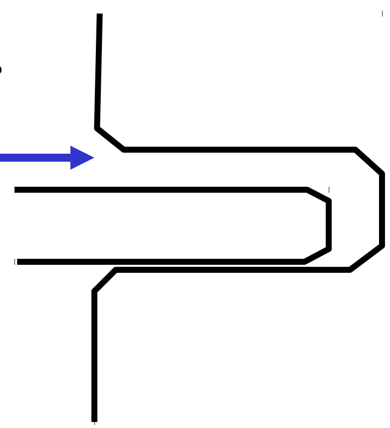
Part III

Example of using sensors in robot tasks

How to assemble?

1. Obtain a given relative positionning of two or more parts.
2. Suppress some or all degrees of freedom between the parts.

- In the case of small mechanical parts, the clearance can be very small.



Robotized assembly

- Difficulties:
  - Robot position errors.
  - Uncertainty/variability in object positions.
  - Manufacturing tolerances.
- Common consequence:

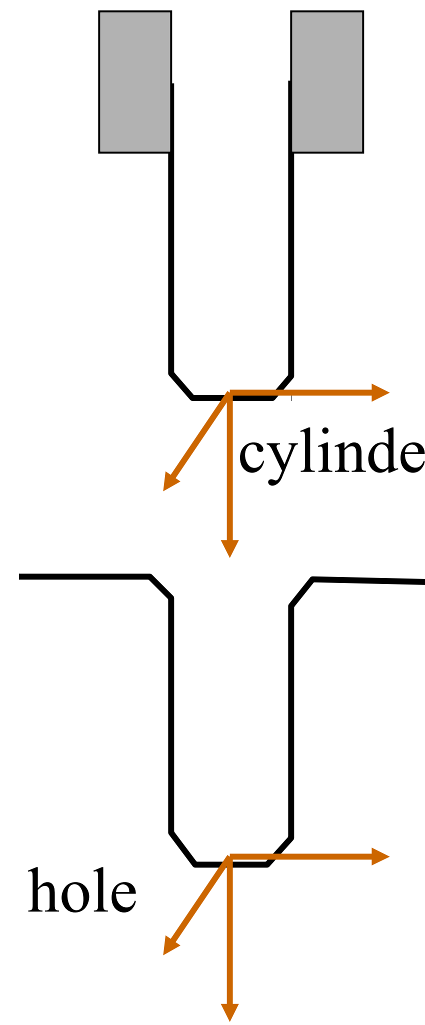
Insufficient precision in relative positionning of parts for assembly

Pre-recorded locations cannot be used

Robotized assembly

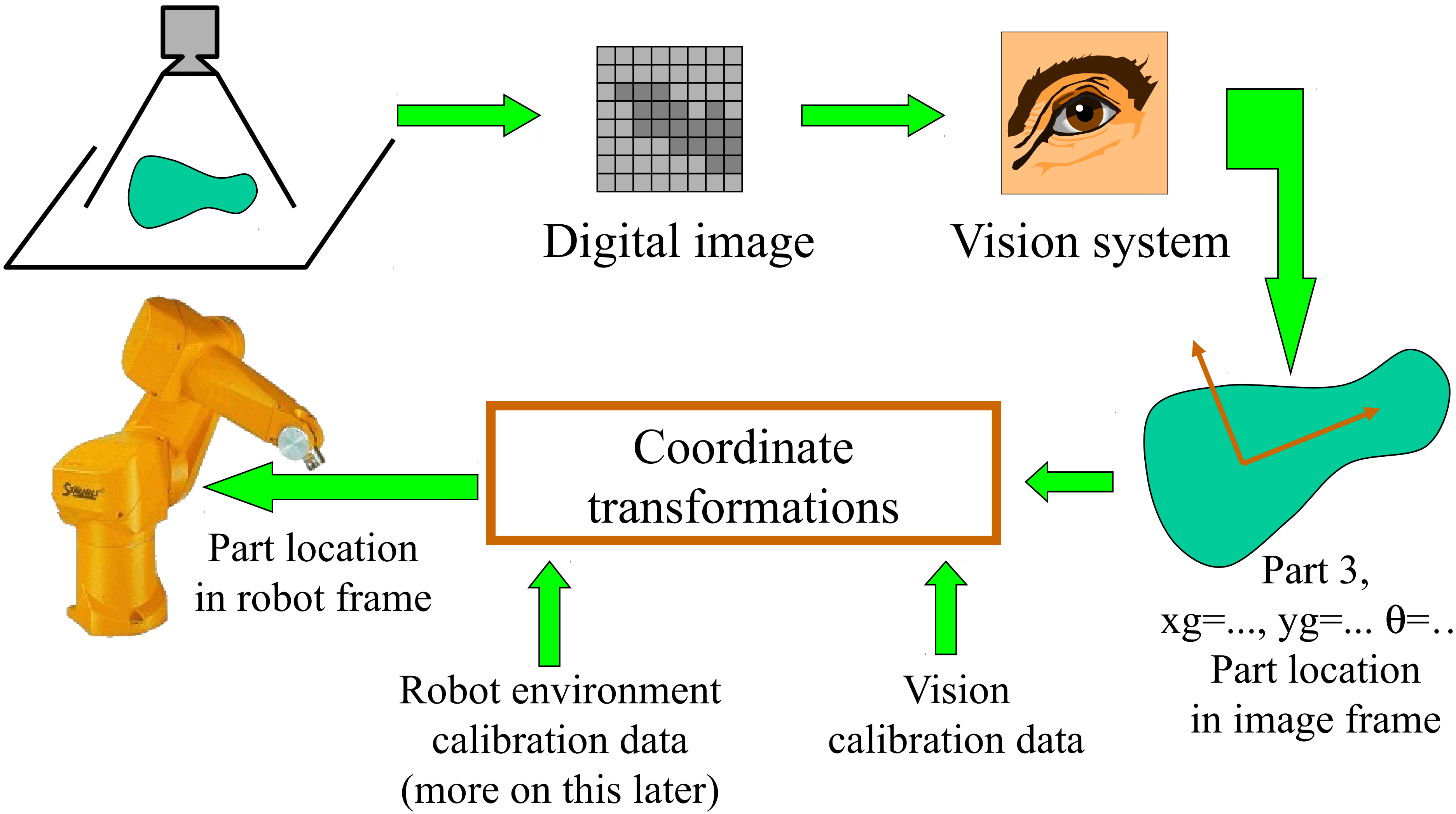
- Possible solutions:
  - Force sensors, vision sensors …
  - Special-purpose end effectors (compliant systems).
- Consequences
  - Applications solved on a case by case basis.
  - High cost and long ROI.

```
Function boolean insert (frame cylinder, frame hole, real fzmax, real eps, integer nmax)

integer N; vector effort ; frame hole1 ;

begin
    N=0;  hole1 := hole * translation( VZ , 10*mm) ;
    while N < nmax do
        move cylinder to hole1 until fz > fzmax with wait ;
        if distance(cylinder,hole) < 1 * mm then
            open gripper to 40 * mm with wait ;
            return(true) ;
        else
            effort = vector( FX, FY, 0 ) ;
            move cylinder by translation(effort,eps) with wait ;
            eps := eps/2 ;   N := N+1 ;
        endif
    endwhile
    write ("Insertion failed.");
    return(false);
end
```



http://www.youtube.com/watch?v=AsBOEEo05G

Using a vision system



Example: http://www.youtube.com/watch?v=3JDwuLe6mc0

General remarks about the use of sensors

- Locations are not predefined: they are determined at execution time.
- Hence, taught locations may be necessary to automate the task, but they are not sufficient.
- Using sensors may require specific knowledge and expertise:
  - Signal processing techniques;
  - Vision systems knowledge…

Examples of sensor uses

- Identification and localisation of objects.
- Quality control.
- Sensory control: the robot motion or force is adjusted in accordance with outputs of external sensors (polishing, seam following).
- Unstacked parts picking …

Very fast development!

