

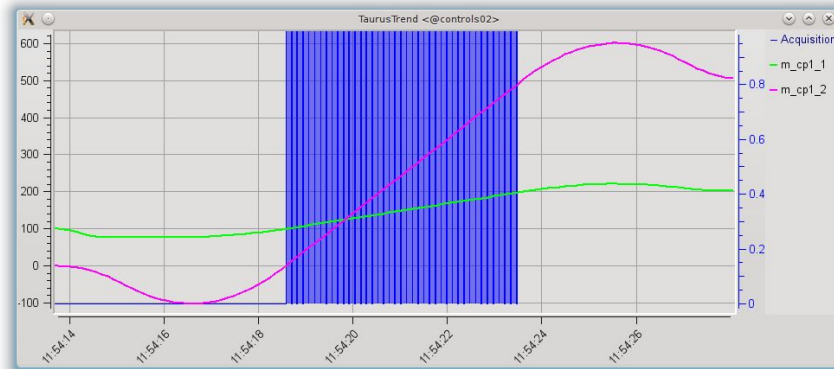
Sardana Trajectories: Preliminary Core Design for Synchronized Motion and Acquisition

Jordi Aguilar
Roberto Homs
Zbigniew Reszela
Oriol Vallcorba

on behalf of
ALBA Synchrotron
Controls Section

Pseudo-motors limitations

- Do not implement velocity and acceleration
- Sardana continuous scan framework implements constant velocity for all physical motors of one pseudo-motor, does not follow the trajectory.
- Pseudo-motors do not implement movement logic, only calculation.



Objective: Support different Motion Controller Hardware



Aerotech
Delta-tau Pmac
Smaract MCS2
Icepap
Beckhoff

.....
Support: Virtual Axis, PVT/PT, G-Code.



Sardana core requirements

- New motion states
- New attributes/method for configuration and execution on the Motor controller
- New motion loop action
- Adapt Generic Scan Framework

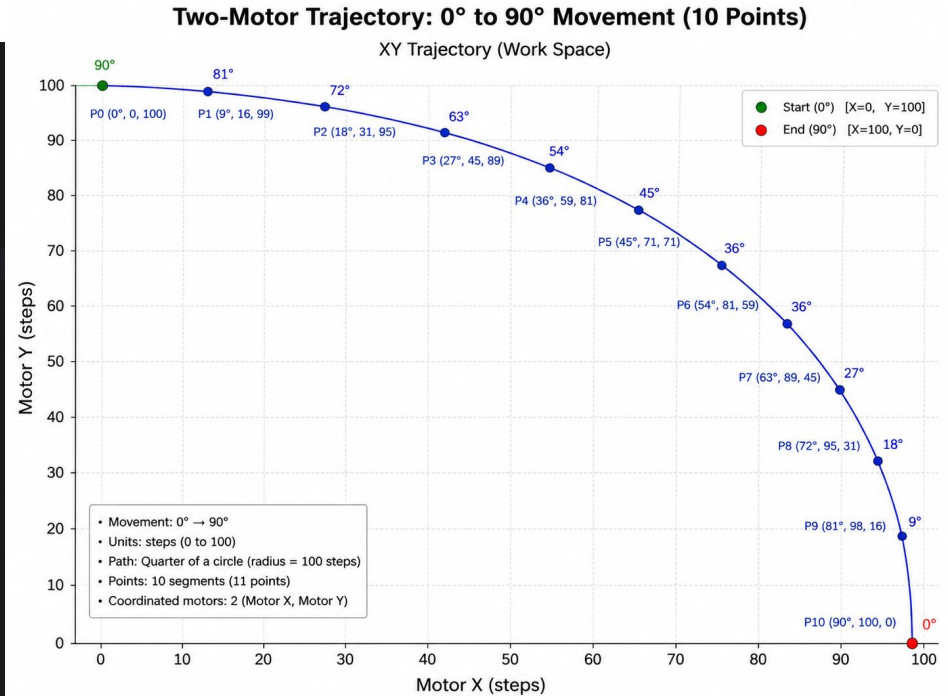
Out of the scope: Sardana is not responsible for synchronizing two controllers
e.g: Icepap-Aerotech, Icepap-Icepap

Helpers (sardana.util.motion.trajectory)

```
8 @dataclass
9 class PVTPoint:
10     time: float
11     position: float
12     velocity: float
13
14
15 class PointTrajectory: 4+ usages
16 >     def __init__(self):...
21
22     def build(
23         self,
24         time_array: list[float] | np.ndarray,
25         positions: dict[str, list[float] | np.ndarray],
26 >     ):...
57
58 @property
59 > def max_velocity(self) -> dict[str, float]:...
61
62 @property
63 > def max_acceleration(self) -> dict[str, float]:...
67
68 @property
69 > def limits(self) -> dict[str, tuple[float, float]]:...
74
75 > def pvt(self, json_serialized: bool = True) -> dict[str, np.ndarray]:...
98
```

