

Deck 02 · What learning replaces, and the notation for frames

2.1 Which boxes of the stack can learning replace?

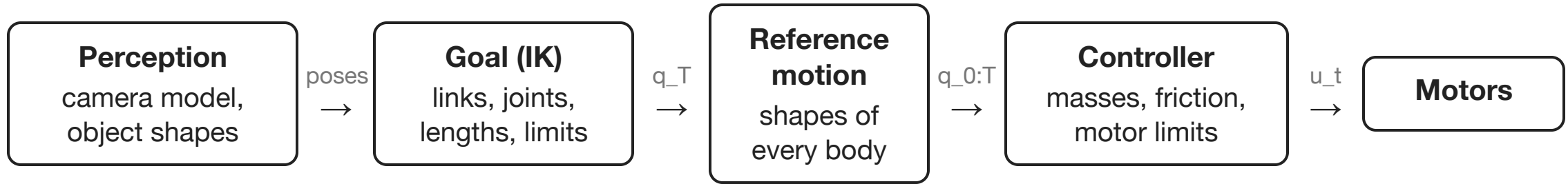
2.2 Frames, rotations, and monogram notation

Özgür S. Öğüz · Bilkent University · Fall 2026

What must the classical stack know before it can act — its "map" — and when is that knowledge wrong?

Perception reports the mug as three numbers. Measured from what, along whose axes, and which way is the mug turned?

Recap: the five boxes, and the map each one reads



Each box reads its part of the **map**; the arrows are what they hand each other: object **poses**, the joint angles **q** , a command **u** for the motors.

Today: what is on the map, and where it stops being right.

The words on the map, read off one mug

A map prints its legend in the corner. Here is ours, for the mug on the table:

In plain words	The map's word for it
the mug, the table, the arm's links, and which is bolted to which	the multibody system
how heavy the mug is, and how the weight is spread out	its inertia
where it stands, and which way it is turned: six independent freedoms	its pose — six degrees of freedom
how far each of the arm's seven joints has turned	the joint angles $\mathbf{q}$
the gap from the fingers to the mug: make it small, or demand it be zero	a cost , or a constraint
small joint changes have a well-defined local effect on that gap	the cost is differentiable
all of it as currently believed, from cameras and encoders	a state estimate

The next slide is these seven lines, compressed into three.

What the classical stack assumes to know

1. **The multibody system:** every solid part — links, table, mug — what is jointed to what, each part's shape and mass
2. **Every goal as a cost or a constraint:** a number to minimise, or a requirement such as a joint limit
3. **A state estimate:** $\mathbf{q}$, the object poses and friction *right now* — every number a guess from sensors

These three lines are the **map**. Each solver uses the parts relevant to its problem.

Not assumed knowledge: the kinematic tree **2.3** · cost, and what *differentiable* buys, **3.1** · constraint **4.1** · state estimation **weeks 5–6**

When the map is good enough: the robot by itself

- The arm's structure comes from the data sheet; encoders measure $\mathbf{q}$ to a fraction of a degree
- Moving through free space to a pose, or around known obstacles: **well-supported by classical methods**, when the model fits

"It is sometimes non-sensical to explore learning methods for these aspects."

M. Toussaint

Challenge 1a: interaction — locomotion

- A walking robot moves by pushing on the ground; its base is not bolted down
- Contact forces **switch on and off**: a foot pushes or it does not
- Continuous dynamics with a discrete on–off structure on top: **hybrid**

Challenge 1b: interaction — manipulation

- A loose mug has six **degrees of freedom**, but no joint motor of its own
- Supported mug, joint motion alone: **before contact, stays put; with a rigid no-slip grasp, follows the hand**
- Friction, slip, surface contact, deformable shapes

Where the analogy breaks. A route map treats the road network as fixed.
When the gripper closes, **the map is redrawn.**

Computed: how much the mug moves per joint

Before the grasp. Supported mug on the table, 10.5 cm below the **flange** (the plate the hand bolts onto), between the open fingers.

$$\mathbf{J}_{\text{mug}} = \frac{\partial {}^W \mathbf{p}^M}{\partial \mathbf{q}} = \mathbf{0}_{3 \times 7}, \quad \text{rank } 0$$

