

WVU Robotic Technology Center



Assembling Large Infrastructures in Space using Intelligent Robots

Workshop at IROS 2023 in Detroit, MI, USA

October 1, 2023

Hardware-in-the-loop simulation of modular antenna assembly

A. Gailey, G. Marani

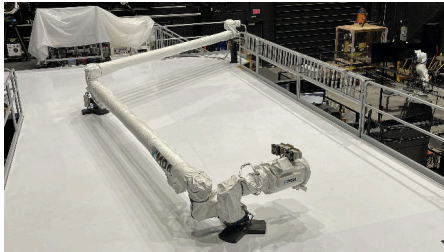
- 1 Preface
 - West Virginia Robotic Technology Center
 - Some of the past and current WVRTC activities
 - Facility: Robot Hardware Systems
- 2 Introduction
 - Hardware-in-the-loop simulation: Objectives
 - Applications
- 3 Hardware-in-the-loop simulation: overview
 - Workspace discretization
 - High level organization
 - Components
- 4 Real-Time Dynamic Simulator
 - System Modeling
 - Dynamics solver
 - Code Generation
 - Real-time simulation of flexibility
- 5 Contact Forces Extractor
 - Isolate contact forces from gravity and inertial forces
 - Contact load example 1
 - Contact load example 2
 - Dynamic Parameters Estimation
- 6 Demonstrations
 - Demo 1: testing the robot flexibility
 - Demo 2: assembling parts
- 7 Arm Programming Language
 - Overview
 - Client-Server architecture of APL
- 8 Conclusions

1 - Preface

West Virginia Robotic Technology Center



Located in Fairmont, WV, the West Virginia University Robotic Technology Center is a world class space simulation facility supporting multiple tasks in the field of In-Space Servicing, Assembly, and Manufacturing



1 - Preface

Facility: Robot Hardware Systems

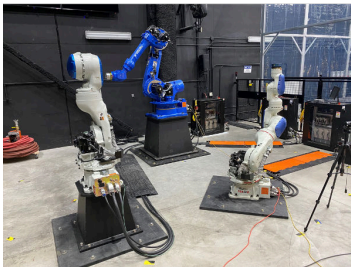


- Yaskawa Manipulators:

- Yaskawa Motoman SIA10D
- Yaskawa Motoman SIA20D
- Yaskawa Motoman SIA50D (3x)
- Yaskawa Motoman MH250

- Mikrolar Platforms:

- Mikrolar R-3000 Rotopod (2x)



2 - Introduction

Hardware-in-the-loop simulation: Objectives



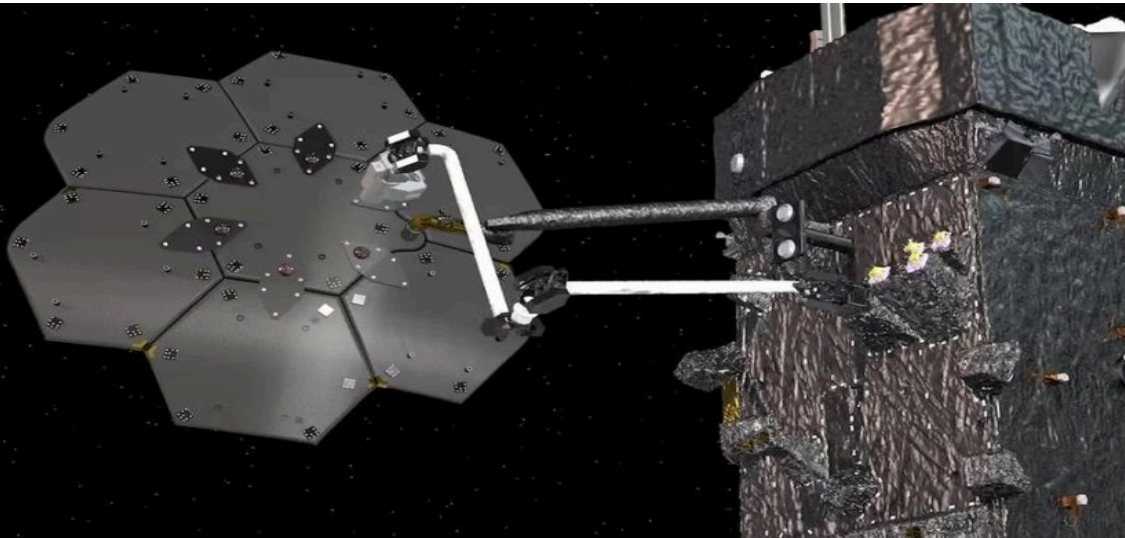
- Hybrid simulation of a generic robotic space intervention system in 1-g environment with:
 - Actual hardware for manipulation task with a suitable industrial robot
 - A real-time dynamic simulator for simulating the space manipulator kinematics and dynamics, in its own space environment (on-orbit, zero-g, etc.)
- Possibility of choosing which capabilities of the space robotic system are to be simulated or replicated in hardware
- Reduce cost of on-Earth test/validation/verification of a space robotic systems by pushing to real-time dynamic simulations the aspects difficult to replicate on Earth (e.g. simulating the effect of thrust on a long flexible manipulator)
- Perform a parameterized performance study of a space robotic mission
- Discover the benefits, limitations and sensitivities of the robotic system to known performance contributors such as properties of the robot controller and sensors
- Verify the performance sensitivity to key hardware and software characteristics.
- Earlier steps: Use Monte Carlo simulation methods to predict robot arm performance against selected hardware and software characteristics
- Later steps: validate these Monte Carlo predictions on robot hardware

