INESCTEC

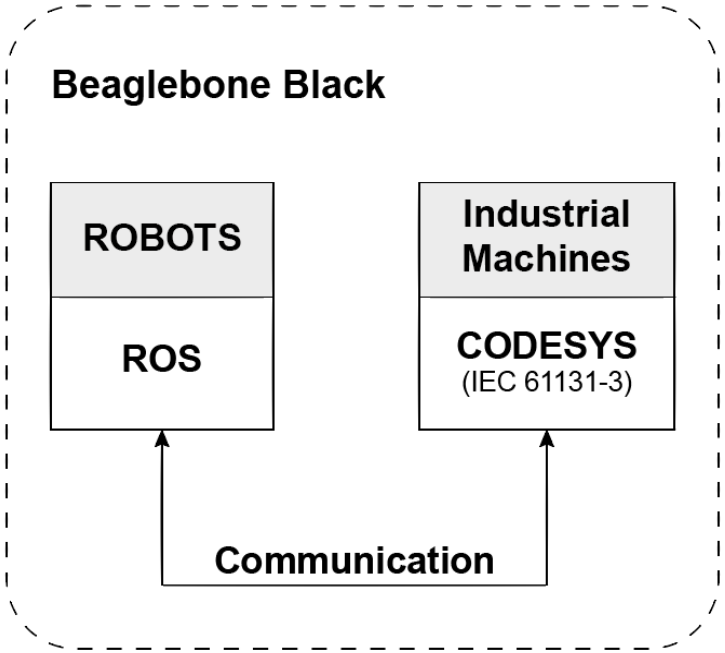
Bridging Automation and Robotics

An Interprocess Communication between IEC 61131-3 and ROS

Tiago Pinto
tiago.f.pinto@inesctec.pt

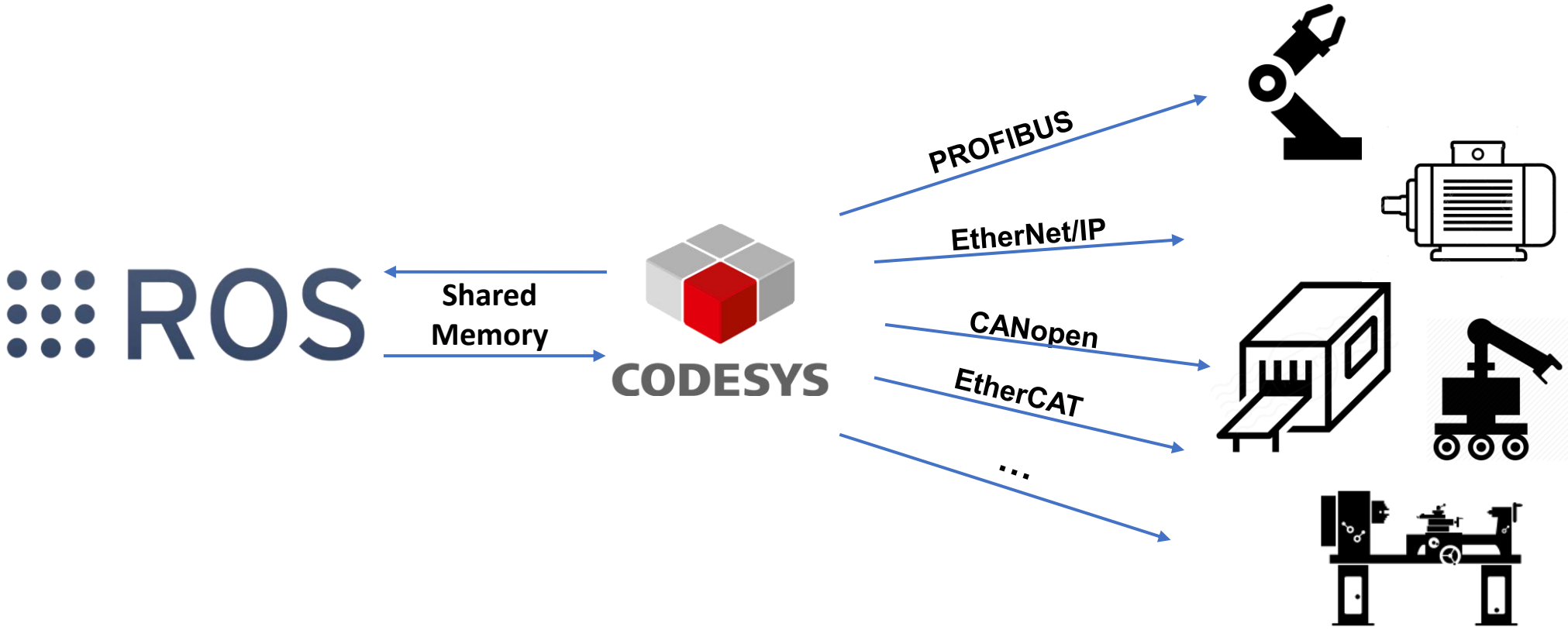
Concept

- **Problem:**
  - Robot re-programming by automation technitians who do not have the required skills to delve into complicated robot systems.
  - Implementing **multi-vendor and multi-protocol communication** between ROS and industrial equipment.
  - Time consumed programming drivers for actuators.

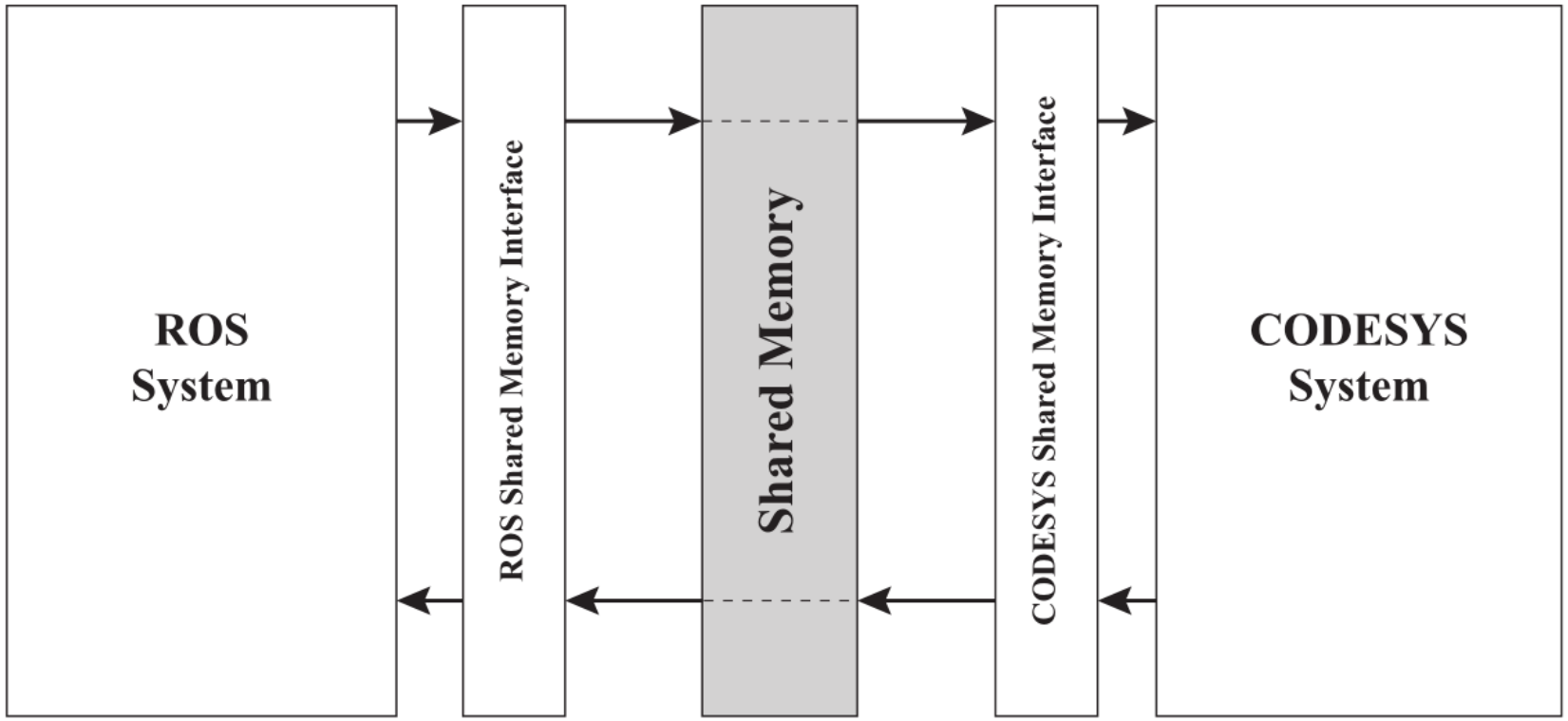


- **Solution:**
  - Embedded System Development with ROS and CODESYS softPLC Integration

