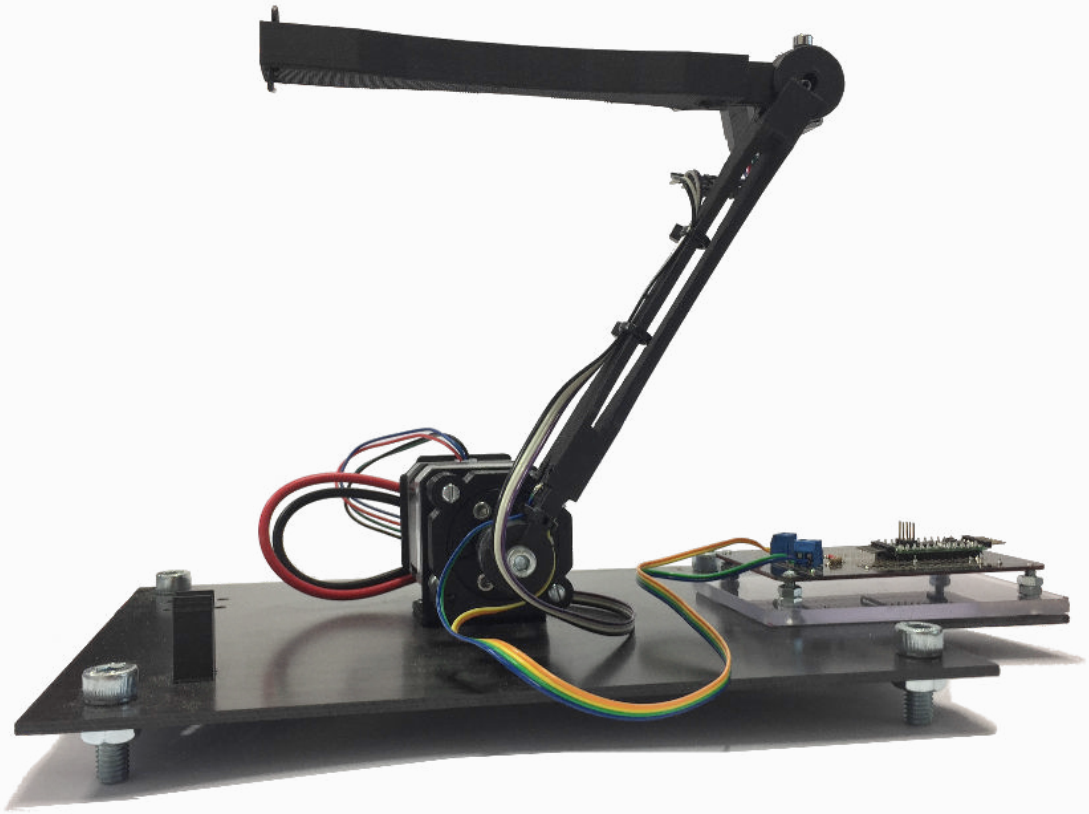
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

MPERL:
Hardware and Software Co-design for Robotic Manipulators

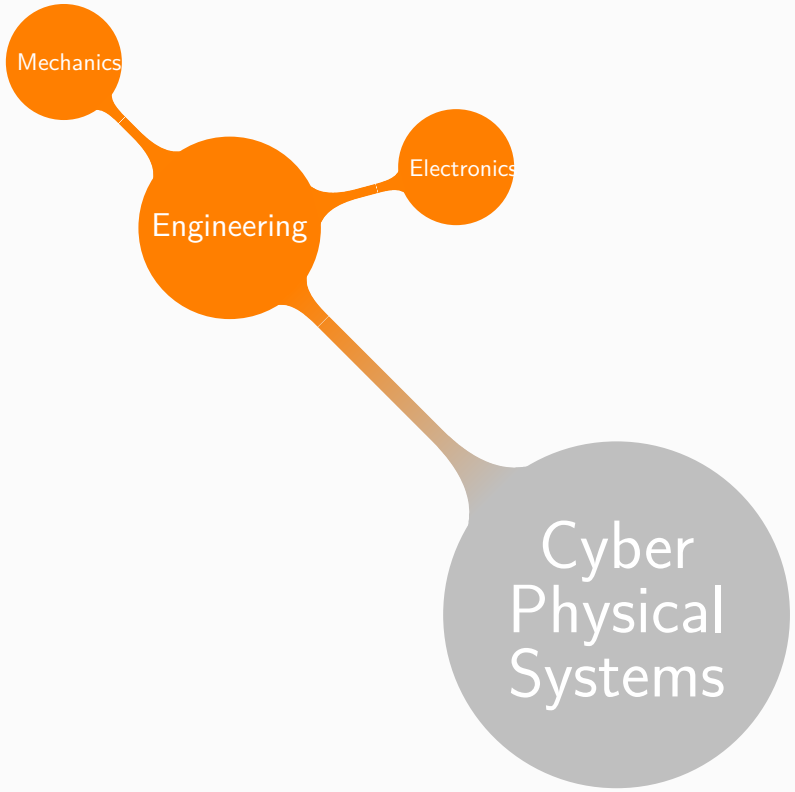
Marcus Pirron, mpirron@mpi-sws.org
Damien Zufferey, zufferey@mpi-sws.org
IROS 2019
November 7, 2019