2 - Introduction

Applications



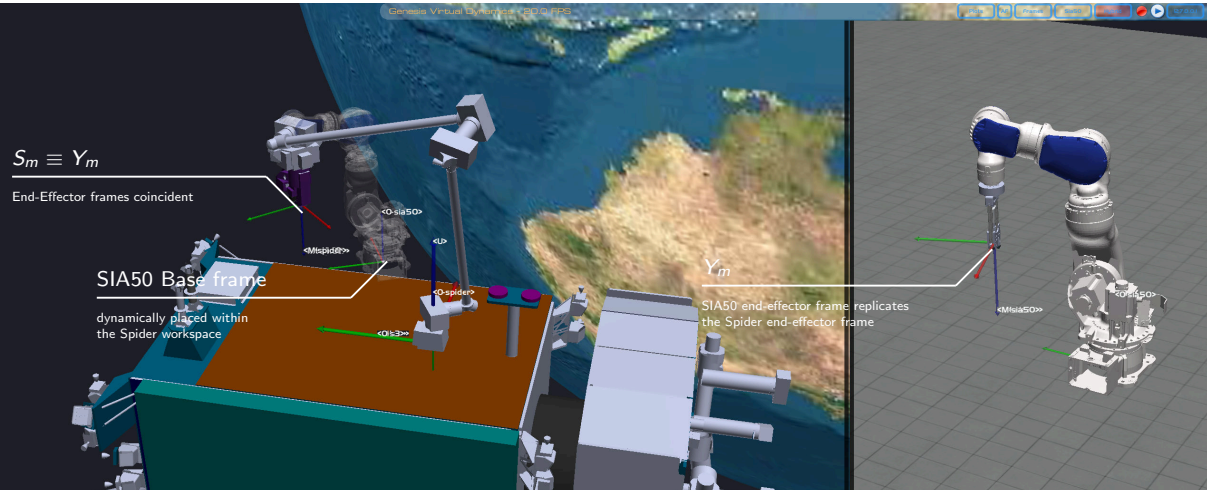
The presented HIL approach is currently used to perform IV&V of the SPIDER mission



3 - Hardware-in-the-loop simulation: overview

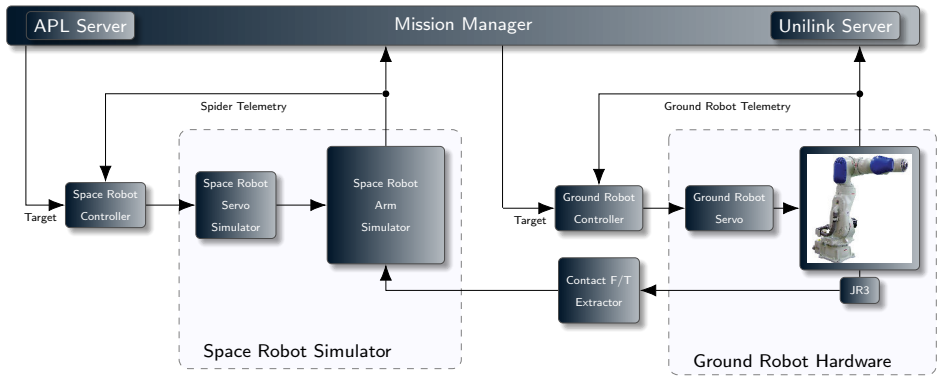
Workspace discretization

- The smaller workspace of the industrial manipulator operates in a subset of the larger space robot workspace
- The selected space robot workspace volume is dynamically changed according to the task