Helpers (sardana.util.motion.trajectory)


```
8 @dataclass
9 class PVTPoint:
10     time: float
11     position: float
12     velocity: float
13
14
15 class PointTrajectory: 4+ usages
16 > def __init__(self):...
21
22     def build(
23         self,
24         time_array: list[float] | np.ndarray,
25         positions: dict[str, list[float] | np.ndarray],
26 >     ):...
27
28
29 @property
30 > def max_velocity(self) -> dict[str, float]:...
31
32
33 @property
34 > def max_acceleration(self) -> dict[str, float]:...
35
36
37 @property
38 > def limits(self) -> dict[str, tuple[float, float]]:...
39
40
41 > def pvt(self, json_serialized: bool = True) -> dict[str, np.ndarray]:...
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
```



Helpers (sardana.util.motion.trajectory)

```
15
16  ✓ def createTrajectory():
17      alpha = numpy.linspace(0,90,10)
18      times = numpy.linspace(0,10,10)
19      x_positions = numpy.array(numpy.cos(numpy.deg2rad(alpha))*100, int)
20      y_positions = numpy.array(numpy.sin(numpy.deg2rad(alpha))*100, int)
21      test_traj_ipap = PointTrajectory()
22  ✓ test_traj_ipap.build(times, positions: {TRAJ_MOT01: x_positions, TRAJ_MOT02: y_positions,
23      'param': alpha})
24  ⚡ return test_traj_ipap
25
```

Helpers (sardana.util.motion.trajectory)

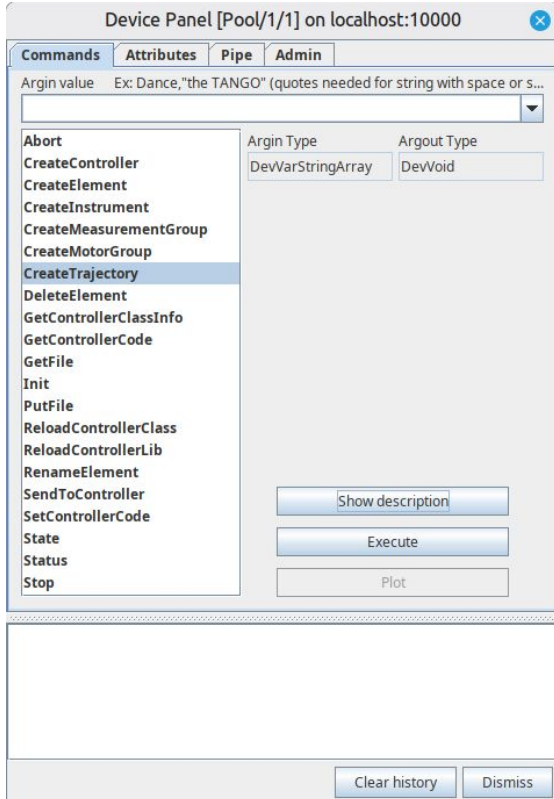
```
100  ▾ class MeshScanTrajectory:
101      def __init__(
102          self,
103          x_start: float,
104          x_stop: float,
105          x_points: int,
106          y_start: float,
107          y_stop: float,
108          y_points: int,
109          point_time: float,
110          snake: bool = True,
111      ): ...
120
121  > def generate_points(self) -> tuple[np.ndarray, dict[str, np.ndarray]]: ...
147
148  >  def build_trajectory(self) -> PointTrajectory: ...
155
```

Motor Controller Interface

```
▼ © Trajectory(object)  
  (m) LoadPVT(self, pvt)  
  (m) SynchronizeAll(self, axes, position)  
  (m) SynchronizeOne(self, axis, position)  
  (m) StartTrajectory(self, position, velocity, acceleration, axes)  
  (m) StopTrajectory(self, axes)  
  (m) ReadTrajectoryPosition(self, axis)  
  __class__(self)
```

