg_{(j)}}(t_k) = a_1 \cdot I_{mg_{(j)}}(t_{k-1}) + a_2 \cdot I_{mg_{(j)}}(t_{k-2}) + b_1 \cdot R_{(j)}(t_k)$$



$$\tau_{mg}(t) = K_I \cdot I_{mg}(t)$$

$$F_{mg}(t) = \left(J(q)^T\right)^{-1} \tau_{mg}(t)$$

$$\begin{cases} F_{mg_{(ax)}}(t) > Th1_{H_{(ax)}}(t) \\ F_{mg_{(ax)}}(t) < Th1_{L_{(ax)}}(t) \end{cases}$$

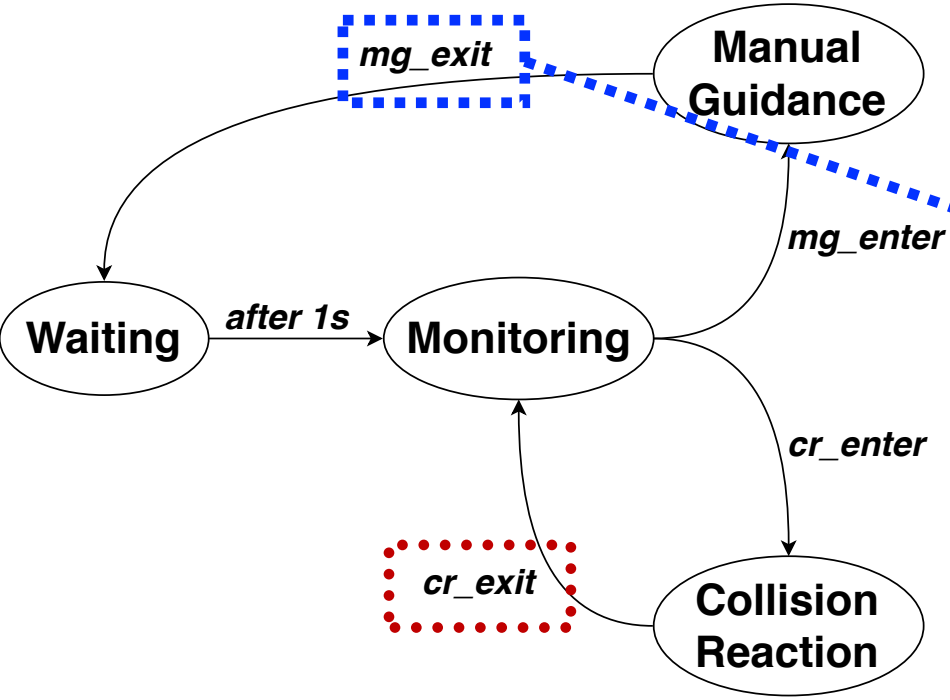
$$I_{cd_{(j)}}(t) > Th_{cd_{(j)}}(t)$$

$$Th_{cd_{(j)}}(t) = k_{cd_{c_{(j)}}} + k_{cd_{v_{(j)}}} \frac{|\dot{q}_j(t)|}{\dot{q}_{j,max}} + k_{cd_{a_{(j)}}} \frac{|\ddot{q}_j(t)|}{\ddot{q}_{j,max}}$$