3 - Hardware-in-the-loop simulation: overview

High level organization



Principal components of the hybrid loop

Dynamic Simulator

- Real-time simulator with elasticity

Contact forces extractor

- Dynamic parameters estimation

Arm Programming Language (APL)

- Advanced Task Description Language for robotic systems

Advanced cartesian controller

- Kinematic and algorithmic singularities avoidance
- Joint limits optimization

Industrial Robot Controller

- Real-time link with servo drives
- Synchronous Ethercat Drivers for minimal latency

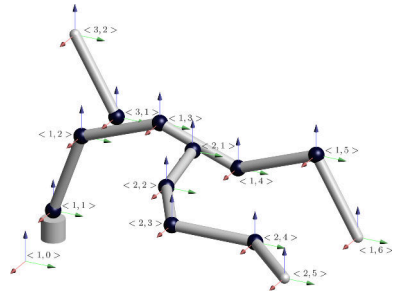
Statistical Analysis

- Accelerated multi-thread simulation for Monte Carlo analysis
- Parameters sensitivity analysis

The Plasma library

- Geometric, kinematic and dynamic modeling of a multi-DOF general multibody system (ramified chain)^a
- Automatic C++ code generation from model description using Maple
- Iterative optimization

^aTheoretical details available in *Marani, Yuh: Introduction to Autonomous Manipulation, Springer, 2014, Sec. 3.*



Dynamic model:

- Lagrange equation for quasi-coordinates¹:

$$\mathbf{A}(\mathbf{q})\dot{\mathbf{p}} + \mathbf{B}(\mathbf{q}, \mathbf{p})\mathbf{p} + \mathbf{V}^T \mathbf{C}(\mathbf{q}) = \mathbf{V}^T \boldsymbol{\mu}^e \quad (1)$$

where:

- $\mathbf{q}$ is the configuration vector
- $\mathbf{p}$ is the quasivelocity vector
- The matrix $\mathbf{A}$ is known as the *inertia matrix* of the structure.
- $\mathbf{B}(\mathbf{q}, \mathbf{p})$ is the Coriolis-centrifugal matrix
- The quantity $\mathbf{V}^T \mathbf{C}(\mathbf{q})$ represents the projection in the joint velocity space of the gravitational forces
- $\mathbf{V}^T \boldsymbol{\mu}^e$ is the projection in the joint velocity space of all the other external forces (for example the actuator forces)

Numeric solver:

- Custom C++ implementation of Runge-Kutta 4

¹Marani, Yuh: Introduction to Autonomous Manipulation, Springer, 2014, Sec. 2. 

Code Generation



```

15 # Link data:=
16 # [m]
17 # [cm]
18 # [cm]
19 # [center-of-mass],m
20 # [outboard joint],m
21 # mass,m
22 # [inertia-matrix]=
23 # ],m
24 # ... (repeated for each link)=
25 # ],m
26 # ... (repeated for each branch)=
27 # ]=
28 Bodylist :=m
29 [m]
30 [m]
31 [m]
32 [L1_cm(0), L1_cm(1), L1_cm(2)],m
33 [0, 0, 1],m
34 bodyMass,m
35 matrix[[bodyInertia(0,0), bodyInertia(0,1), bodyInertia(0,2)],m
36 [bodyInertia(1,0), bodyInertia(1,1), bodyInertia(1,2)],m
37 [bodyInertia(2,0), bodyInertia(2,1), bodyInertia(2,2)]=m
38 ]m
39 ]m
40 ]m
41 ]=
42 # Joint matrices=
43 # [ [ [joint-dof], [joint-matrix] ], ... (repeated for each branch) ]=
44 Jointlist :=m
45 [m]
46 [0], matrix[[1, 0, 0, 0, 0, 0],m
47 [0, 1, 0, 0, 0, 0],m
48 [0, 0, 1, 0, 0, 0],m
49 [0, 0, 0, 1, 0, 0],m
50 [0, 0, 0, 0, 1, 0],m
51 [0, 0, 0, 0, 0, 1, 0],m
52 [0, 0, 0, 0, 0, 0, 1 ]]=m
53 ]m
54 ]=