Some conclusions

- Sensors and computers are the only way to make robots actually flexible tools.
- Robot system (ISO 8373):
  - Robot
  - End effectors
  - Any equipment, devices or sensors required for the robot to perform its task
  - Any communication interface that is operating and monitoring the robot, equipment and sensors, as far as these peripheral devices are supervised by the robot control system.

Part IV

The various levels of textual robot programming

Joint level languages

- A language is a **joint level language** if poses (resp. displacements) of the end effector can be specified only through the positions (resp. displacements) of each joint of the robot.

- Likewise, only joint velocities can be specified.

Advantages / Drawbacks

- Advantage: easy to implement: direct relation between position informations in the program and desired positions sent to the control.

- Drawback: impossible to define frames, points relative to other points, etc. Reduced possibilities.

Frequent characteristics

Joint level languages are usually not sophisticated:

- Non structured programming
- No subroutines or without parameters
- Short mnemonic names
- Short variable names (or predefined variables)
- …

Of course, this is not inherent to joint level languages but to the will of making something simple (not to use, but to implement).

Example

SP1, TEST	subroutine 1, named “test”
IM	initialisations
IP	
EF1	label n° 1
RA1, 15.	absolute joint displacements
RA2, 27.	
FG	motion synchronization
D03	start of a loop to perform 3 times
RR1, 10.	relative joint displacements
RR2, 5.	
FG	
IF E0137, 5	if digital inputs 0,1,3,7 are at level 1 go to label 5
DE	end of loop
WT O 24, 1	wait until input 2 or input 4 is at level 1
EF5	label 5
OP, 50	open gripper
FG	
JP1	unconditional jump to label 1
ED1	end of subroutine 1

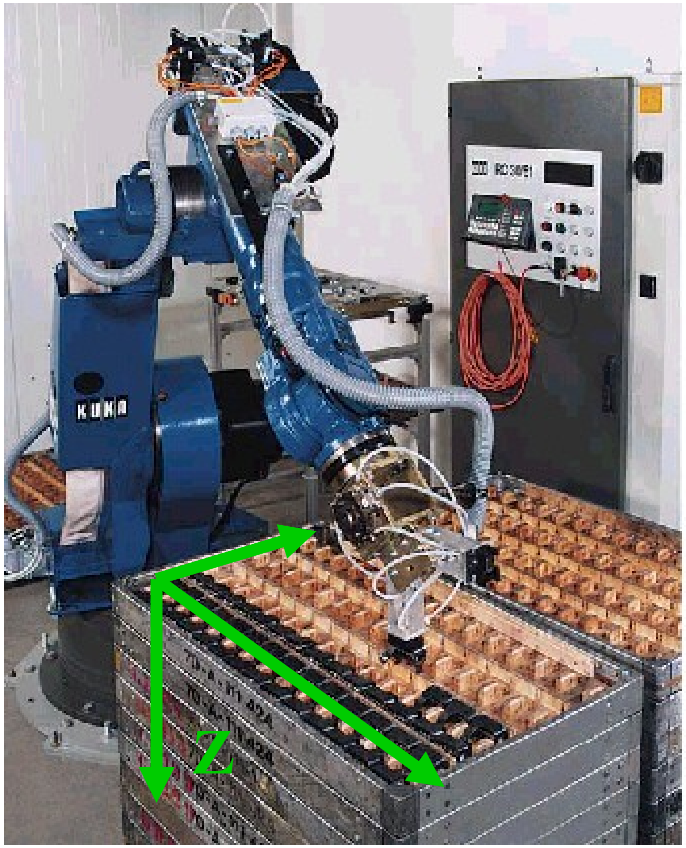
End effector level languages

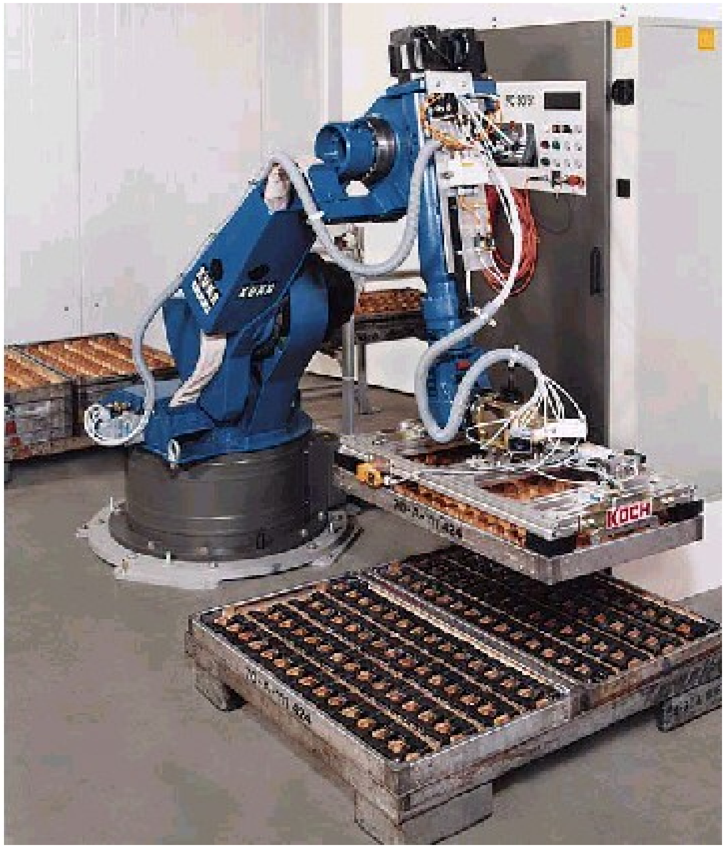
- A language is an end effector level language allows to specify the poses (resp. displacements) of the end effector in Cartesian coordinates (or equivalent system).
- Likewise, Cartesian velocities of the end effector can be specified.

Requires direct and inverse geometric models.

Interest of defining frames

With Cartesian coordinates and frames, somes problems become simple.

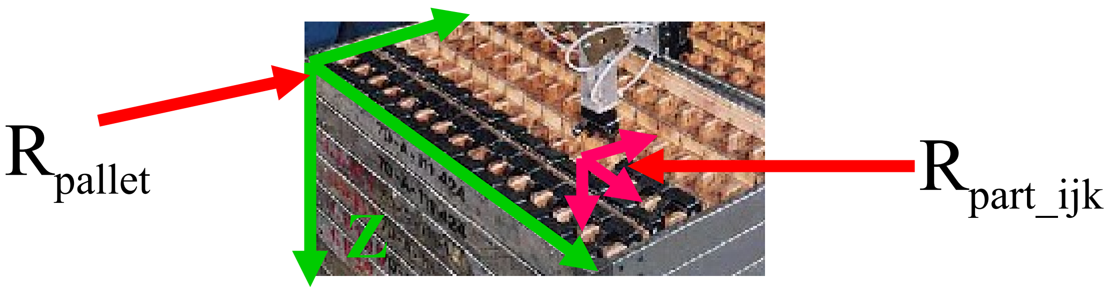




In the frame represented here, the coordinates of all objects are simple. How such a frame can be defined and why its Z axis should point downward will be addressed later.

