Part VI

Advanced Val II / V+ features

Val II tasks
and examples of their use

The Val II Process Control Program

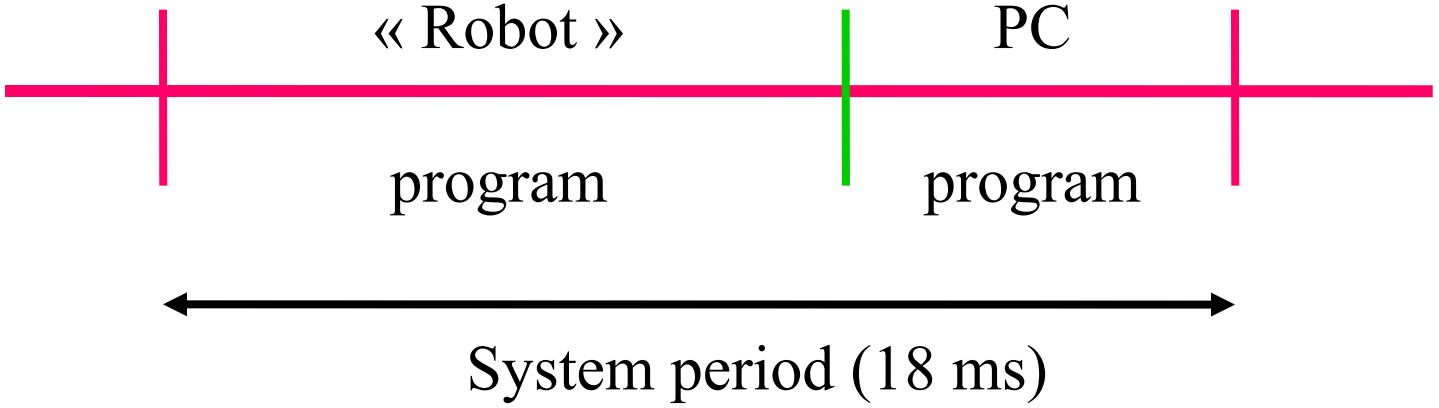
- The PC program is executed « in parallel » to the « robot » program (time sharing).
- The time balance between robot program and PC program is fixed.
- If there is no PC program, its time slice remains unused:
  - A very simple solution
  - Executing a PC program does not slow down the robot program.

The Val II Process Control Program

- PC program influence on motion **only** by:
  - The brake instruction (**not** break!)
  - Real-time trajectory modification (the « alter » mode, see later).
- Typical uses:
  - Process control actions
  - Sensor monitoring
- Communication with robot program
  - Global variables
  - Internal bits (2001-2032)

