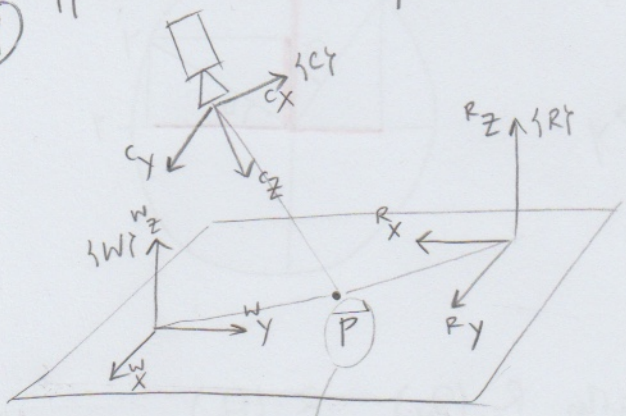
3D TRANSFORMATIONS (RIGID)

① Different coordinate systems



Point $\vec{P}$ observed from 3 different coord. systems

WATCH OUT:
$\vec{P}$ IS A VECTOR (embedded in a rigid solid)
WE CAN APPLY ROTATIONS ON VECTORS !!!

② Homogeneous transformations

$$ {}^{C}\vec{P} = \begin{bmatrix} {}^{C}(x_P) \\ {}^{C}(y_P) \\ {}^{C}(z_P) \\ 1 \end{bmatrix} \qquad {}^{R}\vec{P} = \begin{bmatrix} {}^{R}(x_P) \\ {}^{R}(y_P) \\ {}^{R}(z_P) \\ 1 \end{bmatrix} $$

but, careful: in drawings arrow is the other direction!!

$$ {}^{R}\vec{P} = {}^{R}_{C}H \; {}^{C}\vec{P} = {}^{R}_{C}R \cdot {}^{C}\vec{P} + {}^{R}\vec{t} $$

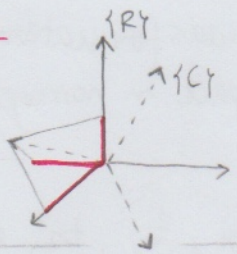
$$ {}^{R}_{C}H = \left[\begin{array}{c|c} {}^{R}_{C}R & {}^{R}\vec{t} \\ \hline 0\;0\;0 & 1 \end{array}\right] $$

" from R, where is C "

$$ {}^{R}_{C}R = \left[{}^{R}\underset{\sim}{X}_C \;\middle|\; {}^{R}\underset{\sim}{Y}_C \;\middle|\; {}^{R}\underset{\sim}{Z}_C \right] $$

- $\|R\| = 1$
- $R \cdot R^{-1} = I$
- symmetric

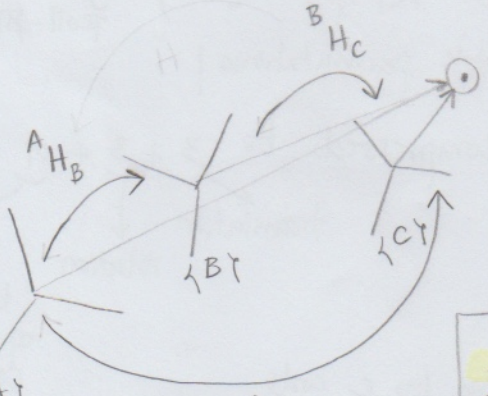
$\underset{\sim}{X}$ from/of C expressed in R (old expressed in new)



translation necessary from/of C to move to R expressed in R !

or: the origin of C viewed from R (new)