10Hz High-Pass filter

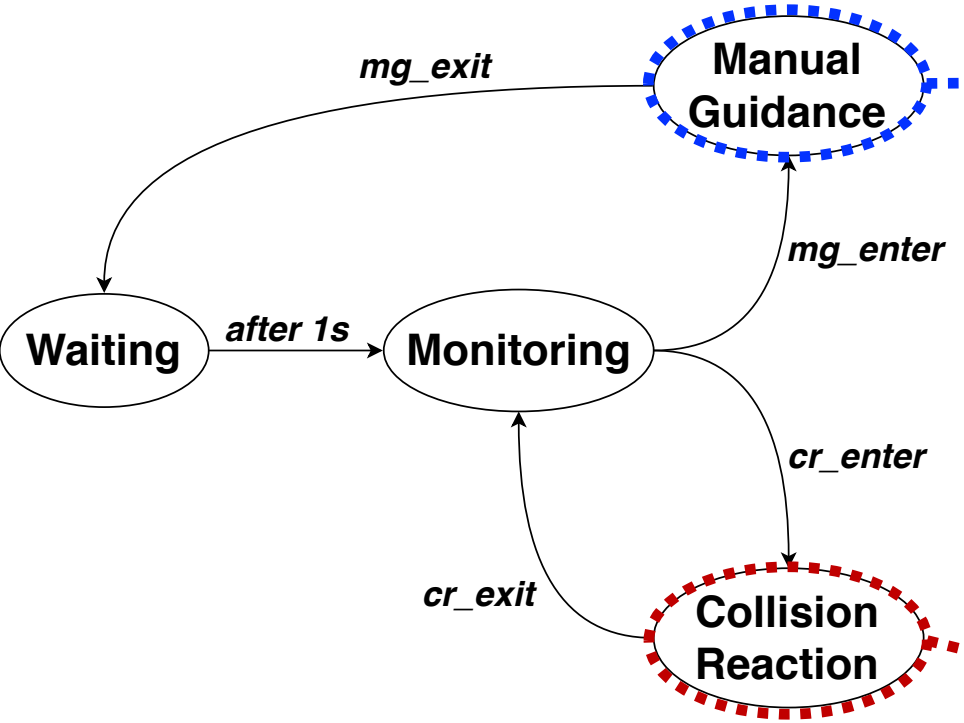
$$I_{cd_{(j)}}(t_k) = c_1 \cdot I_{res_{(j)}}(t_k) + c_2 \cdot I_{res_{(j)}}(t_{k-1}) + c_3 \cdot I_{res_{(j)}}(t_{k-2}) + c_4 \cdot R_{(j)}(t_{k-3})$$

Post-collision reaction and Manual Guidance



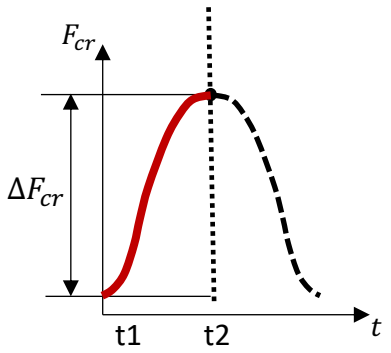
$$\begin{cases} Th2_{L_{(ax)}}(t) < F_{mg_{(ax)}}(t) < Th2_{H_{(ax)}}(t) \\ \left| \frac{1}{N_{mg}} \sum_{k=1}^{N_{mg}} \left(F_{mg,s_{(ax)}}(t-k-1) \right) \right| < C_{flat_{(ax)}} \end{cases}$$

Post-collision reaction and Manual Guidance



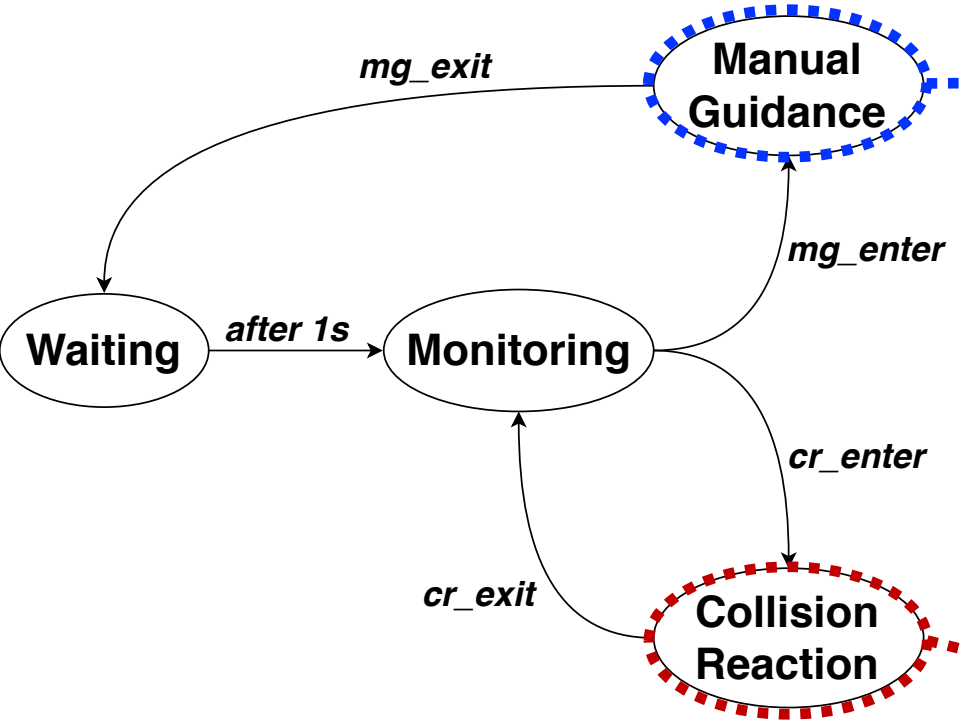
$$\Delta p_{mg} = K_{mg}^{-1} \cdot F_{mg}$$