Interest of defining frames

In terms of homogeneous transforms, the situation can be expressed like this:



$$ {}^{0}T_{part\_ijk} = {}^{0}T_{pallet} * {}^{pallet}T_{part\_ijk} = {}^{0}T_{pallet} * \begin{bmatrix} 1 & 0 & 0 & i*step.x \\ 0 & 1 & 0 & j*step.y \\ 0 & 0 & 1 & k*step.z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

We will see how to define ${}^{0}T_{pallet}$ using the language of the robots available in the lab.

LM: a reference ancestor


```
Program Demo

frame pallet_1, pallet_2...;
ext procedure pick( frame; real; real);
ext procedure place( frame; real; real);

begin
Init_pos := robot; Col := 0; ...
set robot speed to 0.7;
read pallet_1, pallet_2,... in pallet;
while col<5 do
    lgn:=0;
    while lgn<4 do
        grasp_loc:=pallet_1
            * translat(vx,Rea(lgn)*incr_lgn
            * translat(-vy,Rea(col)*incr_col;
        pick(grasp_loc,50.0,90.0);
        place(place_loc,50.0,50.0);
    .....
```

K-RA, another Pascal-like

```
program
const
    OPEN=1; ...
var
    Convey, Pallet : path ;
    AwayPos : position ; ...
begin
    toolaction( OPEN, GRIPPER, 0,0,0,0 );
    for rg_pc_y=0 to NB_PCS_Y-1 do
        for rg_pc_x=0 to NB_PCS_X-1 do
            shift_amp = vect( rg_pc_x*SPACE_X,  rg_pc_y*SPACE_Y, 0 )
            move along Pallet
            move relative shift_amp
            toolaction(CLOSE, GRIPPER, 0,0,0,0 )
            ...
        endfor
    endfor
    move to AwayPos
end depalletization
```

Usual difficulties …

- The manipulation of coordinates requires a quantitative knowledge of the environment.
  Help: teaching, sensors.
- Languages are a superset of standard programming languages and require more training.
- If sensors are used, it requires some experience and specific knowledge.

Usual difficulties …

Difficulties related to the complexity of tasks:

- Anticipate and handle all events, especially related to abnormal situations (absence of a part, unrecognized object…)
- An apparently simple task can turn into a complex program.
- Complex debugging:
  - Exception handling
  - Non repeatable environment
  - Real-time aspects …

Object level languages

- Goal: free the programmer from manipulation and displacement details.

- Characteristic: the task is described by instructions which define the operations to be performed on objects, independently of the robots and tools.

Under development. No industrial product.

Part V

The Val II and V+ languages

Characteristics

- End effector level languages.
- Structured.
- Interpreted.
- Subroutines:
  - Val II: no parameters and all variables are global.
  - V+: parameters, automatic variables, global variables
- No explicit declarations but variables must be initialized before use.

Types in Val II

- Real (no explicit integer type).
- Vectors (no matrices)
- « Precision points »: vector of joint angles.
- Cartesian points: X, Y, Z in mm + 3 angles in degrees.

Types in V+

- Same + characters and strings.
- Also matrices.
- Different orientation representation.

Operators

- Arithmetic operators
  - +, -, *, /, MOD
- Logical operators
  - AND, OR (inclusive), NOT
- Relational operators
  - > , >= , < , <=
  - <>
  - ==
- Bitwise operators

Mathematical functions

- Abs(expr)
- sign(expr)
- fract(expr)
- int(expr)
- sqr(expr)
- sqrt(expr)
- Sin(expr)
- cos(expr)
- atan2(y,x)

Structures

Conditional structure

```
if  j == 2 then
    move A
end
```

Alternative structure

```
if  sign(x) == 1 then
    move A
else
    move B
end
```

Multiple choice

```
case i of
    value 1,2 :
        move A
    value 3 :
        move B
    any
        move C
end
```

Loop structures

« for » loop

```
for i = binf to bsup [step 2]
    ...
end
```

« while » loop

```
while abs(x-y) > eps do
    ...
end
```

« repeat - until » loop

```
do
    ...
Until abs(x-y) < eps
```

NB: brackets […] denote an optional parameter.

Jumps

Unconditional jump

goto <label>

label: integer value

Conditional jump

if sign(x) == -1 goto 10

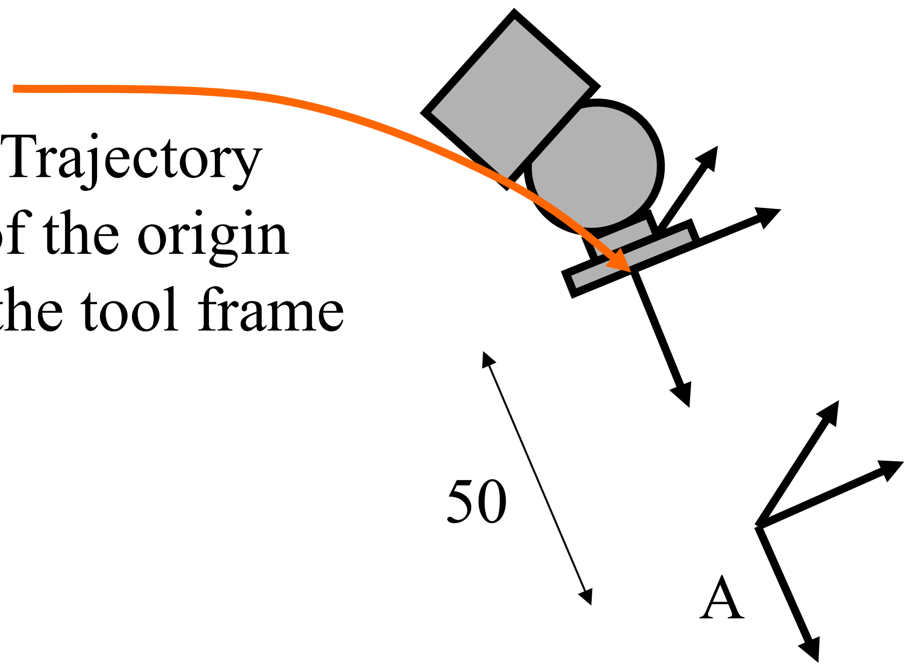
Jumps should be reserved to specific situations, like the handling of exceptions.
Otherwise, use algorithmic structures.