ROS – CODESYS Communication

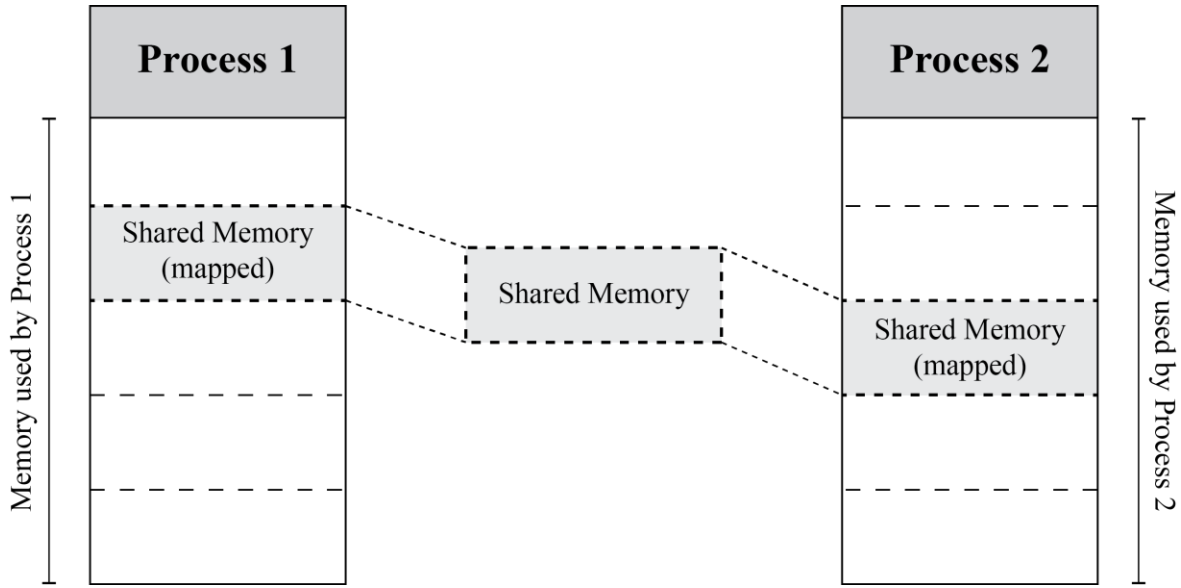


Architecture of the system

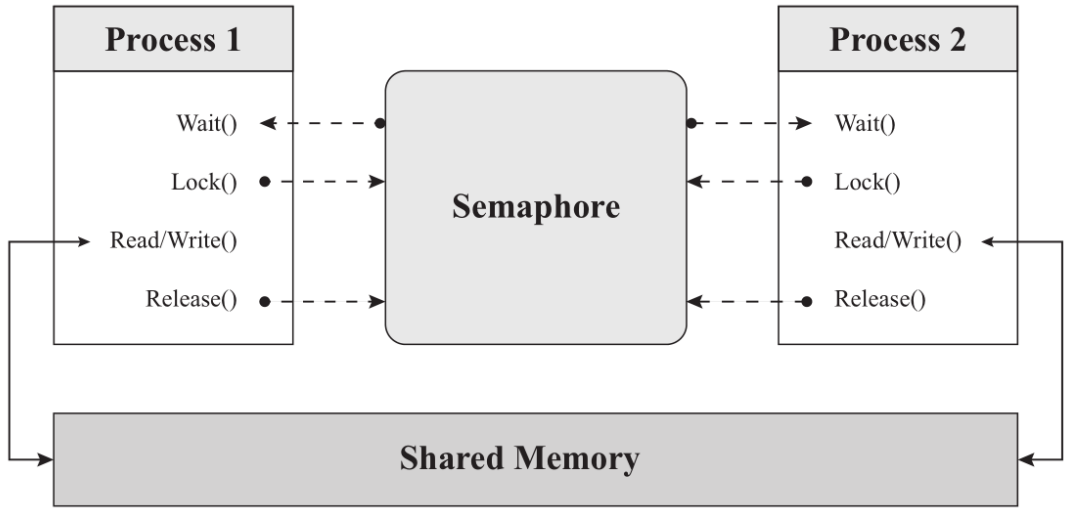


POSIX Shared Memory

- CODESYS Provides implementation function for the **POSIX Named Shared Memory**.
- It is known as **the fastest way of passing data** between two processes on the same host system.
- This method needs a syncronization mechanism.



