Project: Kino-Dynamic Interpolation

by: Mirko Borich

Kino-Dynamic Interpolation

Motivation

Pick & place movements between workspace obstacles. Must be as fast as possible. No need to specify any velocity, robot should go in minimum time from place to place without hitting anything on it's path.

Kino-Dynamic Interpolation

We are speaking about ***near*** **Kino-Dynamic** straight-line interpolation.

- Defined in http://en.wikipedia.org/wiki/Kinodynamic_planning as:

  "*In robotics and motion planning,* ***kinodynamic planning*** *is a class of problems for which velocity and acceleration bounds must be satisfied*"

AMCS system:

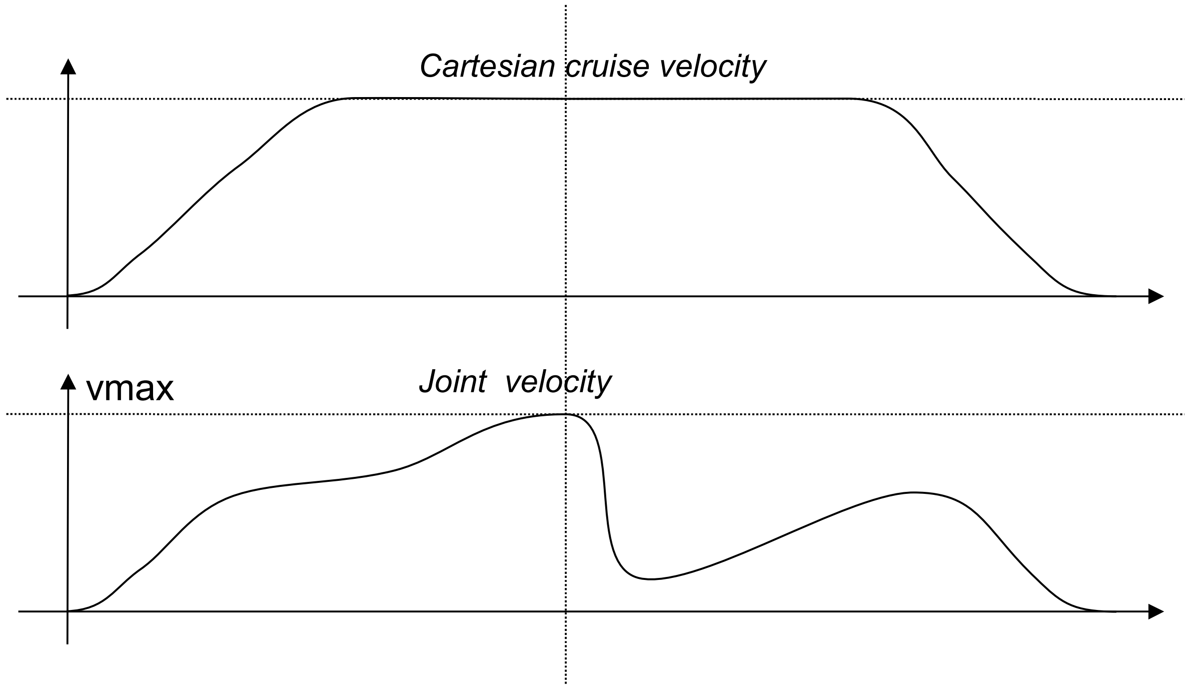
Straight line motion interpolation where one of the axes always ***reaches*** but not exceeds it's maximum values of velocity, acceleration and jerk (*and stays there as long as possible* - desirable). The other axes must not ***breach*** these values.

Note: *No requirements on motions Cartesian velocity, it just needs to start and end at zero. No* ***cruising*** *demanded.*

Today (version 4.7.12 and before)

Straight line motion (MOVES) – keeps cruise velocity constant.

- This means that in cases the joint velocity exceeds it's maximum the whole movement will be slowed down (instead only at this place) to keep the joint velocity value below max.



Additional problem: It is hard to predict points of max joint values on the path!