Motion instructions

Approaching a point
joint space interpolation

- appro <point> , <distance in mm>

Trajectory
of the origin
of the tool frame



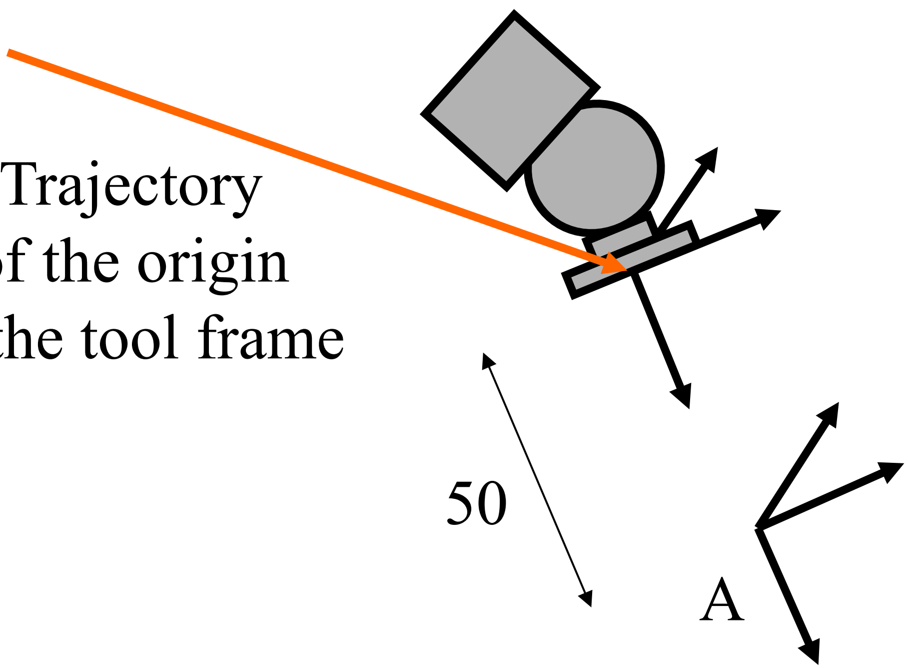
Position after
appro A , 50

The orientation is the same as that of A.

Approaching a point
straight line motion

- appros <point> , <distance in mm>
  Use only when necessary.

Trajectory
of the origin
of the tool frame



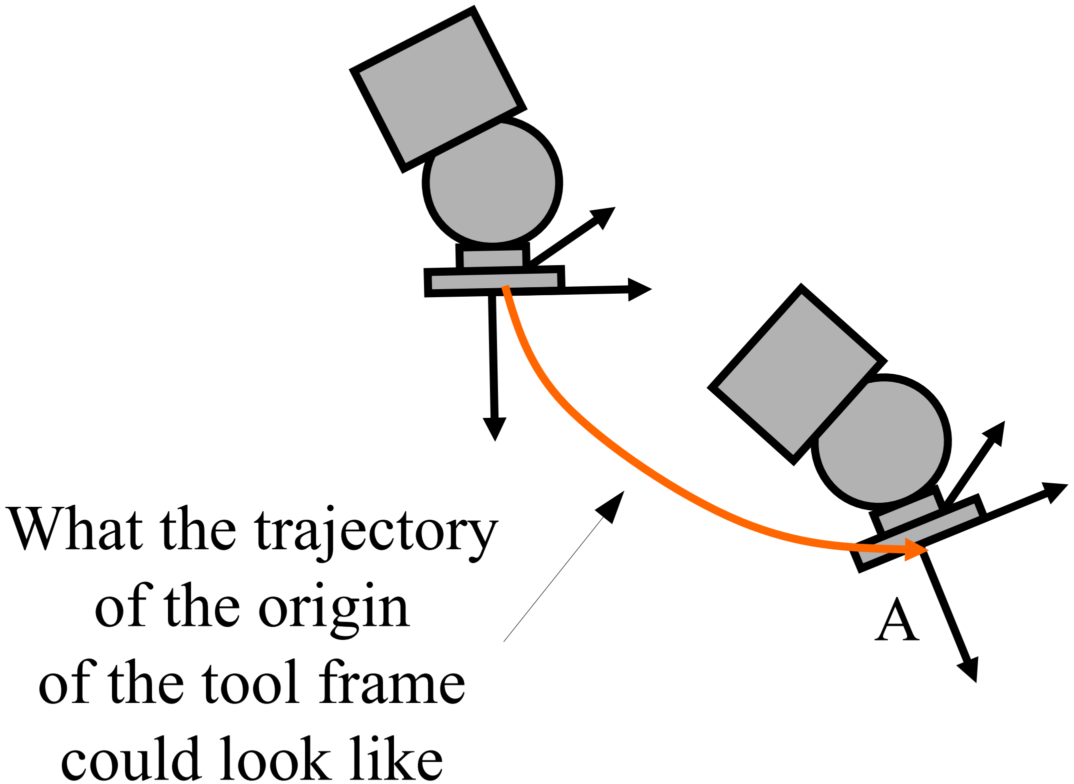
Position after
appros A , 50

The orientation is the same as that of A.

Same final position as Previously

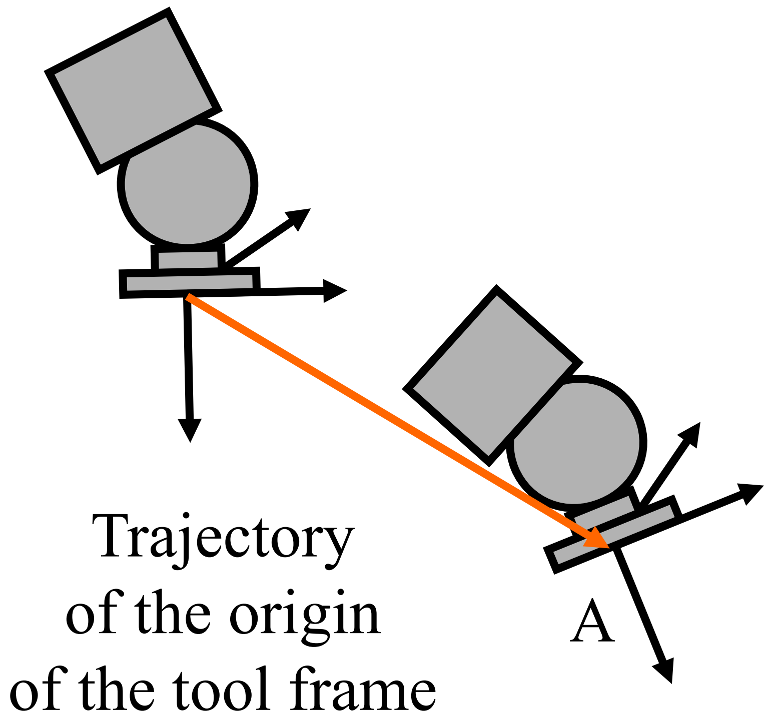
Moving to a point
joint space interpolation

- move <point>



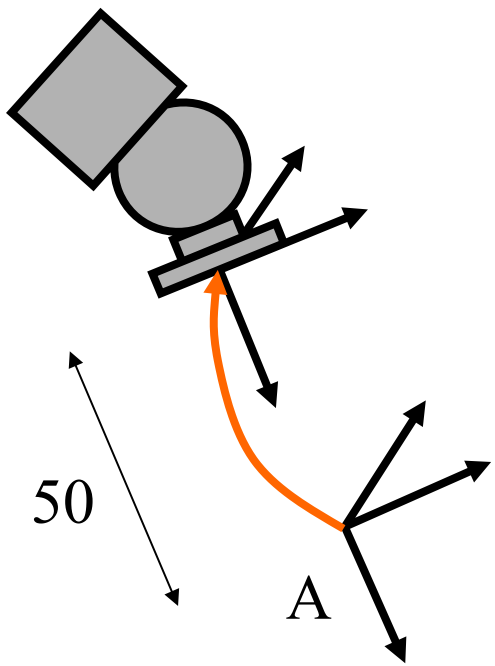
Moving to a point
straight line motion

- moves <point>
  Use only when necessary.