Imagine we want to build a robotic manipulator

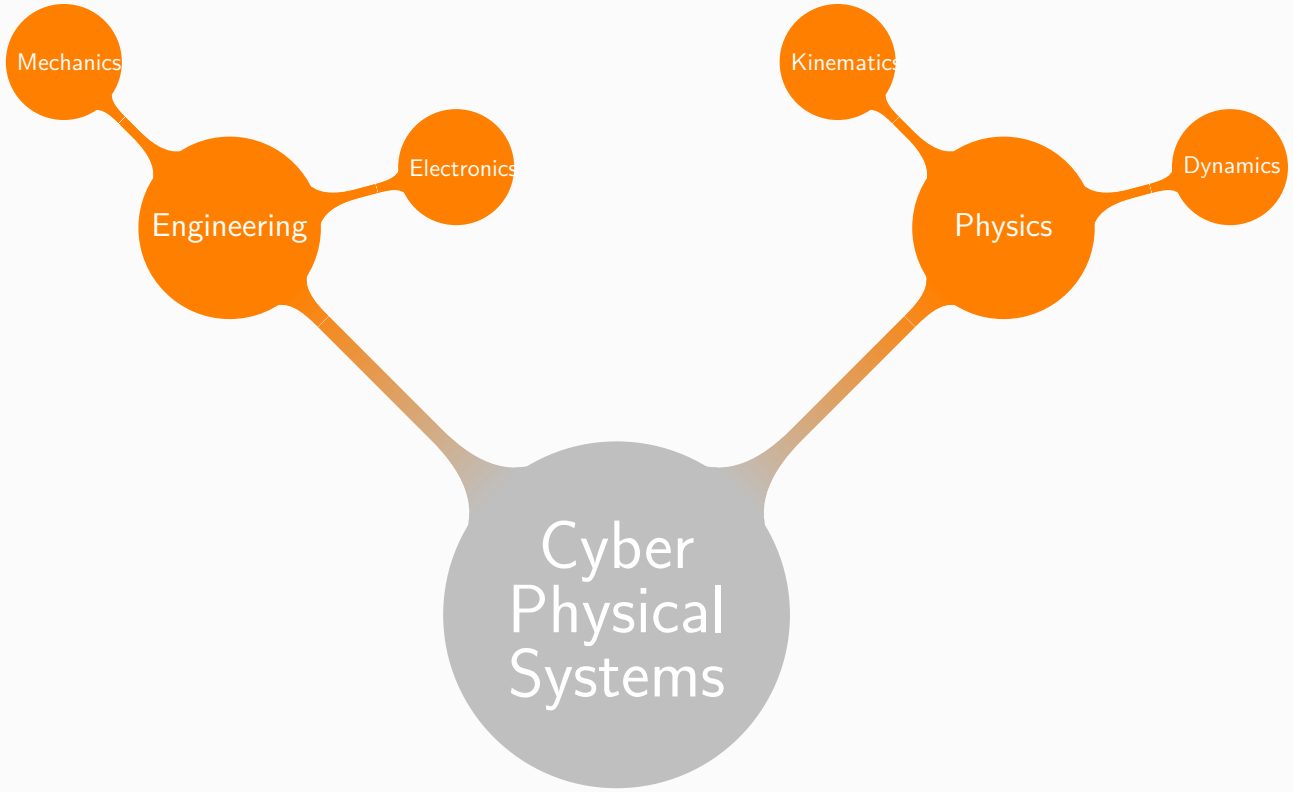


CPS draw frome multiple domains ...