First phase



$$\Delta p_{cr}(t) = K_{cr}^{-1} \cdot F_{cr}(t)$$

Post-collision reaction and Manual Guidance



$$\Delta p_{mg} = K_{mg}^{-1} \cdot F_{mg}$$

Second phase

$$N_{dec} = a_s P_{cr} + b_s$$

S1: $P_{cr} = (\|K_{cr}^{-1} \Delta F_{cr}\| / \Delta t)$

S2: $P_{cr} = \|K_{cr}^{-1} \Delta F_{cr}\|$

S3: $P_{cr} = 0$

Post-collision reaction and Manual Guidance

Conclusions

On robotic cell programming :

- Task model able to take into account both physical and functional constraints
- Automatic task oriented programming based on the task model
- Collaboration with UNIVPM to integrate the task programming approach with the OTE methodology
- Verification of the methodology for realistic robotic cells

On service algorithms :

- Improvement of the robot dynamic model using a new framework for friction identification
- Adaptive collision detection algorithm (implemented in COMAU controller)
- Payload check (implemented in the COMAU controller)
- Post collision reaction
- Manual guidance

Publications

❖ Task-Oriented Programming

1. Task Modeling for Task-Oriented Robot Programming, Trapani, Stefano; Indri, Marina, 22nd IEEE International Conference on Emerging Technologies And Factory Automation (ETFA 2017)
2. Integration of a production efficiency tool with a general robot task modeling approach, Indri, Marina; Trapani, Stefano; Bonci, Andrea; Pirani, Massimiliano, IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA 2018)
3. Programming robot work flows with a task modeling approach, Indri, Marina; Trapani, Stefano, 44th Annual Conference of the IEEE Industrial Electronics Society (IECON 2018)

❖ General procedures and service algorithms of general validity

4. Development of a Virtual Collision Sensor for Industrial Robots, Indri, Marina; Trapani, Stefano; Lazzero, Ivan, journal SENSORS, 17(5), 1-23, 2017
5. A general procedure for collision detection between an industrial robot and the environment , Indri, Marina; Trapani, Stefano; Lazzero, Ivan, 20th IEEE International Conference on Emerging Technologies and Factory , Automation (ETFA 2015)
6. Development of a general friction identification framework for industrial manipulators, Indri, Marina; Trapani, Stefano; Lazzero, Ivan, IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society
7. Using Virtual Sensors in Industrial Manipulators for Service Algorithms Like Payload Checking, Indri, Marina; Trapani, Stefano, 26th International Conference on Robotics in Alpe-Adria-Danube Region, RAAD 2017, Springer series on MECHANISMS AND MACHINE SCIENCE
8. Smart Sensors Applications for a New Paradigm of a Production Line, Indri, Marina; Lachello, Luca; Lazzero, Ivan; Sibona, Fiorella; Trapani, Stefano. - In: SENSORS. - ISSN 1424-8220. - ELETTRONICO. - 19:3, 650(2019).

Thanks