Departing motion
joint space interpolation

- depart <distance in mm>
  Rarely used: close to objects, one likes to know the exact trajectory…

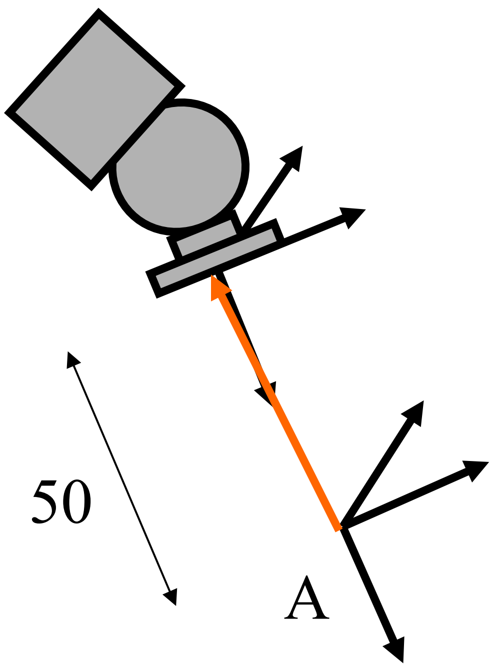


Starting from A, position after depart 50

Orientation is not constant along the motion but is identical in the initial and final positions.

Departing motion
straight line motion

- departs <distance in mm>



Starting from A, position after depart 50

Orientation is constant along the motion.

Standard pick and place sequences

Pick sequence

appro pick.loc , 100
moves pick.loc
closei
departs 50

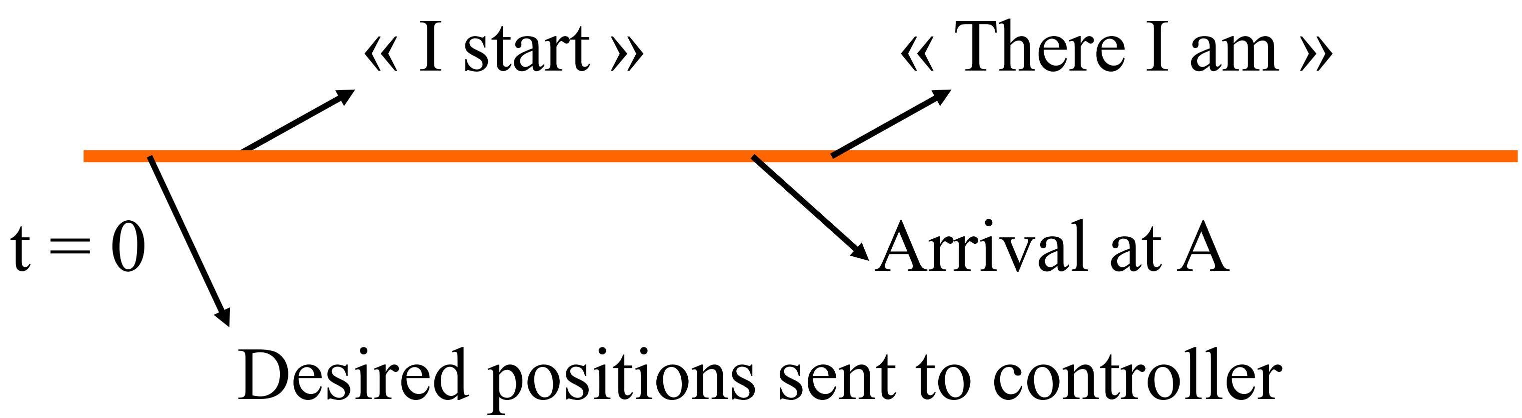
Place sequence

appro place.loc , 50
moves place.loc
openi
departs 100

- Closei (close « immediate » is for closing fast (typically pneumatic) grippers. Closing takes place once the destination point is reached. It is a synchronous gripper closing. Close is not.
- Also note that motions which happen close to objects are straight line motions.

Synchronous vs asynchronous motions

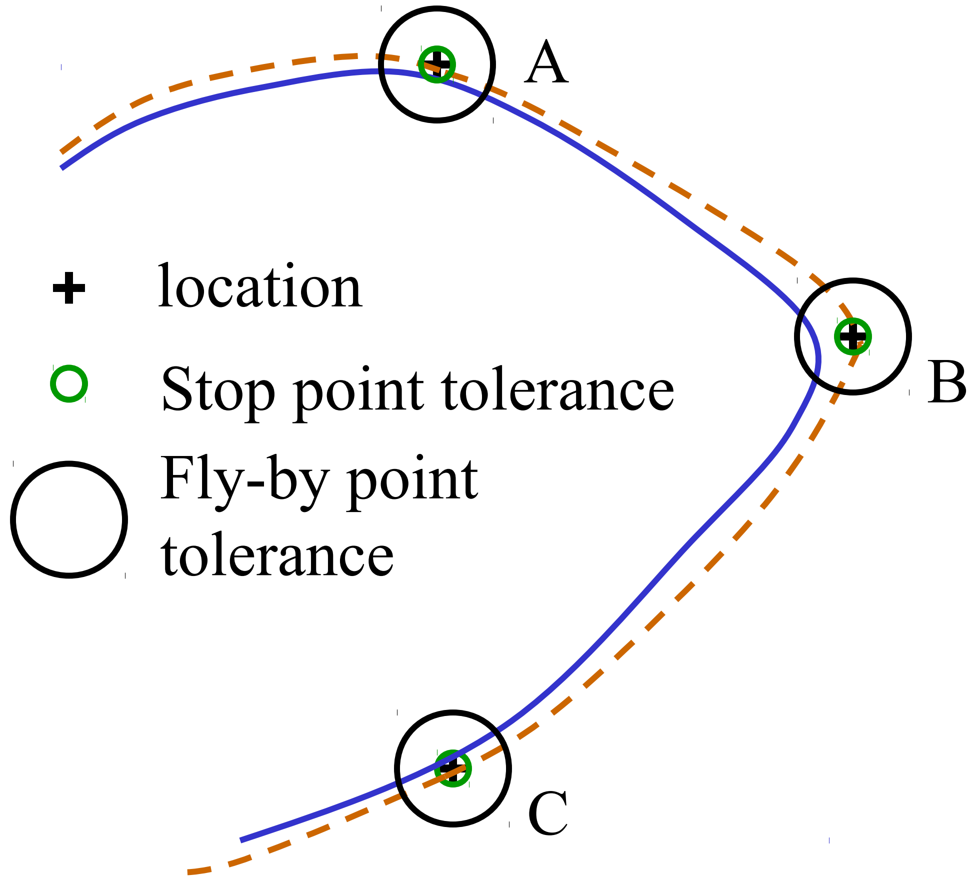
move A
type « I start » ; Instructions executed
... ; during the motion.
break ; Wait until current motion is over.
type « There I am »



Breaking continuous path

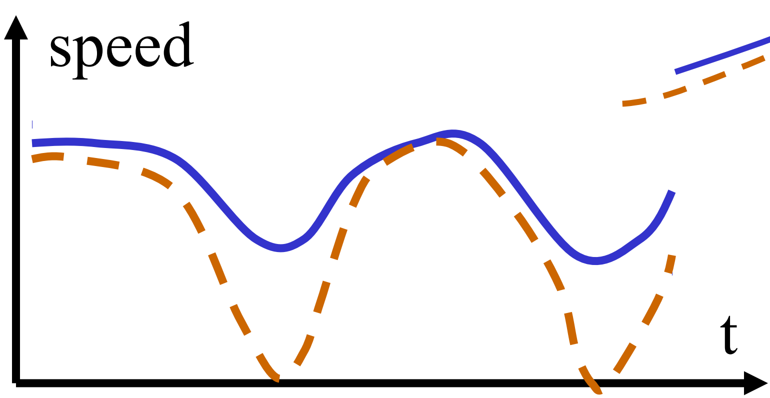
Fly-by points:
move A
move B
move C
move …

path



Stop points:
move A
break
move B
break
move C
break
move…

path



« break » forces an error nulling process which in turn forces « zero » speed.

