Imperial College London

SMART ROBOTICS LAB

25/01/2021

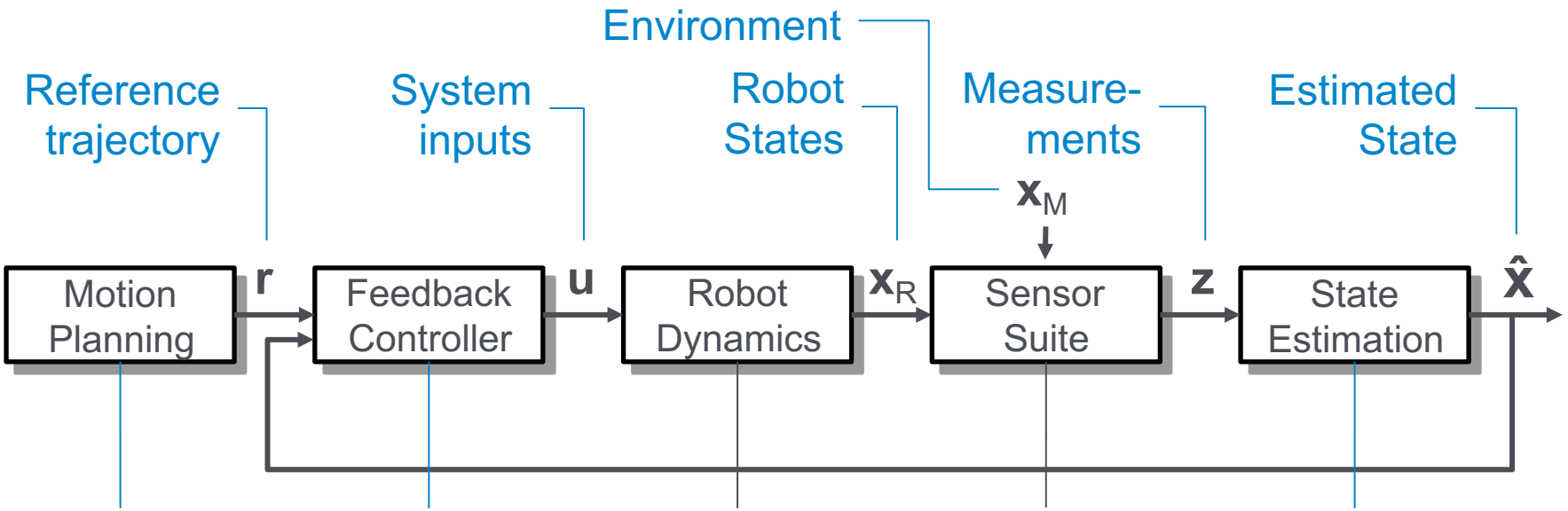
Lecture 2: Representations and Sensors

70003 Advanced Robotics

Dr Stefan Leutenegger

Teaching Assistants: Dr Masha Popovic,
Sotiris Papatheodorou, Binbin Xu, and Nils Funk

A More Realistic Setting



What trajectory that is safe do we want the robot to follow?

What commands should we send to the robot so it follows the reference trajectory?

How does the robot react to inputs given initial states?

How are measure-ments are generated given the robot states?

What is the best explan-ation for the robot states given the measure-ments?

To Be Estimated: Robot States and Environment

Robot state

- Typical components:
- Position
- Orientation
- Velocity
- Sensor-internal states
- …

Characteristics:

- Typically *time-varying*

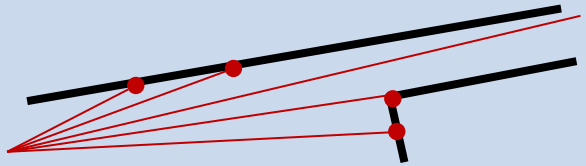
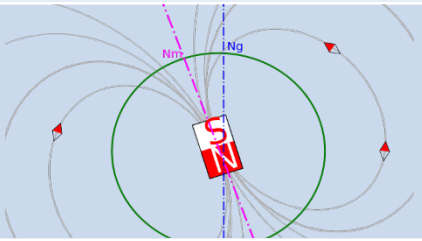
Map – the environment

- Popular representations:
- 3D points (sparse)
- 3D pointcloud/surfels (dense)
- Occupancy grid / voxels
- Triangulated mesh
- …

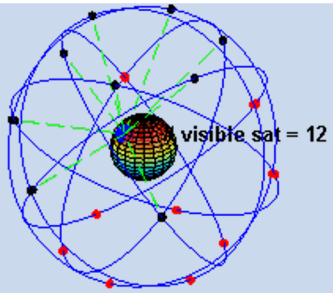
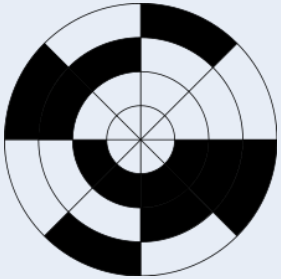
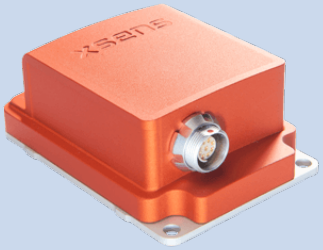
Characteristics:

- Typically, a large part is assumed to be *static*

Typical Sensors – Exteroceptive

Sensor	Measurement	
Laser Scanner	3D points	
Camera	(Colour) image (RGB-D: with depth!)	
Magnetometer	3D magnetic field	
Pressure sensor	Air pressure (altitude / airspeed)	

Typical Sensors – Proprioceptive

Sensor	Measurement	
GPS	pseudo-ranges (position)	
Encoders	Joint / wheel angles	
Inertial Measurement Unit (IMU)	Rotation rates and accelerations (with caution: orientation)	

Probabilistic Estimation

We use Bayesian Statistics:

Measurements $\mathbf{z}$ are modeled as samples from a **distribution** $p(\mathbf{z}|\mathbf{x})$ given the variables $\mathbf{x}$ (here: robot states plus possibly the map).

- **Maximum Likelihood (ML) Estimation:**
  What values for variables best explain all the measurements?

- **Maximum a Posteriori (MAP) Estimation:**
  What values for variables best explain all the measurements and a prior belief $p(\mathbf{x})$ about those variables?