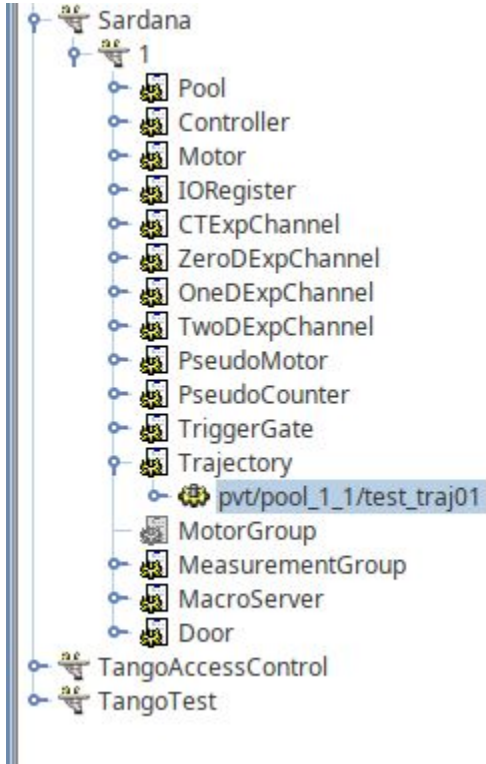
```
42  
43 class IcepapController(MotorController, Trajectory): 5+ usages  👤 Roberto Javier Homs Puro +9 *  
44     """  
45     This class is the Sardana motor controller for the ICEPAP motor controller.  
46     As part from the standard Pool motor interface per axis, it provides extra  
47     attributes for some firmware attributes of the driver.  
48     """  
49
```

Tango Interface



```
if __name__ == "__main__":
    pool = tango.DeviceProxy('tango://localhost:10000/pool/1/1')
    pool.CreateTrajectory([TRAJ_NAME, TRAJ_MOT01, TRAJ_MOT02])
```

Tango Interface



Device Panel [pvt/pool_1_1/test_traj01] on localhost:10000

Commands | Attributes | Pipe | Admin

Argin value

Command	Argin Type	Argout Type
Abort	DevVoid	DevVoid
Init		
LoadPVT		
Release		
Restore		
State		
Status		
Stop		
SynchronizeAll		
SynchronizeOne		

Show description

Execute

Plot

Clear history | Dismiss

Device Panel [pvt/pool_1_1/test_traj01] on localhost:10000

Commands | Attributes | Pipe | Admin

Argin value

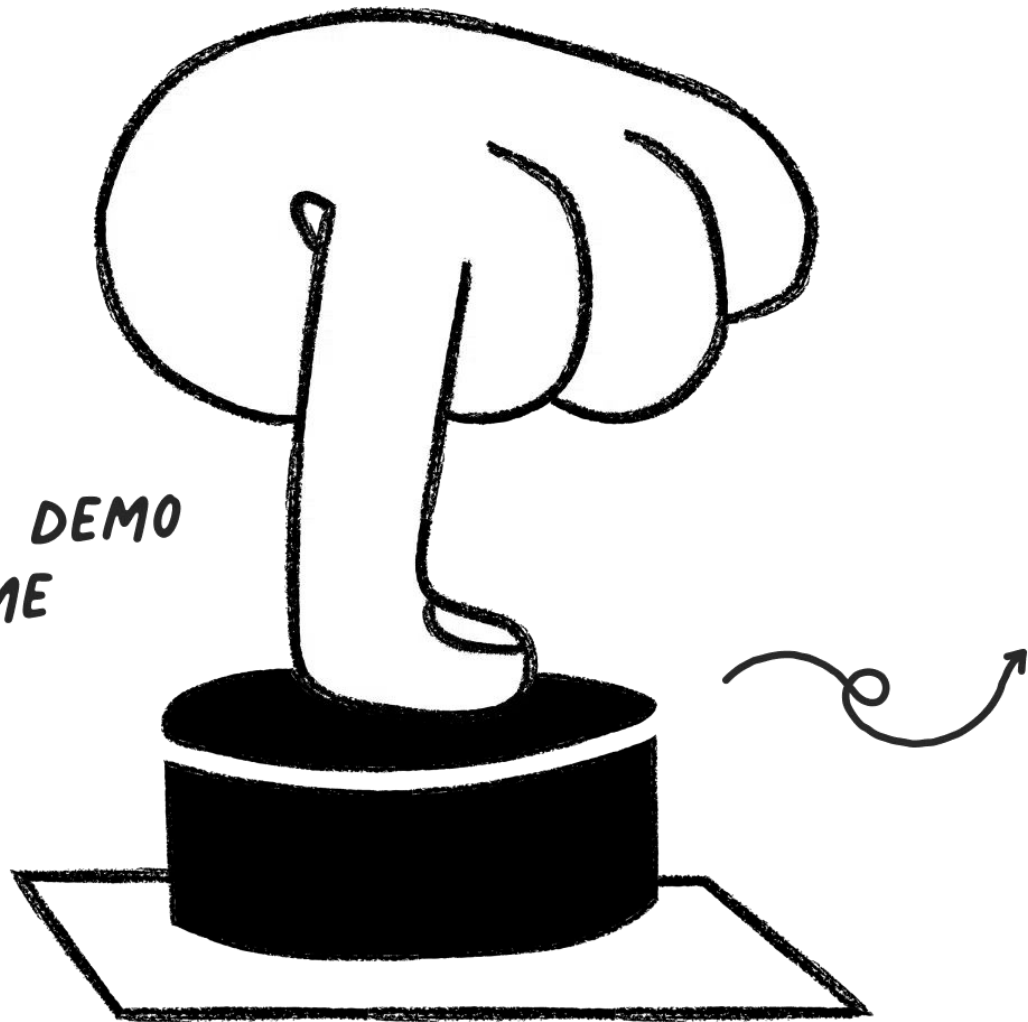
Attribute	Name	ElementList
Acceleration	Label	ElementList
ElementList	Desc	No description
MotorsPosition	Writable	READ
Position	Data format	Spectrum
PVT	Data type	DevString
State	Max Dim X	4096
Status	Max Dim Y	0
TrajectoryState	Unit	
Velocity	Std Unit	No standard unit
	Disp Unit	No display unit
	Format	%s
	Min value	Not specified