Breaking continuous path

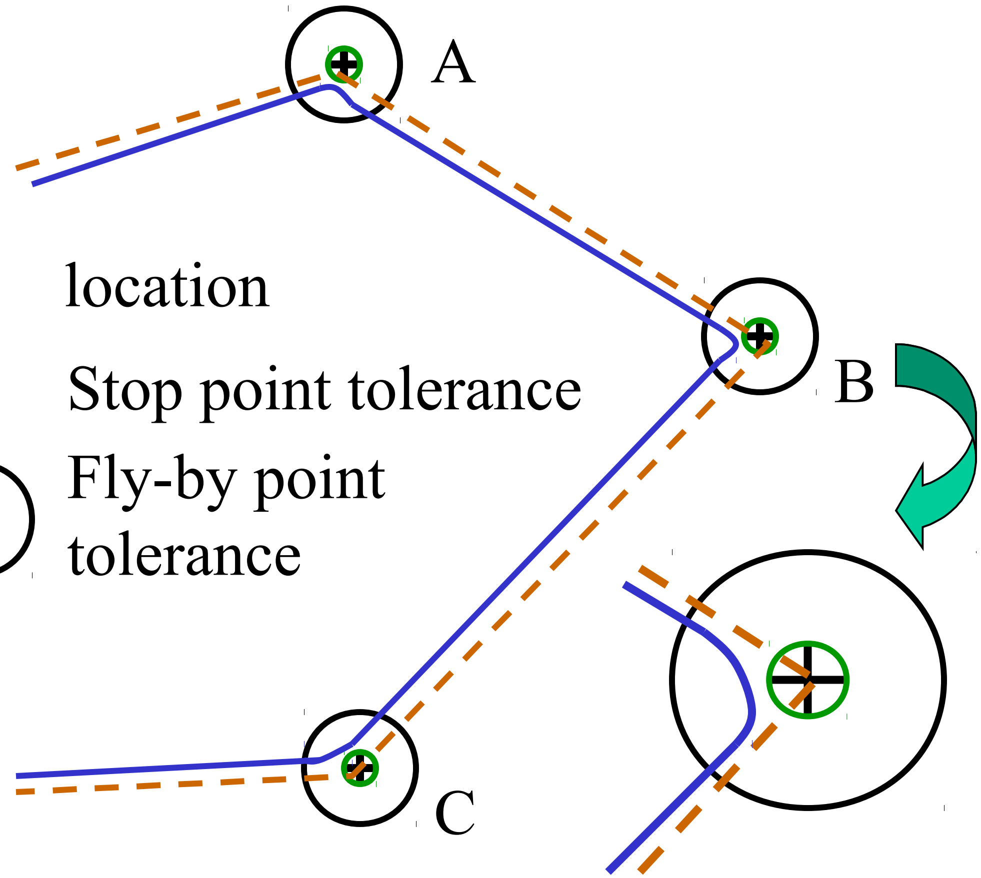
Fly-by points:
moves A
moves B
moves C
moves …

path

+ location

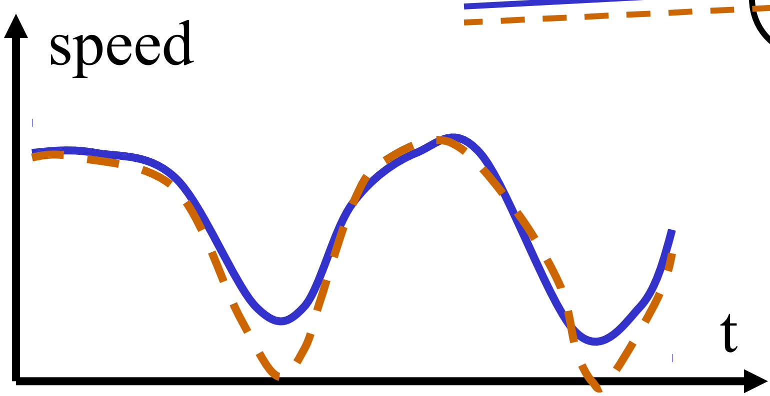
Stop point tolerance

Fly-by point tolerance



Stop points:
moves A
break
moves B
break
moves C
break
moves…

path



The robot will slow down more with straight line motions, even without « break »

Setting the speed

- The speed (sp) **monitor command**:
  - Is **not** a Val II / V+ instruction
  - Sets a system parameter value, which has a default value.
  - Is expressed in percent
  - All desired speeds mentionned in the program will be multiplied by this percentage.
  - Monitor speed can be progressively increased when testing a task program.

Setting the speed

- The speed **instruction**:
  - speed <percentage>: sets the speed for the next joint-interpolated motion.
  - speed <percent> always: sets the **default** speed for subsequent joint interpolated motions
  - speed <v> mmps: set the speed in mm/s for the next straight line motion.
  - speed <v> mmps always: sets the **default** speed in mm/s for subsequent straight line motions

Points, frames and transformations

Their manipulation in Val II and V+

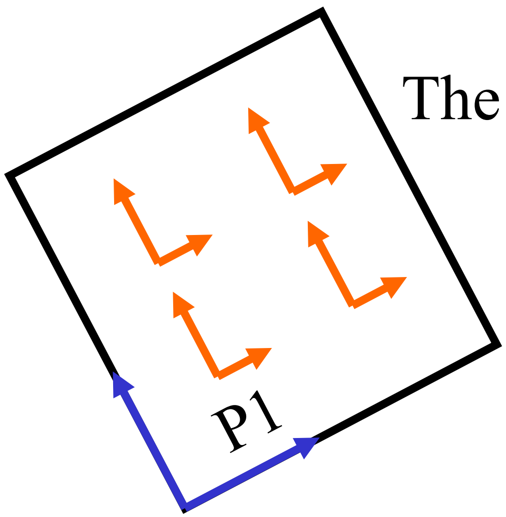
The three notions are the same

- A point represents a pose (position and orientation).
- It can also be considered as a frame in 3D space.
- A point is defined by six coordinates: it also corresponds to the transformation between two frames (by default between the considered frame and the base frame of the robot).

