Imperial College London

SMART ROBOTICS LAB

15/02/2021

Lecture 5:
Feedback Control

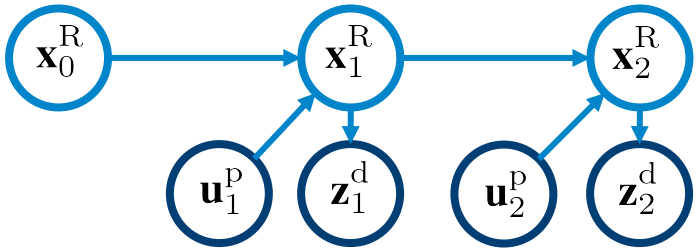
70003 Advanced Robotics

Dr Stefan Leutenegger

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

Recursive Estimation and the Bayes Filter

- So far we have looked at batch estimation, i.e. estimating every variable at once.
- Recall, however, the typical structure of a robot state estimation problem:



$\mathbf{x}_k^{\mathrm{R}}$: the robot state at time step k.
$\mathbf{z}_k^{\mathrm{d}}$: a distance measurement.
$\mathbf{u}_k^{\mathrm{p}}$: wheel odometry (measurement).

$$p(\mathbf{x}_0^{\mathrm{R}}, \ldots, \mathbf{x}_N^{\mathrm{R}}, \mathbf{z}_1^{\mathrm{d}}, \ldots, \mathbf{z}_N^{\mathrm{d}}, \mathbf{u}_1^{\mathrm{p}}, \ldots, \mathbf{u}_N^{\mathrm{p}}) = p(\mathbf{x}_0^{\mathrm{R}}) p(\mathbf{x}_1^{\mathrm{R}}|\mathbf{u}_1^{\mathrm{p}}, \mathbf{x}_0^{\mathrm{R}}) p(\mathbf{u}_1^{\mathrm{p}}) p(\mathbf{z}_1^{\mathrm{d}}|\mathbf{x}_1^{\mathrm{R}}) \ldots p(\mathbf{x}_N^{\mathrm{R}}|\mathbf{u}_N^{\mathrm{p}}, \mathbf{x}_{N-1}^{\mathrm{R}}) p(\mathbf{u}_N^{\mathrm{p}}) p(\mathbf{z}_N^{\mathrm{d}}|\mathbf{x}_N^{\mathrm{R}}).$$

Often, we are only interested in the **current state** (e.g. to control the robot) and not the whole (growing!) history of robot states (trajectory).

This can be done by alternating **prediction** (using a temporal model) and **update**:

Bayes Filter – recursive MAP estimation:

- Initialise the prior distribution $p(\mathbf{x}_0^{\mathrm{R}})$.
- At every k[th] iteration do the following two steps:

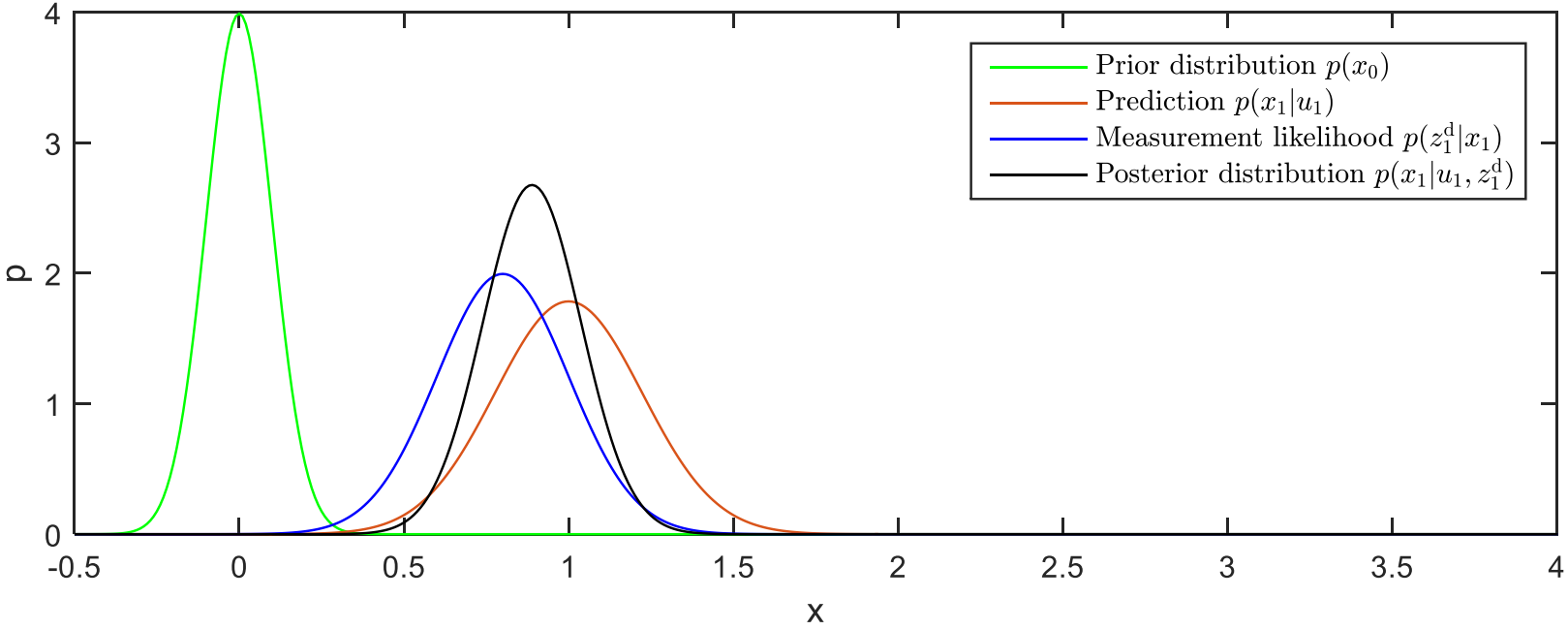
1. **Predict**: $p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_{1:k}^{\mathrm{p}}, \mathbf{z}_{1:k-1}^{\mathrm{d}}) = \int p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_k^{\mathrm{p}}, \mathbf{x}_{k-1}^{\mathrm{R}}) p(\mathbf{x}_{k-1}^{\mathrm{R}}|\mathbf{u}_{1:k-1}^{\mathrm{p}}, \mathbf{z}_{1:k-1}^{\mathrm{d}}) d\mathbf{x}_{k-1}^{\mathrm{R}}$ **Prod./sum rule**
2. **Update**: $p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_{1:k}^{\mathrm{p}}, \mathbf{z}_{1:k}^{\mathrm{d}}) = \eta p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_{1:k}^{\mathrm{p}}, \mathbf{z}_{1:k-1}^{\mathrm{d}}) p(\mathbf{z}_k^{\mathrm{d}}|\mathbf{x}_k^{\mathrm{R}})$ **Bayes' theorem**

1D Kalman Filter (KF) Example

KF: Bayes Filter assuming Gaussian distributions for all random variables as well as linear measurement and state transition models!

Example: robot moving along the x-axis…

- State transition model: $x_k = x_{k-1} + u + w, \quad w \sim \mathcal{N}(0, q)$, where u^{p} is an odometry reading.
- Measurement updates (absolute position): $z^{\mathrm{d}} = x, \quad z^{\mathrm{d}} \sim \mathcal{N}(0, r).$



Nonlinear Dynamic Systems

Typically, we will model as a **nonlinear continuous-time system** of the form:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_c(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t)).$$
$$\mathbf{z}(t) = \mathbf{h}(\mathbf{x}(t)) + \mathbf{v}(t).$$

Linearise: around $\overline{\mathbf{x}}, \overline{\mathbf{u}}$

The **linearised continuous-time system** can be handy for analysis:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{F}_c\delta\mathbf{x}(t) + \mathbf{G}_c\delta\mathbf{u}(t) + \mathbf{L}_c\mathbf{w}(t).$$
$$\delta\mathbf{z}(t) = \mathbf{H}\delta\mathbf{x}(t) + \mathbf{v}(t).$$

Discretise: integrate from t_{k-1} to t_k .