③ Chain & inverse rules:



$$ {}^{A}H_C = {}^{A}H_B \; {}^{B}H_C $$

$$ {}^{C}H_A = ({}^{A}H_C)^{-1} $$

④ Notes on homogeneous transformations

$$ \underset{\sim}{H} = \left[\begin{array}{c|c} \underset{\sim}{R} & \vec{t} \\ \hline 0 & 1 \end{array}\right] = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0\;0\;0 & & & 1 \end{bmatrix} \cdot \begin{bmatrix} \underset{\sim}{R} & 0 \\ 0 & 1 \end{bmatrix} $$

$$ = \underset{\sim}{H}(\vec{t}) \cdot \underset{\sim}{H}(\underset{\sim}{R}) $$

" First translate coord. system, then rotate !!! "

$$ \underset{\sim}{H}_1 \cdot \underset{\sim}{H}_2 = \left[\begin{array}{c|c} \underset{\sim}{R}_1 & \vec{t}_1 \\ \hline 0 & 1 \end{array}\right] \cdot \left[\begin{array}{c|c} \underset{\sim}{R}_2 & \vec{t}_2 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} \underset{\sim}{R}_1 \cdot \underset{\sim}{R}_2 & \underset{\sim}{R}_1 \vec{t}_2 + \vec{t}_1 \\ \hline 0 & 1 \end{array}\right] $$

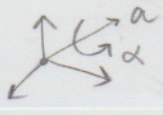
" Don't assume scalar operation order !! "

⑤ Rotations: Representations and conventions

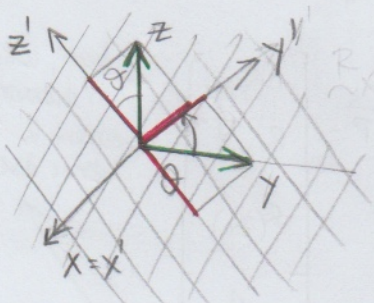
* Only 3 rotation values are necessary, but order of application is important

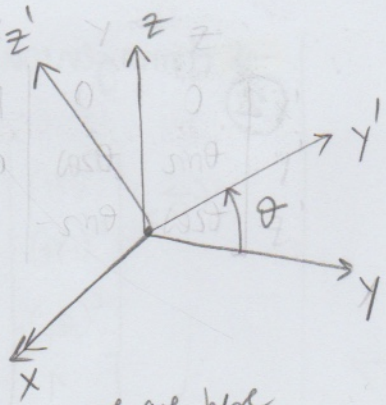
- Types of representations
  - Rotation matrix: 3+3+3 = 9 values
  - Quaternions: 4 values
  - Angle-axis: 3+1 values
  - Dual quaternions: 4+4 → rigid body

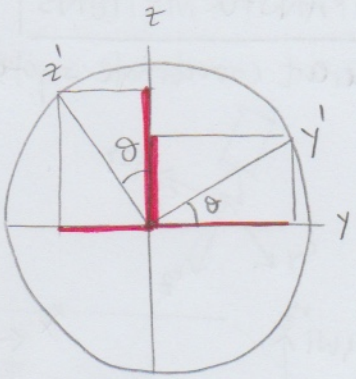
→ ORDER OF ROTATION APPLICATION MATTERS !!



(6) Rotation order







we are here

$$\underset{\sim}{R}_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{matrix} x \\ y \\ z \end{matrix}$$

x' y' z'

y' in new {x,y,z} coordinates

we want to go here

→ similarly $\underset{\sim}{R}_y(\theta_y)$, $\underset{\sim}{R}_z(\theta_z)$ can be computed and assembled with matrix multiplication:

$$\underset{\sim}{R} = \underset{\sim}{R}_z(\alpha) \cdot \underset{\sim}{R}_y(\beta) \cdot \underset{\sim}{R}_x(\gamma) : \text{abg}$$

Yaw-Pitch-Roll

or

$$\underset{\sim}{R} = R_x(\gamma) \cdot \underset{\sim}{R}_y(\beta) \cdot R_z(\alpha) : \text{gba}$$

Roll-Pitch-Yaw

... 24 possible permutations!

(7) Pose: rigid transformation in 3D space ~ homogeneous transformation compressed to 3+3+1 values (tuple)

3 → translation, 3 → rotation, 1 → type of rotation code

create_pose (t_x, t_y, t_z, r_x, r_y, r_z, c)

r_x, r_y, r_z → assembled according to c code

$c \in [0,13]$ and encodes

- Order of transform: 'R·p+t' / 'R(p-t)' (matrix operation)
- Order of rotation: 'abg' / 'gba' / 'rodrigues' (select accordingly)
- View of transform: 'point' / 'coord_system' (pre/post mult)

~ use these, avoid others

if other order of rotation, create pose with homogeneous matrices & then convert to pose.