The Val II Process Control Program

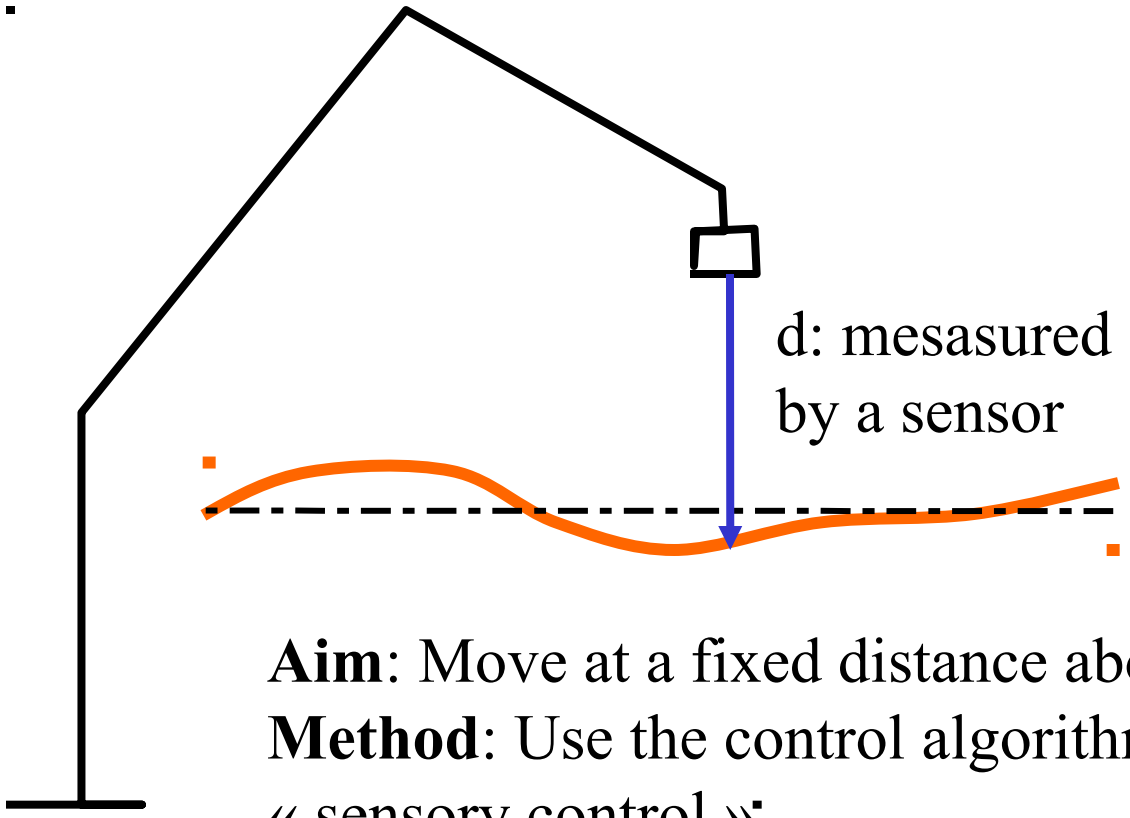
- PC program management by robot program:
  - pcexecute <pc.prog.name> [,<nb.iterations>]
  - pcend
- Time sharing method:



Application: guarded motions

- A guarded motion is a motion that can be stopped by two conditions:
  - The destination is reached (normal condition)
  - Some condition related to the output of a sensor is satisfied.
- LM example:
  move <toolframe> to <dest> until <cond>
- The PC program allows monitoring the condition (at a fixed frequency) and stop the motion (brake)

Real-time trajectory modification by the PC program



Nominal profile:

Actual profile:

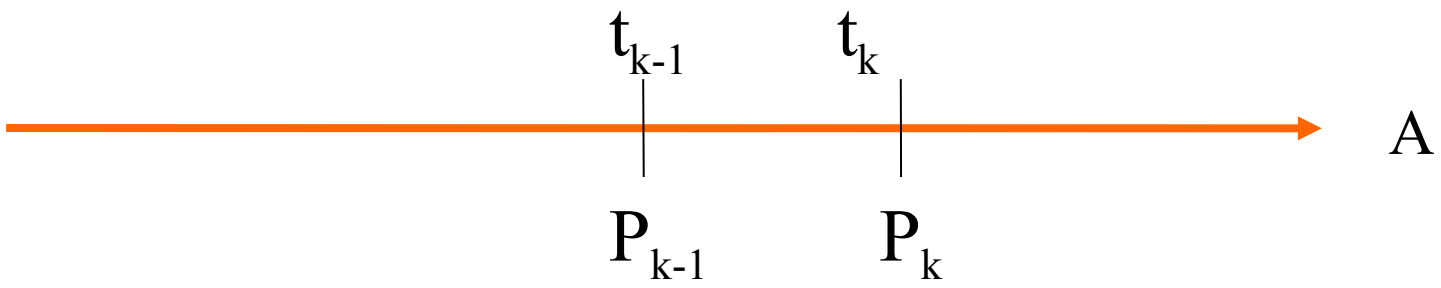
Aim: Move at a fixed distance above actual profile.
Method: Use the control algorithm. This is called « sensory control ».

Note: exists in V+ with a slightly different syntax.

Language instructions

- Enter real-time trajectory modification by alter(type,mode):
  - Type: internal (this case) / external
  - Mode: corrections in base/tool frame and wrt current/nominal location.
- Send corrections by altout(exception , dx,dy,dz , rx,ry,rz)
- Stop alter mode: noalter.
- If no corrections are received when in alter mode, the program is stopped.
- Can be used only for straight line motions.