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k).$$
$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k.$$

Linearise: around $\overline{\mathbf{x}}, \overline{\mathbf{u}}$

$$\delta\mathbf{x}_k = \mathbf{F}_k\delta\mathbf{x}_{k-1} + \mathbf{G}_k\delta\mathbf{u}_k + \mathbf{L}_k\mathbf{w}_k.$$
$$\delta\mathbf{z}_k = \mathbf{H}_k\delta\mathbf{x}_k + \mathbf{v}_k.$$

The **discrete-time nonlinear system** is used in our estimator implementations

Also the **linearised discrete-time system** is needed in the implementation

Extended Kalman Filter (EKF): Nonlinear Version

EKF: Bayes Filter (up to linearisation) assuming Gaussian distributions for all random variables!

Initialisation:

Decide on starting point for states $\mathbf{x}_0$ and their prior covariance $\mathbf{P}_0$.

Prediction:

Odometry/control $\mathbf{u}_k$, Noise, with covariance $\mathbf{Q}_k$

With nonlinear state transition model $\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k)$.

Mean: $\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$

Covariance: $\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{L}_k \mathbf{Q}_k \mathbf{L}_k^\top$

$\mathbf{F}_k$: Linearisation $\partial \mathbf{f}/\partial \mathbf{x}$; $\mathbf{L}_k$: Linearisation $\partial \mathbf{f}/\partial \mathbf{w}$

Measurement update:

With measurement $\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$ — Noise, with covariance $\mathbf{R}_k$

Measurement residual: $\mathbf{y}_k = \tilde{\mathbf{z}}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1})$

Residual covariance: $\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k$

$\mathbf{H}$: Linearisation $\delta \mathbf{h}/\partial \mathbf{x}$

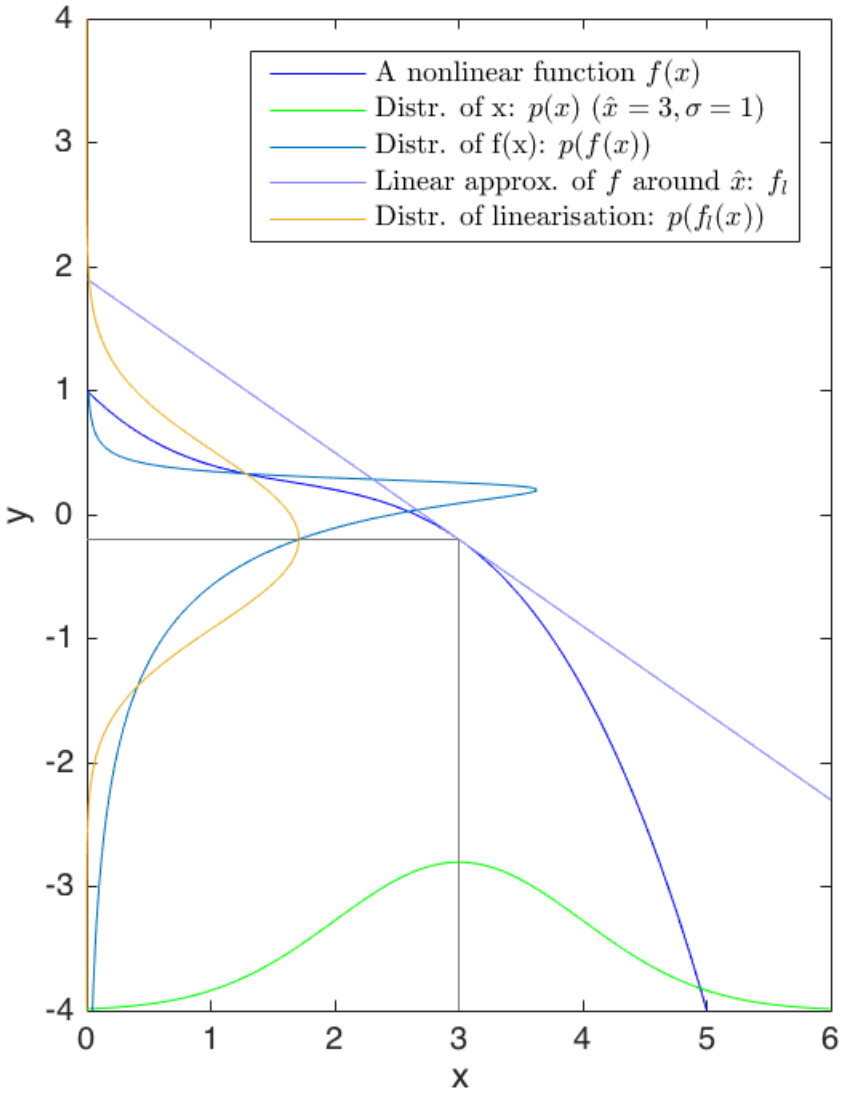
Kalman gain: $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^\top \mathbf{S}_k^{-1}$

Updated state (mean): $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k$

Updated covariance: $\mathbf{P}_{k|k} = (\mathbf{1} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$

Problem with Linearisation in EKF – 1D Example

- The linear approximation might not be a good one for the whole distribution.
- Error propagation through the linearised approximation may e.g.
  - shift peak (**biased solution**) or
  - Lead to higher peak (**overconfidence**)
- Whether or not this causes trouble depends on the problem we are trying to solve:
  - intrinsic nonlinearities in propagation and measurement
  - Noise levels (process noise and measurement noise)



Local Parameterisation of Orientation Again…

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

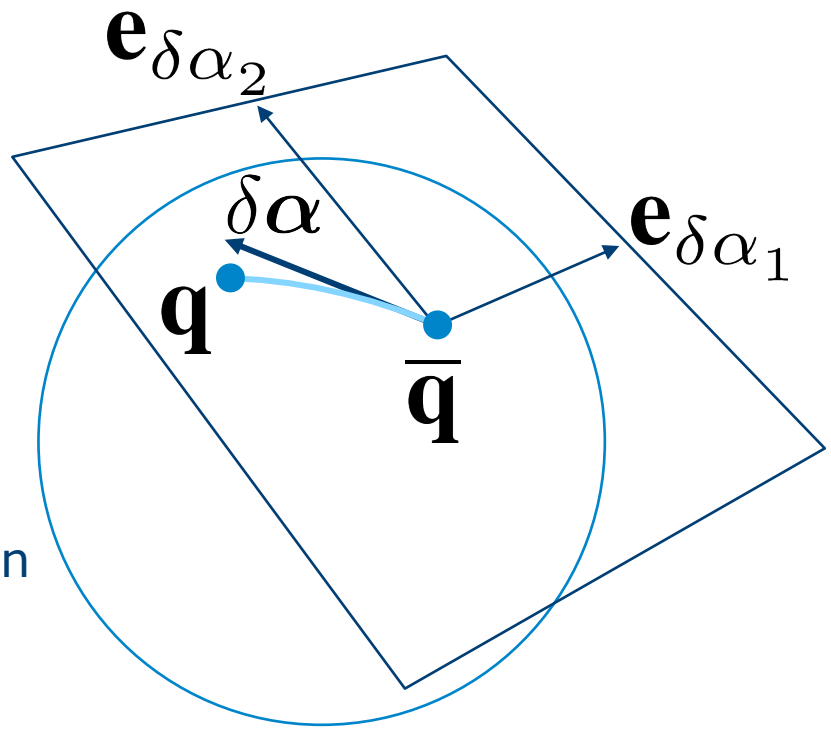
Can be converted into quaternion:

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

Now we write:

$$\mathbf{q}_{WS} = \delta\mathbf{q}(\delta\boldsymbol{\alpha}) \otimes \overline{\mathbf{q}}_{WS}.$$

Linearisation point

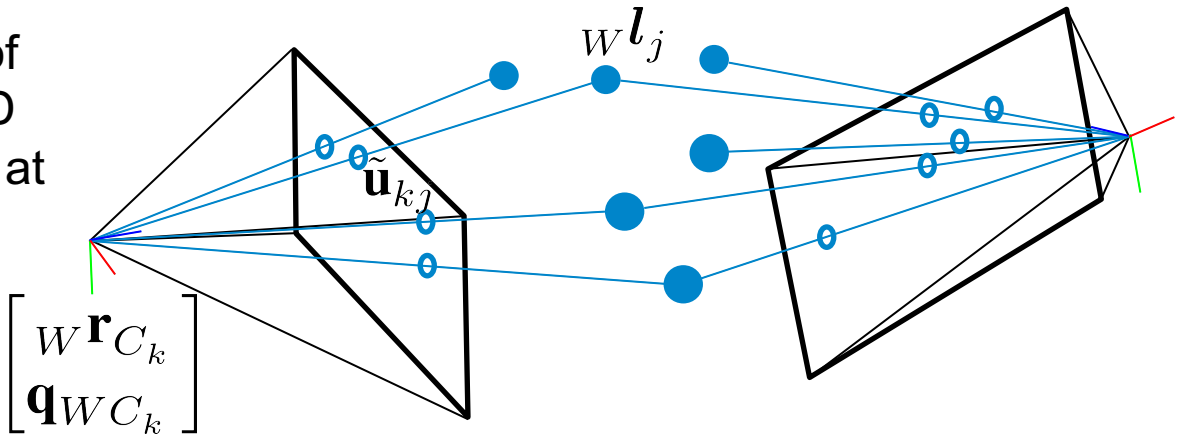


Now, we **compute Jacobians** w.r.t. $\delta\boldsymbol{\alpha} \in \mathbb{R}^3$ instead of the quaternion $\mathbf{q} \in S^4$.

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

Example: Keypoint Measurements

Given: 2D detections $\tilde{\mathbf{u}}_{kj}$ of corresponding unknown 3D landmarks ${}_W\boldsymbol{l}_j$ in images at time step k .



Find: Measurement model $\mathbf{h}(\mathbf{x}) = \mathbf{h}({}_W\mathbf{r}_{C_k}, \mathbf{q}_{WC_k}, {}_W\boldsymbol{l}_j)$ and linearisation $\mathbf{H}$.

Solution:

$$\mathbf{h}({}_W\mathbf{r}_{C_k}, \mathbf{q}_{WC_k}, {}_W\boldsymbol{l}_j) = \mathbf{u}(\boldsymbol{T}_{C_kW}\ {}_W\boldsymbol{l}_j),$$

with $\mathbf{u} = \mathbf{k}(\mathbf{d}(\mathbf{p}({}_C\mathbf{l}_j)))$ (see Lecture 2),

1. project
2. distort
3. scale and centre

Example: Keypoint Measurement Linearisation

We use local perturbations $\delta\boldsymbol{\chi} = [\delta\mathbf{r}, \delta\boldsymbol{\alpha}, \delta\mathbf{l}]^T$ around $\bar{\mathbf{x}} = [{}_W\bar{\mathbf{r}}_C, \bar{\mathbf{q}}_{WC}, {}_W\bar{\mathbf{l}}_j]^T$.

We would like to compute $\frac{\partial \mathbf{h}}{\partial {}_W\mathbf{r}_C}$, $\frac{\partial \mathbf{h}}{\partial \delta\boldsymbol{\alpha}_k}$ and $\frac{\partial \mathbf{h}}{\partial {}_W\mathbf{l}_j}$,

of $\mathbf{h}({}_W\mathbf{r}_C, \mathbf{q}_{WC}, {}_W\boldsymbol{l}_j) = \mathbf{u}({}_C\mathbf{l}_j) = \mathbf{k}(\mathbf{d}(\mathbf{p}({}_C\mathbf{l}_j)))$.

With chain rule $\frac{\partial \mathbf{h}}{\partial {}_W\mathbf{r}_C} = \frac{\partial \mathbf{u}}{\partial {}_C\mathbf{l}_j}\frac{\partial {}_C\mathbf{l}_j}{\partial {}_W\mathbf{r}_C}$, $\frac{\partial \mathbf{h}}{\partial \delta\boldsymbol{\alpha}} = \frac{\partial \mathbf{u}}{\partial {}_C\mathbf{l}_j}\frac{\partial {}_C\mathbf{l}_j}{\partial \delta\boldsymbol{\alpha}}$, $\frac{\partial \mathbf{h}}{\partial {}_W\mathbf{l}_j} = \frac{\partial \mathbf{u}}{\partial {}_C\mathbf{l}_j}\frac{\partial {}_C\mathbf{l}_j}{\partial {}_W\mathbf{l}_j}$.

Jacobian of world2cam: $\mathbf{U}({}_C\bar{\mathbf{l}}_j) := \frac{\partial}{\partial {}_C\mathbf{l}_j}\mathbf{u}({}_C\mathbf{l}_j) = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \mathbf{D} \begin{bmatrix} \frac{1}{\bar{l}_3} & 0 & -\frac{\bar{l}_1}{\bar{l}_3^2} \\ 0 & \frac{1}{\bar{l}_3} & -\frac{\bar{l}_2}{\bar{l}_3^2} \end{bmatrix}$.

Jacobian of distortion

Now, we know (Lecture 2): ${}_C\mathbf{l}_j = \mathbf{C}_{WC}^T {}_W\mathbf{l}_j - \mathbf{C}_{WC}^T {}_W\mathbf{r}_C$, thus

$\frac{\partial {}_C\mathbf{l}_j}{\partial {}_W\mathbf{r}_C} = -\overline{\mathbf{C}}_{WC}^T$,

$\frac{\partial {}_C\mathbf{l}_j}{\partial \delta\boldsymbol{\alpha}} = \overline{\mathbf{C}}_{WC}^T [{}_W\bar{\mathbf{l}}_j - {}_W\bar{\mathbf{r}}_C]^\times$,

$\frac{\partial {}_C\mathbf{l}_j}{\partial {}_W\mathbf{l}_j} = \overline{\mathbf{C}}_{WC}^T$.