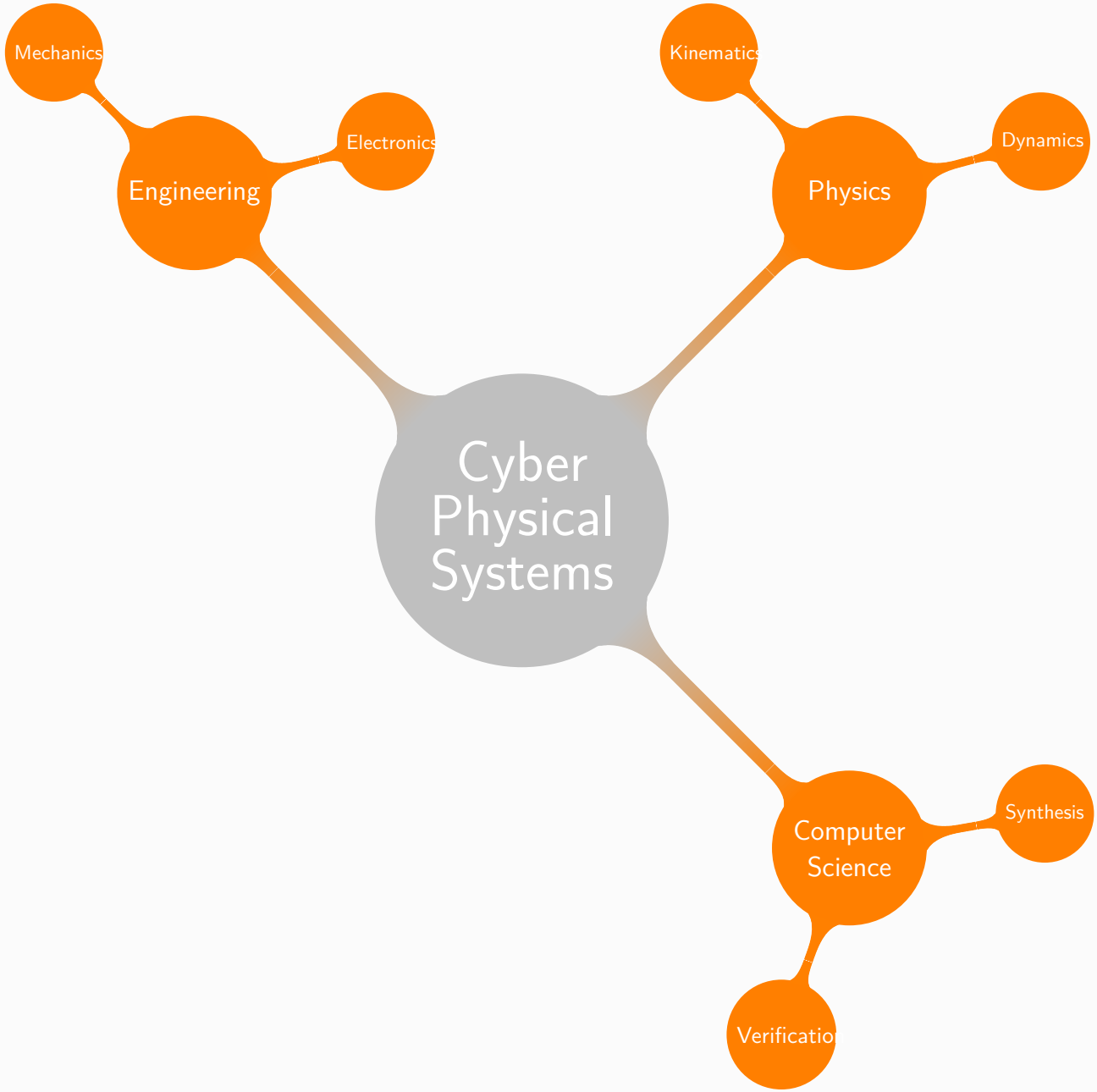
CPS draw frome multiple domains ...



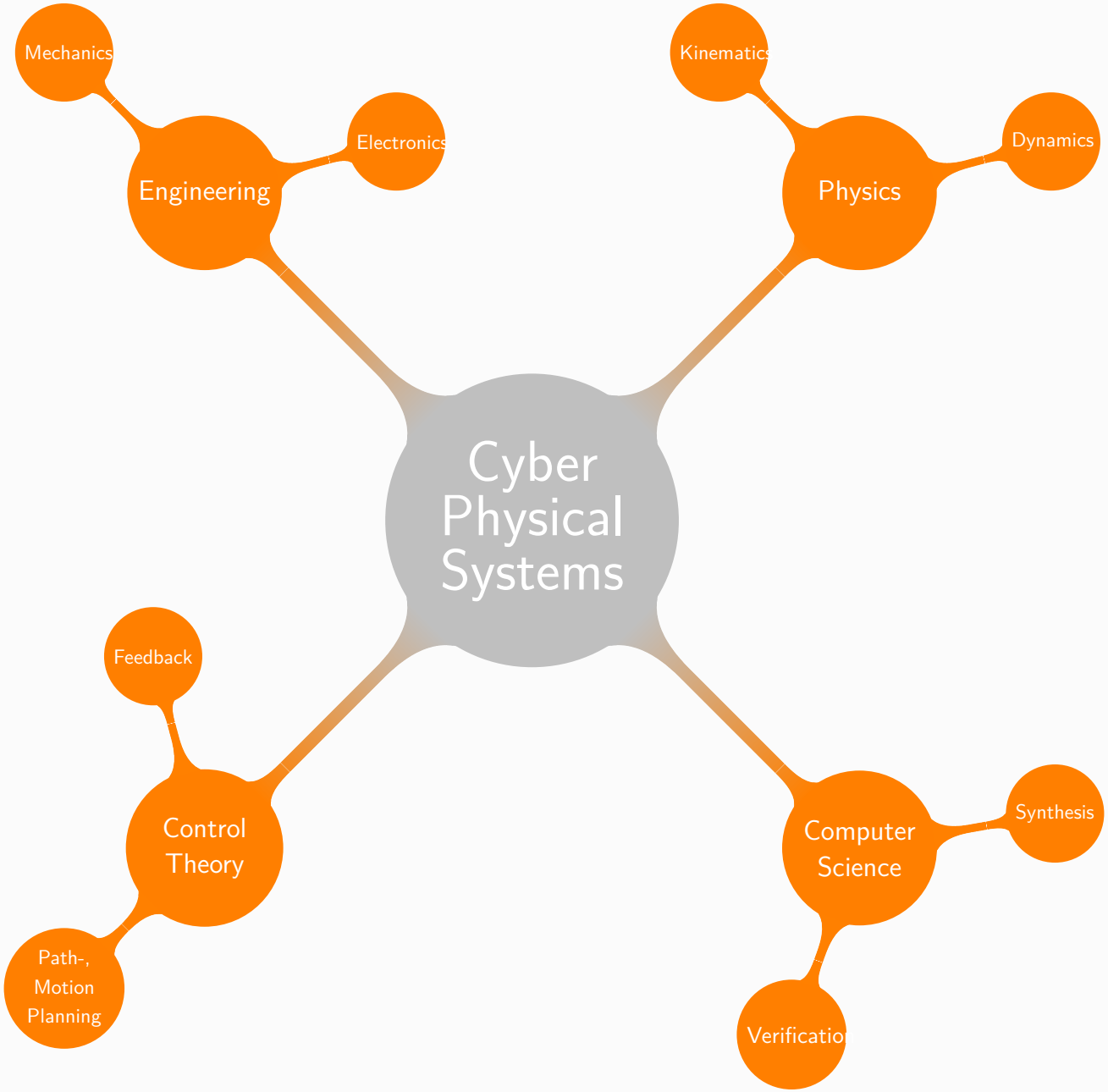
CPS draw frome multiple domains ...



