Imperial College London

SMART ROBOTICS LAB

22/02/2021

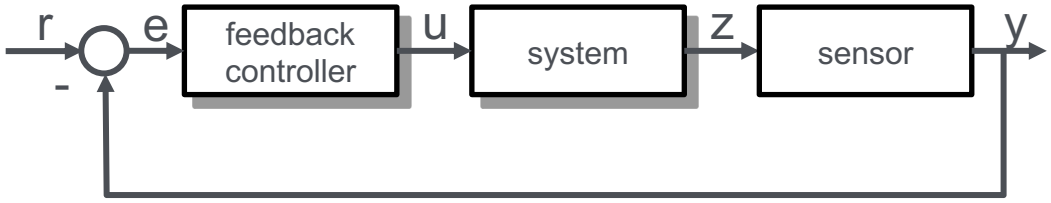
Lecture 6:
Nonlinear Least Squares

70003 Advanced Robotics

Dr Stefan Leutenegger

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

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

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 .

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

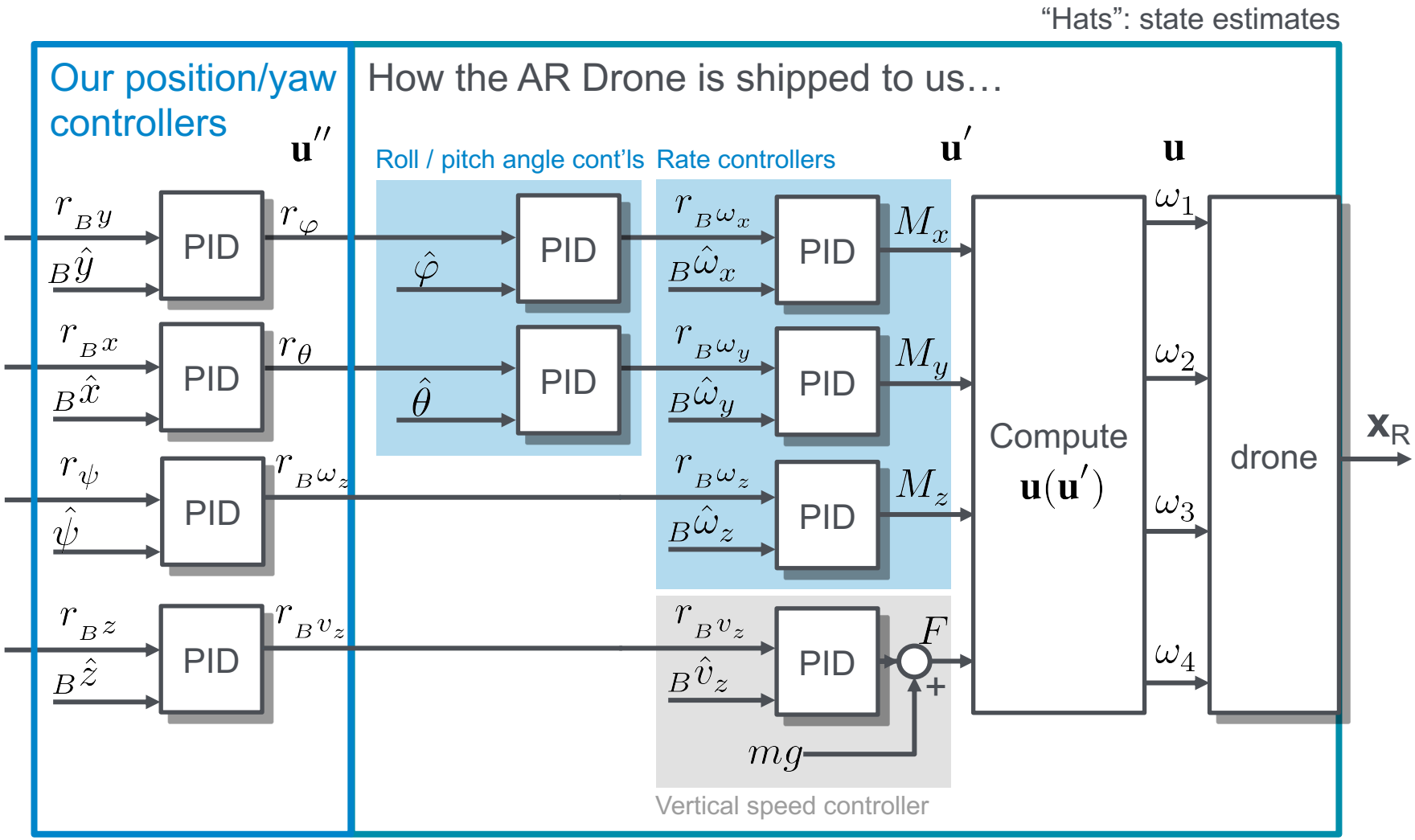
Linearise: around $\overline{\mathbf{x}}, \overline{\mathbf{u}}$

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

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

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

PID Drone Control: Cascaded and Parallel SISO Loops



Model-Based Control: Linear Quadratic Regulator (LQR)