Example: IMU Kinematics With Sensor Intrinsics

In Lecture 3, we derived the **continuous-time nonlinear** model

$$
\begin{aligned}
{}_W\dot{\mathbf{r}}_S &= {}_W\mathbf{v}\,, \\
\dot{\mathbf{q}}_{WS} &= \frac{1}{2}\mathbf{q}_{WS} \otimes \begin{bmatrix} {}_S\tilde{\boldsymbol{\omega}} + \mathbf{w}_\mathrm{g} - \mathbf{b}_\mathrm{g} \\ 0 \end{bmatrix}, \\
{}_W\dot{\mathbf{v}} &= \mathbf{C}_{WS}\,({}_S\tilde{\mathbf{a}} + \mathbf{w}_\mathrm{a} - \mathbf{b}_\mathrm{a}) + {}_W\mathbf{g}, \\
\dot{\mathbf{b}}_\mathrm{g} &= \mathbf{w}_{b_\mathrm{g}}, \\
\dot{\mathbf{b}}_\mathrm{a} &= \mathbf{w}_{b_\mathrm{a}}.
\end{aligned}
$$

IMU biases

We want to find the dynamics of infinitesimal perturbations $\delta\boldsymbol{\chi}$ around the linearisation point $\bar{\mathbf{x}}$ as

$$\delta\dot{\boldsymbol{\chi}}(t) = \mathbf{F}_c(\bar{\mathbf{x}}, t)\delta\boldsymbol{\chi}(t) + \mathbf{L}_c(\bar{\mathbf{x}}, t)\mathbf{w}(t),$$

with the noise terms $\delta\mathbf{w} = [\mathbf{w}_g, \mathbf{w}_a, \mathbf{w}_{b_g}, \mathbf{w}_{b_a}]^T$,

$\delta\boldsymbol{\chi} = [\delta\mathbf{r}, \delta\boldsymbol{\alpha}, \delta\mathbf{v}, \delta\mathbf{b}_\mathrm{g}, \delta\mathbf{b}_\mathrm{a}]^T$, $\bar{\mathbf{x}} = [{}_W\bar{\mathbf{r}}_S, \bar{\mathbf{q}}_{WS}, {}_W\bar{\mathbf{v}}_{WS}, \bar{\mathbf{b}}_\mathrm{g}, \bar{\mathbf{b}}_\mathrm{a}]^T$, and

$\mathbf{x} = [{}_W\bar{\mathbf{r}}_S + \delta\mathbf{r}, \delta\mathbf{q} \otimes \bar{\mathbf{q}}_{WS}, {}_W\bar{\mathbf{v}}_{WS} + \delta\mathbf{v}, \bar{\mathbf{b}}_\mathrm{g} + \delta\mathbf{b}_\mathrm{g}, \bar{\mathbf{b}}_\mathrm{a} + \delta\mathbf{b}_\mathrm{a}]^T$.

Linearised IMU Kinematics and Sensor Intrinsics

Omitting the derivation, we obtain:

$$
\begin{aligned}
\delta\dot{\mathbf{r}} &= {}_W\mathbf{v}\,, \\
\delta\dot{\boldsymbol{\alpha}} &= (\mathbf{1} + \delta\boldsymbol{\alpha}^{\times})\overline{\mathbf{C}}_{WS}\,(\mathbf{w}_{\mathrm{g}} - \delta\mathbf{b}_{\mathrm{g}}), \\
\delta\dot{\mathbf{v}} &= \mathbf{C}_{WS}\,({}_S\tilde{\mathbf{a}} + \mathbf{w}_{\mathrm{a}} - \mathbf{b}_{\mathrm{a}}) + {}_W\mathbf{g}, \\
\delta\dot{\mathbf{b}}_{\mathrm{g}} &= \mathbf{w}_{b_{\mathrm{g}}}\,, \\
\delta\dot{\mathbf{b}}_{\mathrm{a}} &= \mathbf{w}_{b_{\mathrm{a}}}\,.
\end{aligned}
$$

Now, for the overall linearised dynamics $\delta\dot{\boldsymbol{\chi}}(t) = \mathbf{F}_c(\bar{\mathbf{x}}, t)\delta\boldsymbol{\chi}(t) + \mathbf{L}_c(\bar{\mathbf{x}}, t)\mathbf{w}(t)$, we finally obtain finally:

$$
\begin{bmatrix} \delta\dot{\mathbf{r}} \\ \delta\dot{\boldsymbol{\alpha}} \\ \delta\dot{\mathbf{v}} \\ \delta\dot{\mathbf{b}}_{\mathrm{g}} \\ \delta\dot{\mathbf{b}}_{\mathrm{a}} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{1}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\overline{\mathbf{C}}_{WS} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -[\overline{\mathbf{C}}_{WS}\,({}_S\tilde{\mathbf{a}} - \overline{\mathbf{b}}_{\mathrm{a}})]^{\times} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\overline{\mathbf{C}}_{WS} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}}_{\mathbf{F}_c} \begin{bmatrix} \delta\mathbf{r} \\ \delta\boldsymbol{\alpha} \\ \delta\mathbf{v} \\ \delta\mathbf{b}_{\mathrm{g}} \\ \delta\mathbf{b}_{\mathrm{a}} \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{0}_{3\times1} \\ \overline{\mathbf{C}}_{WS}\,\mathbf{w}_{\mathrm{g}} \\ \overline{\mathbf{C}}_{WS}\,\mathbf{w}_{\mathrm{a}} \\ \mathbf{w}_{b_{\mathrm{g}}} \\ \mathbf{w}_{b_{\mathrm{a}}} \end{bmatrix}}_{\mathbf{L}_c\mathbf{w}}\,.
$$

Discrete-Time IMU Kinematics / Sensor Intrinsics

Discretisation (using Trapezoidal Integration):

$$\Delta\mathbf{x}_1 = \Delta t\mathbf{f}_c(\mathbf{x}_{k-1}, t_{k-1}),$$
$$\Delta\mathbf{x}_2 = \Delta t\mathbf{f}_c(\mathbf{x}_{k-1} + \Delta\mathbf{x}_1, t_k)),$$
$$\mathbf{x}_k = \mathbf{x}_{k-1} + \frac{1}{2}(\Delta\mathbf{x}_1 + \Delta\mathbf{x}_2).$$

Make sure to **re-normalise the quaternions** whenever modifying!

Linearised Discrete Error Dynamics (using Trapezoidal Integration):

We apply the chain-rule to the discretisation scheme…

$$\mathbf{F}_k = \mathbf{1} + \frac{\Delta t}{2}\mathbf{F}_c(\mathbf{x}_{k-1}, t_{k-1}) + \frac{\Delta t}{2}(\mathbf{F}_c(\mathbf{x}_{k-1} + \Delta\mathbf{x}_1, t_k)(\mathbf{1} + \Delta t\mathbf{F}_c(\mathbf{x}_{k-1}, t_{k-1}))).$$
$$\mathbf{L}_k = \frac{1}{2}(\mathbf{L}_c(\mathbf{x}_{k-1}, t_{k-1}) + \mathbf{L}_c(\mathbf{x}_{k-1} + \Delta\mathbf{x}_1, t_k)).$$

Covariance Propagation: $\mathbf{P}_k = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^\top + \mathbf{L}_k\mathbf{Q}_k\mathbf{L}_k^\top$.

With: $\mathbf{Q}_k = \text{diag}[\sigma_{c,g}\mathbf{1}_3, \sigma_{c,a}\mathbf{1}_3, \sigma_{c,w_g}\mathbf{1}_3, \sigma_{c,w_a}\mathbf{1}_3]\Delta t.$

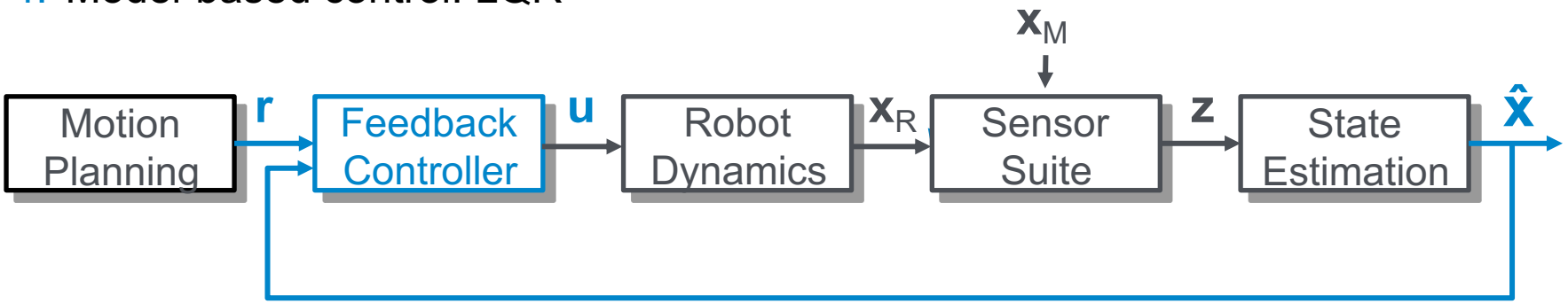
The sigmas denote continuous-time noise densities (e.g. from IMU datasheets).