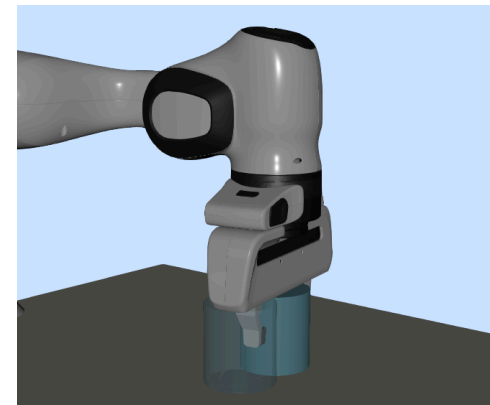
Base joint turns 0.2 rad: the gripper moves $(-0.006, 0.061, 0)$ m; **the mug moves** $(0, 0, 0)$

Same $\mathbf{q}$, same $\Delta \mathbf{q}$, same mug. Only the map changed.

After the grasp, modelled by gluing the mug to the gripper:

$$\mathbf{J}_{\text{mug}} = \begin{bmatrix} 0 & 0.152 & 0 & 0.130 & 0 & 0.212 & 0 \\ 0.307 & 0 & 0.325 & 0 & 0.212 & 0 & 0 \\ 0 & -0.307 & 0 & 0.472 & 0 & 0.088 & 0 \end{bmatrix}$$

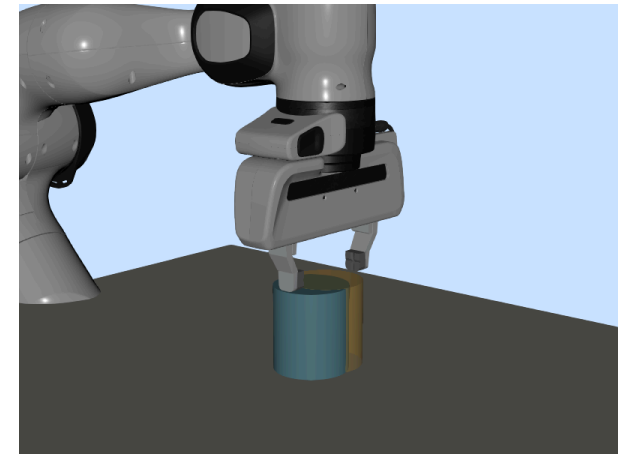
rank 3 — three independent directions. Same 0.2 rad: **the mug moves** $(-0.006, 0.061, 0)$ m.



Challenge 2: state estimation

- Everything so far needs **estimates**: $\mathbf{q}$, object poses, shapes, friction
- Good estimators exist; most are optimization problems **of the same shape** as weeks 3–4
- **Known** object: compare observations with its model. **Unknown** object: shape and pose must be inferred together

The need for state estimation is the main downside of model-based robotics solvers, and a core motivation for learning methods.



Computed: the mug believed 2 cm off (orange); the plan is exact, a finger lands on the rim and the arm stops. Tolerance across the fingers: **5 mm**

Which problems are easy, which are hard

Easy (reliable model-based methods)

- the robot itself, given the map:
kinematics, planning, tracking

Hard (the two challenges)

- predicting and controlling **contact**
- estimating the **state** of a scene

Shared foundations: both optimal control and reinforcement learning choose actions for a long-term objective (6.4)

The question is never *learning or models*.

Which component, and why.

What learning supplies, and what the stack keeps

What is learned	Example: the mug task	What can remain classical
A model or correction	Predict slip better	A planner using that prediction
A score for candidate actions	Predict which grasp succeeds	Search or selection using the score
A proposal or initial guess	Suggest a reaching posture	IK and feasibility checks
A perception map	Image $\rightarrow$ mug pose	Planning and control
A policy	Image $\rightarrow$ next action	The action interface and low-level controller

A policy is a rule mapping observations to actions. Learning can replace one calculation or several.