```

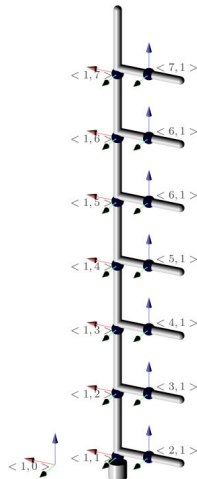
[illegible]

Several thousands lines of code!!!
The strong iterative automatic optimization makes it possible to run the dynamics at more than 30kHz

²Giacomo Marani: ROBOSIM: a program for simulating mechanical systems in Robotics (in Italian)

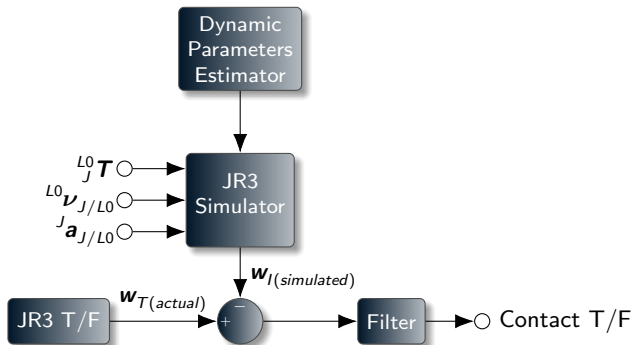
Flexibility with augmented model (+ 1 DoF per joint)

- Elasticity is reduced at the joint level, along the torque transmission path between the motor and the joint
- Dynamic model is augmented by adding extra branches
 - This is equivalent to attaching each motor to the original joint through a spring/mass/damper
 - Total degrees of freedom doubles
- The "spring" acts only on the axis of the motor; every other part of the structure is supposed rigid
- This model assumes the dominant action is the elasticity of the torque transmission between the motor and the joint
- This model requires only $50\mu s$ to perform one update step on a 7-DoF robot



5 - Contact Forces Extractor

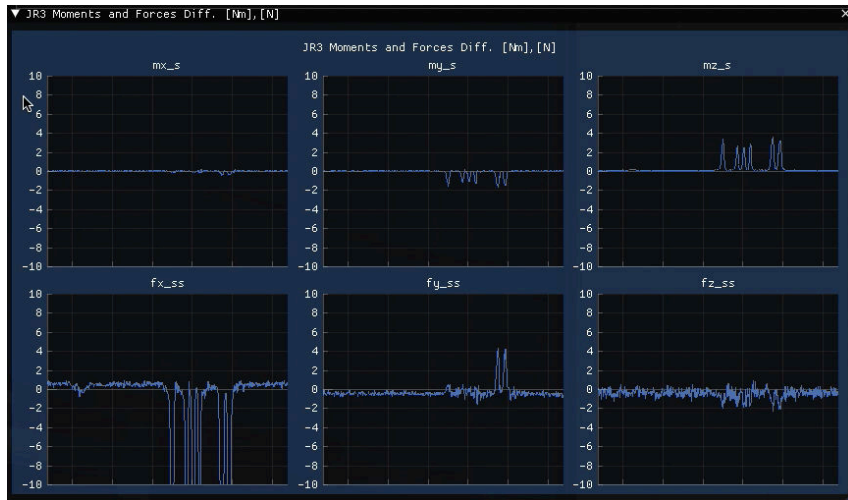
Isolate contact forces from gravity and inertial forces



- Generalized position of JR3 frame ${}^{L0}T_J$ is computed using joint servo data
- Generalized velocity of JR3 frame ${}^{L0}\nu_{J/L0}$ is computed using joint servo data
- Generalized acceleration of JR3 frame ${}^J a_{J/L0}$ is currently computed with numerical differentiation of a filtered version of ${}^{L0}\nu_{J/L0}$. This is being replaced with the acceleration data read directly from the JR3 sensor.