HALCON OPERATORS	Homogeneous matrices	
hom_mat3d_identity	$\underset{\sim}{H} = \underset{\sim}{I}$	
– translate	$\underset{\sim}{H}_2 = \underset{\sim}{H}(t) \cdot \underset{\sim}{H}_1$	(regular)
– translate_local	$\underset{\sim}{H}_2 = \underset{\sim}{H}_1 \cdot \underset{\sim}{H}(t)$	(not regular)
– rotate	$\underset{\sim}{H}_2 = \underset{\sim}{H}(R) \cdot \underset{\sim}{H}_1$	(regular)
– rotate_local	$\underset{\sim}{H}_2 = \underset{\sim}{H}_1 \cdot \underset{\sim}{H}(R)$	(not regular)
– compose	$\underset{\sim}{H}_3 = \underset{\sim}{H}_1 \cdot \underset{\sim}{H}_2$	
– invert	$\underset{\sim}{H}_2 = (\underset{\sim}{H}_1)^{-1}$	
affine_trans_point_3d	$\vec{P}_2 = \underset{\sim}{H} \cdot \vec{P}_1$	

REGULAR: PRE-Multiplication → apply along/around OLD axes

NOT REGULAR: POST-Multiplication → apply along/around NEW "

HALCON OPERATORS	Poses
create_pose (deg expected!)	
hom_mat3d_to_pose	$\underset{\sim}{H}$ ↔ Pose
pose_to_hom_mat3d	
convert_pose_type	change code c or rotation encoding
pose_invert	
pose_compose:	multiply poses sequentially
write_pose	
read_pose	
set_origin_pose	

(8)

Dual Quaternions

- They can represent a complete 3D rigid transformation (translation & rotation) with 8 scalars
- They can:
  - be concatenated, like hom. matrices
  - be used to interpolate between 2 3D rigid transformations
  - be used to easily transform lines

$$\hat{q} = q_r + \varepsilon * q_d$$

- $\hat{q}$: Dual quaternion — consists of 2 unit quaternions
- q_r: Real part: unit quaternion
- ε: unit dual number: $\varepsilon\varepsilon = 0$ (?)
- q_d: Dual part: unit quaternion

each of q_r or q_d:

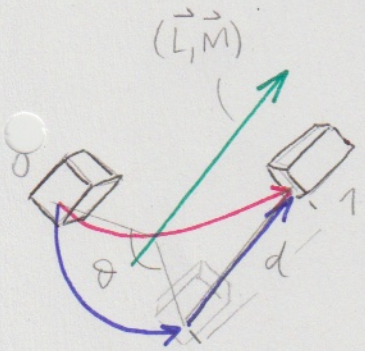
$$q = W + iX + jY + kZ$$

- W scalar part
- $\vec{v} = (X, Y, Z)$ vector part
- $(1, i, j, k)$ basis elements of quaternion space

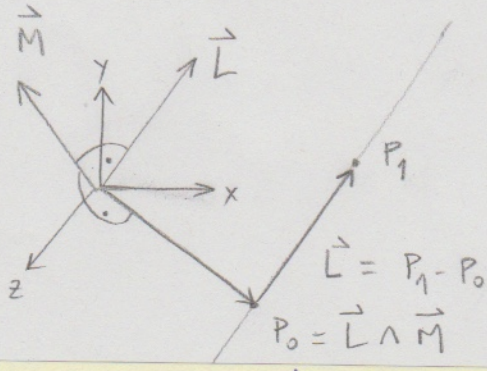
- Dual quaternions: representation in HALCON

$$\hat{q} = [W_r, \underbrace{X_r, Y_r, Z_r}_{\vec{v}_r}, W_d, \underbrace{X_d, Y_d, Z_d}_{\vec{v}_d}] = q_r + \varepsilon * q_d$$