CPS draw frome multiple domains ...



CPS draw frome multiple domains ...



MPERL: Design, Manufacture and Control of robotic systems

Allow non-expert users to create and test robotic manipulators programmatically. Key characteristics:

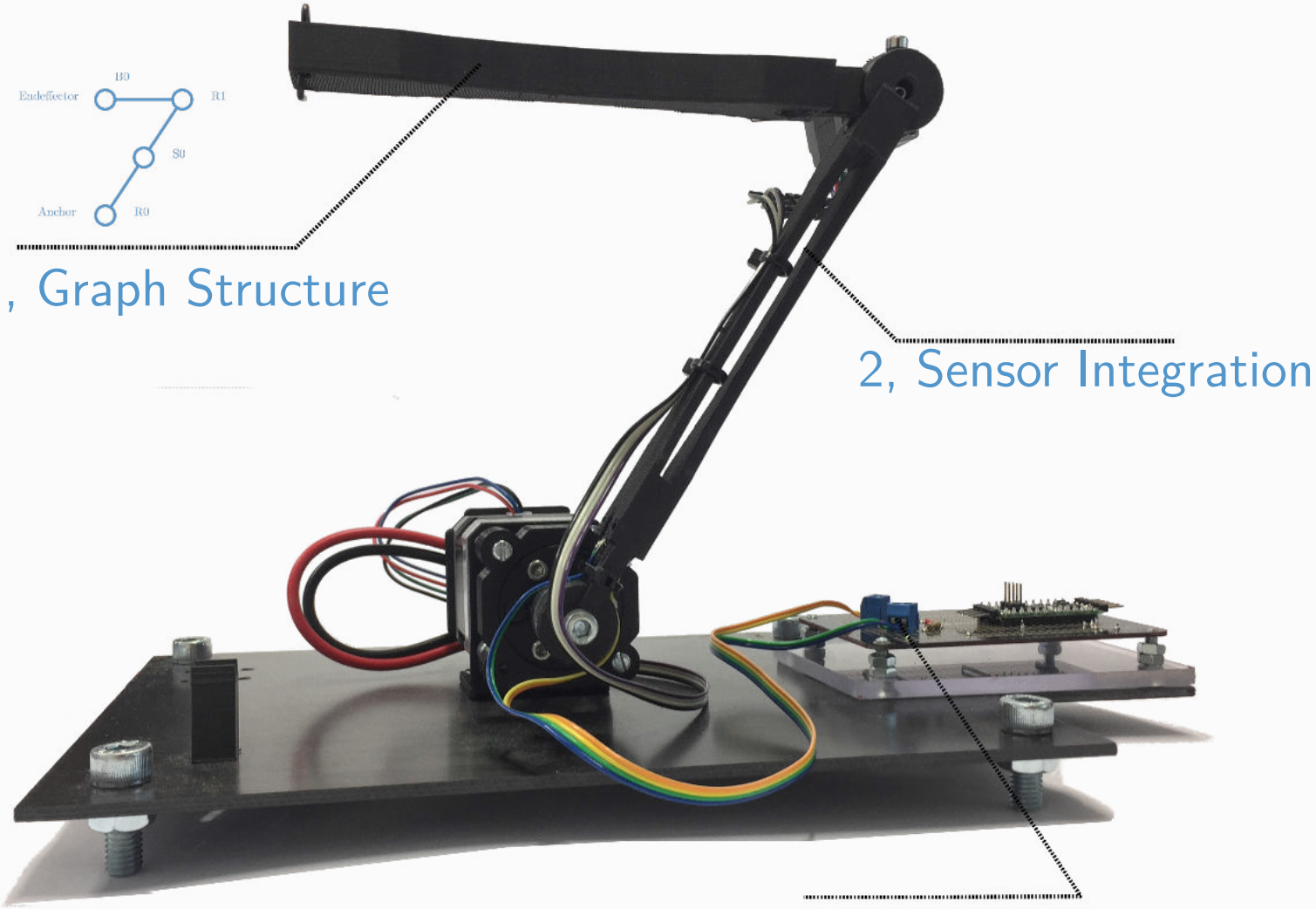
- Graph structure based: **serial** and **parallel** (i.e. interconnected) manipulators
- Support for **actuators**, **sensors**, joints, links, ...
- **Kinematics** solver, Refinable **singularity** Map
- Programming by automatically generating **high-level motion primitives**
- Integrates elements from **computational design and fabrication**, **kinematic solvers**, **synthesis**, and **feedback control**