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. Single-Input-Single-Output (SISO) vs Multiple-Input-Multiple-Output (MIMO)
2. System response
3. PID control of SISO systems
4. Model-based control: LQR

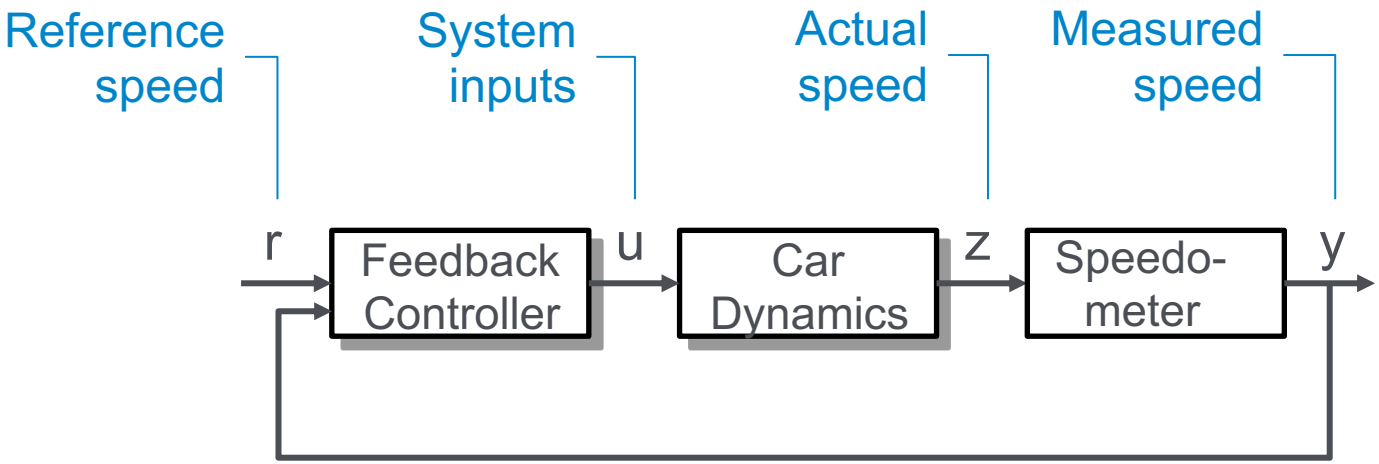


Disclaimer: 'Control Systems' is usually taught over 2-4 courses. This is a very brief summary of a few selected important/practical aspects.

Single-Input-Single-Output (SISO) Systems

Example: cruise control

- Reference: r: the desired speed
- System input: u: the gas pedal position (u>0) / break position (u<0)
- System output: z: the speed of the car



Single-Input-Single-Output (SISO) Systems

Example: cruise control

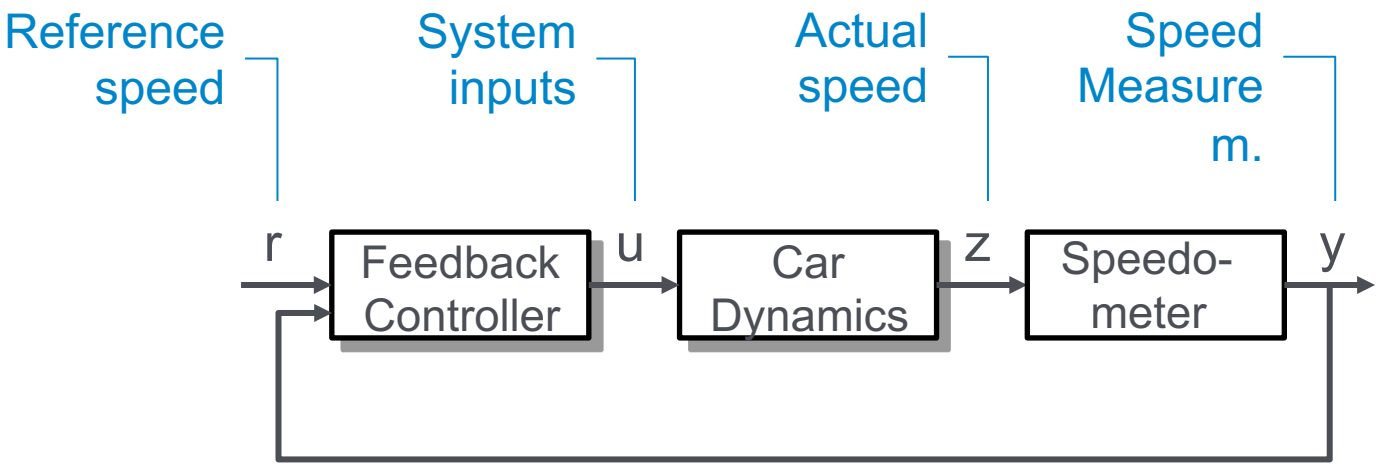
What could a feedback control law look like?

$$u = \begin{cases} u_{\max}, y < r \\ 0, y = r \\ u_{\min}, y > r \end{cases}$$

"Bang-bang" control
(will work, but I don't want to sit in that car)

$$u = k_p(r - y)$$

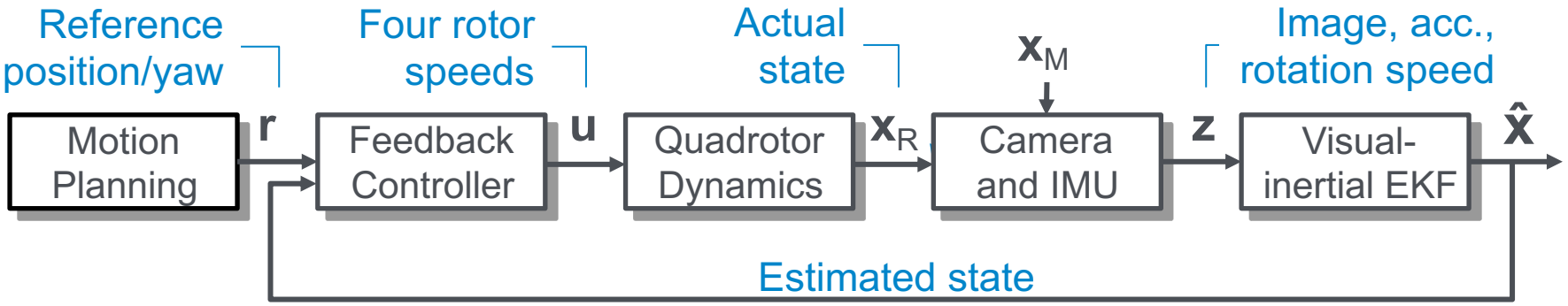
P-control (proportional control)



Most Systems Are Multiple-Input-Multiple-Output (MIMO)

Example: quadrotor

- Reference: **r**: typically: position/yaw (maybe also speed?) trajectory
- System inputs: **u**: four rotor speeds
- Internal states **x**: position, orientation, speed, IMU biases
- System outputs: $\hat{\mathbf{x}}$: typically: position, orientation, speed, (IMU biases)



What could a feedback control law look like?