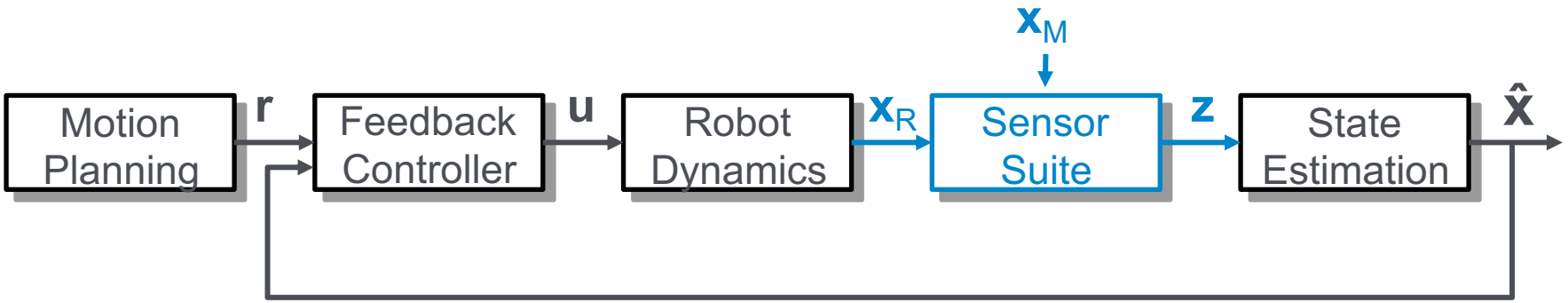
How to compute these values? To be discussed in later lectures.

Schedule: Lectures

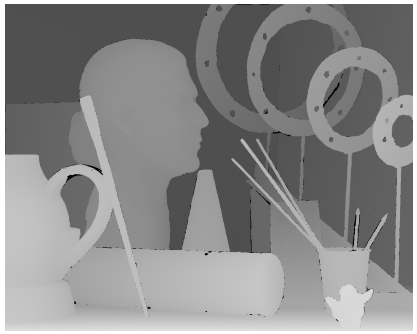
What	Date	Topic
Lecture 1	18/01/21	Introduction, Problem Formulation, and Examples
Lecture 2	**25/01/21**	**Representations and Sensors**
Lecture 3	01/02/21	Kinematics and Temporal Models
Lecture 4	08/02/21	The Extended Kalman Filter
Lecture 5	15/02/21	Feedback Control
Lecture 6	22/02/21	Nonlinear Least Squares
Lecture 7	01/03/21	Vision-Based Simultaneous Localisation and Mapping
Lecture 8	08/03/21	Revision, Q&A

Today

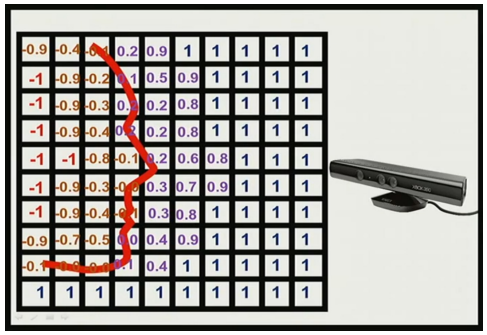
1. Representations of map and robot state
2. Representing uncertainty: basics of probability theory
3. Sensors: IMU and Camera



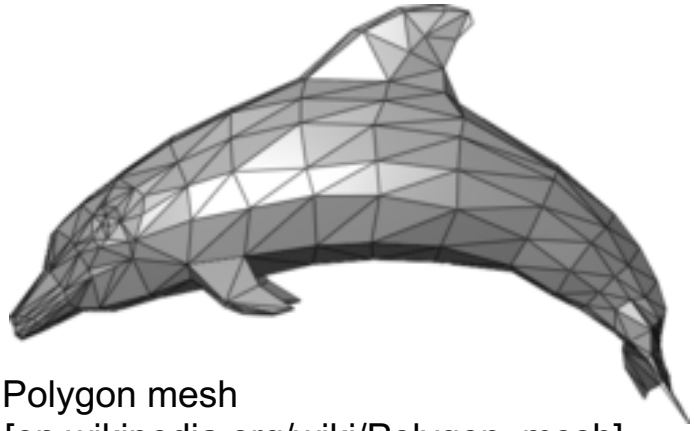
Map Representations



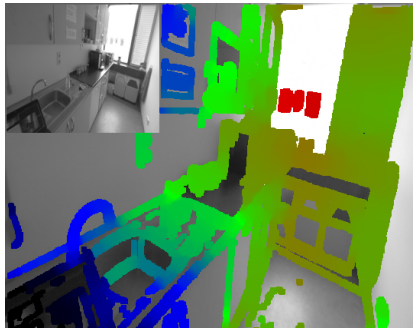
Depth maps
[vision.middlebury.edu]



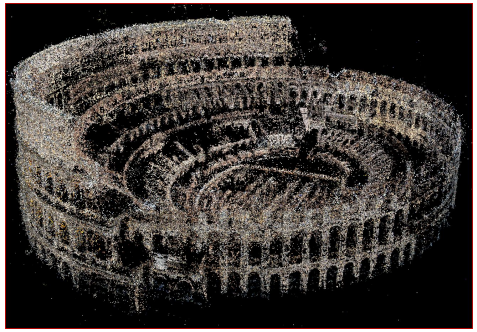
Truncated Signed Distance Function [pointclouds.org]



Polygon mesh
[en.wikipedia.org/wiki/Polygon_mesh]



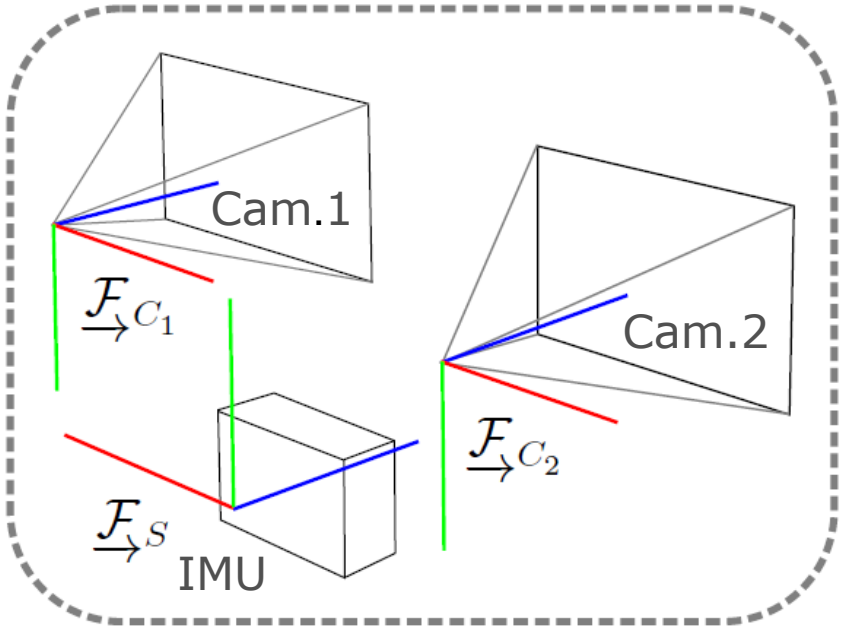
Semi-dense depth maps
[vision.in.tum.de]



Point clouds (here: sparse)
[grail.cs.washington.edu]

Surfel maps
[wp.doc.ic.ac.uk/thefutureofslam]

Frames, Notation, Definitions

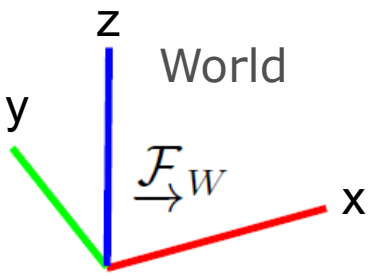


Position vector ${}_W\mathbf{r}_S$
Origin of $\underrightarrow{\mathcal{F}}_S$ expressed in $\underrightarrow{\mathcal{F}}_W$.