Git Repository: https://gitlab.mpi-sws.org/mpirron/mperl

Related Work

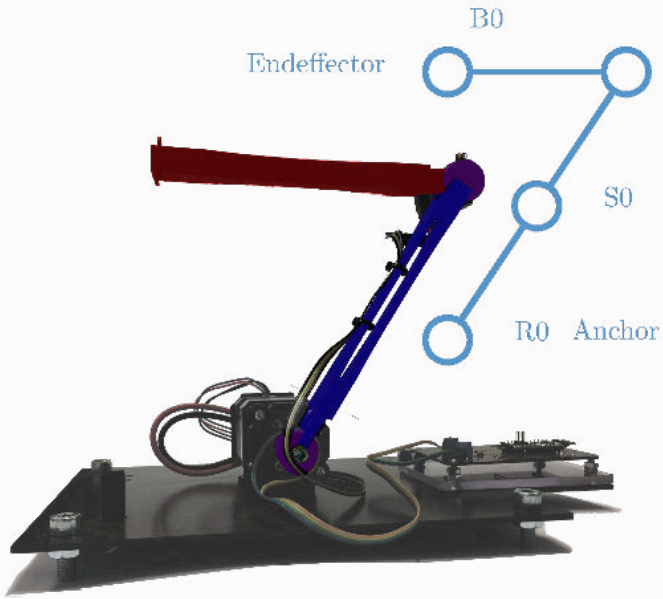
	Input	Output	Controller
Metha et al. [IROS 2014, ASME 2015]	High level structural description	General mechanism, manufacturable parts	FSM, complex temporal behaviors
Ha et al [Trans. on Robotics 2018]	Input Trajectory	Mechanism following input trajectory	Tied to input trajectory
MPERL	High level structural description	General mechanism, manufacturable parts	Motion primitives, (inverse) kinematics, singularities

Example: Single Scara Arm with Deflection Correction



Code example

```
1 S0 = FlexBeam( length=1 )
2 B0 = Beam( length=1 )
3 R0 = Revolute(yaw in (-pi/2, pi/2),actuated)
4 R1 = Revolute(yaw in (-pi/2, pi/2),actuated)
5
6 Arm = R0 → S0 → R1 → B0
7 Anchor( R0, (0,0,0) )
8
9 EndEffector( B0 )
```

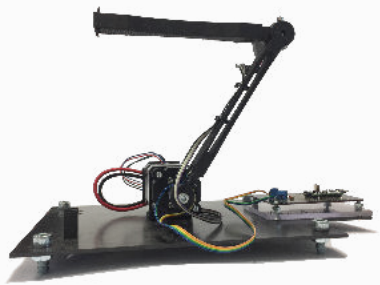


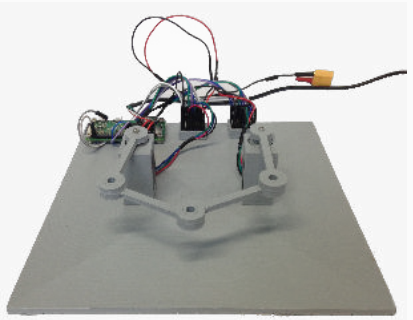
From serial to parallel systems

MPERL also supports **parallel** robotic systems.

```
1 Arm = R0 → S0 → R1 → B0
2
3
4
5 Anchor( R0, (0,0,0) )
6
7 EndEffector( B0 )
```

```
1 Arm1 =  R0 → B0 → R1 → B1 → R2
2 Arm2 = Clone( Arm1, preserve=R2 )
3 DualScara = Merge(Arm1, Arm2)
4
5 Anchor( R0, (0,0,0) )
6 Anchor( Arm2.head, (2,0,0) )
7 EndEffector( R2 )
```