Where these learned components appear in the course

Classical component	Learned counterpart	Asg
Dynamics model	8.1 world models in the small · 12.1 sim-to-real, domain randomization	project
Action selection and control	10.4–11.2 policy gradients, value methods, RL as post-training	A3
Constraint solving / IK	5.1 learned samplers · 7.1 a policy from demonstrations	A2
State estimation	6.2 learned keypoints, grasps · 6.3 VLM perception	A2
Whole system	8.3–9.2 vision-language-action models · 12.2 world-action models	A3

What a learned component silently assumes

Learned component	What we must validate
Dynamics model	Predictions hold for deployment states and motions
Reward-based score or policy	Task score (reward) is appropriate; observations and data coverage support its choices
IK proposal	Goals and robot geometry are covered; constraints are checked
Perception map	Pose accuracy holds across deployment cameras, viewpoints and objects

Both need validation. Explicit models expose assumptions; learned components also depend on training coverage.

Three questions for every learned component, from now on

What does it replace? What does it keep? What does it silently assume?

Apply them: a network predicts the mug's world-frame pose from an image. Its output goes to the existing IK solver, planner and controller. The camera moves to a new viewpoint.

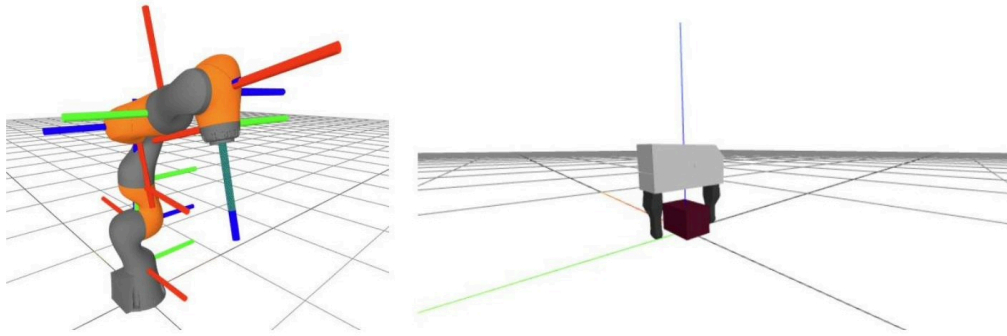
Summary of 2.1

- The map: **the parts and their joints, shapes and masses; goals as costs; a state estimate**
- For a well-modelled arm, start with reliable classical tools
- Two things break the map: **contact** and **state estimation**
- Learning can supply a model, score, proposal, estimate or action; check what the remaining stack needs

Break

Second half: frames, rotations, and one worked example you can check by hand.

Why frames



Every rigid body carries a **frame**: world, base, each link, gripper, camera, each object.

Perception reports poses; the goal box wants a pose; planning checks shapes at poses.

Frames are the language the boxes speak to each other.

A frame is an origin plus right-handed axes

- **Origin:** a point. **Axes:** three unit vectors, mutually perpendicular, right-handed
- A **vector** is an arrow; its **coordinates** depend on the axes you measure against
- Names: W world, G gripper, O object, C camera, M mug (today), T tray

Turn the axes by 90° : the arrow $(1, 0)$ becomes $(0, -1)$. Same arrow, different numbers.

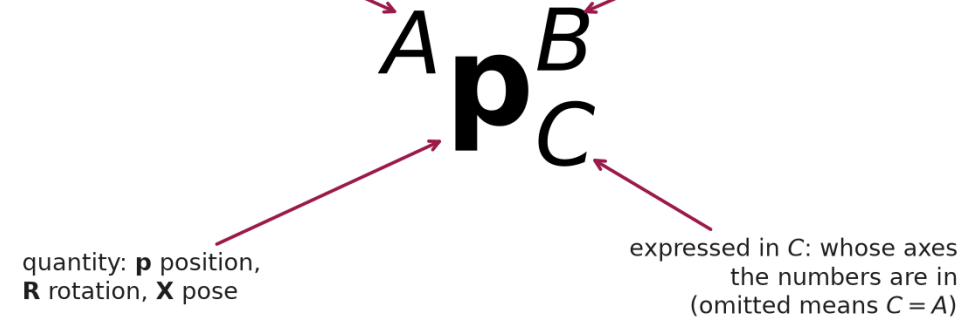
Monogram notation: three slots, three facts

$${}^A\mathbf{p}_C^B$$

- **left superscript A** : the reference, where the arrow starts
- **right superscript B** : the target, where it ends
- **subscript C** : whose axes the numbers are in; omitted means $C = A$

reference A
where the arrow starts
(a point or a frame)

target B
where the arrow ends
(a point or a frame)