How it works



- While the robot moves to P_{k-1}:
  1. Calculate P_k.
  2. Add any correction ΔP_k sent by altout.
  3. Calculate $q_k = IGM(P_k + \Delta P_k)$.
  4. At t_{k-1}, set desired position for t_k as q_k.
- Only step 2 is added to the standard straight line motion process.

Profile tracking program

Robot Program:

```
edit profile

pcexecute correction, -1, 0
alter (-1, mode)
moves shift(start.loc by 0,
        profil.length, 0)
moves start.loc
break
noalter
pcend

e
```

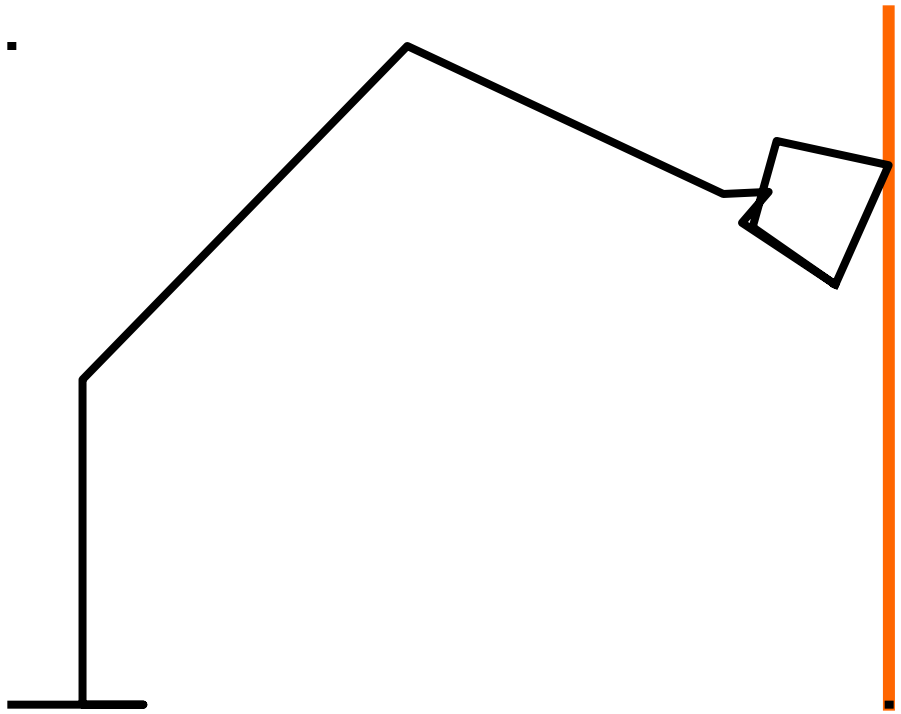
PC program
(Proportional controller) :

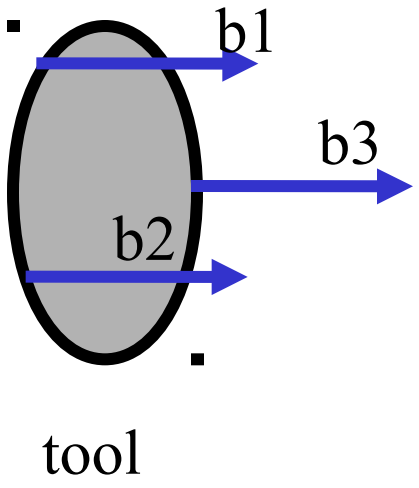
```
edit correction

v = adc(sensor.channel);

correct.z = desiredv-v

altout no.except, 0, 0,
    kp*correct.z*todis, 0, 0, 0
return

e
```

Comments

- Use of the PC program guarantees a fixed sampling period.
- The robot program is not overloaded with control instruction: just the nominal task.
- A more complex control could be implemented in the PC program.
- The mode can be set to « tool frame » and « corrections with respect to current location ».

Other example: brushing a surface





Three sensors measure three distances

Position constraints of the tool wrt the surface:

- Angle wrt surface normal.
- Distance to surface (corresponds to a pressure using a spring).