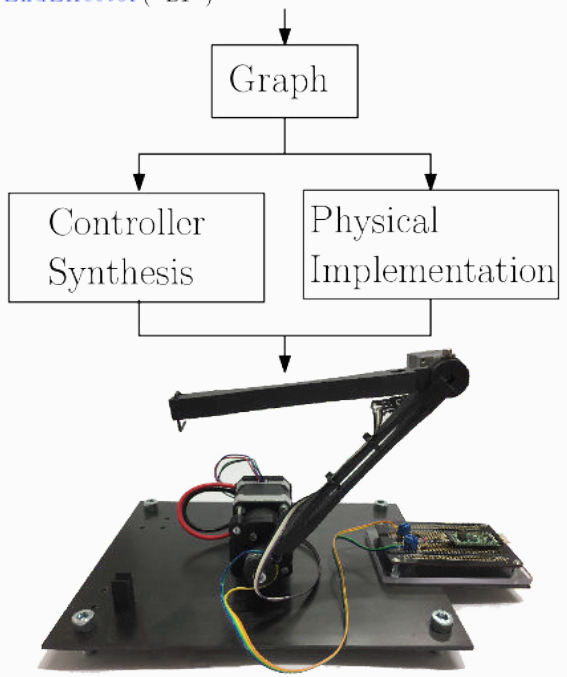


What do we do with this description?

From this description, MPERL generates the **graph structure**.

The graph forms the base for the controller (e.g. kinematics or singularities)

```
S0 = FlexBeam( length=1 )
B1 = Beam( length=1 )
R0 = Revolute(yaw in (-pi/2, pi/2), actuated)
R1 = Revolute(yaw in (-pi/2, pi/2), actuated)
Arm = R0 -> S0 -> R1 -> B1
Anchor( R0, (0,0,0) )
Anchor( Arm.head, (2,0,0) )
EndEffector( B1 )
```

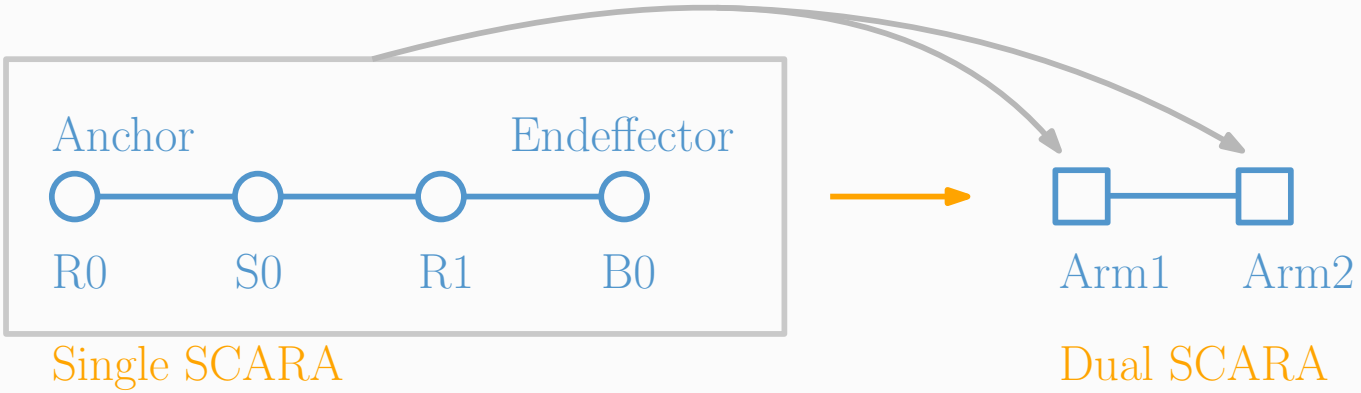


Graph Structure

The graph structure represents the **functional dependencies** of the robotic system, consisting of interconnected components (**actuators**, **sensors**, ..). It is the basis for calculating e.g. kinematics and singularities.

- Vertices: Components with resp. frame of reference
- Edges: Connections between components
- Labels: Type (Actuator, Sensor ..) and Properties (Anchors, Effector ..)

A robotic system can contain **other robotic systems**.

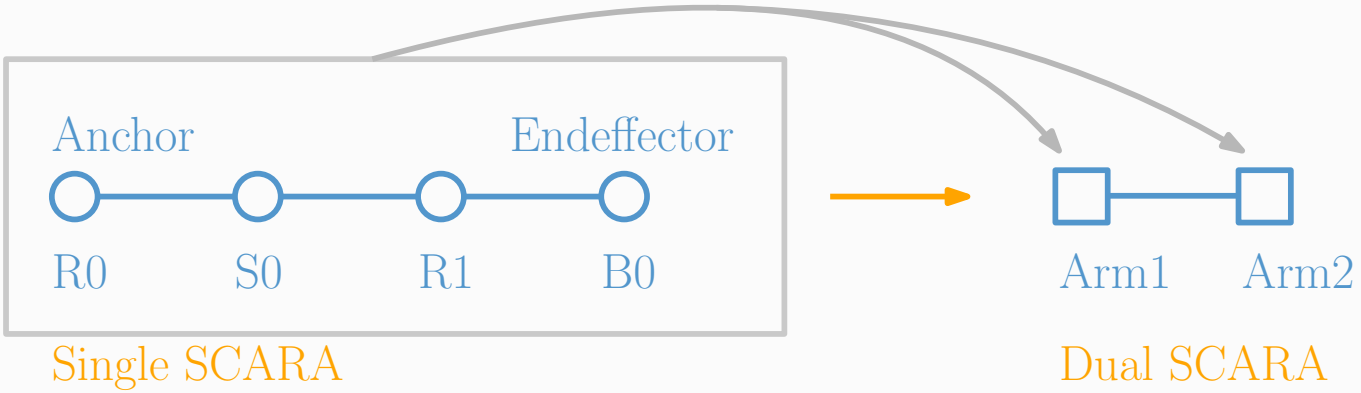


Graph Structure

The graph structure represents the **functional dependencies** of the robotic system, consisting of interconnected components (**actuators**, **sensors**, ..). It is the basis for calculating e.g. kinematics and singularities.