- Screws:
  3D rigid transformations can be represented as screws



- object is transformed from 0 to 1 with
  - (screw) translation d
  - (screw) rotation θ
- directions are given by $\vec{L}$ & $\vec{M}$
  - $\vec{L}$: direction of screw translation, $\|\vec{L}\| = 1$
  - $\vec{M}$: moment of screw: $\vec{L} \cdot \vec{M} = 0$, $\vec{M} = \vec{P}_0 \wedge \vec{L}$



- Dual quaternion representation with screw model:

$$\hat{q} = \begin{pmatrix} \cos\theta/2 \\ \vec{L}\sin\theta/2 \end{pmatrix} + \varepsilon * \begin{pmatrix} -\frac{d}{2}\sin\theta/2 \\ \vec{M}\sin\theta/2 + \vec{L}\frac{d}{2}\cos\theta \end{pmatrix}$$

- Properties of dual quaternions:
  - * $\hat{q}$ and $-\hat{q}$ represent same 3D rigid transformation
  - * inverse of $\hat{q}$ is its CONJUGATE!!

HALCON OPERATORS

- pose_to_dual_quat
- dual_quat_to_pose
- dual_quat_compose
- dual_quat_interpolate
- dual_quat_to_screw
- screw_to_dual_quat
- dual_quat_to_hom_mat3d
- dual_quat_trans_line_3d
- dual_quat_conjugate
- dual_quat_scale
- ...

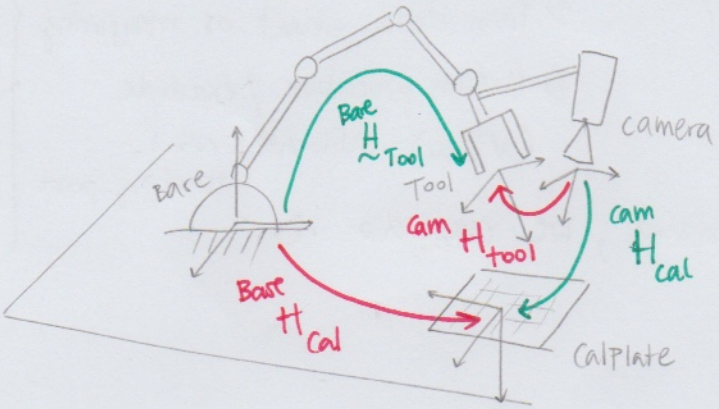
ROBOT VISION | HAND-EYE CALIBRATION

- Goal: transform object poses obtained by camera to robot coord. system.
- Variations
  - Robot type: articulated (6DoF) / SCARA (4 DoF - faster & more precise)
  - Camera mounting: stationary / moving
  - Calibration object (↔ camera): calibration plate (2D) / 3D object

(1)

→ Principle: "Close the transformation chain robot – camera – calibration object"

1) MOVING



$$^{cam}H_{cal} = {}^{cam}H_{tool} \cdot ({}^{base}H_{tool})^{-1} \cdot {}^{base}H_{cal}$$

(INPUT: $^{cam}H_{cal}$; RESULT: $^{cam}H_{tool}$; INPUT: $^{base}H_{tool}$; RESULT: $^{base}H_{cal}$)

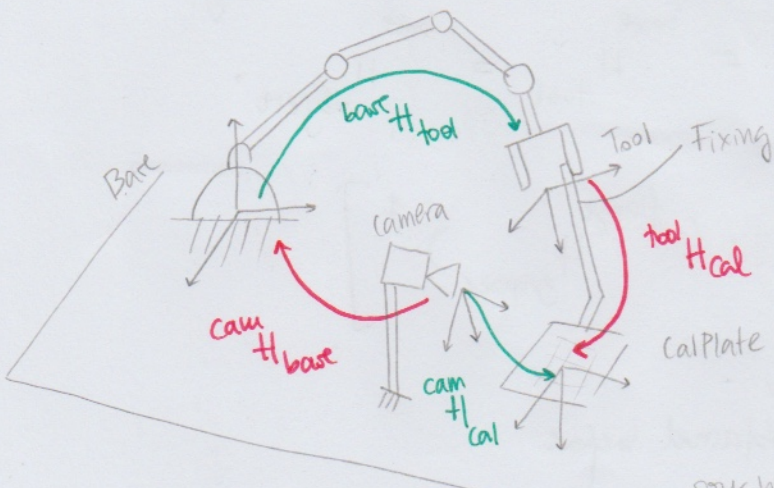
they must correspond!