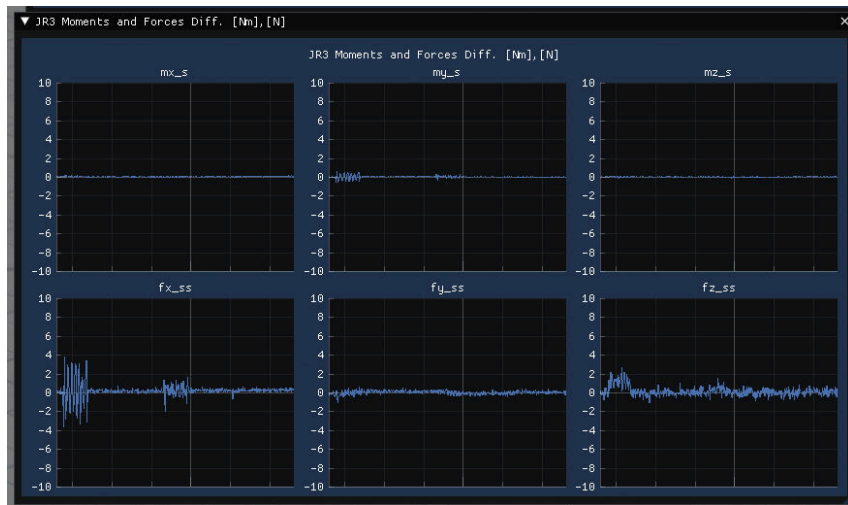
5 - Contact Forces Extractor

Contact load example 1



5 - Contact Forces Extractor

Contact load example 2



5 - Contact Forces Extractor

Dynamic Parameters Estimation



The Force/Torque sensor simulator needs to know the dynamic parameters of its load.

Dynamic Parameters Estimator objectives:

- Identifies dynamic parameters of a robot end-effector
- Estimates velocities and accelerations of the end effector center of mass using an Extended Kalman Filter (EKF)
- Estimates the externally applied forces and moments acting on the end effector using the EKF
- Primary use: contact forces isolation in a 1-G environment for the HIL simulator

How it works:

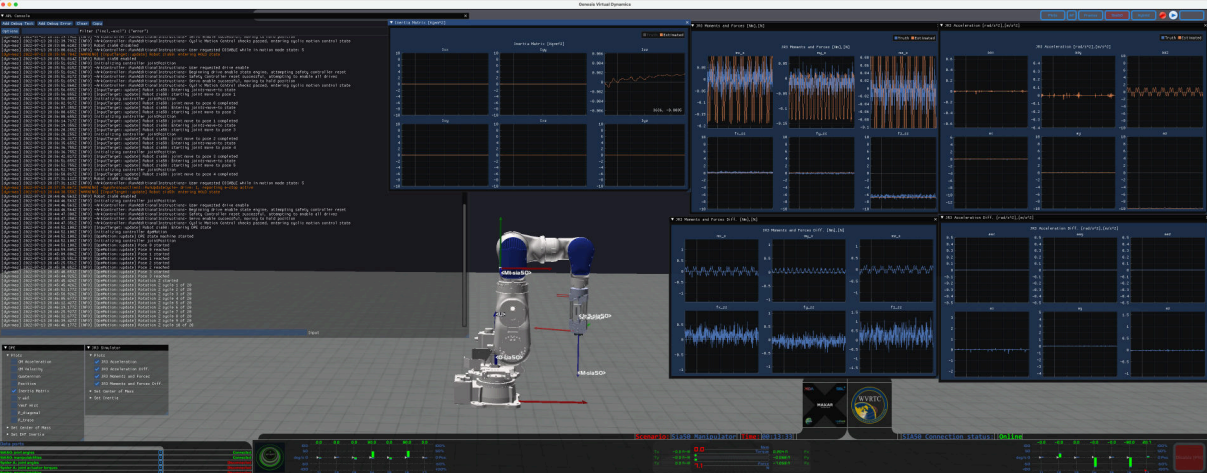
- Inputs: 12-DoF sensor data (linear/angular accelerations, forces and moments)
- Inputs: end effector pose w.r.t the robot base, which is provided by the robot controller
- A mathematical model of the end effector dynamics developed using Maple CAS provides a solution for the F/T sensor inertial forces
- Simulated inertial forces are subtracted from the F/T readings to isolate the contact forces