Rotation Matrix $\mathbf{C}_{WS}$
Transforms vector coordinate representation as ${}_W\mathbf{a} = \mathbf{C}_{WS}\,{}_S\mathbf{a}$.

Velocity vector ${}_W\mathbf{v} = {}_W\mathbf{v}_{WS}$
Velocity of origin of $\underrightarrow{\mathcal{F}}_S$ w.r.t. $\underrightarrow{\mathcal{F}}_W$

A typical example sensor setup



Robot State

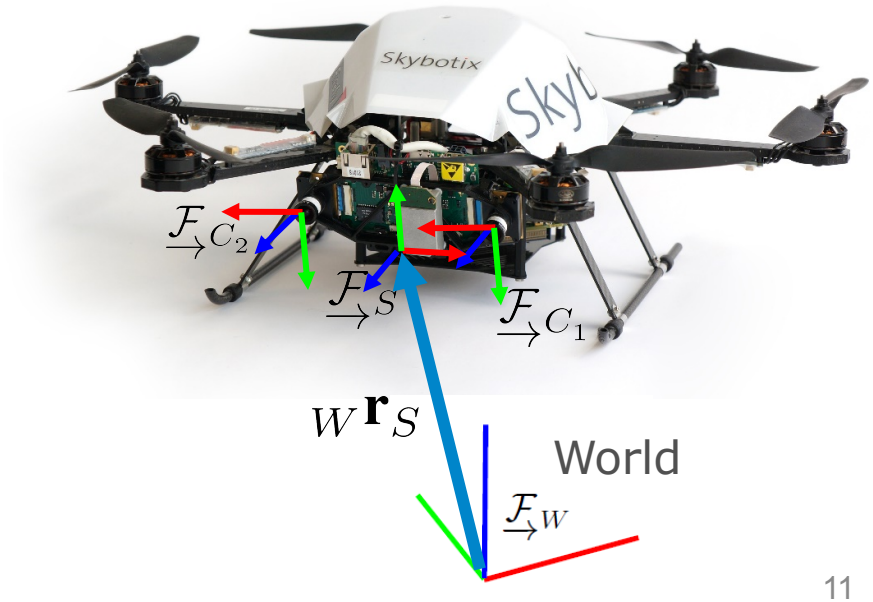
An example state vector:

$$\mathbf{x}_{\mathrm{R}}^{T} = \left[{}_{W}\mathbf{r}_{S}^{T}, \mathbf{q}_{WS}^{T}, {}_{W}\mathbf{v}^{T}, {}_{S}\mathbf{r}_{C_1}^{T}, \mathbf{q}_{SC_1}^{T}, {}_{S}\mathbf{r}_{C_2}^{T}, \mathbf{q}_{SC_2}^{T}, \mathbf{b}_{g}^{T}, \mathbf{b}_{a}^{T} \right]$$

Position and orientation of frame S origin w.r.t. in W	Velocity of frame S origin expressed in W	Poses of cameras 1 and 2 relative to frame S	IMU gyro and accelerometer biases
Physical robot states		**Sensor extrinsics (here: camera)**	**Sensor intrinsics (here: IMU)**

Notes:

- This is just an example.
- You decide, which quantities should be estimated.
- You decide, how those quantities are expressed / parameterised.



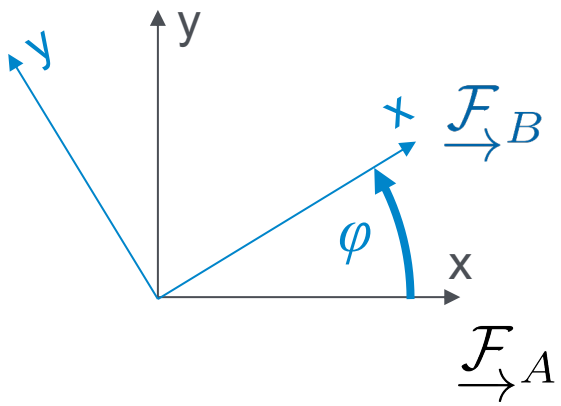
How Should We Represent Orientation?

2D case:

Seems simple:

1 angle, φ, fully describes the relationship between the two frames…

$$\mathbf{C}_{AB} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}.$$



Note: account for wrap-around...

3D case:

???

We have **3** Degrees of Freedom (DoF) to express the orientation between two frames.

How do we best parameterise them?

https://desandro.github.io/3dtransforms/docs/cube.html

Representing Orientation (SO3): Rotation Matrix

Orientation in 3D Euclidean space (SO3) has 3 Degrees of Freedom (DoF).

You can express it using a rotation matrix, e.g. $\mathbf{C}_{WS}$:

- The rotation matrix changes the coordinate representation of a vector as
  $${}_W\mathbf{a} = \mathbf{C}_{WS}\ {}_S\mathbf{a}.$$
- The rotation matrix is orthonormal. Consequently:
  $$\mathbf{C}_{SW} = \mathbf{C}_{WS}^{-1} = \mathbf{C}_{WS}^{T},$$
  $$\det\mathbf{C}_{SW} = 1.$$
- The columns of $\mathbf{C}_{WS}$ are the basis vectors of $\underrightarrow{\mathcal{F}}_S$ expressed in $\underrightarrow{\mathcal{F}}_W$. (This is handy to know for visualisations.)