Read | Write | Plot

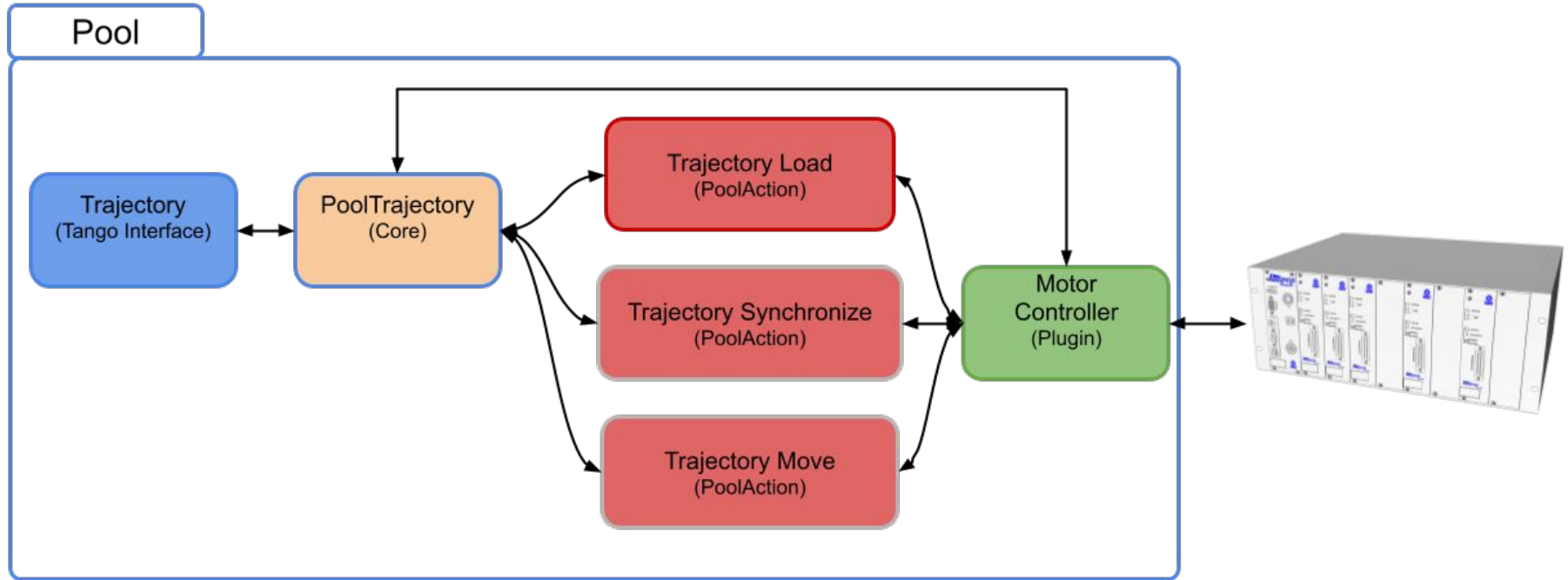
Attribute: pvt/pool_1_1/test_traj01/ElementList
Duration: 5 msec
measure date: 08/06/2026 14:25:01 + 230ms
quality: VALID
dim x: 2
Read length: 2
Read [0] mott01
Read [1] mott02

Clear history | Dismiss

IT'S DEMO
TIME



What's new?



Next steps.....

- First MR:
 - Events generation (positions, state machine, ...)
 - Adapt/re-define TrajectoryState attribute to Tango Device Server State or keep it
 - Position, Velocity and Acceleration validation
 - Transformation from user unit to motor units
 - Create meshtraj macro (ideas for name are welcome)
 - Documentation :)
- Design solution for pseudo-motors
- Validate PoC with other controllers (Aerotech, Beckoff,.....)

Thank You for Your Attention