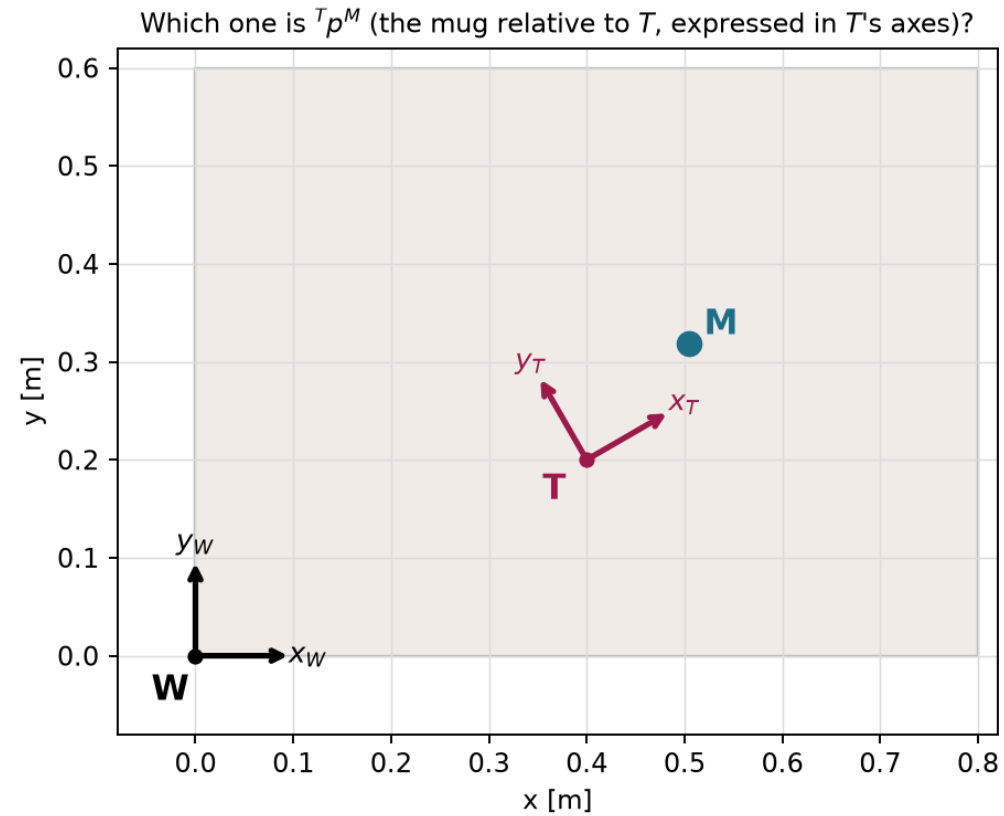


In code: `p_AB_C` , `R_AB` , `X_AB` .

Vectors and matrices in bold, frames plain.

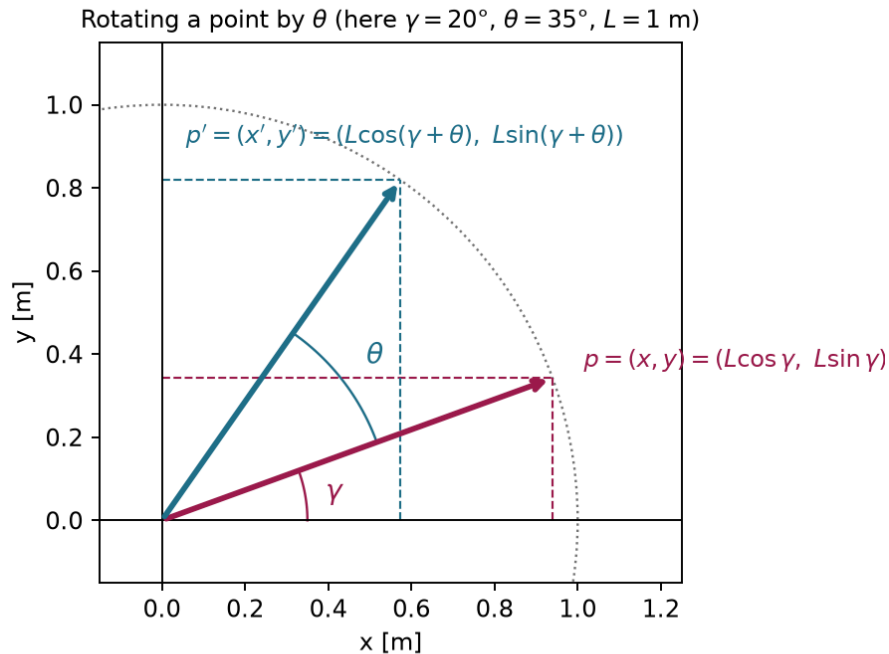
$\mathbf{R}$: the orientation of B , ${}^A\mathbf{R}^B$. $\mathbf{X}$: the pose, ${}^A\mathbf{X}^B = ({}^A\mathbf{R}^B, {}^A\mathbf{p}^B)$, written as a 4×4 matrix in 2.3.