Advantages of this parameterisation:

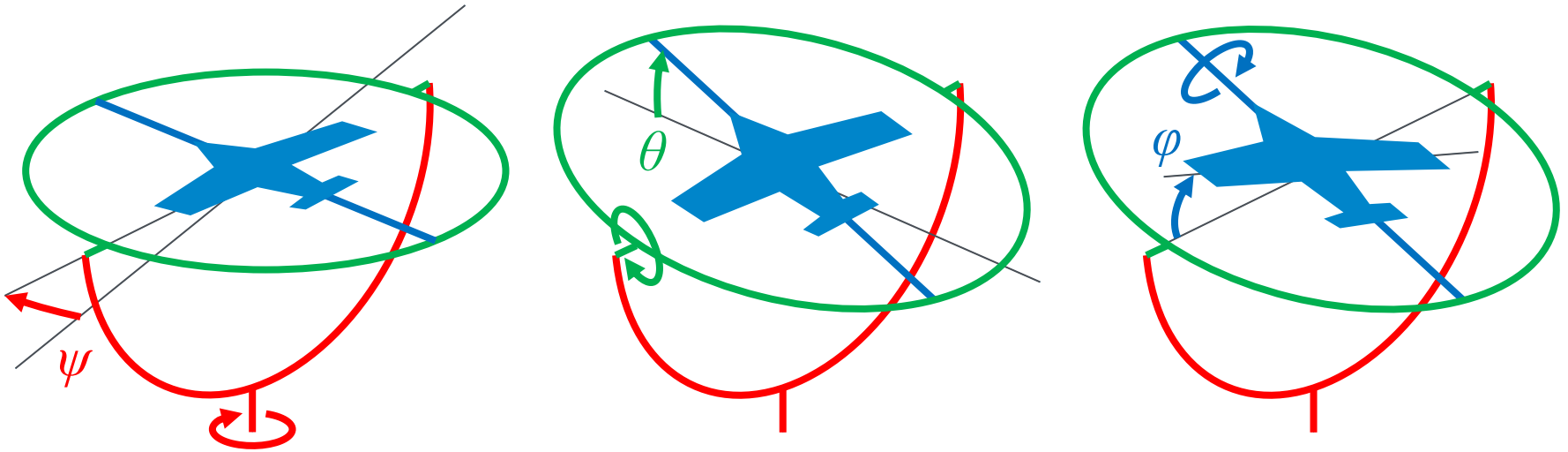
- No singularities

Drawbacks of this parameterisation:

- 9 parameters to represent 3 DoF (overkill).
- Must enforce the above orthonormality constraints.

Representing Orientation (SO3): Euler Angles (Tait-Brian)

Rotation Matrix (e.g. $\mathbf{C}_{EB}$) is parametrized with **3 successive rotations** using the **zyx Tait-Brian Angles** (specific kind of Euler Angles):



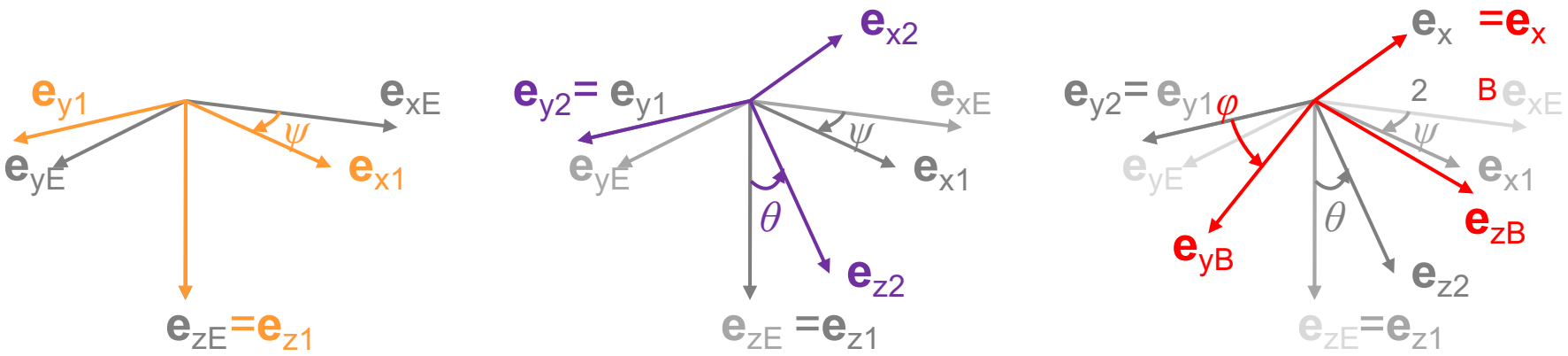
1 **Yaw:** ψ

2 **Pitch:** θ

3 **Roll:** φ

Representing Orientation (SO3): Euler Angles (Tait-Brian)

Rotation Matrix (e.g. $\mathbf{C}_{EB}$) is parametrized with **3 successive rotations** using the **zyx Tait-Brian Angles** (specific kind of Euler Angles):



1 **Yaw:**
ψ around $\mathbf{e}_{zE}$: $\mathbf{C}_{E1}(\psi)$
⇒ Frame 1

2 **Pitch:**
θ around $\mathbf{e}_{y1}$: $\mathbf{C}_{12}(\theta)$
⇒ Frame 2

3 **Roll:**
φ around $\mathbf{e}_{x2}$: $\mathbf{C}_{2B}(\varphi)$
⇒ Frame B

$$\mathbf{C}_{EB} = \mathbf{C}_{E1}(\psi) \cdot \mathbf{C}_{12}(\theta) \cdot \mathbf{C}_{2B}(\varphi)$$

(post-multiply for rotations around new axes…)

Euler Angles (Tait-Brian) Continued

The rotation matrix calculated (s(.) for sin(.), c(.) for cos(.)):

$$\mathbf{C}_{E1}(\psi)=\begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{C}_{12}(\theta)=\begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \quad \mathbf{C}_{2B}(\varphi)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & s(\varphi) & c(\varphi) \end{bmatrix}$$

$$\mathbf{C}_{EB}=\mathbf{C}_{E1}\mathbf{C}_{12}\mathbf{C}_{2B}=\begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\varphi)-s(\psi)c(\varphi) & c(\psi)s(\theta)c(\varphi)+s(\psi)s(\varphi) \\ s(\psi)c(\theta) & s(\psi)s(\theta)s(\varphi)+c(\psi)c(\varphi) & s(\psi)s(\theta)c(\varphi)-c(\psi)s(\varphi) \\ -s(\theta) & c(\theta)s(\varphi) & c(\theta)c(\varphi) \end{bmatrix}$$

Be careful with the boundaries: **Roll** ($-\pi<\varphi<\pi$) **Pitch** ($-\pi/2<\theta<\pi/2$) **Yaw** ($-\pi<\psi<\pi$)

Computing angles from rotation matrix (not the only way, and not most stable):

$$\begin{aligned} \psi &= \arcsin\left(C_{21}/\sqrt{1-C_{31}^2}\right) \\ \theta &= \arcsin\left(-C_{31}\right) \\ \varphi &= \arcsin\left(C_{32}/\sqrt{1-C_{31}^2}\right) \end{aligned}, \text{ with } \mathbf{C}_{EB}=\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

Euler Angles (Tait-Brian): Gimbal Lock

What happens at pitch angle $\theta = \pm\pi/2$?

- Yaw and roll axes align…
- 1 DoF is thus lost at this point
- It is a singularity!
- **Note:** we can still describe this orientation (just not uniquely – set yaw or roll to zero)

Advantages of Tait-Bryan parameterisation:

- Minimal representation, thus no constraints to enforce.
- Reasonably easy to interpret, e.g. in a plot against time.