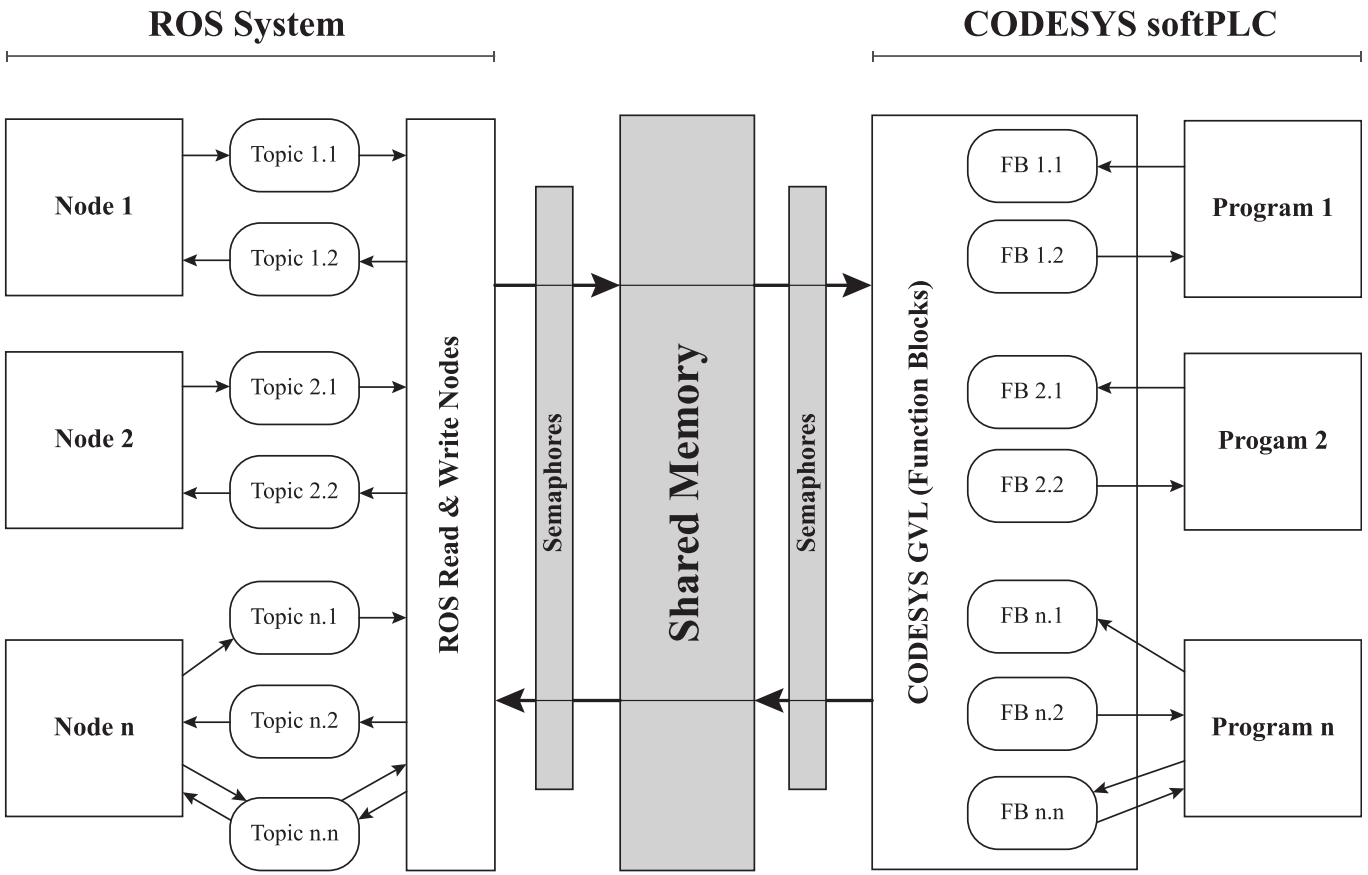
Access Synchronization



T. Pinto, R Arrais and G. Veiga, "Bridging Automation and Robotics: an Interprocess Communication between IEC 61131-3 and ROS", 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), Porto, 2018. In Press.

- CODESYS has a library with ***Semaphore*** functions.
- This library implements the **POSIX Named Semaphores**.
- **Can be implemented both on ROS and CODESYS**

Architecture of the system



T. Pinto, R Arrais and G. Veiga, "Bridging Automation and Robotics: an Interprocess Communication between IEC 61131-3 and ROS", 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), Porto, 2018. In Press.

ROS Messages to IEC 61131-3 Data Types

```
1  string first_name
2  string last_name
3  uint8 age
```



```
4  VAR
5      first_name : STRING[100];
6      last_name : STRING[100];
7      age : USINT;
8  END_VAR
```

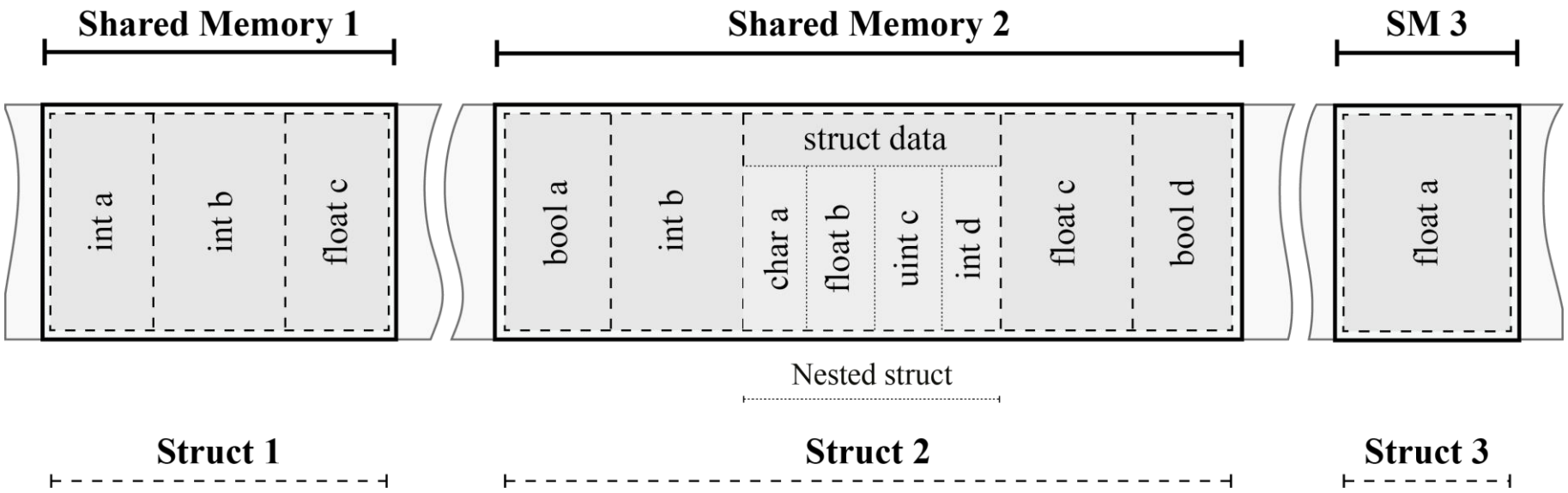
Description	ROS Messages Primitive Type	C++	IEC 61131-3
Unsigned 8-bit Integer	bool	uint8_t	USINT
Signed 8-bit Integer	int8	int8_t	SINT
Unsigned 8-bit Integer	uint8	uint8_t	USINT
Signed 16-bit Integer	int16	int16_t	INT
Unsigned 16-bit Integer	uint16	uint16_t	UDINT
Signed 32-bit Integer	int32	int32_t	DINT
Unsigned 32-bit Integer	uint32	uint32_t	UDINT
Signed 64-bit Integer	int64	int64_t	LINT
Unsigned 64-bit Integer	uint64	uint64_t	ULINT
32-bit IEEE Float	float32	float	REAL
64-bit IEEE Float	float64	double	LREAL
ASCII String	string	std::string	STRING
Time (secs/nsecs)	time	ros::Time	TIME
Time (secs/nsecs)	duration	ros::Duration	TIME

Converted to Bool on CODESYS

Converted to a fixed length array of char on ROS implementation

Mapping data structures

- **ROS Messages** are organized into structures
- **IEC61131-3** also supports structures



T. Pinto, R Arrais and G. Veiga, "Bridging Automation and Robotics: an Interprocess Communication between IEC 61131-3 and ROS", 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), Porto, 2018. In Press.

Implemented ROS Messages

std_msgs:

- Already implemented

common_msgs:

- Almost all implemented
- Problem with variable length arrays on CODESYS

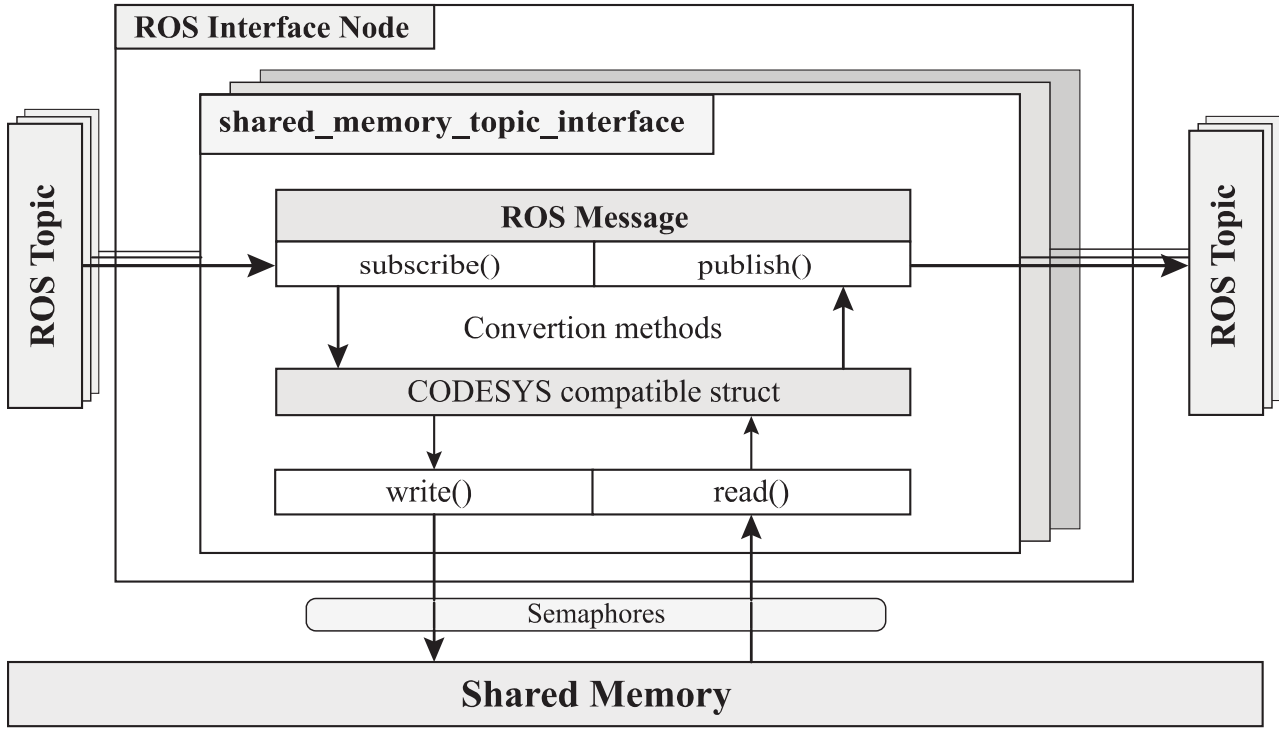
Custom messages:

- Needs to be implemented by the user.
- Will be automated in the future.

ROS StandardMessages	ROS Common Messages		
Bool	**actionlib_msgs**	**geometry_msgs**	**sensor_msgs**
Byte	GoalID	Accel	BatteryState
ByteMultiArray	GoalStatus	AccelWithCovariance	CameraInfo
Char	GoalStatusArray	Inertia	ChannelFloat32
ColorRGBA		Point	CompressedImage
Duration		Point32	FluidPressure
Empty	**diagnostic_msgs**	Polygon	Illuminance
Float32	DiagnosticArray	Pose	Image
Float64	DiagnosticStatus	Pose2D	Imu
Header	KeyValue	PoseArray	JointState
Int16		PoseWithCovariance	Joy
Int32		Quaternion	JoyFeedback
Int64	**nav_msgs**	Transform	LaserEcho
Int8	GridCells	Twist	LaserScan
String	MapMetaData	TwistWithCovariance	MagneticField
Time	OccupancyGrid	Vector3	MultiDOFJointState
UInt16	Odometry	Wrench	MultiEchoLaserScan
UInt32	Path		NavSatFix
UInt64			NavSatStatus
UInt8			PointCloud

Implementation on ROS

- **Topic based** implementation.
- **Automatic shared memory writing** on subscribing callback.
- **Automatic publishing** of the data read from the shared memory.
- Shared Memory access synchronization by semaphores.



Usage on ROS – Writing on the Shared Memory

```
10  PoseStamped_SharedMemory poseStampedSMW("poseStampedWrite_example", WRITE);
11  UInt16_SharedMemory uint16SMW("uint16Write_example", WRITE, PRINT);
12  String_SharedMemory stringSMW("stringWrite_example", WRITE, NOPRINT);
```

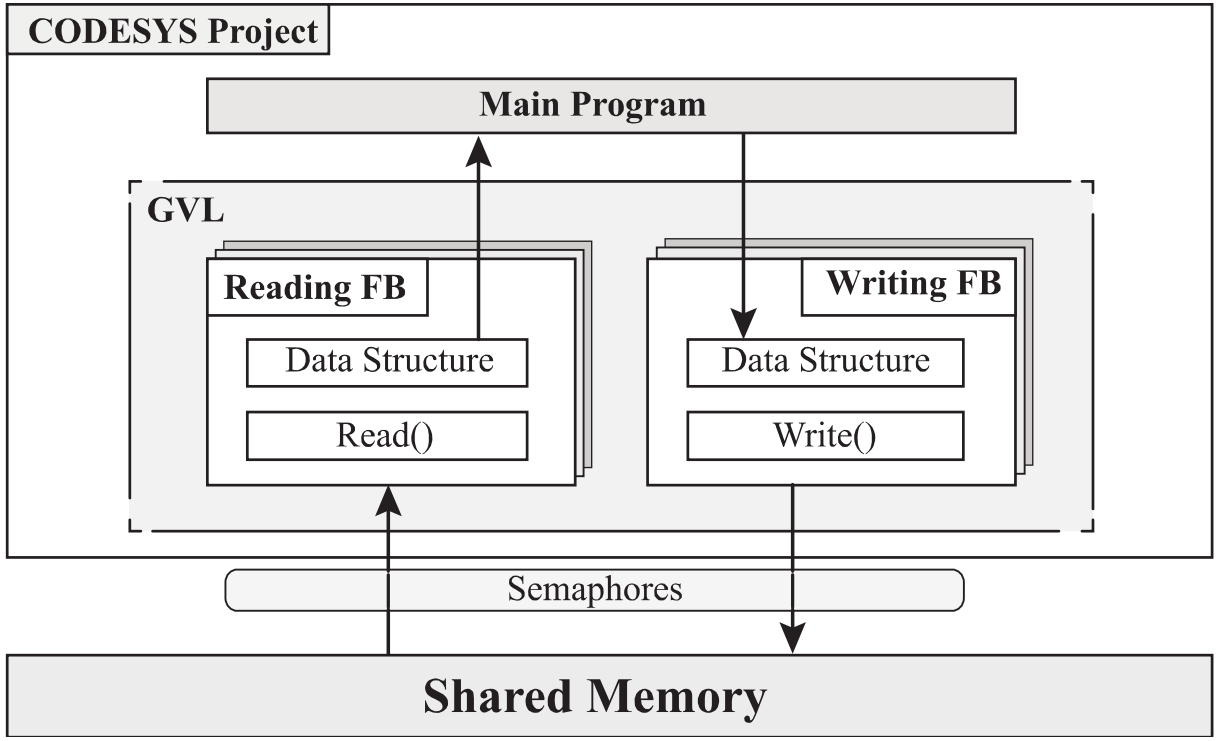
```
14  while (ros::ok())
15  {
16    poseStampedSMW.memoryWrite();
17    uint16SMW.memoryWrite();
18    stringSMW.memoryWrite();
19    ros::spin();
20  }
```

Usage on ROS – Reading from the Shared Memory

```
33  PoseStamped_SharedMemory poseStampedSMR("poseStampedRead_example", READ);
34  Float64_SharedMemory float64SMR("float64Read_example", READ, NOPRINT);
35  Int32_SharedMemory int32SMR("int32Read_example", READ, PRINT);

37  while (ros::ok())
38  {
39    poseStampedSMR.memoryRead();
40    float64SMR.memoryRead();
41    int32SMR.memoryRead();
42
43    ros::spinOnce();
44    loop_rate.sleep();
45  }
```

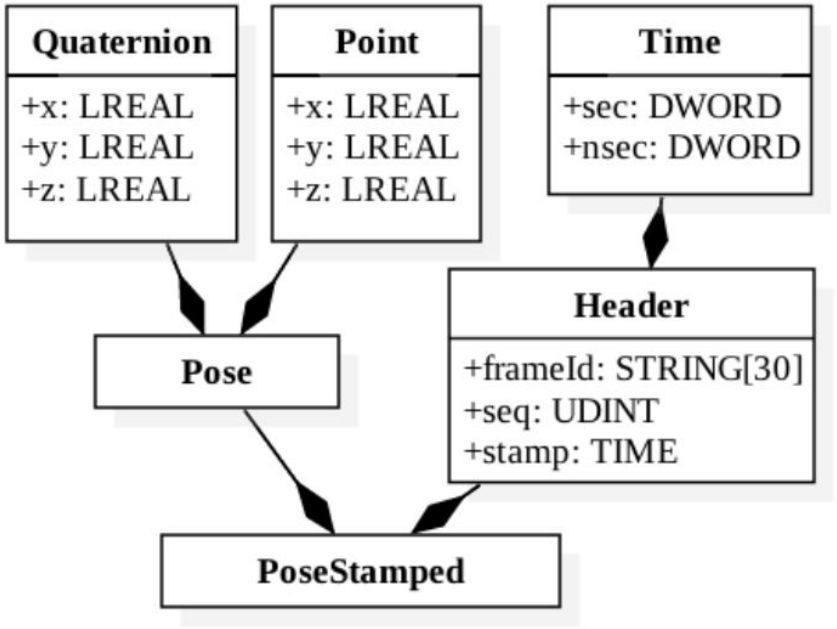
Implementation on CODESYS

- **All variables accessible by GVL.** GVL can be accessed by multiple programs.
- **Automatic update** of the variables.
- Function Blocks for reading and writing **can be used with graphical languages**.
- Shared Memory access synchronization by semaphores.