Kino-Dynamic Interpolation

Today we have two typical scenarios:

- The max joint values are underestimated – cartesian cruise velocity is too high causing motion to exceed its joint maximums which causes either stopping due to drive errors (if values are too high > 120%) or increased position error (if the values are under certain thresholds < 120%).
- The max joint values are overestimated – cartesian cruise velocity is lower than it could be. → *Prolonged system cycle-time (slowdown of machine).*

In both case it is a headache for the application engineer having to fine-tune his application. Manually finding proper values of cruise velocity. → *Increased setup time.*

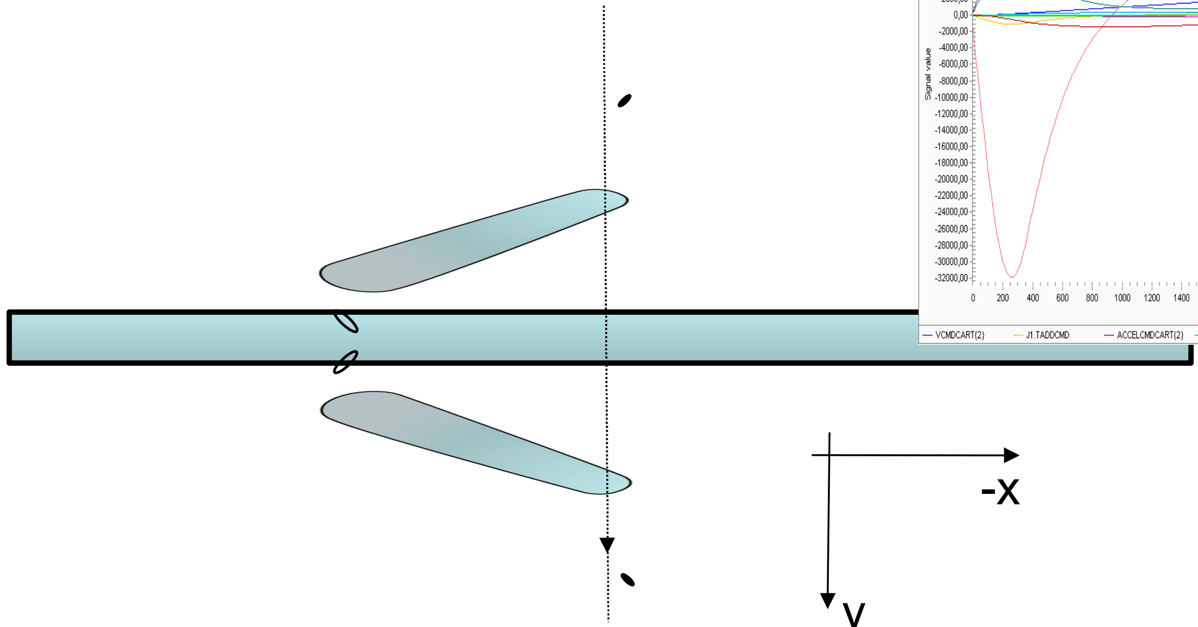
Current solution: Straight line (MOVES) is not used in most of time-critical applications. Joint interpolation (MOVE) is used instead. → *Possibility of collisions. Prolonged cycle-time due to “work-around-ing“ MOVES with MOVE*

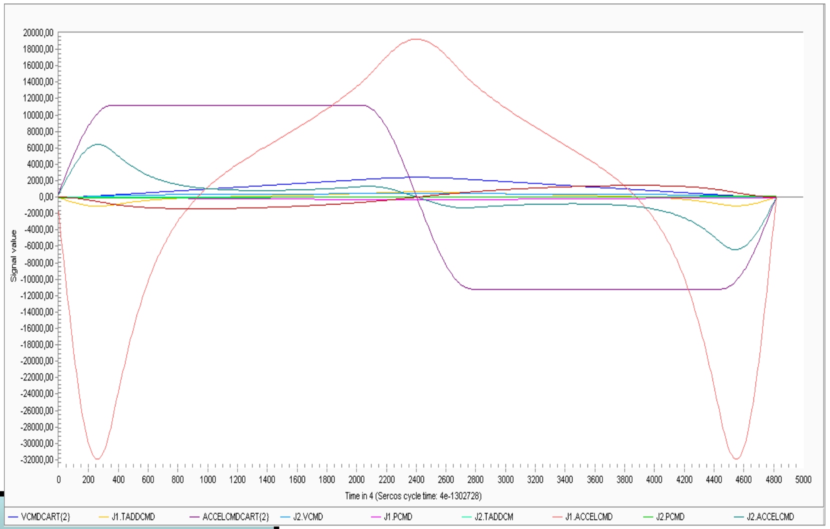
Kino-Dynamic Interpolation

Example (from Artur's MBCRC case study document):

Speedpicker. Moving Straight (MOVES) line from one end to another.

- Cartesian Translational Acceleration is limited by joint's 1 AccMax
- Local limit in joint 1 limits the whole motion
- MOVES: Start: #{0, -290} → Target: #{0, 290}
- Duration: **484** ms.