Drawbacks of this parameterisation:

- **"Gimbal Lock singularity"**
  (all Euler angle representations will have a singularity)

A Little Anecdote on Gimbal Lock in Practice…

04 08 59 35		CapCom	Columbia, Houston. We noticed you are maneuvering very close to gimbal lock. I suggest you move back away. Over.
04 08 59 43		Michael Collins (CMP)	Yes. I am going around it, doing this CMC AUTO maneuvers to the PAD values of roll 270, pitch 101, yaw 45.
04 08 59 52		CapCom	Roger, Columbia.
04 09 00 30		Michael Collins (CMP)	How about sending me a fourth gimbal for Christmas.
04 09 00 40		CapCom	Columbia, Houston. You were unreadable. Say again please.
04 09 00 46		Michael Collins (CMP)	Disregard.

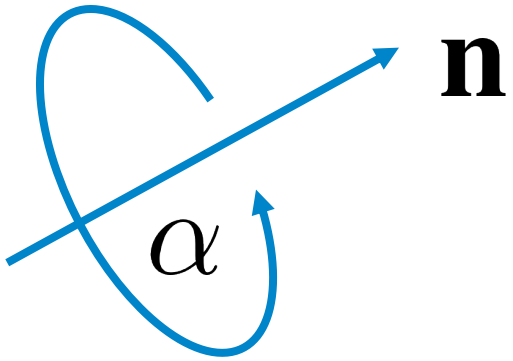
From http://apollo11.spacelog.org/page/04:08:40:46/

Axis-Angle / Rotation Vector

Think of orientation as rotation around a normal $\mathbf{n}$ by an angle α:

Axis-Angle: $[\mathbf{n}^T, \alpha]^T$
(4 parameters, normalise normal)

Rotation Vector: $\boldsymbol{\alpha} = \alpha\mathbf{n}$
(3 parameters, i.e. minimal)



Advantages of Axis-Angle / Rotation Vector parameterisation:

- Rotation Vector: Minimal representation (Axis-Angle: compact)
- Reasonably intuitive concept

Drawbacks of this parameterisation:

- Composition of rotations (rotation sequences) are complex to evaluate
- Wrap-around for angles $\alpha = 2n\pi$
- Axis-Angle: undefined axis at $\alpha = 0$

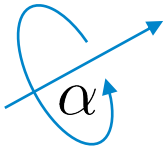
Quaternions of Orientation [9]

Hamiltonian Quaternion:

A general quaternion is a complex number (real part and a 3D imaginary part):
$q = q = q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$, with $\mathbf{i}^2=\mathbf{j}^2=\mathbf{k}^2=-1$, $\mathbf{ij}=-\mathbf{ji}=\mathbf{k}$, $\mathbf{jk}=-\mathbf{kj}=\mathbf{i}$, $\mathbf{ki}=-\mathbf{ik}=\mathbf{j}$.

Unit Quaternion (Q. of Orientation):

Again: think of orientation as rotation around a normal $\mathbf{n}$ by an angle α :



$$\mathbf{v} = \sin\frac{\alpha}{2}\mathbf{n}, \quad a = \cos\frac{\alpha}{2}$$

$$\mathbf{q} = [q_x, q_y, q_z, q_w]^T = [\mathbf{v}(\mathbf{q})^T, a(\mathbf{q})]^T$$

I.e. we use 4 parameters for 3 DoF.

Unit length constraint: $\|\mathbf{q}\| = 1.$

Neutral element: $\boldsymbol{\iota} = [0, 0, 0, 1]^T.$

Inverse: $\mathbf{q}^{-1} = [-\mathbf{v}(\mathbf{q})^T, a(\mathbf{q})]^T.$

[9] kindr cheatsheet, available at https://github.com/ethz-asl/kindr/blob/master/doc/cheatsheet/cheatsheet_latest.pdf, March 2015.

Quaternion Operations

Prerequisite: Cross-product matrix $[\mathbf{a}]^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$.

Multiplication:

Multiply quaternions: $\mathbf{q} \otimes \mathbf{p} = \begin{bmatrix} a(\mathbf{q})\mathbf{v}(\mathbf{p}) + a(\mathbf{p})\mathbf{v}(\mathbf{q}) + \mathbf{v}(\mathbf{q}) \times \mathbf{v}(\mathbf{p})) \\ a(\mathbf{q})a(\mathbf{p}) - \mathbf{v}(\mathbf{q})^T\mathbf{v}(\mathbf{p}) \end{bmatrix}$.
$(\mathbf{q}_{AC} = \mathbf{q}_{AB} \otimes \mathbf{q}_{BC})$

… as matrix multiplication: $\mathbf{q}_{AB} \otimes \mathbf{q}_{BC} = [\mathbf{q}_{AB}]^+ \mathbf{q}_{BC} = [\mathbf{q}_{BC}]^\oplus \mathbf{q}_{AB}$.

With $[\mathbf{q}_{AB}]^+ = \begin{bmatrix} a\mathbf{1} + [\mathbf{v}]^\times & \mathbf{v} \\ -\mathbf{v}^T & a \end{bmatrix}$ and $[\mathbf{q}_{AB}]^\oplus = \begin{bmatrix} a\mathbf{1} - [\mathbf{v}]^\times & \mathbf{v} \\ -\mathbf{v}^T & a \end{bmatrix}$.

Change coordinate representation of a vector: $\begin{bmatrix} {}_A\mathbf{a} \\ 0 \end{bmatrix} = \mathbf{q}_{AB} \otimes \begin{bmatrix} {}_B\mathbf{a} \\ 0 \end{bmatrix} \otimes \mathbf{q}_{AB}^{-1}$.

Conversion to rotation matrix:

Often, we work with the rotation matrix corresponding to a quaternion:

$$\mathbf{C}_{AB} = \mathbf{1}_3 + 2a(\mathbf{q}_{AB})[\mathbf{v}(\mathbf{q}_{AB})]^\times + 2\left([\mathbf{v}(\mathbf{q}_{AB})]^\times\right)^2.$$

Quaternions: Conversions and Efficiency

Chaining efficiency [10]:

Method	Multiply	add/subtract	total
Rotation matrices	27	18	45
Quaternions	16	12	28
Axis-Angle (Rot. Vec)	(convert to Quaternion first)		

Vector rotation efficiency [10]:

Method	multiply	add/subtract	sin/cos	total
Rotation matrix	9	6	0	15
Quaternions *	21 (= 12 + 9)	18 (= 12 + 6)	0	39
Angle/axis	18	12	2	30

Conversion quaternion to Rotation Matrix [10,11]: 12 mult. plus 12 add...

Therefore, when you need to rotate vectors, compute and cache the matrix...