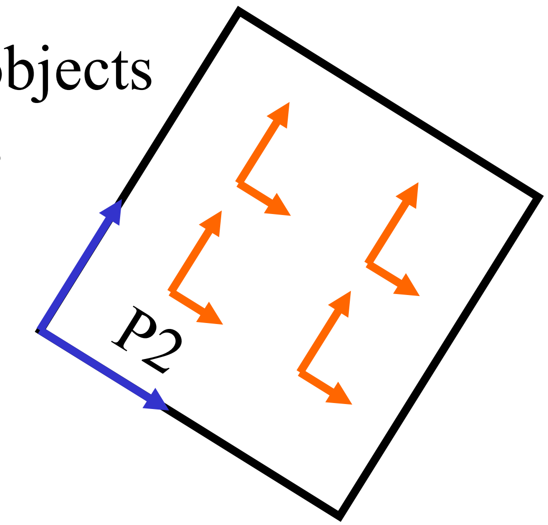
Compound transformations

- move A
  - the coordinates of A are expressed in the base frame of the robot **or** A is the transformation between R_A and R_{base}
- move A:B
  - the coordinates of B are expressed in frame A **or** : B is the transformation between frames R_A and R_B
- move A0:A1: ... An ; is possible

Interest of compound transforms



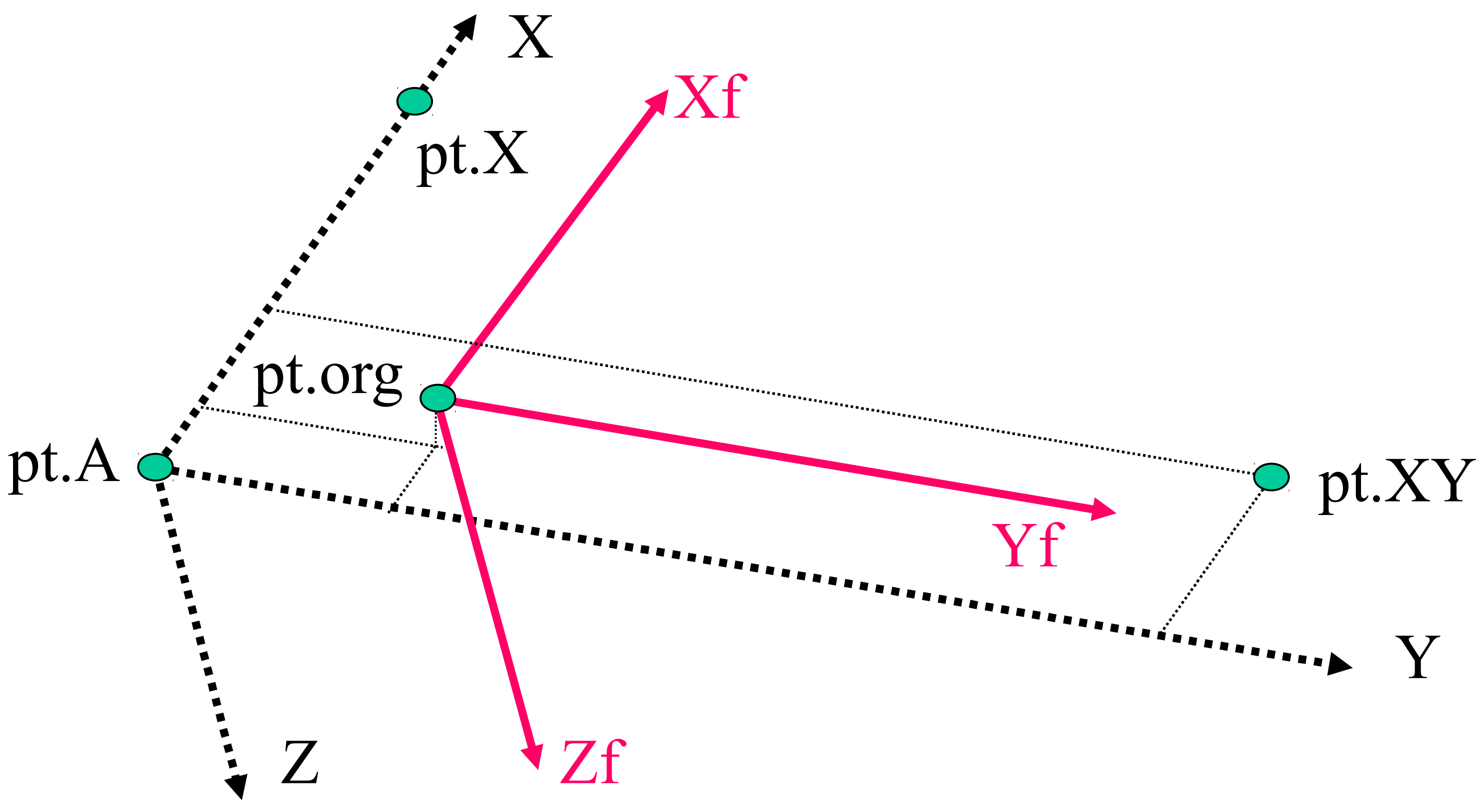
The robot works on two identical objects
The coordinates of the points
relative to both objects
are identical



set object = P1
set A1 = shift(null by d1,d2,0)
appro object:A1, 50
...

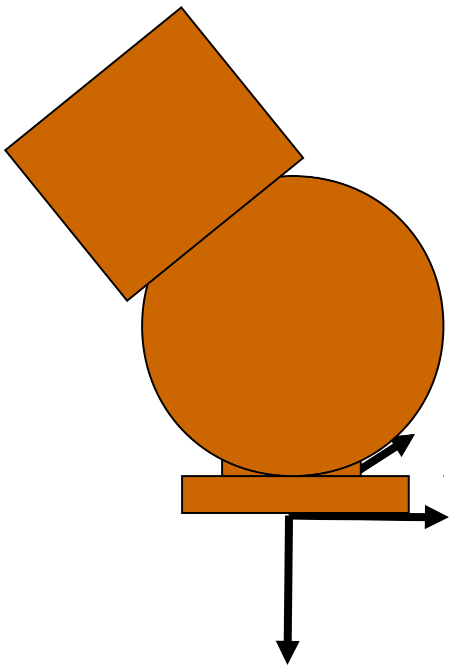
...
set object = P2
appro object:A1, 50
...

Defining a frame using three points

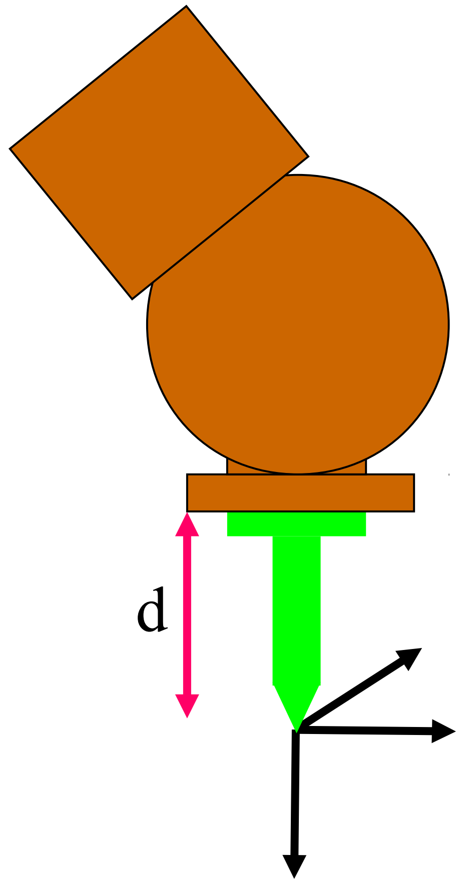


set my.frame = frame(pt.A, pt.X, pt.XY, pt.org)