- Vertices: Components with resp. frame of reference
- Edges: Connections between components
- Labels: Type (Actuator, Sensor ..) and Properties (Anchors, Effector ..)

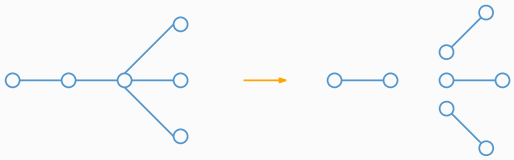
A robotic system can contain **other robotic systems**.



From serial to parallel systems

For kinematics of parallel systems, the graph needs to be processed:

For **serial chains**, no processing is needed.



Implicit chains are decomposed into a collection of serial chains

No additional constraints need to be generated

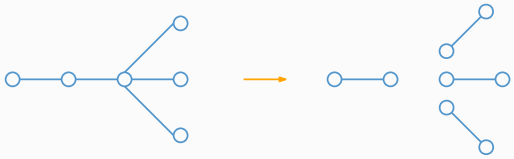
Non-implicit chains are decomposed into a minimal set of paths covering all edges.

Additional constraints are generated (possible node duplication)

From serial to parallel systems

For kinematics of parallel systems, the graph needs to be processed:

For **serial chains**, no processing is needed.



Implicit chains are decomposed into a collection of serial chains

No additional constraints need to be generated

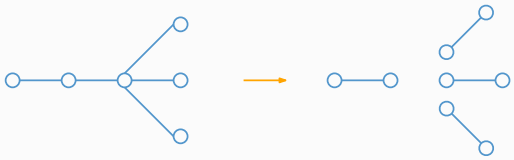
Non-implicit chains are decomposed into a minimal set of paths covering all edges.

Additional constraints are generated (possible node duplication)

From serial to parallel systems

For kinematics of parallel systems, the graph needs to be processed:

For **serial chains**, no processing is needed.



Implicit chains are decomposed into a collection of serial chains

No additional constraints need to be generated

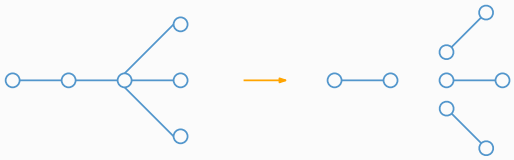
Non-implicit chains are decomposed into a minimal set of paths covering all edges.

Additional constraints are generated (possible node duplication)

From serial to parallel systems

For kinematics of parallel systems, the graph needs to be processed:

For **serial chains**, no processing is needed.



Implicit chains are decomposed into a collection of serial chains

No additional constraints need to be generated

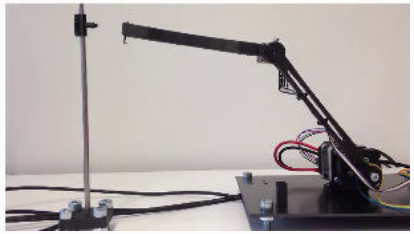
Non-implicit chains are decomposed into a minimal set of paths covering all edges.

Additional constraints are generated (possible node duplication)

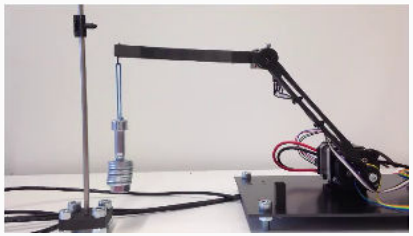
Sensor Integration

The controller adapts **automatically** to the sensor readings

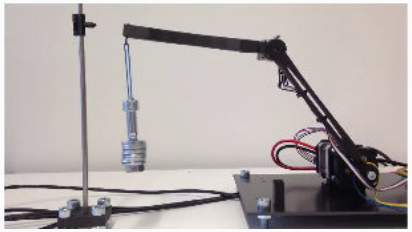
- Sensors can be embedded into the structure at manufacturing time
- Here: Use a flex sensor to compensate for load induced deflection



Position at rest



Load is applied, end position deviates

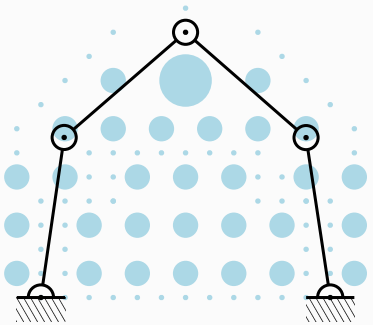


Deviation is automatically corrected

Controlling the system: Kinematic and Singularities

The **controller** evaluates the graph structure and provides **kinematics w.r.t. singularities**.