[10] https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation
[11] https://www.geometrictools.com/Documentation/RotationIssues.pdf

Quaternions: Minimal Local Parameterisation

A quaternion can be interpreted as defined in a 3-dimensional manifold.
A manifold is a topological space that locally resembles Euclidean space near each point [12]. We can express e.g. uncertainty in this Euclidean space (tangent space).

Minimal local parameterisation:

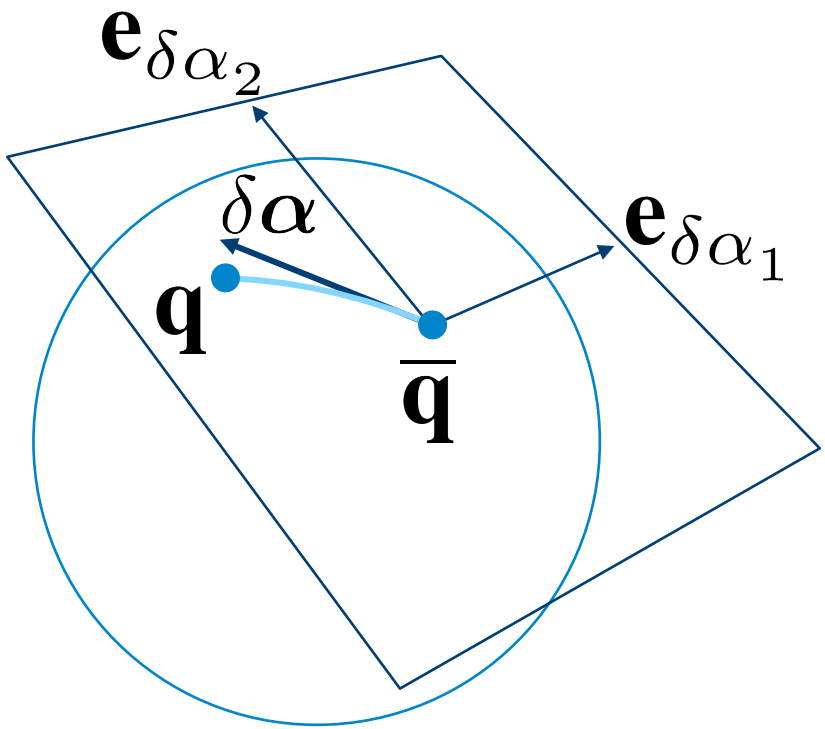
$\delta\boldsymbol{\alpha} \in \mathbb{R}^3$: (small) rotation vector

Can be converted into quaternion:

$$\delta\mathbf{q} := \exp\left(\delta\boldsymbol{\alpha}\right) = \begin{bmatrix} \operatorname{sinc}\left\|\frac{\delta\boldsymbol{\alpha}}{2}\right\| \frac{\delta\boldsymbol{\alpha}}{2} \\ \cos\left\|\frac{\delta\boldsymbol{\alpha}}{2}\right\| \end{bmatrix}.$$

Now we write:

$$\mathbf{q}_{WS} = \delta\mathbf{q} \otimes \overline{\mathbf{q}}_{WS}\,.$$



[12] https://en.wikipedia.org/wiki/Manifold

Quaternions: Pros and Cons

Advantages of this parameterisation:

- Reasonably compact representation.
- No singularity.
- Operations are numerically stable and efficient.

Drawbacks of this parameterisation:

- Unit length constraint needs to be enforced.
- Using local parameterisation and respective derivatives takes getting used to.

Some cool visualisations / animations of quaternions: https://eater.net/quaternions

SE3 and Homogenous Transformation Matrices

A 6D pose consists of position and orientation.
We can write the respective change of reference frame of a position vector $\mathbf{r}_P$ as:

$$ {}_A\mathbf{r}_P = \mathbf{C}_{AB}\, {}_B\mathbf{r}_P + {}_A\mathbf{r}_B. $$

Or, we can use the homogeneous transformation matrix:

$$ \boldsymbol{T}_{AB} = \begin{bmatrix} \mathbf{C}_{AB} & {}_A\mathbf{r}_B \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. $$

Now: $ {}_A\boldsymbol{r}_P := \begin{bmatrix} {}_A\mathbf{r}_P \\ 1 \end{bmatrix} = \boldsymbol{T}_{AB} \begin{bmatrix} {}_B\mathbf{r}_P \\ 1 \end{bmatrix}. $

Notes

As inverse transform, we obtain $ \boldsymbol{T}_{BA} = \boldsymbol{T}_{AB}^{-1} = \begin{bmatrix} \mathbf{C}_{AB}^T & -\mathbf{C}_{AB}^T\, {}_A\mathbf{r}_B \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. $

We can stack transformations as $ \boldsymbol{T}_{AC} = \boldsymbol{T}_{AB}\, \boldsymbol{T}_{BC}. $

Working with Discrete Random Variables

Example: two random variables, X and Y that can assume discrete values.

Joint probability: P(X=x,Y=y), short P(X,Y), “the probability of X and Y”
Conditional probability: P(X=x|Y=y), short P(X|Y), “the probability of X given Y”

Sum rule: $$P(X) = \sum_Y P(X, Y).$$

Product rule: $$P(X, Y) = P(X|Y)P(Y).$$

With these two rules and symmetry P(X,Y) = P(X|Y) P(Y) = P(Y|X) P(X) we get

Bayes’ Theorem: $$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}.$$

Working with Continuous Random Variables

Example: two random variables, x and y that can assume real values.

Joint probability density: p(x,y).
Conditional probability density: p(x|y).

Meaning: $$p(x \in (a,b), y \in (c,d)) = \int_a^b \int_c^d p(x,y)dydx.$$

"Sum" rule: $$p(x) = \int_{-\infty}^{+\infty} p(x,y)dy.$$

Product rule: $$p(x,y) = p(x|y)p(y).$$

With these two rules and symmetry p(x,y) = p(x|y) P(y) = p(y|x) p(x) we get

Bayes' Theorem: $$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

Independence and Conditional independence

Independence:

Two random variables x and y are called independent, iff

$$p(x, y) = p(x)p(y).$$

Conditional Independence:

Two variables x and y can become independent, if a third one (z) is given:

$$p(x, y|z) = p(x|z)p(y|z).$$

Note: conditional and absolute independence are not the same thing!

$$p(x, y|z) = p(x|z)p(y|z) \implies p(x, y) = p(x)p(y).$$
$$p(x, y) = p(x)p(y) \implies p(x, y|z) = p(x|z)p(y|z).$$

Expectation and Covariance

Expectation:

Average value of some function **f**: $E[\mathbf{f}(\mathbf{x})] = \int_{-\infty}^{+\infty} \mathbf{f}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}.$

Average value of **x**: $E[\mathbf{x}] = \int_{-\infty}^{+\infty} \mathbf{x} p(\mathbf{x}) d\mathbf{x}.$