Reading exercise



Which one is ${}^T \mathbf{p}^M$? a. (0.10, 0.12) b. (0.15, 0.05) c. (0.50, 0.32)

The rotation matrix, derived in two dimensions



$\mathbf{p} = (L \cos \gamma, L \sin \gamma)$, rotated by θ :

$$p'_x = p_x \cos \theta - p_y \sin \theta$$

$$p'_y = p_x \sin \theta + p_y \cos \theta$$

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

Rotation columns tell us which way to convert

- Columns of ${}^A\mathbf{R}^B$: B 's axes written in A
- B coordinates $\rightarrow A$ coordinates: multiply by ${}^A\mathbf{R}^B$
- A coordinates $\rightarrow B$ coordinates: multiply by $({}^A\mathbf{R}^B)^\top$

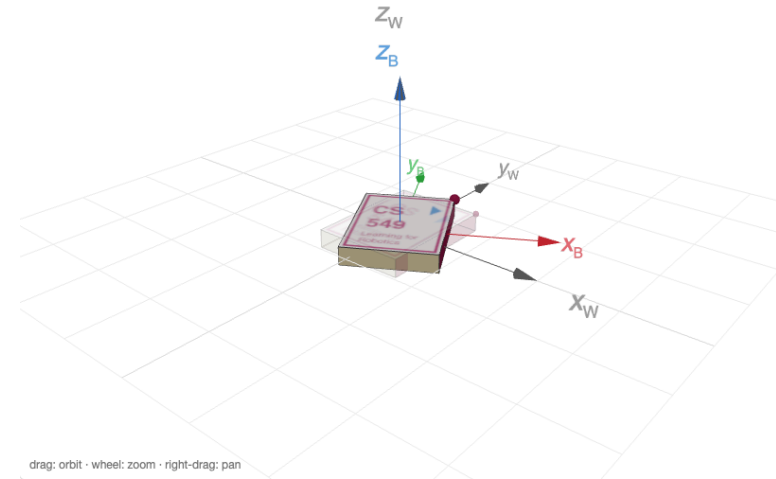
$$\mathbf{R}^\top \mathbf{R} = \mathbf{I}, \quad \det \mathbf{R} = 1, \quad \mathbf{R}^{-1} = \mathbf{R}^\top$$

Turn an **arrow**: active rotation. Change its **measuring axes**: passive conversion.
State the input and output frames.

The three elementary rotations

$$\mathbf{R}_z(\theta) = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \quad \mathbf{R}_y(\theta) = \begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}$$

- $\mathbf{R}_z$: the 2-D matrix in the upper-left block;
 $\mathbf{R}_x, \mathbf{R}_y$: rename axes cyclically $x \rightarrow y \rightarrow z \rightarrow x$
- **Check any rotation matrix you write: set $\theta = 90^\circ$ and read the columns**
- Radians in every formula and every line of code