5 - Contact Forces Extractor

Dynamic Parameters Estimation



The DPE interface



5 - Contact Forces Extractor

Dynamic Parameters Estimation

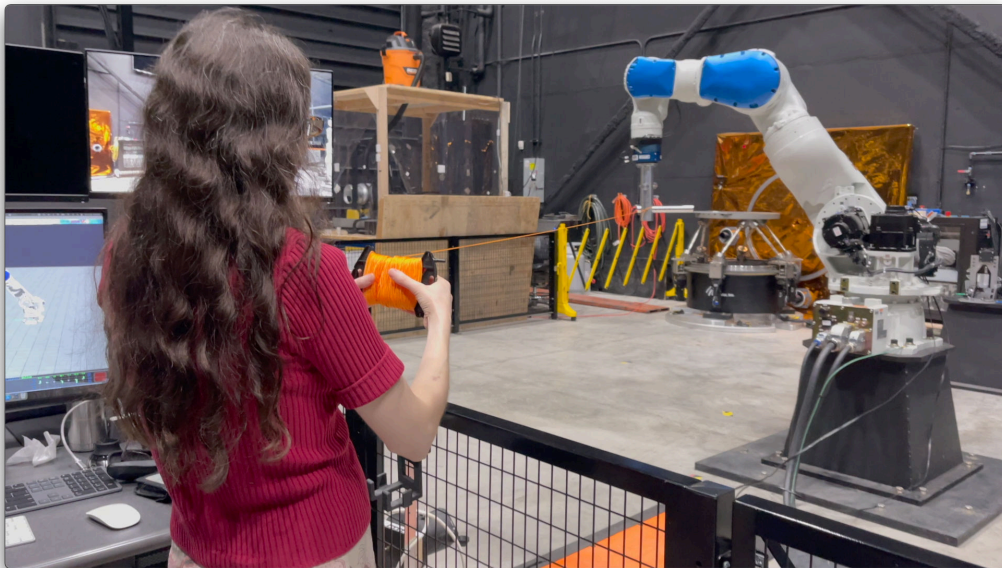


Example of parameters convergency:



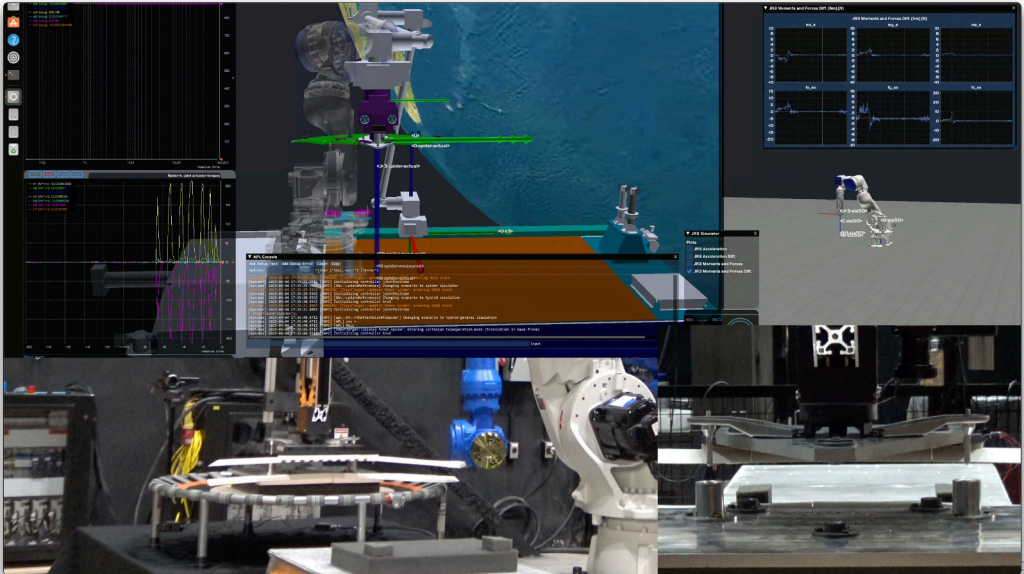
6 - Demonstrations

Demo 1: testing the robot flexibility



6 - Demonstrations

Demo 2: assembling parts



7 - Arm Programming Language

Overview



- Advanced Task Description Language for robotic systems

```
Notes:-
1. sthEmpty: preprocessor-
   handler for do-nothing function with 0 arguments.-
2. sthEmpty: preprocessor-
   handler for do-nothing function with 1 expression argument.-
3. sthEmpty: preprocessor-
   handler for do-nothing function with 2 expression arguments.-
4. rthEmpty: run-time-
   handler for do-nothing function (no push, no pop)-
5. ~: handler for do-nothing function (no push, no pop)-
6. ~: 1 arguments means variable number of arguments.-
7. ~: ~-

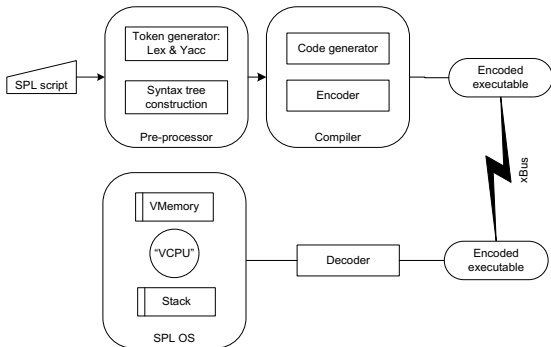
builtinFncs[i++] = {
    "q-", 1, ABf::sthEmpty, ABf::sthEmpty } // 000
builtinFncs[i++] = {
    "q+", 2, ABf::sthPush, ABf::sthPush } // 001
builtinFncs[i++] = {
    "q-", 2, ABf::sthMinus, ABf::sthEmpty } // 002
builtinFncs[i++] = {
    "q+", 2, ABf::sthMul, ABf::sthMul } // 003
builtinFncs[i++] = {
    "q-", 2, ABf::sthDiv, ABf::sthEmpty } // 004
builtinFncs[i++] = {
    "sin", 1, ABf::sthSin, ABf::sthSin } // 005
builtinFncs[i++] = {
    "cos", 1, ABf::sthCos, ABf::sthCos } // 006
builtinFncs[i++] = {
    "eval", 1, ABf::sthEval, ABf::sthEval } // 007
builtinFncs[i++] = {
    "sig", 2, ABf::sthAssign, ABf::sthAssign } // 008
builtinFncs[i++] = {
    "eval", 1, ABf::sthEmpty, ABf::sthEmpty } // 009
builtinFncs[i++] = {
    "m+", 2, ABf::sthEmpty, ABf::sthMul } // 010
builtinFncs[i++] = {
    "print", 1, ABf::sthPrint, ABf::sthPrint } // 011
builtinFncs[i++] = {
    "type", 2, ABf::sthType, ABf::sthType } // 012
builtinFncs[i++] = {
    "error", 1, ABf::sthError, ABf::sthError } // 013
builtinFncs[i++] = {
    "coldin", 1, ABf::sthEmpty, ABf::sthColdin } // 014
builtinFncs[i++] = {
    "rowdin", 1, ABf::sthEmpty, ABf::sthRowdin } // 015
builtinFncs[i++] = {
    "power", 2, ABf::sthPower, ABf::sthPower } // 016 (193)
builtinFncs[i++] = {
    "heat", 1, ABf::sthHeat, ABf::sthHeat } // 017 (272)
builtinFncs[i++] = {
    "m+", 2, ABf::sthMul, ABf::sthMul } // 018 (293)
builtinFncs[i++] = {
    "m+", 2, ABf::sthGreater, ABf::sthGreater } // 019 (213)
builtinFncs[i++] = {
    "m+", 2, ABf::sthEq, ABf::sthEq } // 020 (163)
builtinFncs[i++] = {
    "m+", 2, ABf::sthLess, ABf::sthLess } // 021 (183)
builtinFncs[i++] = {
    "m+", 2, ABf::sthOrEq, ABf::sthOrEq } // 022 (193)
builtinFncs[i++] = {
    "mot", 1, ABf::sthMot, ABf::sthMot } // 023 (163)
builtinFncs[i++] = {
    "and", 2, ABf::sthAnd, ABf::sthAnd } // 024 (173)
builtinFncs[i++] = {
    "or", 2, ABf::sthOr, ABf::sthOr } // 025 (183)
builtinFncs[i++] = {
    "op", 2, ABf::sthOp, ABf::sthOp } // 026 (183)
builtinFncs[i++] = {
    "matrix", 1, ABf::sthMatrix, ABf::sthMatrix } // 027 (193)
builtinFncs[i++] = {
    "eval", 1, ABf::sthEmpty, ABf::sthEval } // 028 (183)
builtinFncs[i++] = {
    "moveJointsto",
    1,
    ABf::sthEmpty,
    ABf::sthMoveJointsto
    } // 29 (186)
}
```

```