Covariance:

$$\mathrm{Cov}[\mathbf{x}] = E[(\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}])^T] = E[\mathbf{x}\mathbf{x}^T] - E[\mathbf{x}]E[\mathbf{x}]^T.$$

The Multivariate Gaussian Distribution

We write $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

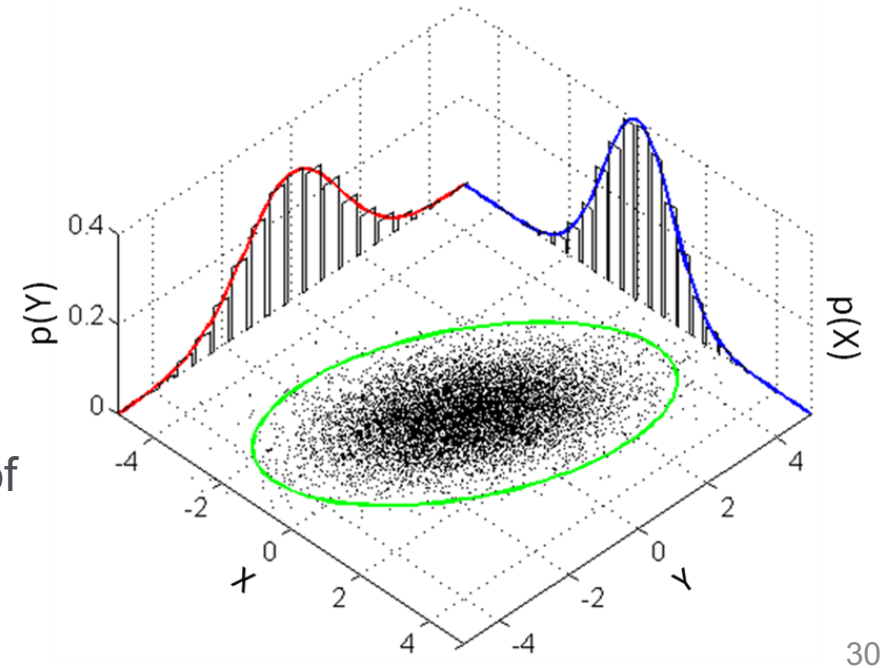
Probability density function:

$$p(\mathbf{x}) = p(x_1, \ldots, x_k)$$

$$= \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right).$$

$\boldsymbol{\mu}$: Mean

$\boldsymbol{\Sigma}$: Covariance matrix



Right: Samples from Gaussian distribution of two variables X and Y.

Measurement Models: IMU and in General

A MEMS IMU measures **rotation rates** ${}_S\tilde{\boldsymbol{\omega}}$ and **acceleration** ${}_S\tilde{\mathbf{a}}$.

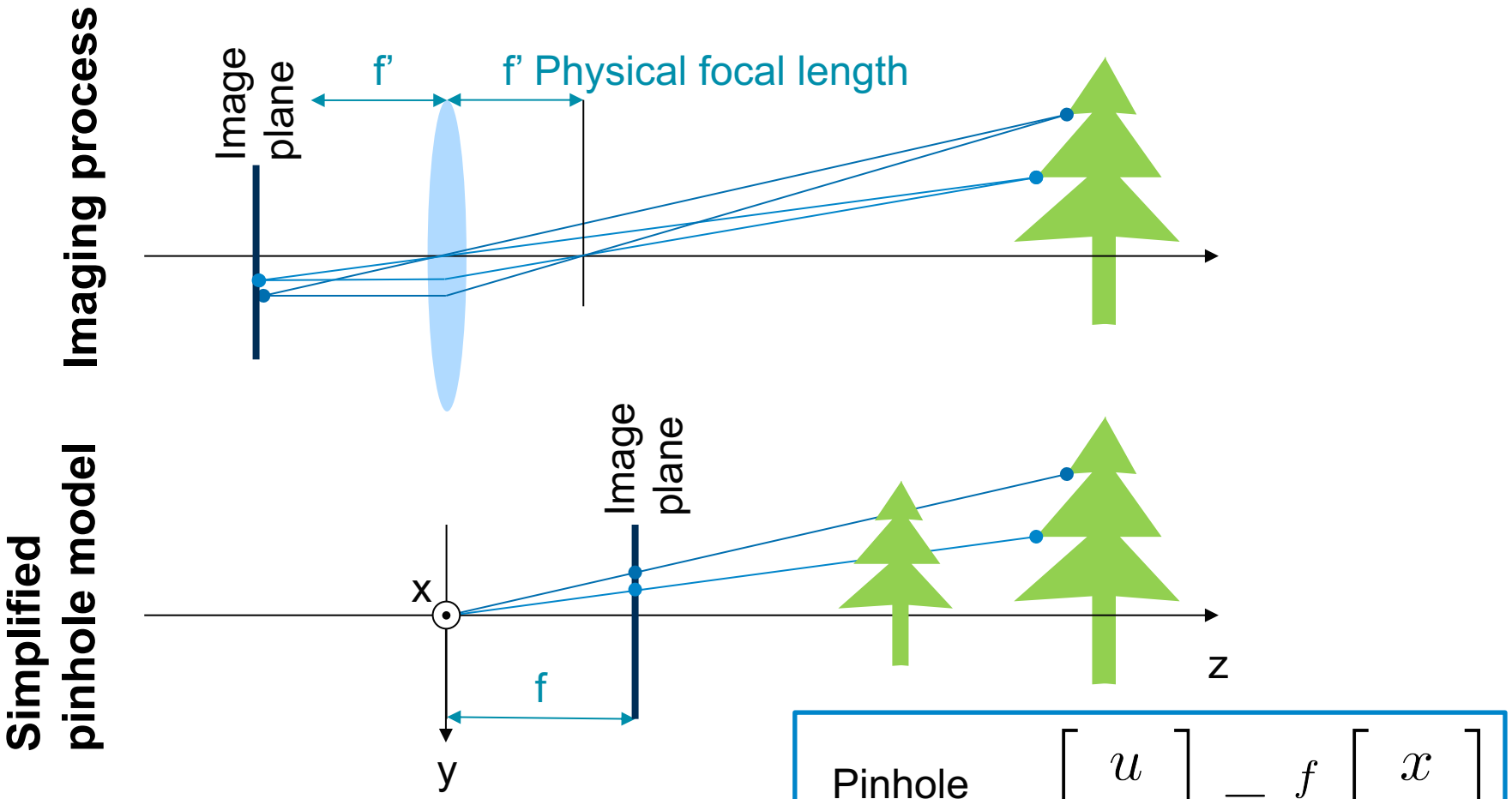
We model a (discrete-time) measurement as:

$$\tilde{\mathbf{z}} = \underbrace{\mathbf{b}_C}_{\text{Constant, calibrate.}} + \underbrace{s\mathbf{M}\mathbf{z}}_{\text{Often constant, calibrate.}} + \underbrace{\mathbf{b}}_{\text{Model \& estimate!}} + \underbrace{\mathbf{n}}_{\text{Model!}} + \mathbf{o}.$$