Interaction matrix

It relates elementary position/orientation variations that should be applied to tool frame to sensor output errors wrt to desired output.

$$ {}^{E}J_{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 \\ \sin(0°) & \sin(120°) & \sin(240°) \\ \cos(0°) & \cos(120°) & \cos(240°) \\ 0 & 0 & 0 \end{bmatrix} $$

$$ {}^{E}dX = \begin{bmatrix} 0 \\ 0 \\ dz \\ \Omega_x \\ \Omega_y \\ 0 \end{bmatrix} = {}^{E}J_{B} \begin{bmatrix} db1 \\ db2 \\ db3 \end{bmatrix} = {}^{E}J_{B}db $$

altout corrections on
dz, rx, ry components

Remarks about Val II tasks

- The multi-tasking possibilities of Val II are simple: no general scheduling process.
- Existing tools are easy to use.
- Together with alter/altout, provides a good basis for guarded motions and sensory control of the robot, provided the system period is adapted.

V+ Programs and Tasks

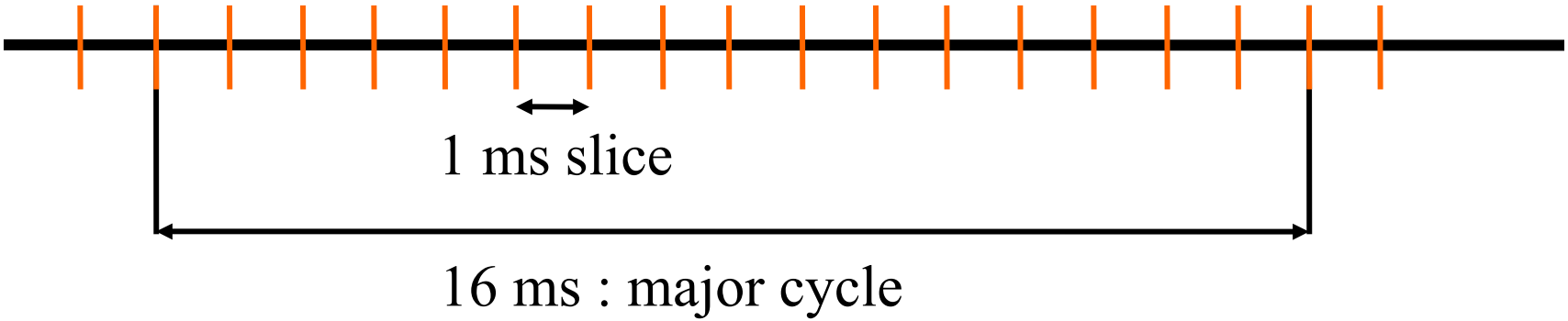
Robot control programs

- Definition: a V+ program which directly controls a robot or other mechanism.
- Generally executed by task 0, which automatically attaches the robot.
- If a robot program is executed by a task other than task 0, it must the the attach instruction to attach the robot.
- No other task can then control the robot.

General programs

- Do not control robots.
- Several can execute concurrently.
- Can only stop the robot using brake.
- Cannot use the tool instruction.
- Communication between programs:
  - Global variables
  - Internal bits (2001-2512), which are manipulated with sig and signal can be associated to a react/reacti (internal bits 2001 to 2008).

Time slices and tasks



- Seven tasks (0..6) can be defined.
- The trajectory generator is invoked at each major cycle.
- The time available to a task depends on:
  - The time slices (0..15) in which it has been defined to run.
  - Its prioriy (-1..63).

Task execution

Optional, to attach the robot

Task number

Program name and parameters.

- execute /C **task_num program(**param_list**)**, cycles, step, priority[i]

Number of cycles (-1 for infinity)

Optional: step at which execution should start.

Optional: task priority in each of the 16 1 ms slice.

Changing the priority in the various slices is hardly ever necessary…

Meaning of priority values

- -1: do not execute in this time slice.
- 0: execute only if no other task is ready to execute in this time slice.
- 1..64: execute in this time slice according to priority.
  - 1..31: normal user task priorities.
  - 32..63: V+ system tasks and drivers.
  - 63: trajectory generator.