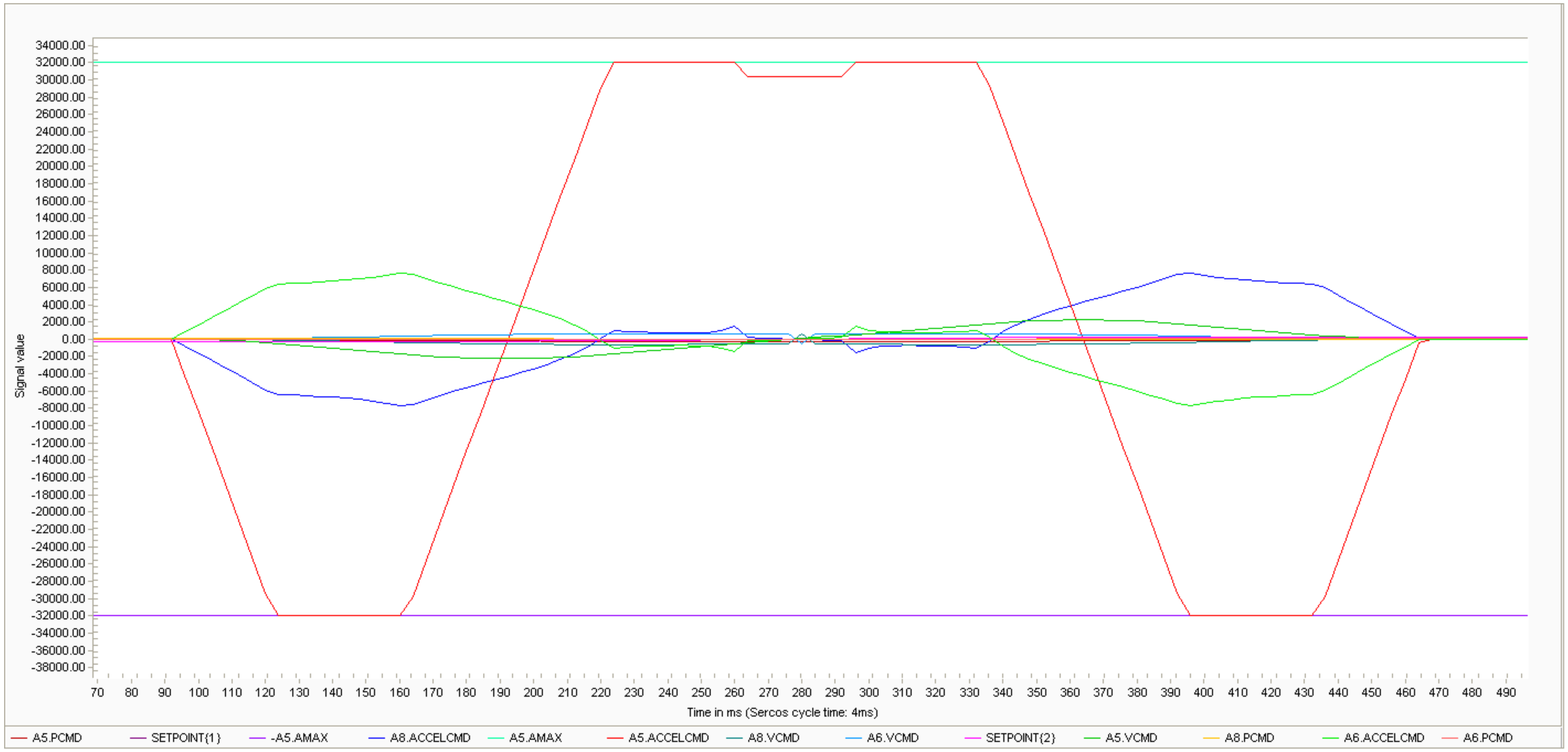




Kino-Dynamic Interpolation

KinoDynamic Interpolation takes only <u>372 ms</u>!

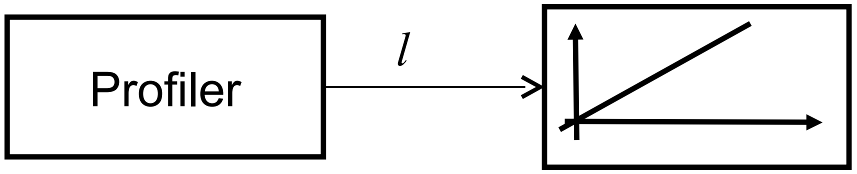
- Which is **30%** faster.
- And this is not the most critical example! There are even bigger differences!