86 ~ builtinFncs[i++] = {
87 ~ ~ ~ ~ ~ "moveMfTo",
88 ~ ~ ~ ~ ~ 1,
89 ~ ~ ~ ~ ~ &Bf::sthEmpty1,
90 ~ ~ ~ ~ ~ &Bf::rthMoveMfTo
91 ~ ~ ~ ~ ~ }; ~ // 30: (14)~
92 ~ builtinFncs[i++] = {
93 ~ ~ ~ ~ ~ "isCsmRunning",
94 ~ ~ ~ ~ ~ 0,
95 ~ ~ ~ ~ ~ &Bf::sthEmpty0,
96 ~ ~ ~ ~ ~ &Bf::rthIsCsmRunning
}
```

```
11 ~
12 ~ ~ ~ ~ ~ // MARK: --- ST and RT functions additional handlers -----
13 ~
14 ~ ~ ~ ~ ~ // Last value of base class built-in functions counter:-
15 ~ ~ ~ ~ ~ // bcc += 29~
16 ~
17 ~ ~ ~ ~ ~ // Fcn. bcc += 1: enableDpe~
18 ~ ~ ~ ~ ~ rtFcnHandlerType rthEnableDpe;~
19 ~
20 ~ ~ ~ ~ ~ // Fcn. bcc += 2: disableDpe~
21 ~ ~ ~ ~ ~ rtFcnHandlerType rthDisableDpe;~
22 ~
23 ~ ~ ~ ~ ~ // Fcn. bcc += 3: attachSia50ToSpider~
24 ~ ~ ~ ~ ~ rtFcnHandlerType rthAttachSia50ToSpider;~
25 ~
26 ~ ~ ~ ~ ~ // Fcn. bcc += 4: setKeyboardTeleoperationLinearVelocity~
27 ~ ~ ~ ~ ~ rtFcnHandlerType rthSetKeyboardTeleoperationLinearVelocity;~
28 ~
29 ~ ~ ~ ~ ~ // Fcn. bcc += 5: setKeyboardTeleoperationAngularVelocity~
30 ~ ~ ~ ~ ~ rtFcnHandlerType rthSetKeyboardTeleoperationAngularVelocity;~
31 ~
32 ~ ~ ~ ~ ~ // Fcn. bcc += 6: enableSia50SimulatedCollisionCylinder~
33 ~ ~ ~ ~ ~ rtFcnHandlerType rthEnableSia50SimulatedCollisionCylinder;~
34 ~
35 ~ ~ ~ ~ ~ // Fcn. bcc += 7: selectSia50~
36 ~ ~ ~ ~ ~ rtFcnHandlerType rthSelectSia50;~
37 ~
38 ~ ~ ~ ~ ~ // Fcn. bcc += 8: setSpiderServoLoopPGains~
39 ~ ~ ~ ~ ~ rtFcnHandlerType rthSetSpiderServoLoopPGains;~
40 ~
41 ~ ~ ~ ~ ~ // Fcn. bcc += 9: setSpiderServoLoopDGains~
42 ~ ~ ~ ~ ~ rtFcnHandlerType rthSetSpiderServoLoopDGains;~
43 ~
44 ~ ~ ~ ~ ~ // Fcn. bcc += 10: setSpiderElJointViscousFrictions~
45 ~ ~ ~ ~ ~ rtFcnHandlerType rthSetSpiderElJointViscousFrictions;~
46 ~
```

7 - Arm Programming Language

Client-Server architecture of APL



- High-level programming language for **Semi-Autonomous operations** written in C++
 - Grammar compiled via Flex/Bison
- Client/Server architecture connected through unilink
 - Client resides in the GUI, using console input or script file
- Virtual CPU for sampled-mode execution
- Basic symbolic manipulation
- Customizable to many different real-time robot controllers

- Investigated feasibility of an hybrid hardware-in-the-loop simulation environment of a generic robotic space intervention system in 1-g environment with:
 - Actual hardware for manipulation task with a suitable industrial robot
 - A real-time dynamic simulator for simulating the space manipulator kinematics and dynamics, in its own space environment (on-orbit, zero-g, etc.)
- Major challenge: implementation of real-time dynamics
- Approach seems promising; this solution allowed us assess several possible problems such as tele-manipulation in zero-g in case of large and flexible manipulators
- Cost of the simulation environment was greatly reduced with respect the adoption of large mechanical platforms
- The optimized implementation is suitable for advanced statistical analysis such as a parameter sensitivity analysis that requires a very large number of simulation runs

Thank you