Swapping tasks

- A task becomes inactive if:
  - It terminates.
  - It performs an I/O operation.
  - It executes a wait or a release.
- The highest priority task is executed.
- A special process is used when equal priority tasks are ready to run.
- A task can release the procesor voluntarily.
- Wait used alone gives up the procesor until next major cycle.

Task priority example

- Task 0 executes in all slices, priority=20.
- Task 1 executes in all slices, priority =10.
- Task 2 executes in all slices, priority =20.
- System tasks are ignored.
- System interrupts are ignored.

- No « react » and no « lock »: the priority of programs executed by tasks is always 0.
- Note: unless tasks 0 and 2 both execute a « wait » or a « release » in favor of task 1, task 1 never executes.

Sequence of events

❶ prog_a issues a **WAIT.EVENT**. This suspends prog_a and passes execution to the next highest task which is task 2 running prog_c.

❷ prog_c runs until it issues a **RELEASE** instruction. Since the RELEASE has no arguments, execution is passed to the next highest task with a program to run. Since task 0 is waiting on a **SET.EVENT**, the next task is task 1.

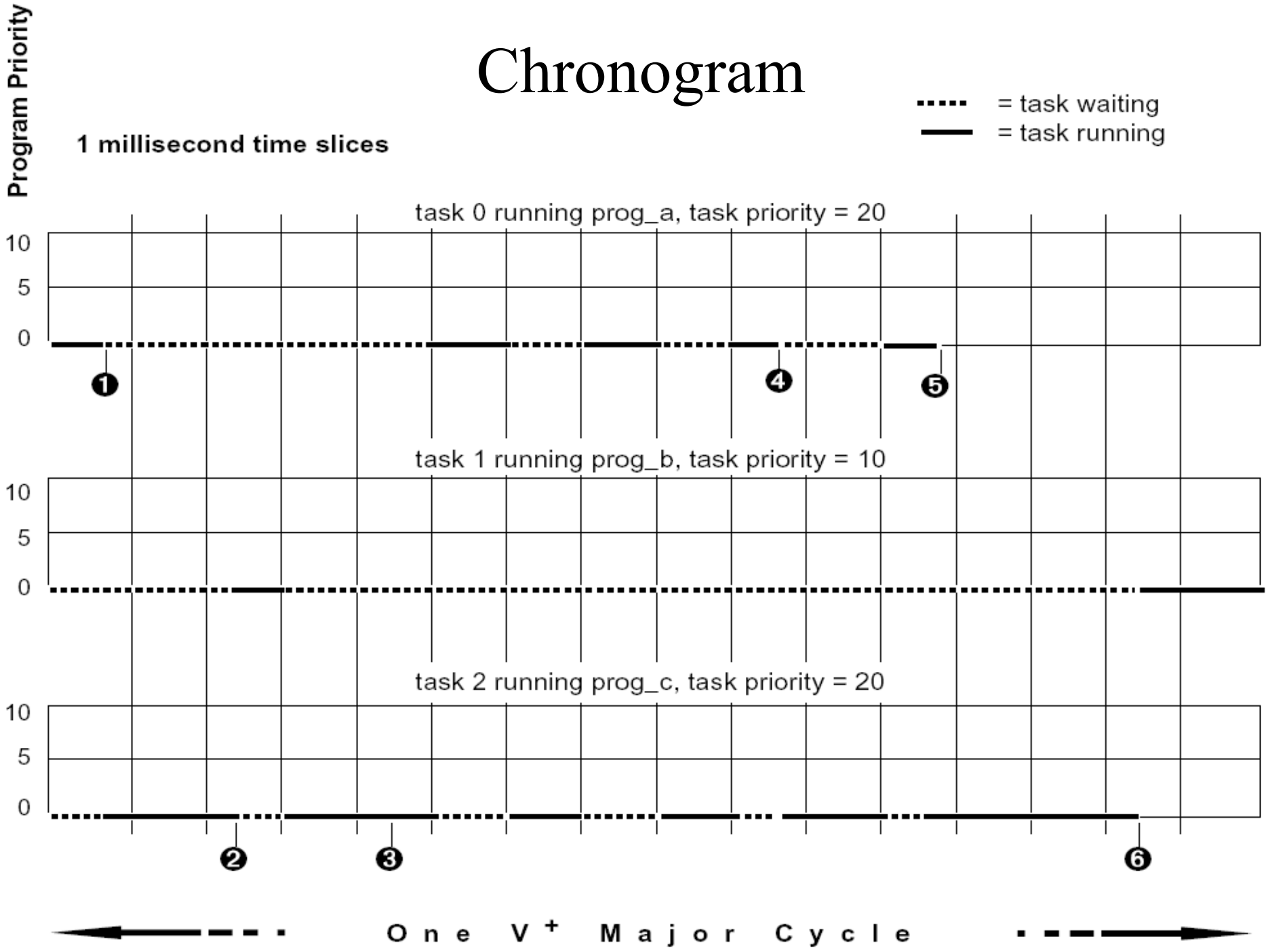
❸ Task 2 issues a SET.EVENT to task 0 and runs until the end of a time slice at which time task 0 runs. Tasks 0 and 2 have the same priority so they swap execution. (If two tasks with equal priority are ready to run, the least recently run task runs.)