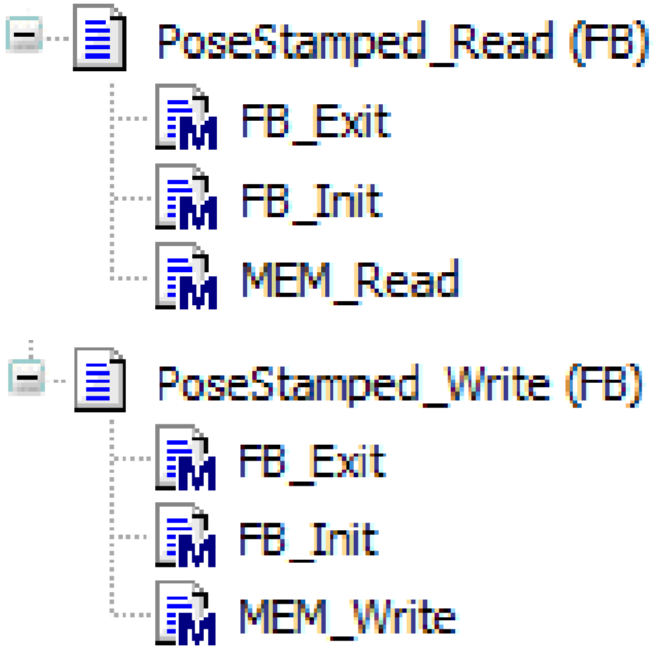


Implementation example on CODESYS

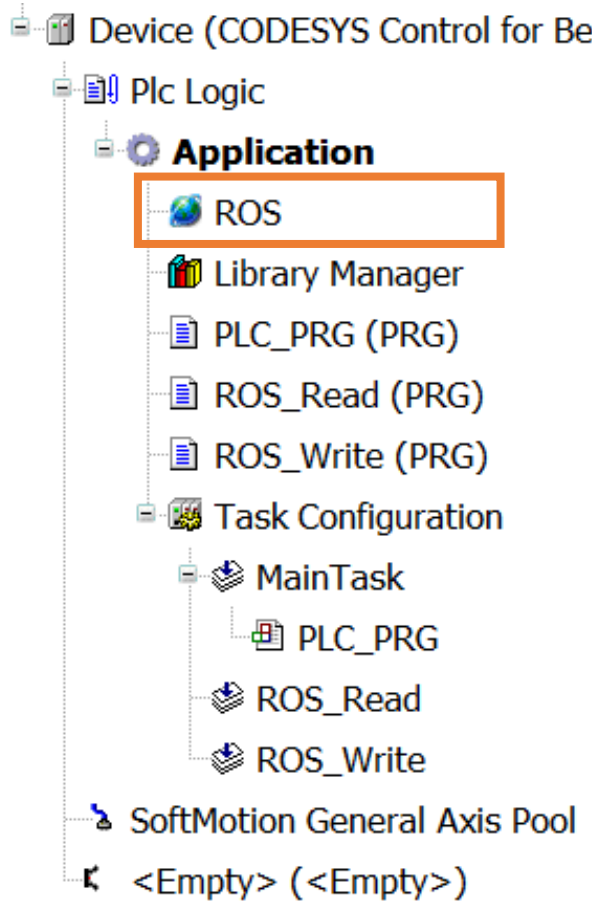
CODESYS Structures



Function Blocks

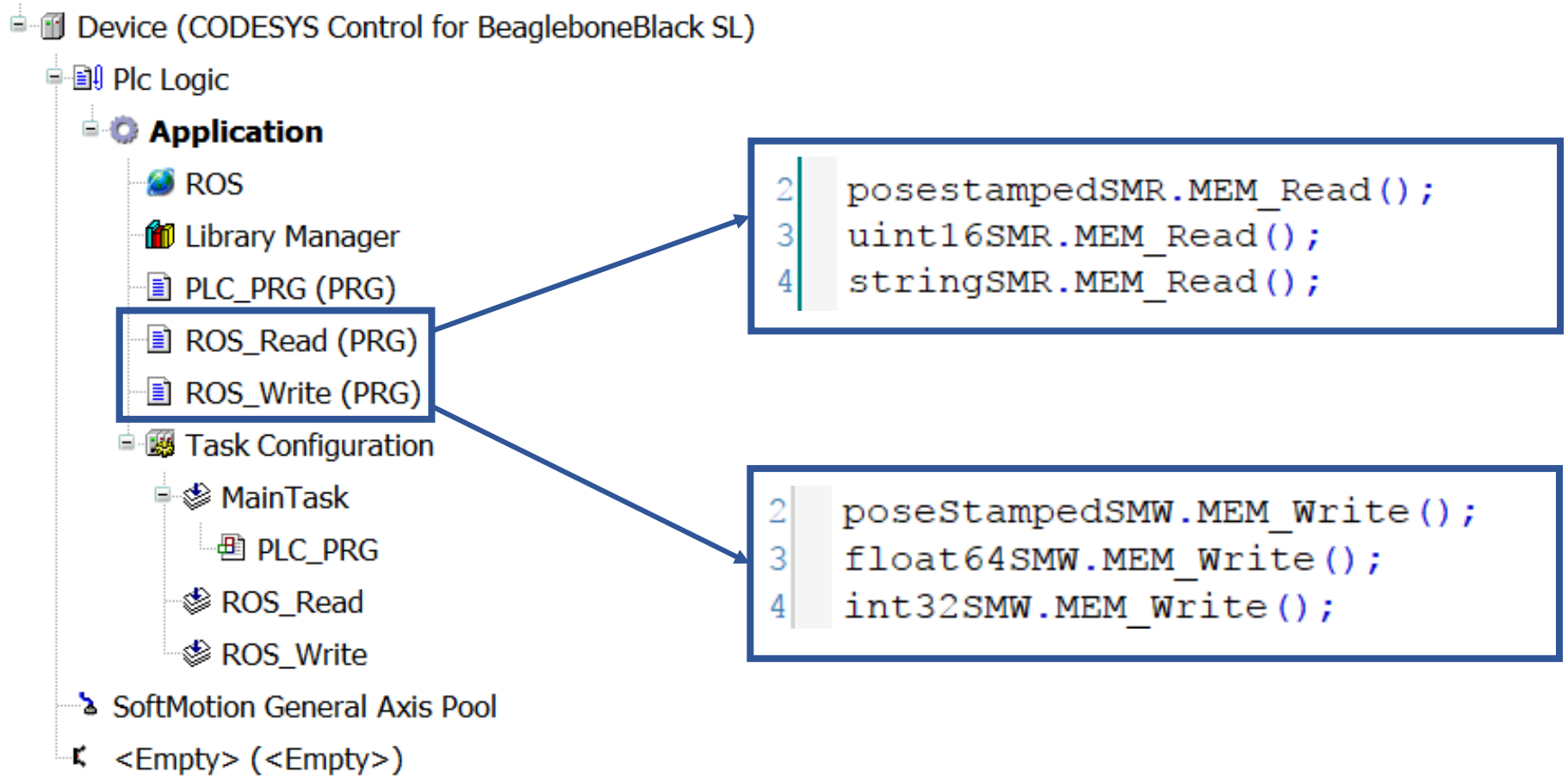


Usage on CODESYS

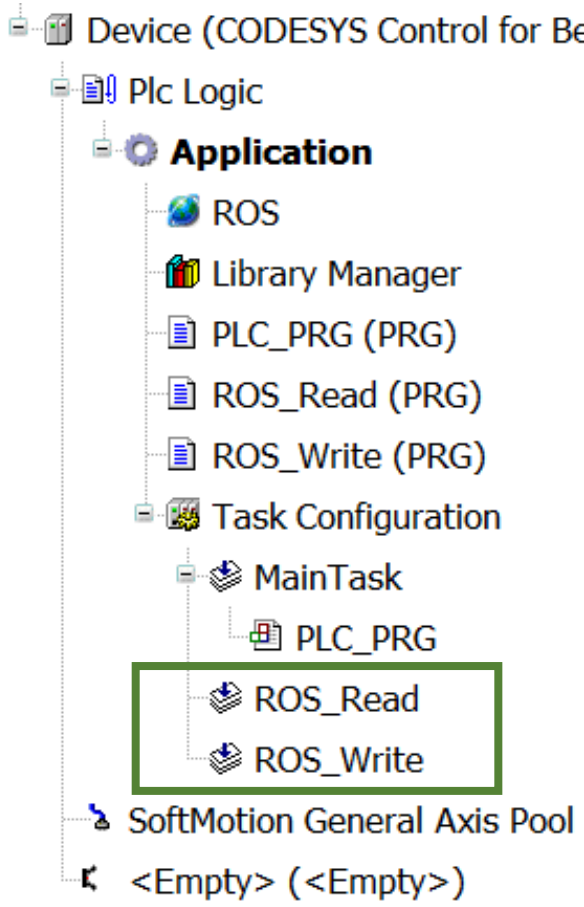



```
3  VAR_GLOBAL
4      poseStampedSMR : PoseStamped_Read('poseStampedWrite_example');
5      uint16SMR : UInt16_Read('uint16Write_example');
6      stringSMR : String_Read('stringWrite_example');