- This is not obvious at all… We will see
- how we can still do SISO control with some trickery,
- or we can do actual MIMO control (e.g. model-predictive).

Predicting the Future

We can use the discrete-time system (neglecting uncertainty here)

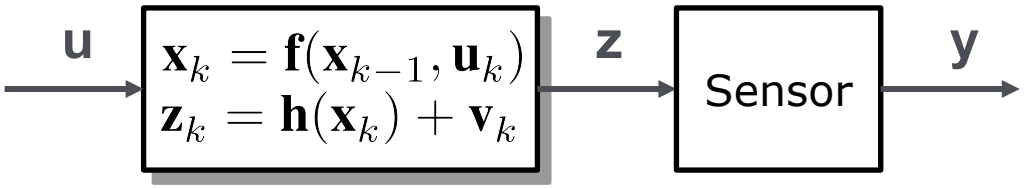
$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k)$,

$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$.

to simulate / predict the future!

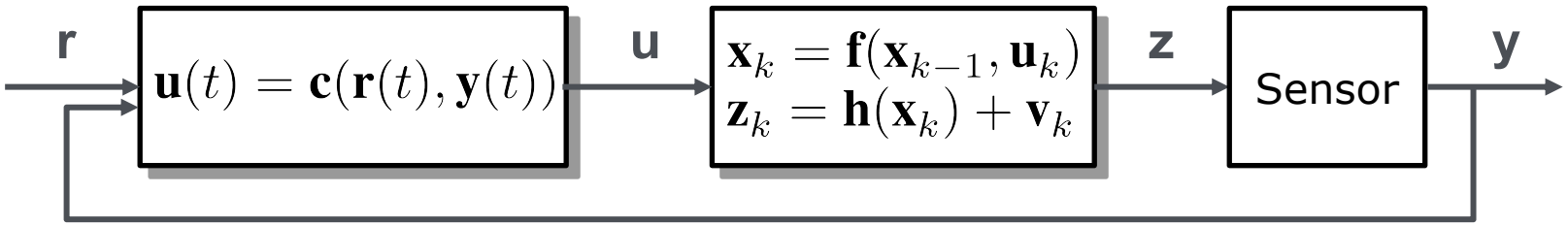
Open-loop response

We can simulate how the state $\mathbf{x}$ reacts to e.g. a step in the control input $\mathbf{u}$:



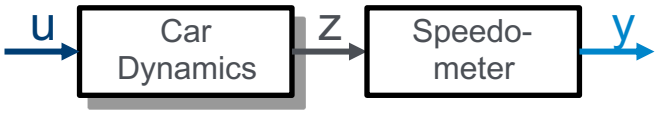
Closed-loop response

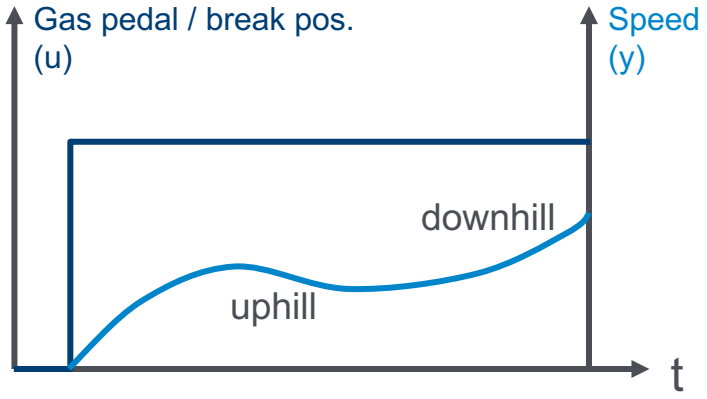
Having designed a controller $\mathbf{u}(t) = \mathbf{c}(\mathbf{r}(t), \mathbf{y}(t))$, we can simulate the reaction of $\mathbf{x}$ to e.g. a step in the reference $\mathbf{r}$



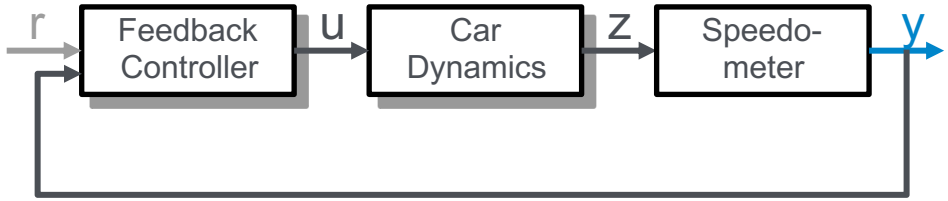
Predicting the Future: Cruise Control Example

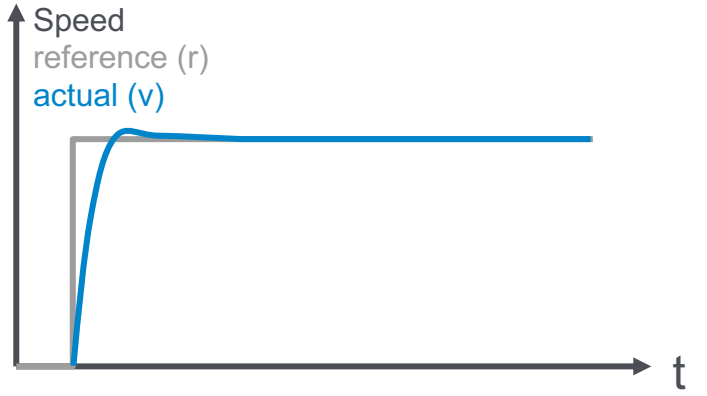
Open-loop response



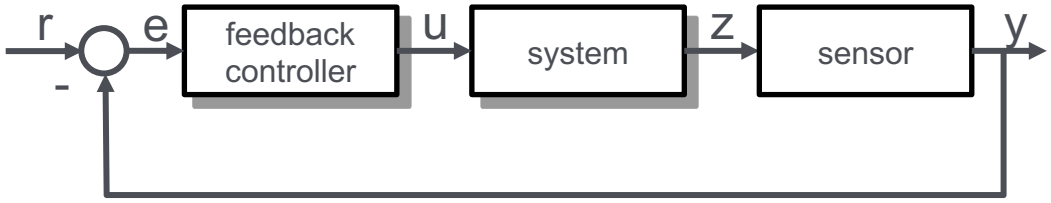


Closed-loop response





Proportional-Integral-Differential (PID) Control



$$u(t) = k_p e(t) + k_i \int_{t_0}^{t} e(\tau)d\tau + k_d \frac{de(t)}{dt}$$

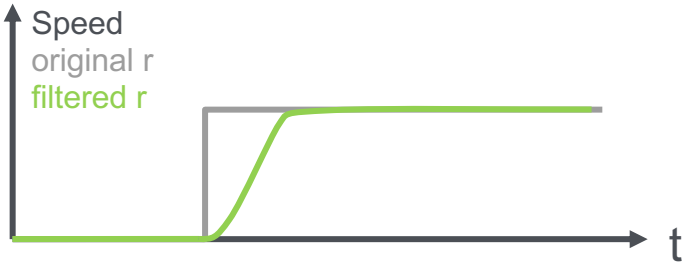
Proportional gain
reduces current error

Integral gain
reduces long-term offset

Differential gain
considers error trend,
reduces overshoot

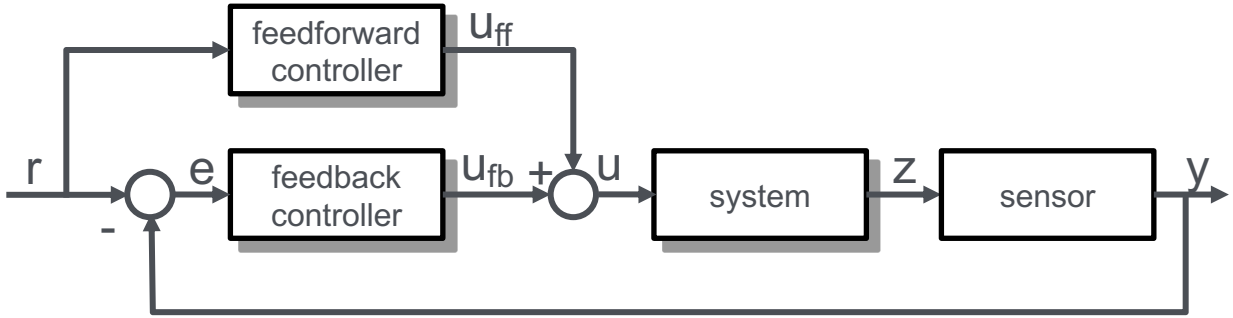
PID Control: Popular Extensions

1. **Reference filtering**: respect physical limits already in the reference. E.g.