❹ prog_c waits for a disk I/O operation to complete. The next highest priority task is 2 which runs until the I/O operation completes and task 0 becomes the least recently run task.

❺ prog_a completes, passing control to task 2.

❻ prog_c completes, passing control to task 1.

Chronogram


Program Priority
1 millisecond time slices
= task waiting
= task running
task 0 running prog_a, task priority = 20
10
5
0
1
4
5
task 1 running prog_b, task priority = 10
10
5
0
task 2 running prog_c, task priority = 20
10
5
0
2
3
6
One V+ Major Cycle

« Test and set » instruction

- Typical use: semaphores

```
while tas(reserved.data,true) do
    wait; or release: do not misuse procesor time
end
; process data
…
reserved.data = false
```

V+ conveyor tracking features

Characteristics

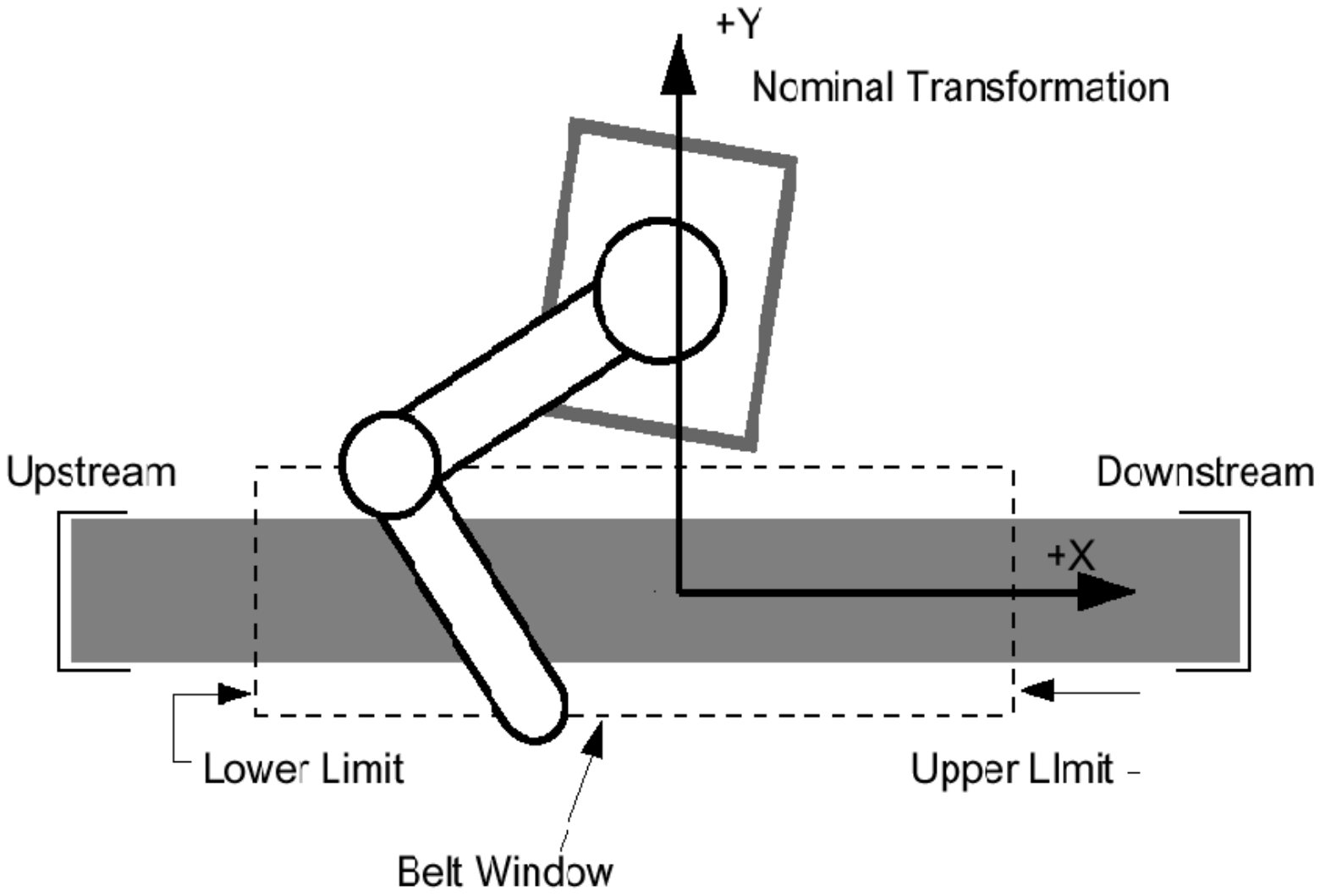
- Requires V+ extension, an additional encoder board and an encoder attached to the conveyor.
- Allows to define locations which are automatically modified depending on the instantaneous position of the conveyor.
- The conveyor must move in a straight line.
- Motions to locations attached to a conveyor must be straight line motions.