7
8       poseStampedSMW :  PoseStamped_Write('poseStampedRead_example');
9       float64SMW : Float64_Write('float64Read_example');
10      int32SMW : Int32_Write('int32Read_example');
11 END_VAR
```

Usage on CODESYS



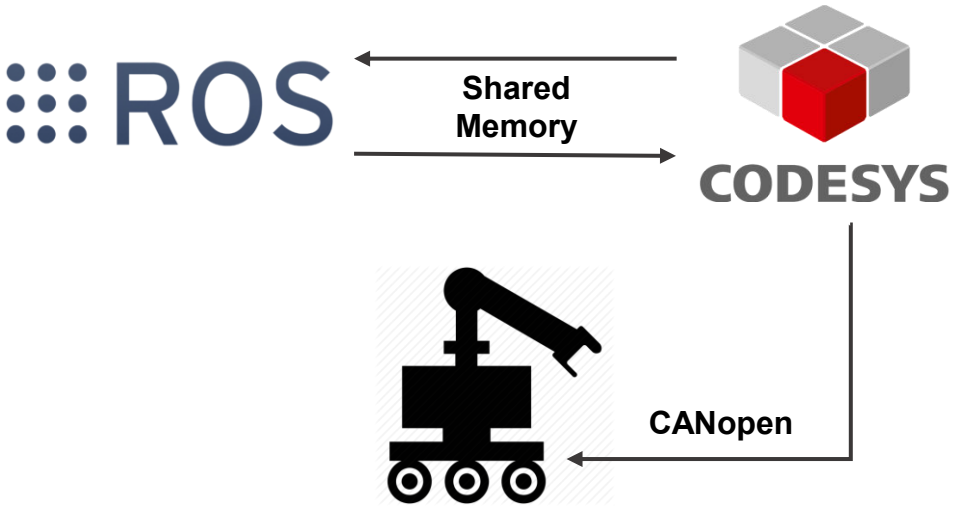
Usage on CODESYS



Practical Applications

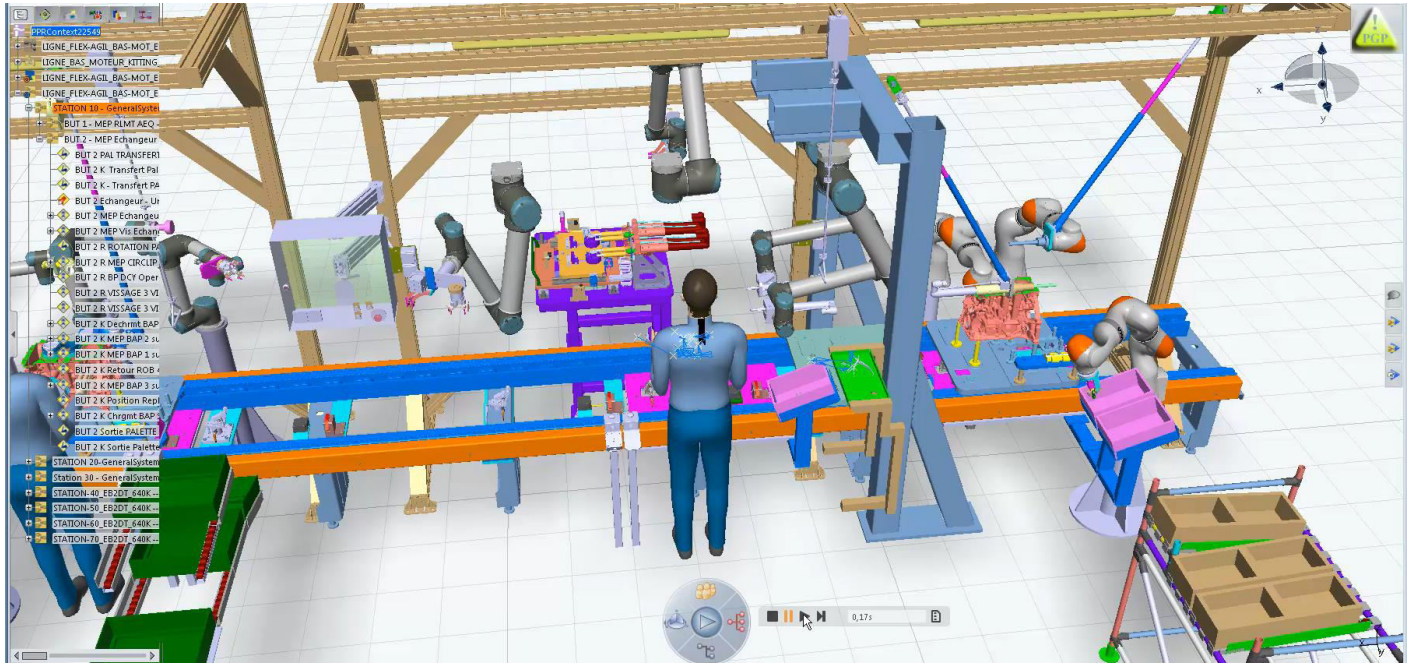
INESC Omnidirectional mobile manipulator

- Using CODESYS to manage the CANopen drivers:



Practical Applications

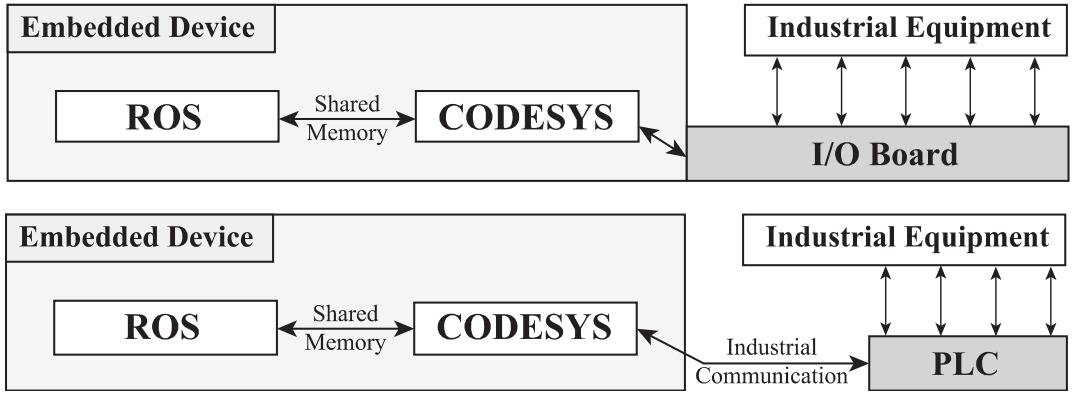
Scalable Project – PSA use case



Overview and Future Work

Overview

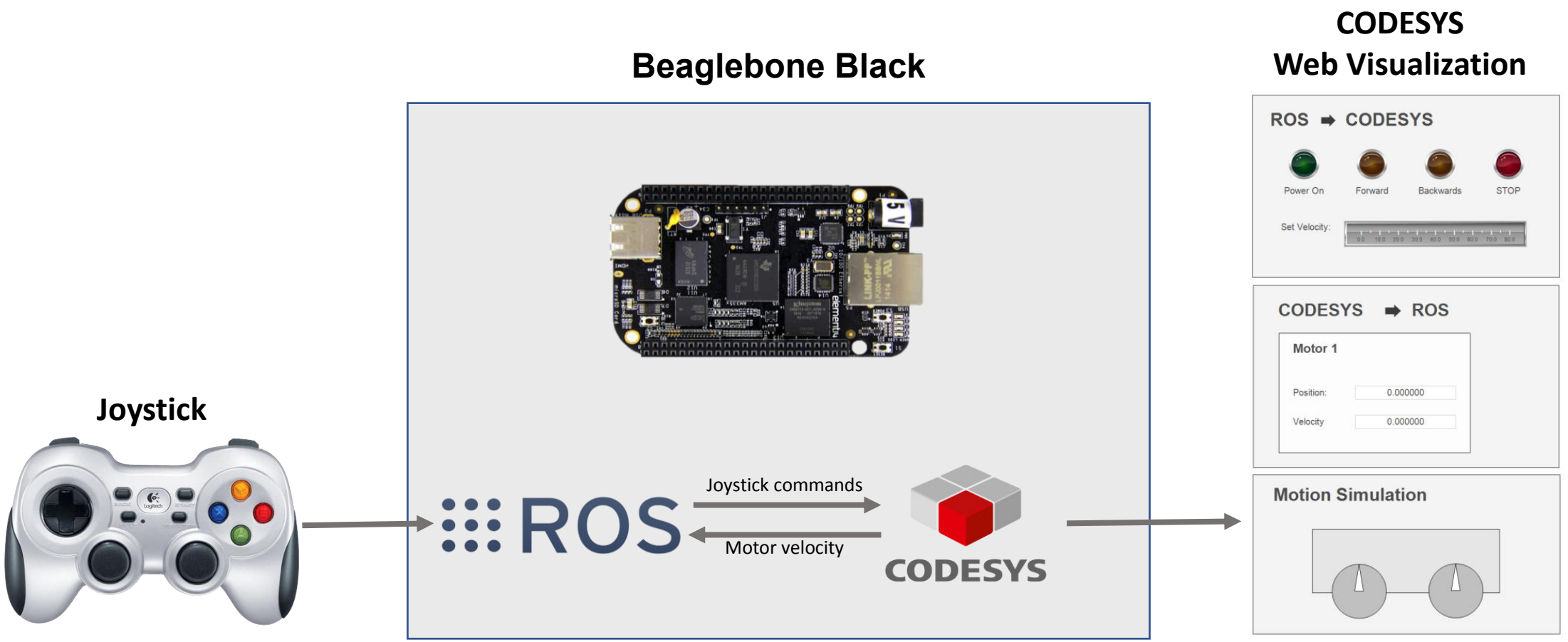
- A communication system between ROS and CODESYS was implemented using **shared memory**.
- Three potential practical apllications:
  1. **Using CODESYS as Communication Bridge**
  2. **Horizontal Integration between ROS and CODESYS**
  3. **Programing robotic tasks in IEC 61131-3 programming languages**



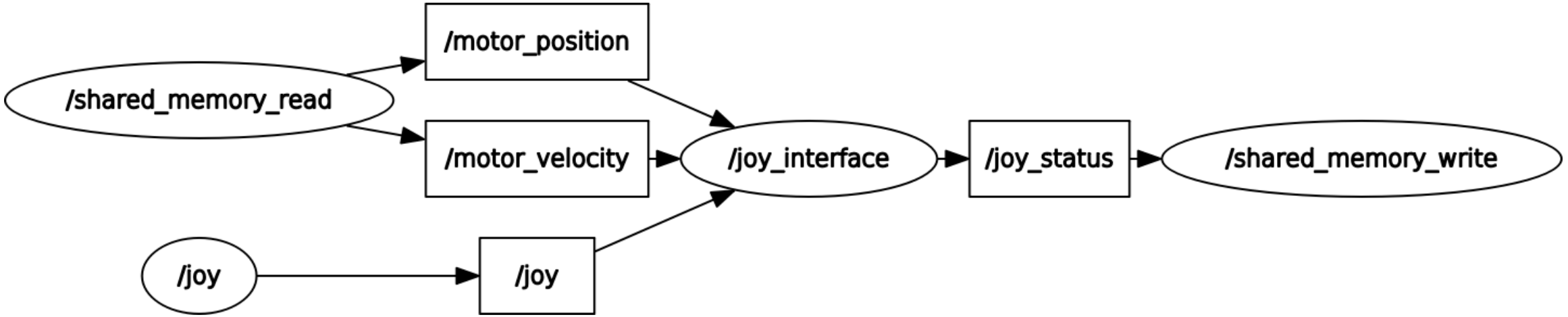
Future Work

- **Simplify the introduction of ROS Custom Messages to be used by this bridge:**
  - A tool will be developed to handle custom messages.
  - On CODESYS, auto generation of code and easy code import will be investigatedImplement a solution to overcome the variable length arrays issue on CODESYS.
- Include **ROS Services and ROS Actions**.

Live Demo


Beaglebone Black
CODESYS
Web Visualization
Joystick
ROS
Joystick commands
Motor velocity
CODESYS
ROS ➡ CODESYS
Power On
Forward
Backwards
STOP
Set Velocity:
CODESYS ➡ ROS
Motor 1
Position:
0.000000
Velocity
0.000000
Motion Simulation

Demo development on ROS



Demo development on ROS

```cpp
#include "ros/ros.h"
#include "codesys_shared_memory/class_headers/Std_Msgs_SharedMemory.h"

int main(int argc, char **argv){
  ros::init(argc, argv, "shared_memory_read");
  Float64_SharedMemory velocity("motor_velocity", READ);
  Float64_SharedMemory position("motor_position", READ);

  while (ros::ok()){
    velocity.memoryRead();
    position.memoryRead();
    ros::spin();
  }
  return 0;
}
```

```cpp
#include "ros/ros.h"
#include "codesys_shared_memory/class_headers/Custom_Msgs_SharedMemory.h"

int main(int argc, char **argv){
    ros::init(argc, argv, "shared_memory_write");
    JoyInterface_SharedMemory joy_interface_write("joy_interface", WRITE);

  while (ros::ok()){
      joy_interface_write.memoryWrite();
      ros::spin();
  }
    return 0;
}