2. **Anti-Reset-Windup**: stops integrating the error for the I-part, when $\mathbf{u}$ is at its physical limit.

3. **Feed-forward controller**: reduce "work" for the feed-back controller. E.g. nominal gas pedal positions resulting in a certain speed.



PID Control – Cruise Control Example

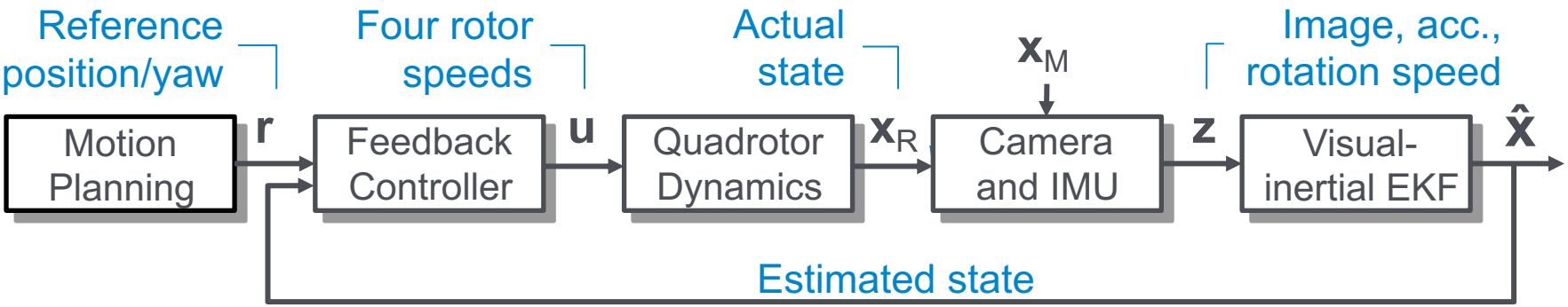
```
class PidController {
 public:
  /// \brief Default constructor, set all zero.
  PidController() : lastError_(0.0), lastErrorDerivative_(0.0),
integratedError_(0.0) {}
  /// \brief Compute control output.
  /// @param[in] referenceSpeed The desired speed [mph].
  /// @param[in] currentSpeed The current speed as measured [mph].
  /// @param[in] deltaT Time elapsed since last call [s]. Assumes deltaT>0.0.
  /// \return The control output u=[-1,...,1]. Negative for breaking.
  double computeOutput(double referenceSpeed, double currentSpeed, double deltaT);
 private:
  double lastError_ = 0.0;  ///< The error last time computeOutput called.
  double lastErrorDerivative_ = 0.0; ///< The error time derivative last time.
  double integratedError_ = 0.0; ///< The integrated error last time.
  const double k_p_ = 1.0; ///< Proportional gain.
  const double k_i_ = 0.1; ///< Integral gain.
  const double k_d_ = 0.5; ///< Differential gain.
};
```

PID Control – Cruise Control Example

```
double PidController::computeOutput(double referenceSpeed,
                                    double currentSpeed, double deltaT) {
  // compute error:
  const double error = referenceSpeed - currentSpeed;
  // compute error derivative:
  double errorDerivative = (error - lastError_) / deltaT;
  // do some smoothing (numeric derivatives are noisy):
  errorDerivative = 0.8 * lastErrorDerivative_ + 0.2 * errorDerivative;
  // compute output:
  double output = k_p_ * error + k_i_ * integratedError_ + k_d_ * errorDerivative;
  // saturate:
  if (output < -1.0) {
    output = -1.0;  // clamp -- and DO NOT INTEGRATE ERROR (anti-reset windup)
  } else if (output > 1.0) {
    output = 1.0;  // clamp -- and DO NOT INTEGRATE ERROR (anti-reset windup)
  } else {
    integratedError_ += error * deltaT; // safe to keep integrating
  }
  // save:
  lastError_ = error;
  lastErrorDerivative_ = errorDerivative;
  return output;
}
```

PID Drone Control – Input Remapping

Can we still use PID, even if this is a complicated MIMO system…?



Interestingly, this is done a lot in practice. But we need to apply some tricks.

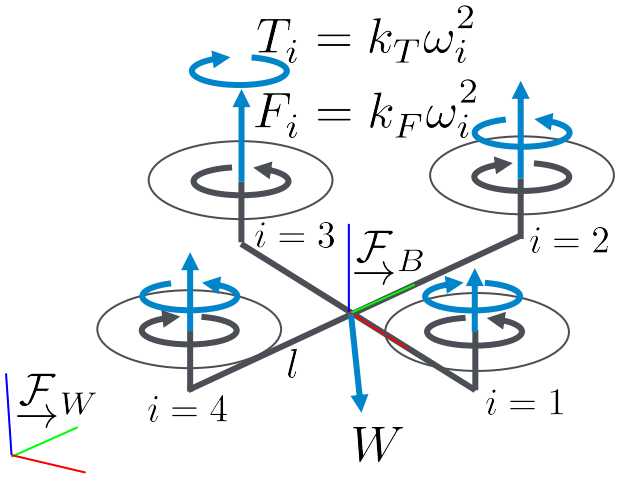
Re-map the input.

From the dynamic model (Lecture 3) we know:

$$ {}_B\mathbf{F} = [0, 0, \underbrace{F_1 + F_2 + F_3 + F_4}_{=:F}]^T - \mathbf{C}_{BW}\,[0, 0, mg]^T, $$

$$ {}_B\mathbf{M} = [\underbrace{F_2 l - F_4 l}_{=:M_x}, \underbrace{-F_1 l + F_3 l}_{=:M_y}, \underbrace{-T_1 + T_2 - T_3 + T_4}_{=:M_z}]^T. $$