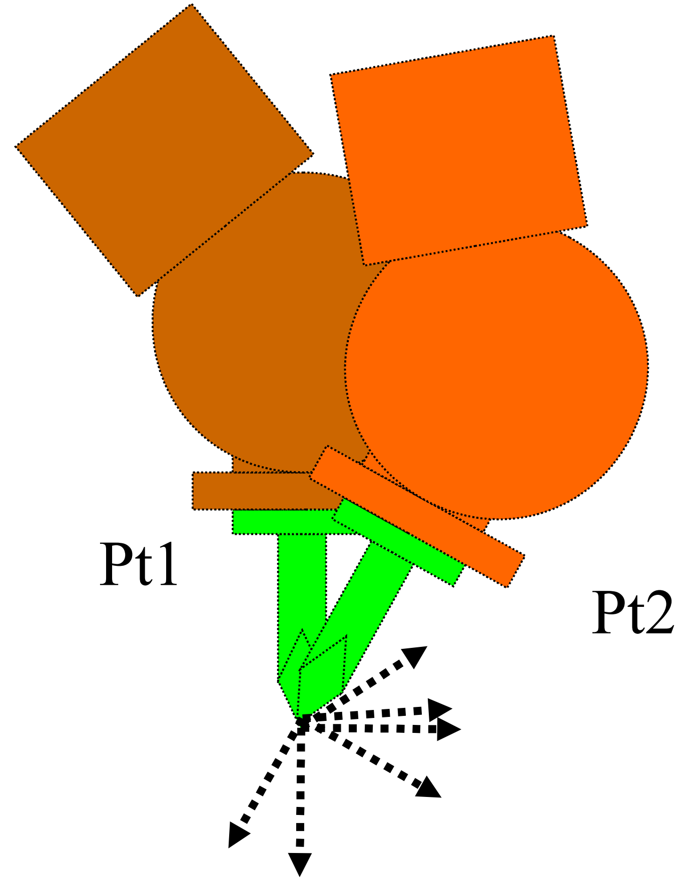
The frame teaching tool



Default tool frame

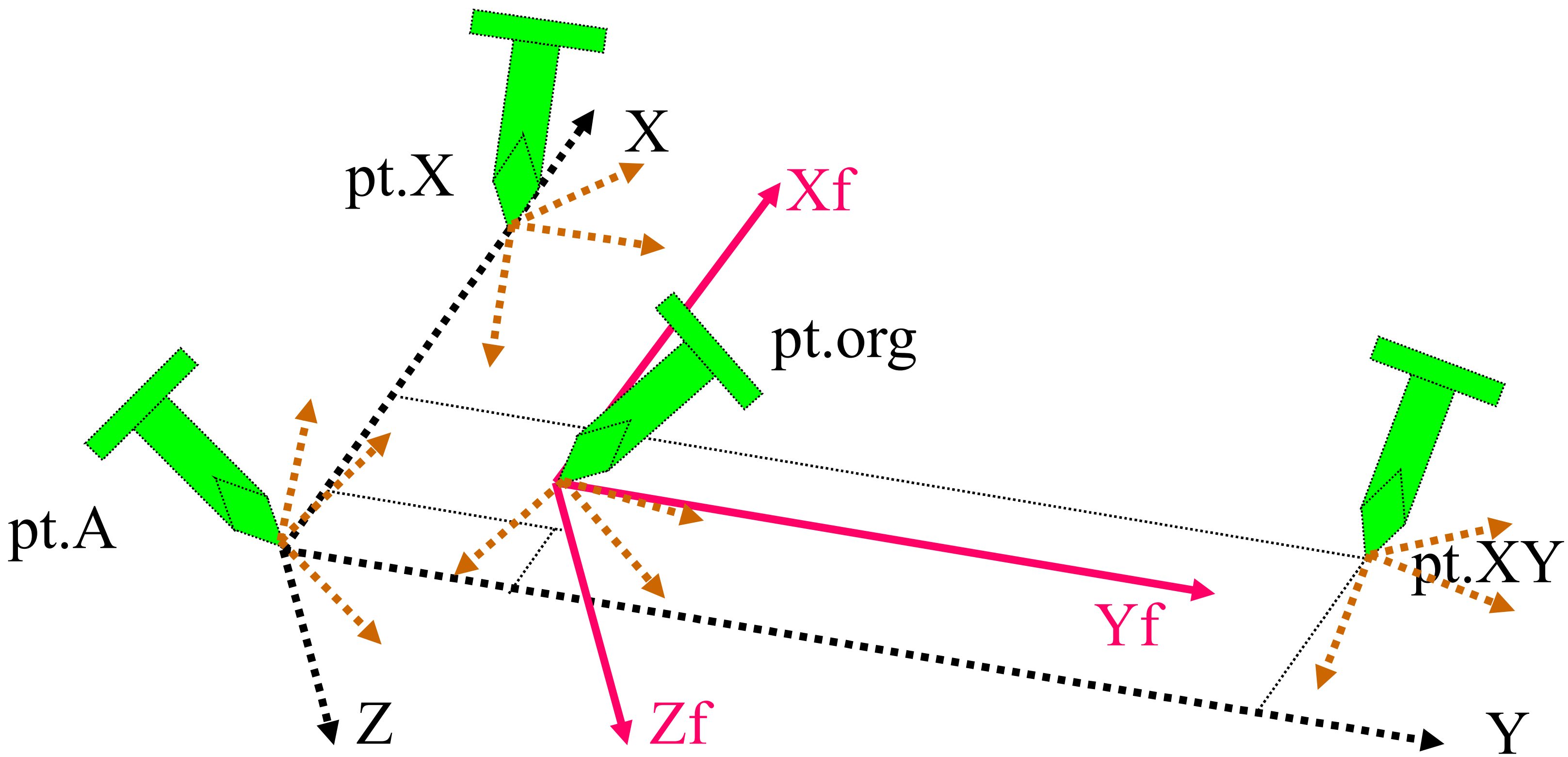


New tool frame after tool shift(null by 0,0,d)



Pt1 and Pt2 equivalent when used in « frame »
Don’t bother about orientation…

Possible positioning of the pointer tool



The orientation of the pointer tool does not matter if its tool frame has been correctly set.

Digital inputs and outputs

The various ways to use them

Input/output numbering

- Inputs :
  - numbered 1001 to 1032
- Outputs :
  - numbered 1 to 32

Example :

```
; Inputs
part.detected = 1005

; outputs
conveyor.start = 1
conveyor.stop = -1
```

```
...
wait sig(part.detected)  ; clear and
signal(conveyor.stop)    ; autodocumented
```

~~wait sig(1005)~~ ; not as clear.

sig reads an input
signal sets an output

Synchronous mode

The input state is tested at a particulier point of the program, chosen by the programmer.

```
if sig(part.det.P1) then
    set P = P1
else
    set P =P2
end

...
wait sig( part.detected )
```

Asynchronous mode

or reaction mode

- react <input> , <subroutine> [, <priority>]
- reacti <input> , <subroutine> [, <priority>]

• With « reacti », the current motion is stopped (robot speed is immediately nulled).

•With « react », the current motion is unaffected.

If the interrupt subroutine starts by instructions which are not motion instructions, they are performed while the current motion finishes.

Internal bits

- Internal bits are internal boolean variables.
- They are numbered 2001…2032 (2256 in V+).
- They are manipulated similarly to inputs/outputs with sig (read) and signal (set).
- React/reacti instructions can be associated to the first 8 internal bits.

- Typical use: inter-process communication.

Switch to case study…