```

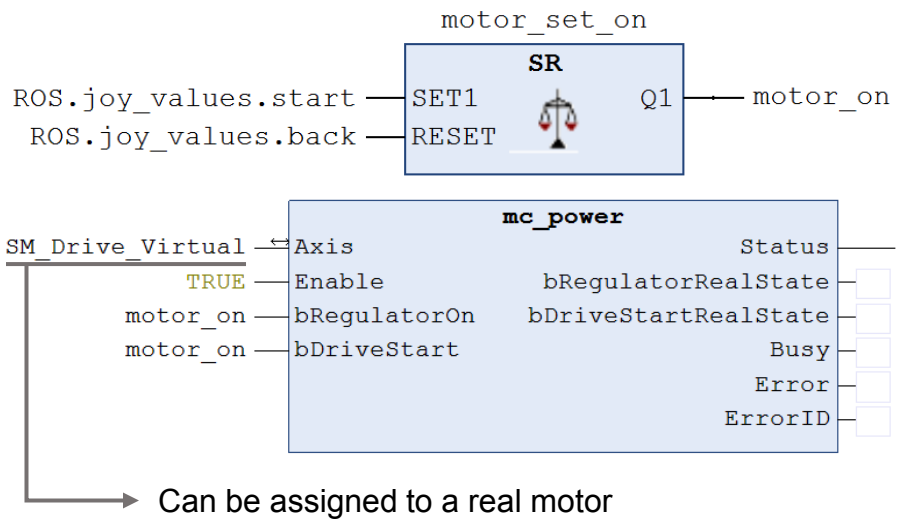
Demo development on CODESYS

```
{attribute 'qualified_only'}
VAR_GLOBAL

    joy_values: JoyInterface_Read('joy_status');

    motorCurrentVelocity : Float64_Write('motor_velocity');
    motorCurrentPosition : Float64_Write('motor_position');

END_VAR
```

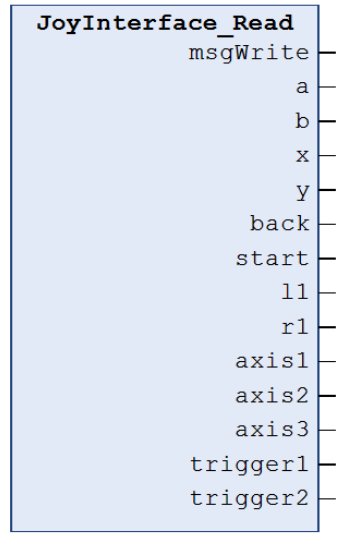
GVL Definition

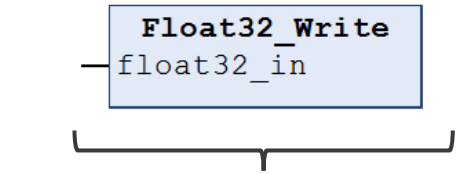
```
ROS.joy_values.MEM_Read();
ROS.motorCurrentVelocity.float64_in := SM_Drive_Virtual.fActVelocity;
ROS.motorCurrentPosition.float64_in := SM_Drive_Virtual.fSetPosition;
ROS.motorCurrentVelocity.MEM_Write();
ROS.motorCurrentPosition.MEM_Write();
```

Shared memory operation program



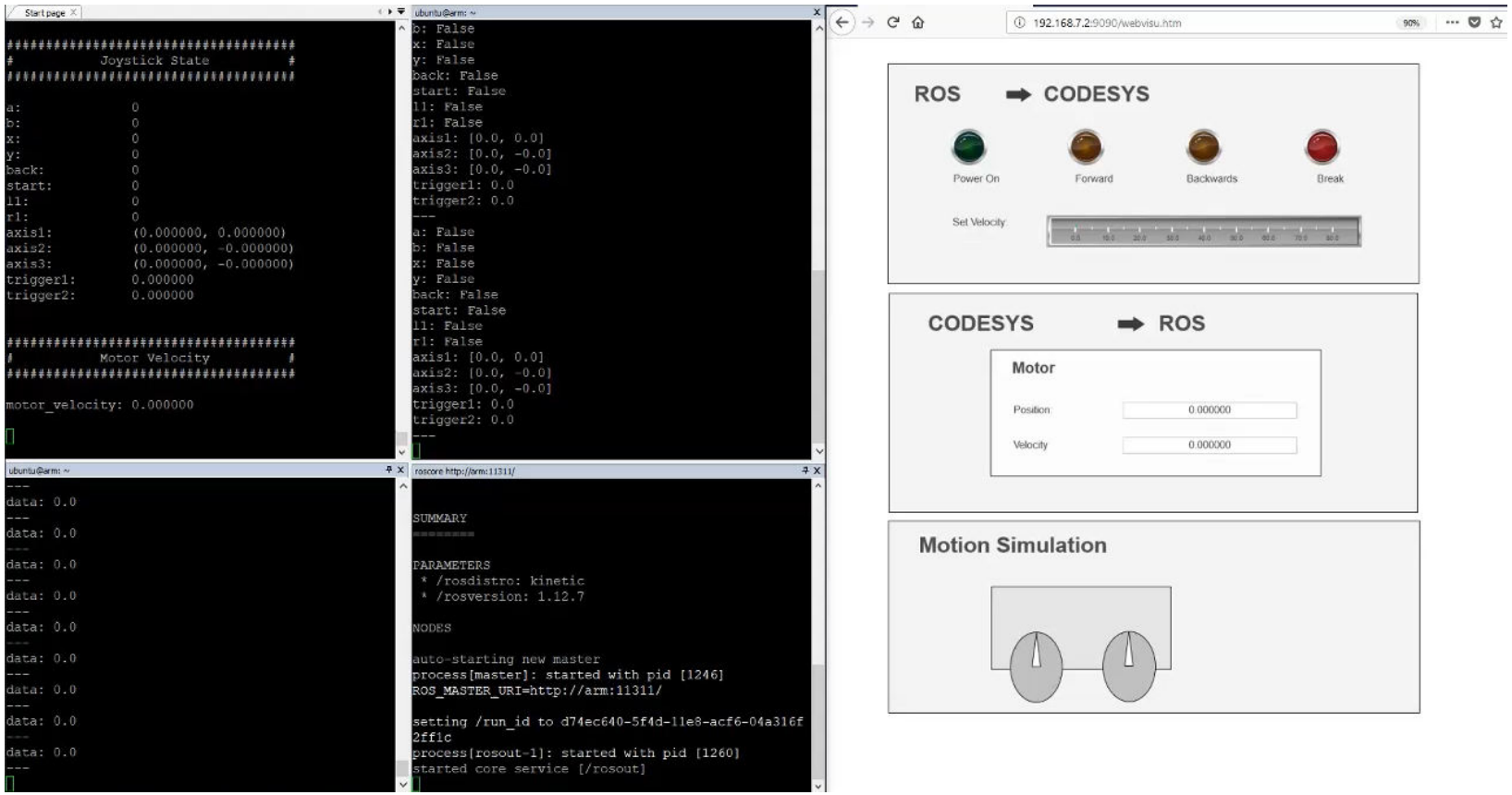
Portion of Motor program FBD





Data can be read or written directly on graphical languages

Demo


Start page
ubuntu@arm: ~
Joystick State
a: 0
b: 0
x: 0
y: 0
back: 0
start: 0
l1: 0
r1: 0
axis1: (0.000000, 0.000000)
axis2: (0.000000, -0.000000)
axis3: (0.000000, -0.000000)
trigger1: 0.000000
trigger2: 0.000000
Motor Velocity
motor_velocity: 0.000000
b: False
x: False
y: False
back: False
start: False
l1: False
r1: False
axis1: [0.0, 0.0]
axis2: [0.0, -0.0]
axis3: [0.0, -0.0]
trigger1: 0.0
trigger2: 0.0
data: 0.0
roscore http://arm:11311/
SUMMARY
PARAMETERS
* /rosdistro: kinetic
* /rosversion: 1.12.7
NODES
auto-starting new master
process[master]: started with pid [1246]
ROS_MASTER_URI=http://arm:11311/
setting /run_id to d74ec640-5f4d-11e8-acf6-04a316f2ffic
process[rosout-1]: started with pid [1260]
started core service [/rosout]
192.168.7.2:9090/webvisu.htm
ROS
CODESYS
Power On
Forward
Backwards
Break
Set Velocity
CODESYS
ROS
Motor
Position
0.000000
Velocity
0.000000
Motion Simulation