Live: the columns are the coloured arrows

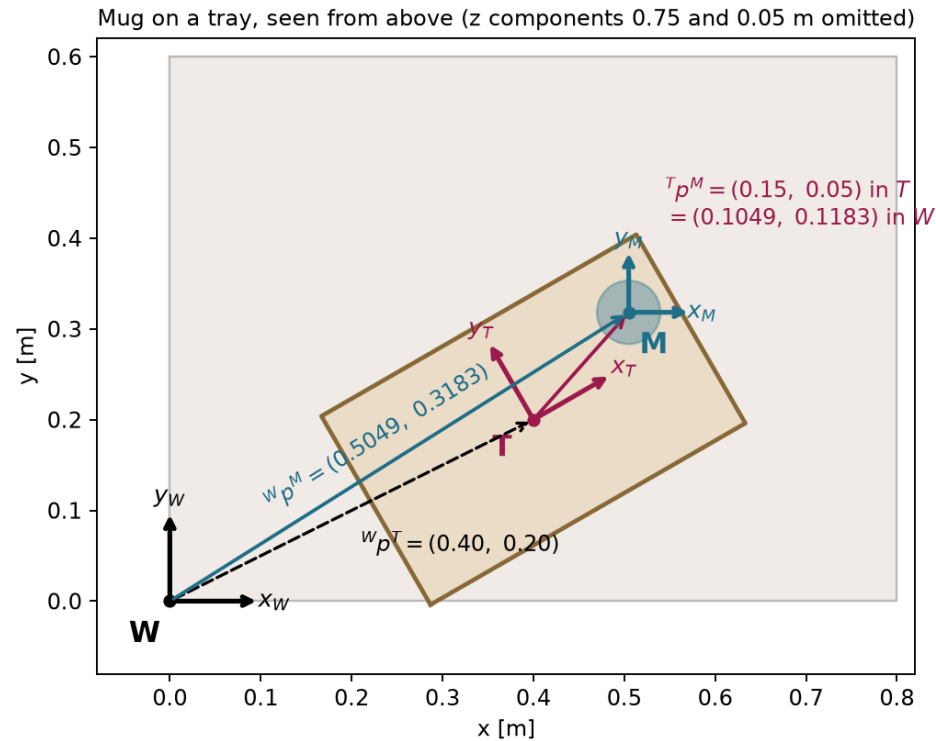
Changing the reference frame: the one equation of today

$${}^A\mathbf{p}^P = {}^A\mathbf{p}^B + {}^A\mathbf{R}^B {}^B\mathbf{p}^P$$
$${}^B\mathbf{p}^P = ({}^A\mathbf{R}^B)^\top ({}^A\mathbf{p}^P - {}^A\mathbf{p}^B)$$

Arrows add head to tail; the second arrow is known in B 's axes and must be converted first.

The inverse: **subtract the offset, then rotate back.**

Worked example (a): a mug on a tray on a table



$${}^A\mathbf{p}^P = {}^A\mathbf{p}^B + {}^A\mathbf{R}^B {}^B\mathbf{p}^P \quad {}^B\mathbf{p}^P = ({}^A\mathbf{R}^B)^\top ({}^A\mathbf{p}^P - {}^A\mathbf{p}^B)$$

$${}^W\mathbf{p}^T = (0.40, 0.20, 0.75) \text{ m}, \quad {}^W\mathbf{R}^T = \mathbf{R}_z(30^\circ)$$

$${}^T\mathbf{p}^M = (0.15, 0.05, 0.05) \text{ m}$$

$${}^W\mathbf{R}^T {}^T\mathbf{p}^M = \begin{bmatrix} 0.8660 \cdot 0.15 - 0.5 \cdot 0.05 \\ 0.5 \cdot 0.15 + 0.8660 \cdot 0.05 \\ 0.05 \end{bmatrix} = \begin{bmatrix} 0.1049 \\ 0.1183 \\ 0.05 \end{bmatrix}$$

$${}^W\mathbf{p}^M = (0.40 + 0.1049, 0.20 + 0.1183, 0.80) = (0.5049, 0.3183, 0.80)$$