Equation is solved using optimization method ⇒ several pose pairs (robot & cam) are necessary

→ LATER USE: known: $^{cam}H_{obj}$, $^{base}H_{tool}$

sought: $$^{base}H_{obj} = {}^{base}H_{tool} \cdot ({}^{cam}H_{tool})^{-1} \cdot {}^{cam}H_{obj}$$

2) STATIONARY



$$^{cam}H_{cal} = {}^{cam}H_{base} \cdot {}^{base}H_{tool} \cdot {}^{tool}H_{cal}$$

(INPUT; RESULT; INPUT; RESULT)

corresponding!

→ LATER USE

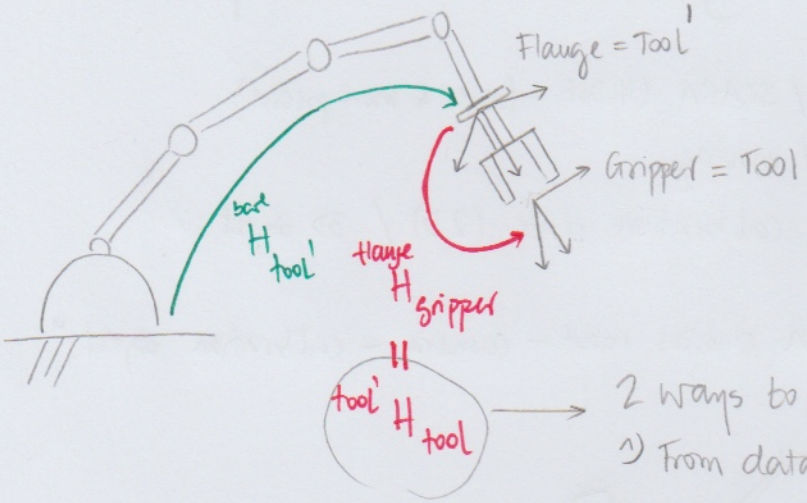
known: $^{base}H_{tool}$, $^{cam}H_{obj}$

sought: $$^{base}H_{obj} = ({}^{cam}H_{base})^{-1} \cdot {}^{cam}H_{obj}$$

(2) Gripper ≠ Tool : usually the tool that the robot thinks is the tool is the flange, and the real tool is the gripper mounted on the flange which it doesn't know...

↓

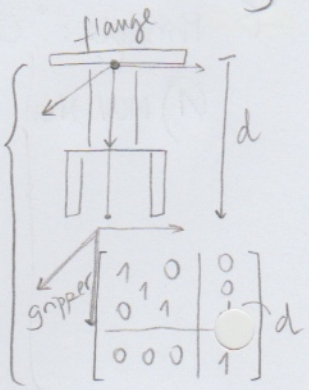
a correction is necessary



2 ways to compute

1) From data sheet or measuring

2) Halcon procedure / example (video): calibrate_robot_touching-point



After $^{flange}H_{gripper}$ is obtained, we can do there two things:

(1) CORRECT EVERY ROBOT POSE OBTAINED

$$\left[{}^{base}H_{tool} = {}^{base}H_{tool'} \cdot {}^{tool'}H_{tool} \right]$$

- $^{base}H_{tool}$ → real TCP pose of robot
- $^{base}H_{tool'}$ → obtained before!
- $^{tool'}H_{tool}$ = $^{flange}H_{gripper}$

(2) CORRECT THE OBJECT POSE SENT TO THE ROBOT

$$^{base}H_{tool'} \cdot {}^{tool'}H_{tool} = {}^{base}H_{tool} = {}^{base}H_{object}$$

$$\left[{}^{base}H_{tool'} = {}^{base}H_{object} \cdot \left({}^{flange}H_{gripper} \right)^{-1} \right]$$