Conveyors are defined by « belt variables »

Belt variables include:

- Position/orientation of conveyor frame.
- Associated encoder number (1..6).
- Encoder dots per mm.
- Encoder offset.
- « Window parameters » which define the space in which the robot is able to access to the conveyor.

Belt vocabulary



Motions wrt a conveyor

- moves %belt.var:relative.location
  - Only one such variable in a compound transform
  - Always the leftmost in the expression.
- The robot tracks the conveyor until a motion to a non conveyor related point is performed.
- The motion to the fixed point that ends conveyor tracking must be a straight line motion.

Motion end

- End condition: joint position in a narrow sphere around desired joint position.
- Because the desired position evolves, the condition may never be met.
- The tolerance (sphere diameter) can be increased by the coarse instruction.
- In extreme cases (irregular speed for example), the error nulling process can be suppressed using the nonull instruction.

Conveyor related functions

- belt(%belt.var) reads the belt encoder.
- setbelt %belt.var = value ; sets the belt variable offset.
- Typical uses:

Teach a belt relative location:

```
wait sig(object.detector)
sig(stop.conv)
setbelt %belt.var = belt(%belt.var)
; Move robot to location and then:
here %belt.var:some.location
```

Move to a belt relative location:

```
wait sig(object.detector)
setbelt %belt.var = belt(%belt.var)
appros %belt.var:some.location , 50
…
```

How it works

- Exactly like for the real-time trajectory modification using « alter ».
- Explains why, again, straight line motions are required.
- It's almost certainly « alter in disguise »…

- Sorry, but why the terminating motion to a fixed point should be a straight line motion remains unclear to me…

Exercises

- Use react_ and reading the belt encoder to determine object diameter as objects pass.

- Use a binary line connecting RX90's output 5 to Puma560's input 11 to communicate 8 bit messages from RX90 to Puma560, using a « Morse type » code.

Exercise

- Use a V+ task to switch on and off the lamp attached to output 4 when an object reaches a location on the conveyor located 10000 encoder dots after the detector attached to binary input 8. Hypothesis: the distance between objects is larger than 10000 encoder dots.
- Same question without the hypothesis.

Exercise

- Situation and task:
  - Objects pass detector connected to input 7.
  - The robot processes the objects: track the object at track.loc relative to belt.
  - Processing is finished when input 9 goes high.
  - If a new object comes while the previous one is processed, its processing is delayed.