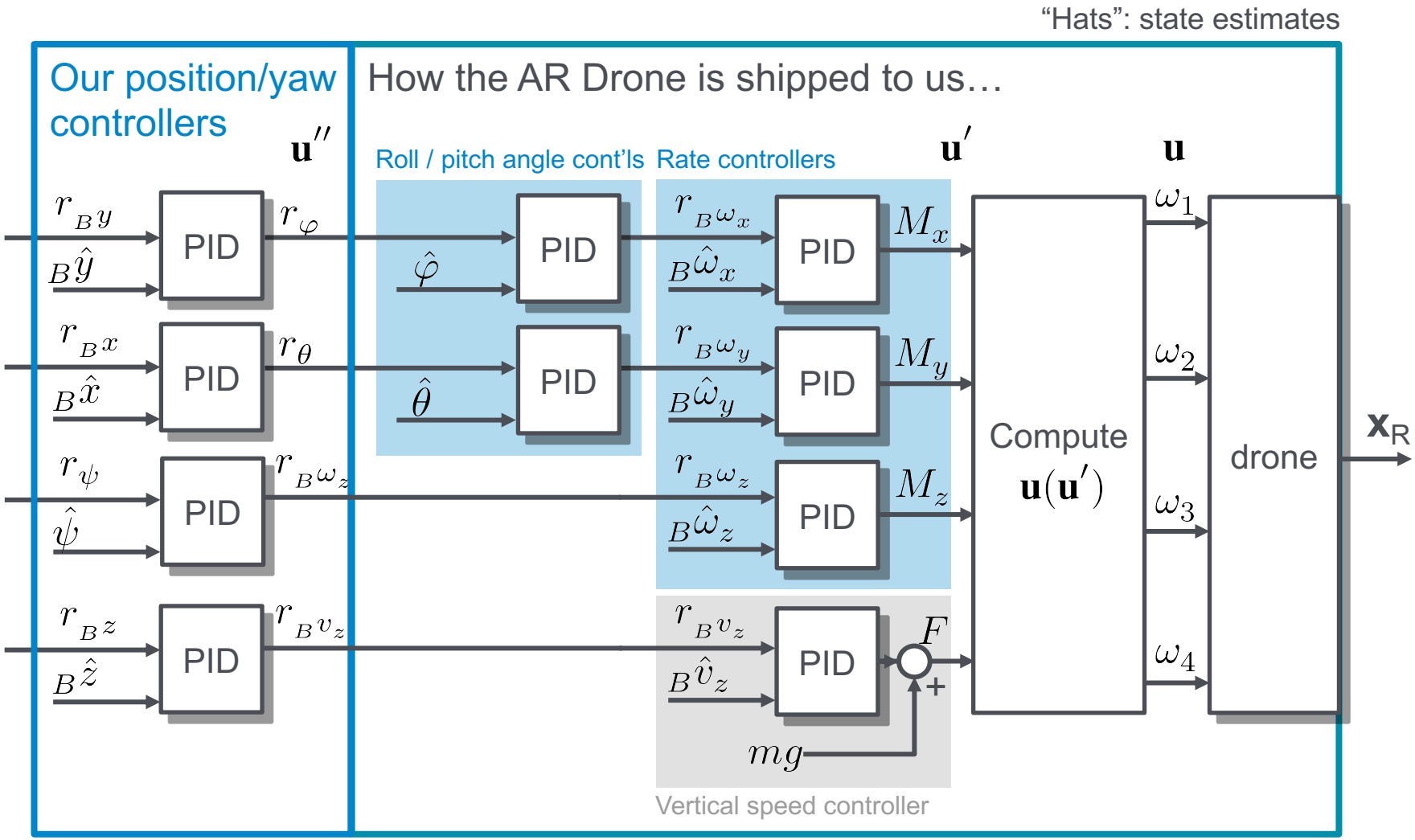
So, instead of $\mathbf{u} = [\omega_1, \omega_2, \omega_3, \omega_4]^T$, we use $\mathbf{u}' = [M_x, M_y, M_z, F]^T$.



Imperial College London

SMART ROBOTICS LAB

PID Drone Control: Cascaded and Parallel SISO Loops



PID Drone Control – Dangers

- There are several assumptions, limitations, and dangers with the approach:
- Operation must be near hover (roll/pitch parameterisation).
- Parallel loops may interfere with each-other (disturb each-other) because they are never perfectly decoupled.
- Cascaded loops: outer loops assume the inner loops can perfectly follow the reference! This is never the case…
  - Tune parameters of innermost loop first, then work your way outward.
  - Have reduced expectation on fast reference tracking on outer loops (i.e. use more conservative gains in outer loops).

Model-Based Control: Linear Quadratic Regulator (LQR)

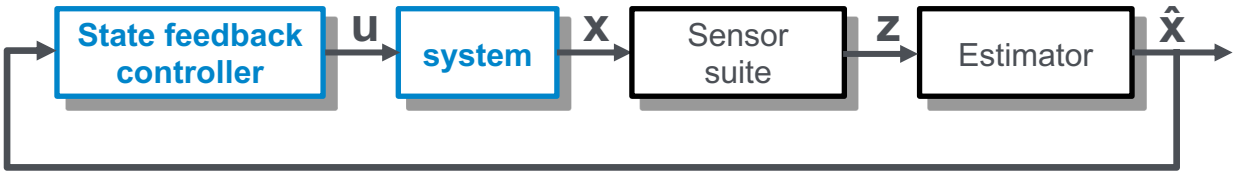
Let's assume linear continuous-time dynamics

$\dot{\mathbf{x}}(t) = \mathbf{F}_c\mathbf{x}(t) + \mathbf{G}_c\mathbf{u}(t),$

That we would like to regulate to $\mathbf{x}(t) = \mathbf{0}$.

Now, let's try and find a **State-Feedback Control Law** of the form

$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t).$



Now we want to minimise a cost functional

$$J = \int_t^\infty \underbrace{\mathbf{x}(\tau)^T\mathbf{Q}\mathbf{x}(\tau)}_{\text{Penalise deviation of state } \mathbf{x}(t) \text{ from } \mathbf{0}} + \underbrace{\mathbf{u}(\tau)^T\mathbf{R}\mathbf{u}(\tau)}_{\text{Penalise deviation of control input } \mathbf{u}(t) \text{ from } \mathbf{0}} d\tau.$$

(i.e. infinite-horizon; finite-horizon variants exist)

Model-Based Control: Linear Quadratic Regulator (LQR)

The solution (proof omitted here) is given by

$$\mathbf{K} = \mathbf{R}^{-1}(\mathbf{G}_c^T\mathbf{P}),$$

where $\mathbf{P}$ is found by solving the so-called Algebraic Ricatti Equation

$$\mathbf{F}_c^T\mathbf{P} + \mathbf{P}\mathbf{F}_c - (\mathbf{P}\mathbf{G}_c)\mathbf{R}^{-1}(\mathbf{G}_c^T\mathbf{P}) + \mathbf{Q} = \mathbf{0}.$$

This is a matrix equation that we cannot solve by hand normally.

But it's one line in Matlab: `K = lqr(F_c, G_c, Q, R);`

Notes:

- Solving the (say with Matlab) for the feedback matrix $\mathbf{K}$ is quite expensive, but is done **offline**! At runtime we only evaluate $\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$.
- We still have to set (i.e. **tune**) the weighting matrices $\mathbf{Q}$ and $\mathbf{R}$. It's usually a good idea to use diagonal matrices.
- The optimality is nice, but **assumes no limits** on the system input $\mathbf{u}$. In practice, there will be limits and too high values on $\mathbf{Q}$ will result in a bang-bang-style controller that is no longer optimal (or not even stable).
- We can use a **non-zero reference** point as well: $\mathbf{u} = \mathbf{K}(\mathbf{r} - \mathbf{x})$.

LQR With Nonlinear Systems

We can do what we always do: use the linear approximation:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{F}_c\delta\mathbf{x}(t) + \mathbf{G}_c\delta\mathbf{u}(t),$$

Around the current state (estimate) $\bar{\mathbf{x}}$ and the current control input $\bar{\mathbf{u}}$.

Problem: $\mathbf{F}_c$ and $\mathbf{G}_c$ will be changing during runtime, but we don't want to solve the Matrix-Ricatti equation in the control loop (too expensive)!

What we can do is **pre-compute** the feedback matrix $\mathbf{K}$ for many linearisation points $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$ as a **lookup-table**.

Note: all guarantees/optimality is lost here…
(But in practice this might still work very well)

Any Questions?

See you on Thursday at 1pm for the practical!