- MPERL can solve inverse kinematics
- For each system, a refinable **singularity map** is created



The size of dot indicates the distance to singular regions (smaller dots: closer to singularity)

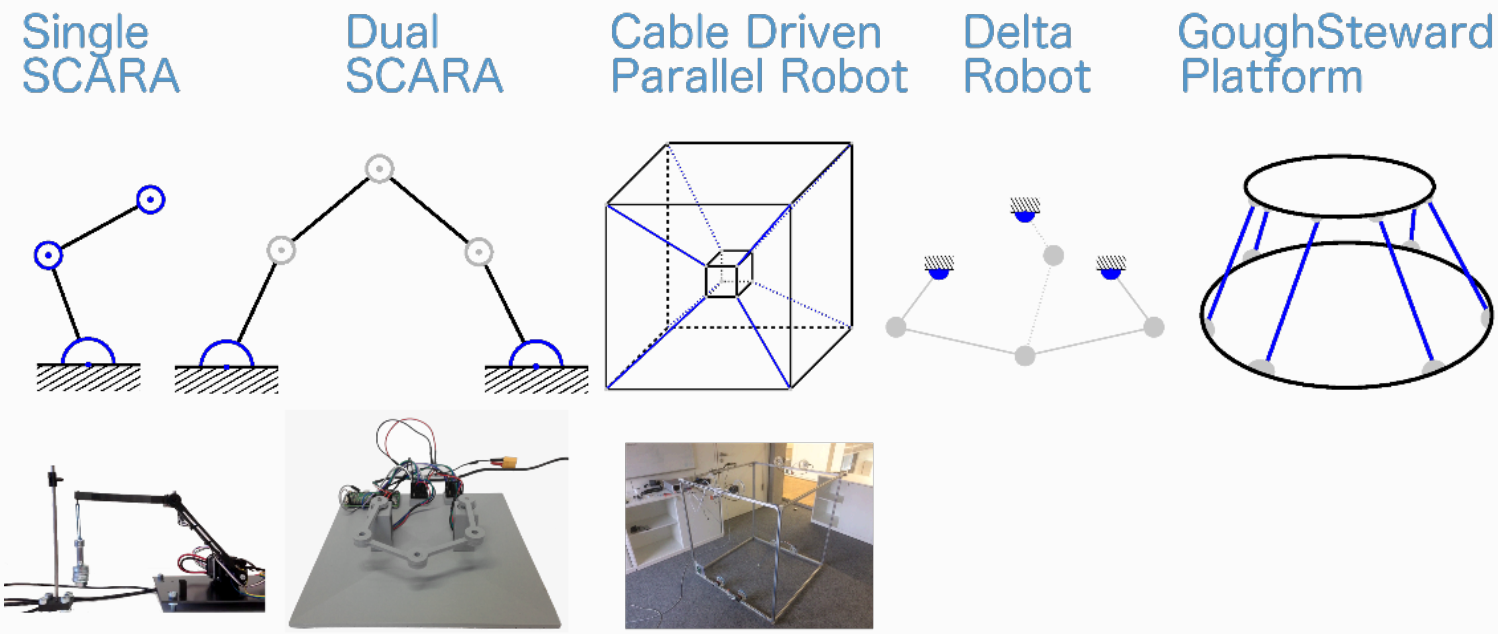
Experiments Overview

- Full implementation of MPERL
- Physical implementation of Single and Dual Scara Arm, Cable Robot
- Kinematic Solvers: Uses dReal (SMT Solver for non-linear theories of reals), Least Squares optimization and CCD

Experiments Overview

MPERL was tested on multiple robotic systems.

We evaluated our tool on three physical implementations and two virtual implementations



Evaluation: Results

Overview of examined robotic systems. $|P_p|$ and $|P_a|$ denotes the number of passive (e.g. joints) and active parameters (e.g. actuators)

Robotic System	Nodes in graph	$\|P_p\|$	$\|P_a\|$	Singularity Map [s]	Inverse kinematics [ms]			Forward kinematics [ms]
					dReal	LS	CCD	
Single Scara Arm	7	2	3	67	52	124	439	44
Dual Scara Arm	13	8	2	34	156	448	918	138
Cable Robot	40	48	8	621	594	955	825	217
Delta Robot	15	4	3	432	367	822	1287	177
Gough Stewart	30	36	6	489	516	755	640	273

Future Work

Currently, the following additions are being made:

- Dynamic effects (e.g. Torque, Load distribution)
- Additional material properties like fatigue, strain, elasticity, ...
- Connecting to PGCD [Banusic et al, ICCPS 2019], for coordinated actions of multiple robotic systems

MPERL

A tool to design, manufacture and control robotic systems

Git Repository: https://gitlab.mpi-sws.org/mpirron/mperl

- Programming by automatically generating **high-level motion primitives**
- Integrates elements from
  - computational design and fabrication
  - kinemetic solvers
  - synthesis
  - feedback control