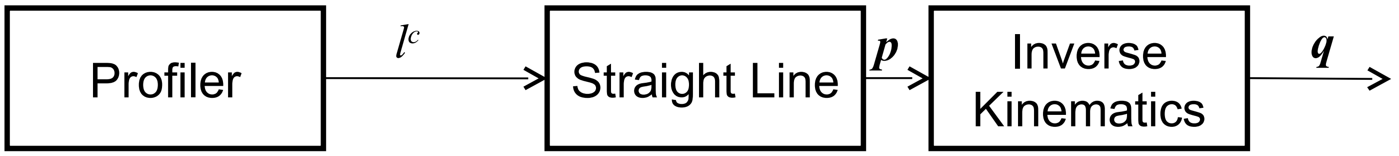
Kino-Dynamic Interpolation

Theory of operation:

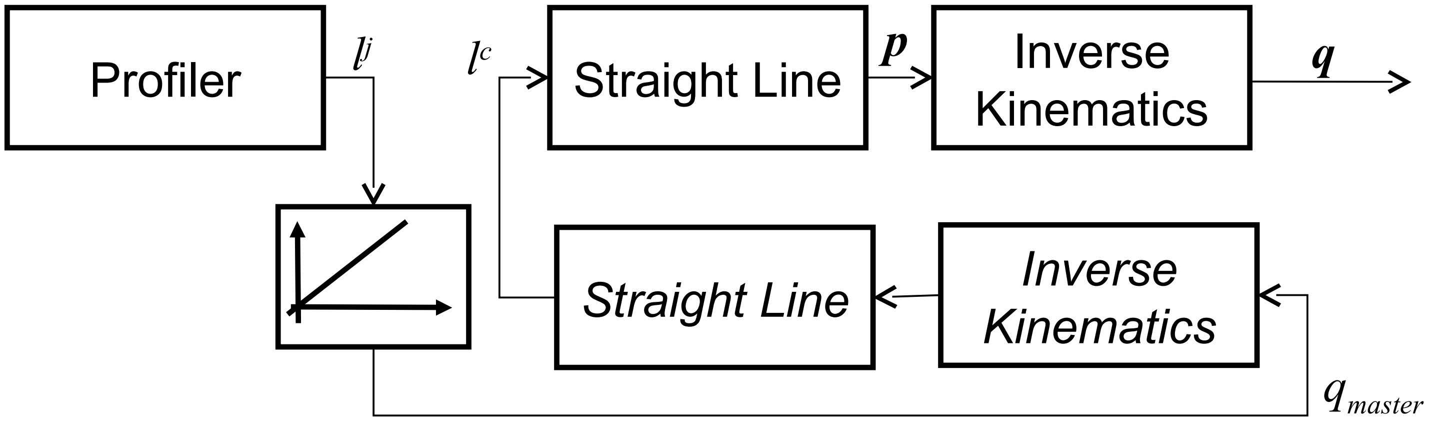
MOVE



MOVES

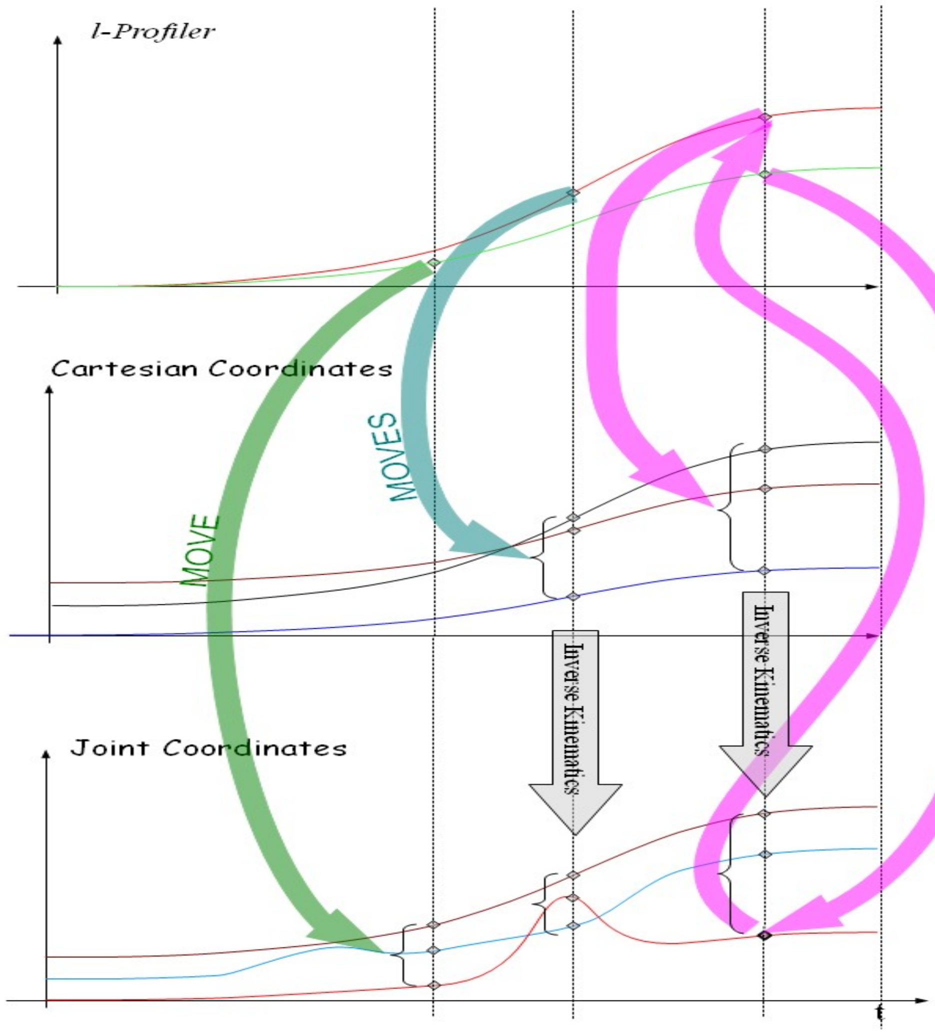


MOVESKD



Kino-Dynamic Interpolation

Theory of operation:



Kino-Dynamic Interpolation

Things to be aware of:

- Kino-Dynamic movements do not have constant Cartesian velocity (not applicable in gluing/cutting applications).

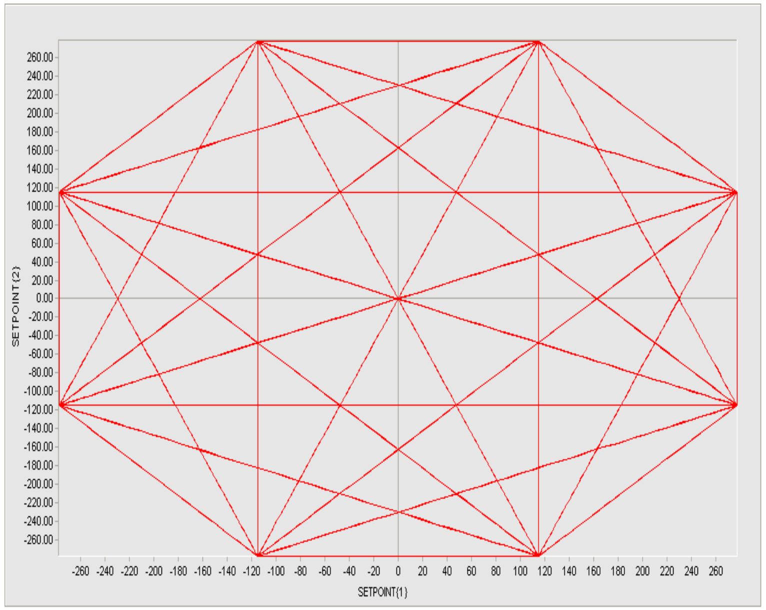
- Kino-Dynamic movements are not usable in Conveyor-Tracking applications (because the link between joint and Cartesian positions is not existing on movement–level).

- Kino-Dynamic movements can not be blended using BlendingMethod =1 (CP). **But SP blending is OK!**

Kino-Dynamic Interpolation

Test Case:

- Moving between points of an octagon inscribed in a circle of radius 300mm (arm length of speed-picker).
- Results:
  - Total time of all motions (8x8 = 64 movements) is:
    - Using regular **MOVES**: 25.213 sec
    - Using **MOVESKD**: 15.645 sec
  - This is **61% faster(total)!**
- Best case: **86%!**



Kino-Dynamic Interpolation

manz passion for efficiency

Even Better (Speed-Picker singular-configuration is treatable!):

angle = 70	factor = 289%
angle = 71	factor = 287%
angle = 72	factor = 284%
angle = 73	factor = 284%
angle = 74	factor = 282%
angle = 75	factor = 279%
angle = 76	factor = 276%
angle = 77	factor = 276%
angle = 78	factor = 274%
angle = 79	factor = 274%
angle = 80	factor = 272%
angle = 81	factor = 272%
angle = 82	factor = 272%
angle = 83	factor = 273%
angle = 84	factor = 297%
angle = 85	factor = 353%
angle = 86	factor = 482%
angle = 87	factor = 487%
angle = 88	factor = 487%
angle = 89	factor = 484%
angle = 90	factor = NA (regular MOVES, can't do it!)