- $^{base}H_{tool'}$ → commanded to robot
- $^{base}H_{object}$ → obtained before

EXAMPLE: hand_eye_stationary_cam_calibration.hdev (STATIONARY)

```
* Create hand-eye calibration model and set parameters: camera, plate, optimization method
create_calib_data ('hand_eye_stationary_cam', 1, 1, CalibDataID)
set_calib_data_cam_param (CalibDataID, 0, [], StartCamParam)
set_calib_data_calib_object (CalibDataID, 0, CalTabFile)
set_calib_data (CalibDataID, 'model', 'general', 'optimization_method', 'nonlinear')

* Loop for at least 3 poses: take plate image and robot configuration for each pose
for I := 0 to NumImages by 1
    read_image (Image, ImageNameStart + I$'02d')
       or grab_image
       you can also save image
             write_image
       find_calib_object (Image, CalibDataID, 0, 0, I, [], [])
             pose automatically saved
       read_pose (DataNameStart + 'robot_pose_' + I$'02d' + '.dat', ToolInBasePose)
             or get robot pose somehow
             important: tool in base pose
             see notes on Poses of the Robot
             same for stationary / moving
             if pose obtained frm robotlive, you can save it also
                   write_pose
       set_calib_data (CalibDataID, 'tool', I, 'tool_in_base_pose', ToolInBasePose)
             must be manually set
             same for stationary / moving
endfor

* Check input pose consistency
check_hand_eye_calibration_input_poses (CalibDataID, 0.05, 0.005, Warnings)

* Calibration
calibrate_hand_eye (CalibDataID, Errors)

* Get camera calibration parameters
get_calib_data (CalibDataID, 'model', 'general', 'camera_calib_error', CamCalibError)
get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)

* Get hand-eye calibration poses
get_calib_data (CalibDataID, 'camera', 0, 'base_in_cam_pose', BaseInCamPose)
get_calib_data (CalibDataID, 'calib_obj', 0, 'obj_in_tool_pose', ObjInToolPose)

* Write to file: camera parameters & hand-eye calibration poses
write_cam_par (CamParam, DataNameStart + 'final_campar.dat')
write_pose (BaseInCamPose, DataNameStart + 'final_BaseInCamPose.dat')
write_pose (ObjInToolPose, DataNameStart + 'final_ObjInToolPose.dat')

* Check (optional): go through SAVED calibration images and robot poses again
for I := 0 to NumImages - 1 by 1
    read_image (Image, ImageNameStart + I$'02d')
    get_calib_data (CalibDataID, 'tool', PoseIds[I], 'tool_in_base_pose', ToolInBasePose)
       this pose is different for each image!
       BUT obtained and constant for each calibrated setup:
             stationary: BaseInCamPose, CalObjInToolPose
             moving: ToolInCamPose, CalObjInBasePose
       pose_compose (BaseInCamPose, ToolInBasePose, ToolInCamPose)
       pose_compose (ToolInCamPose, CalObjInToolPose, CalObjInCamPose)
             case for stationary setup:
             CalObjInCamPose      = cam_H_calplate = cam_H_base * base_H_tool * tool_H_calplate
                                  = BaseInCamPose * ToolInBasePose * CalObjInToolPose
    disp_3d_coord_system (WindowHandle, CamParam, CalObjInCamPose, 0.01)
endfor
```

model created; diff params for SCARA/artzul. Stationary/moving 2D/3D

camera calibration

CalPlate

equation solving method default: 'nonlinear'

CalPlate pose in cam coords automatically saved to model

indices must correspond!

Robot pose must be manually saved

Calibration in single operator! 2 calibrations done: camera calibration + hand-eye calibration

4 value tuple: RMS error for $\vec{t}$ & $\vec{r}$, max error for $\vec{t}$ & $\vec{r}$

camera calibration parameters

$^{cam}H_{base}$, $^{tool}H_{cal}$ for case STATIONARY

In this loop the CalPlate coords are displayed for each image, but these coords are computed with hand-eye calibration info, not with find_calib_object..