Worked example (b), (c): same arrow, and the inverse problem

(b) One arrow, two spellings

$${}^T\mathbf{p}_T^M = (0.15, 0.05, 0.05)$$

$${}^T\mathbf{p}_W^M = (0.1049, 0.1183, 0.05)$$

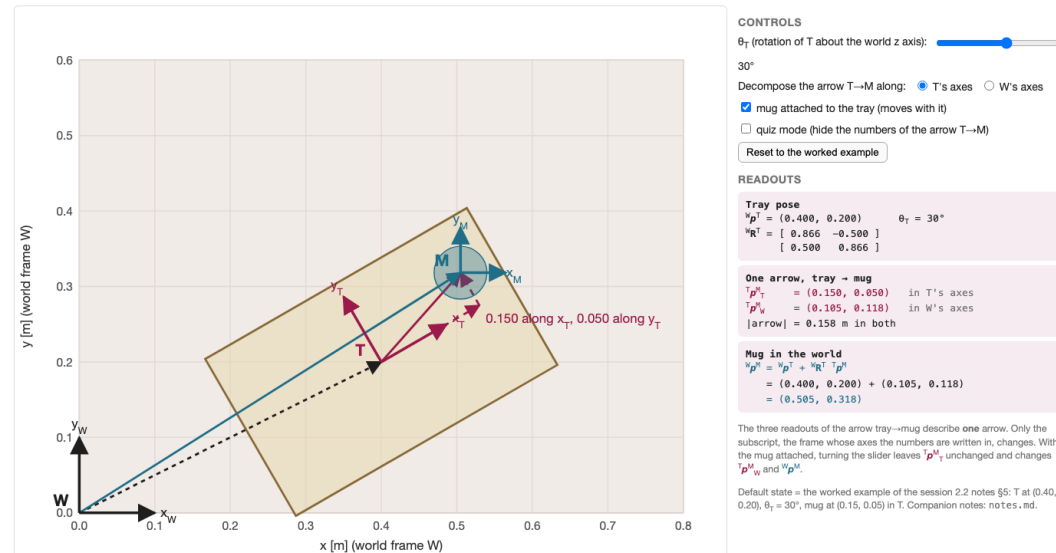
Same length in both: 0.1658 m.

(c) The camera saw ${}^W\mathbf{p}^M$. Where is the mug on the tray?

$$({}^W\mathbf{R}^T)^\top ({}^W\mathbf{p}^M - {}^W\mathbf{p}^T) = \begin{bmatrix} 0.8660 \cdot 0.1049 + 0.5 \cdot 0.1183 \\ -0.5 \cdot 0.1049 + 0.8660 \cdot 0.1183 \\ 0.05 \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.05 \\ 0.05 \end{bmatrix}$$

Live demo: the frames module

Frames and monogram notation CS 549 · session 2.2 · drag the tray frame T or the mug M; turn the tray with the slider. Units: metres, degrees.



<modules/frames-monogram/index.html>

- Default state **is** the worked example
- **Predict:** tray turned to 90° . Which of the three mug numbers change, and to what?
- Reset, uncheck *attached*, turn the tray to 90° again: the mug stays fixed in W , so ${}^T\mathbf{p}^M$ changes (example (c))
- Drag the mug: ${}^T\mathbf{p}^M$ is recomputed by the inverse formula
- Quiz mode hides the tray→mug numbers

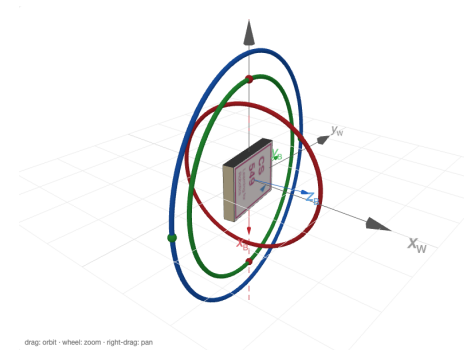
Composition: match frames, then check the order

$${}^A\mathbf{R}^C = {}^A\mathbf{R}^B {}^B\mathbf{R}^C$$

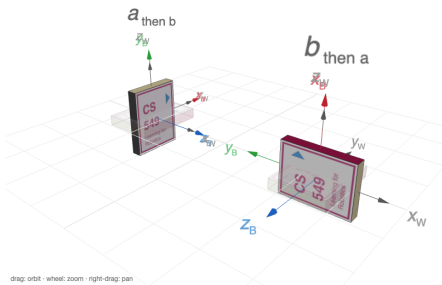
Inner frame letters match.

Different-axis rotations need not commute:

$$\mathbf{R}_z(90^\circ)\mathbf{R}_x(90^\circ) \neq \mathbf{R}_x(90^\circ)\mathbf{R}_z(90^\circ)$$



Euler angles: three ordered turns. At pitch 90° , two angle changes have the same effect: **gimbal lock**. The body still has three rotational freedoms.