- $\mathbf{z}$: Correct measurement
- $\mathbf{b}_C$: Long-term constant bias
- s : Scaling
- $\mathbf{M}$: Misalignment
- $\mathbf{b}$: Time-varying bias (often also a function of temperature)
- $\mathbf{n}$: Noise, typically modelled as $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$
- $\mathbf{o}$: Other (un-modelled) influences

Note: We will model many sensors in a similar way…

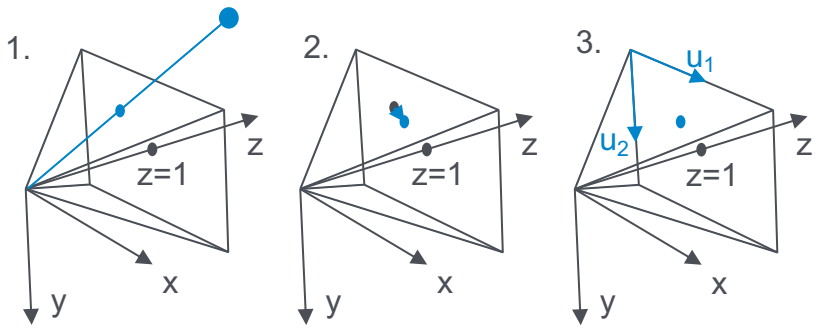
The Camera



Pinhole projection: $\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{z} \begin{bmatrix} x \\ y \end{bmatrix}$

Pinhole Camera Projection with Distortion[1] (World2Cam)

Complete Model: $\mathbf{u} = \mathbf{k}(\mathbf{d}(\mathbf{p}({}_C\mathbf{r}_P)))$



1. Project point to unit plane:

$$\mathbf{x}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \mathbf{p}\left(\begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \right) = \frac{1}{r_3} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}.$$

2. Apply distortion model – e.g. Radial-Tangential (with $r^2 = x_1'^2 + x_2'^2$):

$$\mathbf{x}'' = \mathbf{d}(\mathbf{x}') = \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} + \begin{bmatrix} 2p_1 x_1' x_2' + p_2(r^2 + 2x_1'^2) \\ p_1(r^2 + 2x_2'^2) + 2p_2 x_1' x_2' \end{bmatrix}.$$

3. Scale and centre: $\mathbf{u} = \mathbf{k}(\mathbf{x}'') = \begin{bmatrix} f_1 & 0 \\ 0 & f_2 \end{bmatrix} \mathbf{x}'' + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$

k_1, k_2 : Radial distortion parameters.
k_3-k_6 : (Optional) radial distortion parameters.
p_1, p_2 : Tangential distortion parameters.

f_1, f_2 : x/y focal lengths in pixels.
c_1, c_2 : Image centre in pixels.

[1] Following OpenCV, http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html .

Pinhole Camera with Distortion: Un-Project (Cam2World)

Now we want to invert the model…

Note: we can only find a ray, not the actual 3D point!

1. Change to unit plane coordinates:

$$\mathbf{x}'' = \mathbf{k}^{-1}(\mathbf{u}) = \begin{bmatrix} \frac{1}{f_1} & 0 \\ 0 & \frac{1}{f_2} \end{bmatrix} \left(\mathbf{u} - \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right)$$

2. Un-distort (e.g. Radial-Tangential):

$$\mathbf{x}' = \mathbf{d}^{-1}(\mathbf{x}'').$$

Note: this generally has to be done numerically (e.g. using gradient descent or Gauss-Newton)

3. Compute the ray – simply remember we are operating in the unit plane…

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}' \\ 1 \end{bmatrix}.$$

About Un-Distorting a Whole Image

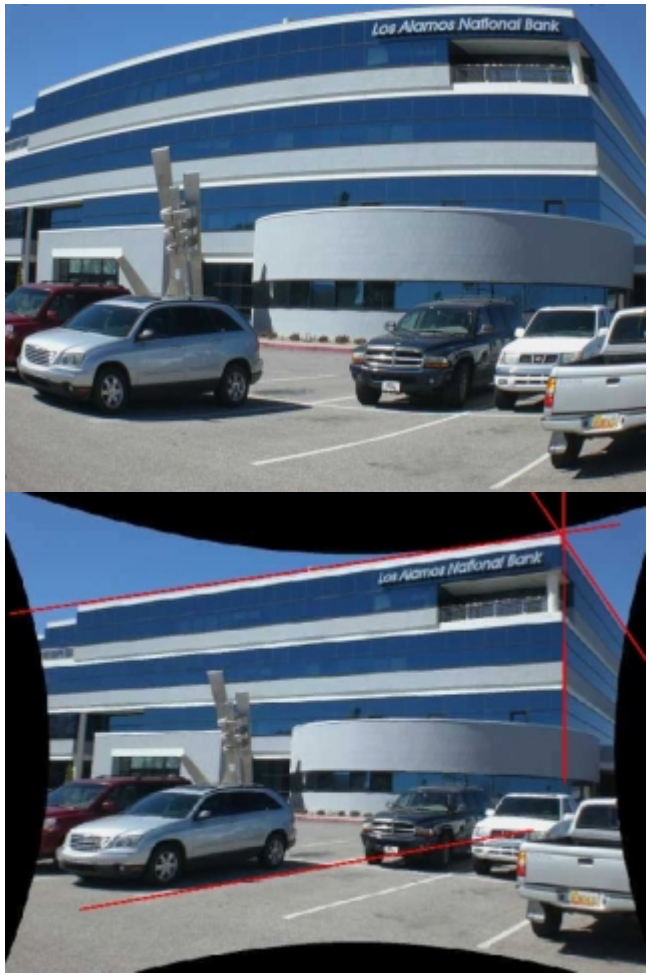


Image source: http://www.edwardrosten.com/work/ .

We can do this (maybe surprisingly) using the **distortion function** in pixel space:

1. For all pixel locations $\mathbf{u}_{i,\text{new}}$ of the desired output image, compute their distorted locations
   $$\mathbf{u}_i = \mathbf{k}(\mathbf{d}(\mathbf{k}_{\text{new}}{}^{-1}(\mathbf{u}_{i,\text{new}}))).$$
   Choose $\mathbf{k}_{\text{new}}$ with desired new focal lengths and image centre.
2. Re-sample the colours at these locations $\mathbf{u}_i$ (using some interpolation scheme between the neighbouring pixels, e.g. bi-linear).

Notes

- Resampling always loses image quality.
- Depending on choices in Step 1, you may get samples from outside the original image (set e.g. black).

Any Questions?

See you on Thursday at 13:00 for the practical!