Same turns, opposite order.

One orientation, several encodings

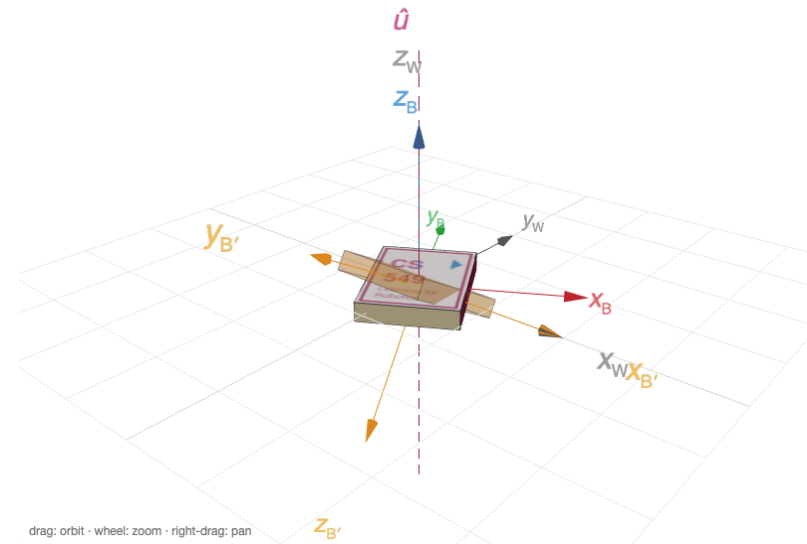
Any rotation is one angle θ about one unit axis $\hat{\mathbf{u}}$. Rodrigues' formula turns the pair into $\mathbf{R}$ (notes §4.6)

Unit quaternion: four numbers built from the same pair

$$r = (w, x, y, z) = \left(\cos \frac{\theta}{2}, \sin \frac{\theta}{2} \hat{\mathbf{u}} \right)$$

The tray's 30° about z :
(0.9659, 0, 0, 0.2588)

The order differs by library. RAI stores `w` first; SciPy and ROS put `w` last. No library can tell which order you meant.



Wrong order. The book is 30° about z . Reading the same numbers as (x, y, z, w) produces the orange frame B' and faint book: 150° about x , almost upside down. No error is raised.

A scene file lists each frame and its parent

```
tray (world): { X: "t(0.40 0.20 0.75) d(30 0 0 1)" }  
mug (tray):   { Q: "t(0.15 0.05 0.05) d(-30 0 0 1)" }
```

- `name (parent)`: a frame, placed relative to its parent
- `X`: pose in the world; `Q`: pose relative to the parent
- `t(x y z)` position
- `d(angle ax ay az)` turn by `angle` degrees about the axis; `0 0 1` is z

The mug's `Q` is ${}^T\mathbf{X}^M$: the position of example (a), turned back 30° so the mug's axes equal the world's (notes §5)

Check in RAI (`C` is the loaded file):

```
C = ry.Config(); C.addFile('tray_scene.g')  
C.getFrame('mug').getPosition()
```

should return ${}^W\mathbf{p}^M$ of example (a),
(0.5049, 0.3183, 0.80)

Failure modes of the week

- **Superscripts read backwards:** ${}^T\mathbf{p}^M$ is the mug relative to the tray
- **Subscript dropped:** about 8.2 cm off, no error message
- **Conversion direction:** B to A uses ${}^A\mathbf{R}^B$; back uses its transpose
- **Quaternion order:** RAI (w, x, y, z) ; SciPy, ROS (x, y, z, w)

The next deck, and A1

- **Deck 03 (2.3–2.4):** the pose as a 4×4 matrix; a two-link arm's kinematics and Jacobian; a 7-joint arm live. **A1 is released.**
- **Before then:** run `frames_example.py` ; change the tray angle; predict the mug first
- **Reading:** essentials §§1–5 before Deck 03 (§6 optional, as a preview); §§6–7 and §9 after Deck 03, before Deck 04 (§8 and the appendices are optional reference); §10 is your self-check
- **Before R 3:** the TAs' RAI videos; `hello_rai.py` ; **quiz 1** on weeks 1–2