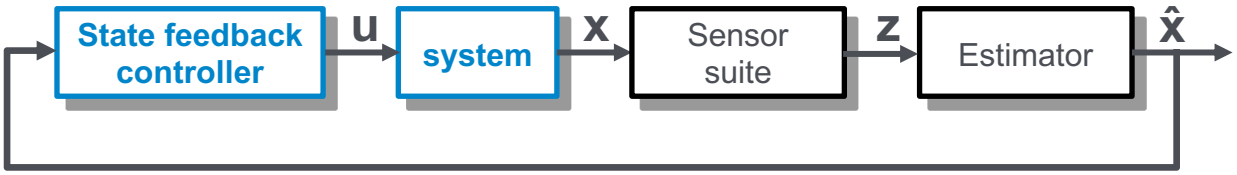
Let's assume linear continuous-time dynamics

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

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)

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?

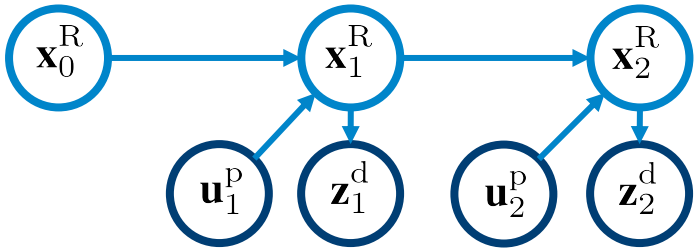
- **Maximum a Posteriori (MAP) Estimation:**
  What values for variables best explain all the measurements and a prior belief $p(\mathbf{x})$ about those variables?

The EKF is one way of solving this problem.

Today we will see another one.

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:

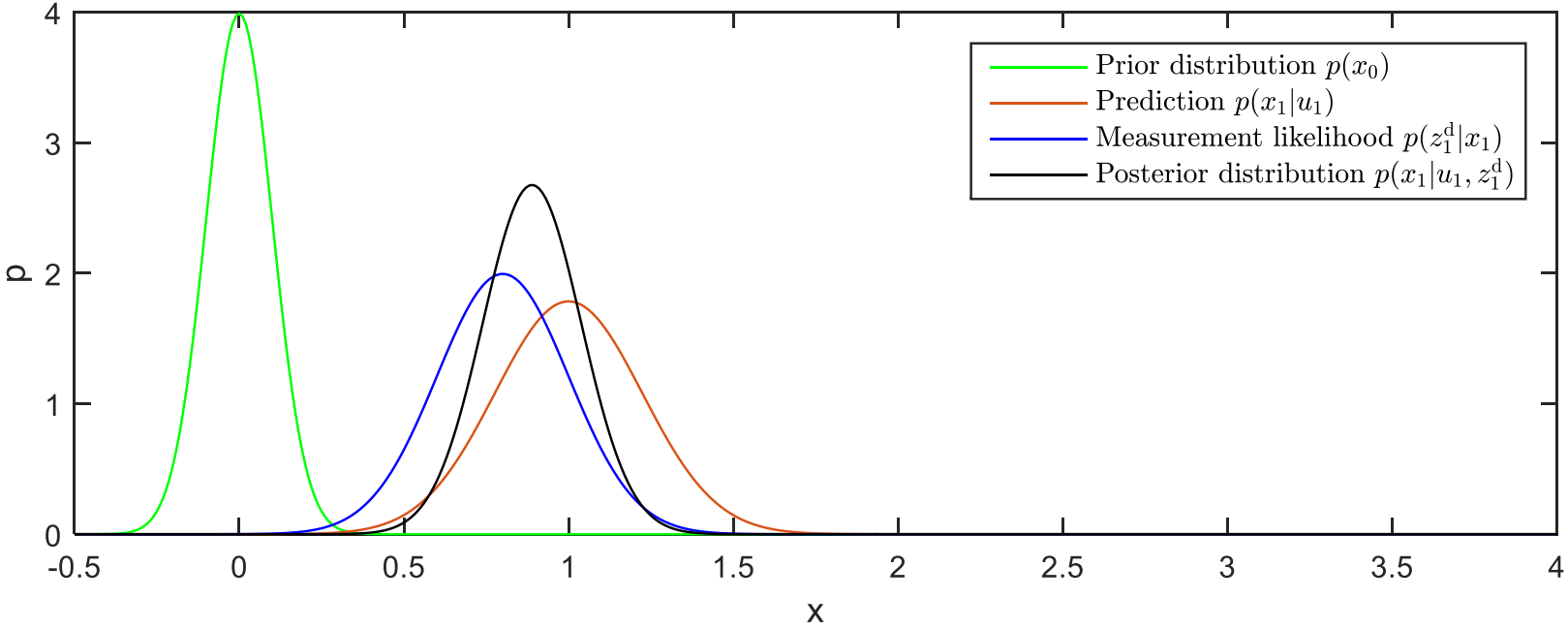
1. **Predict**: $p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_{1:k}^{\mathrm{p}}, \mathbf{z}_{1:k-1}^{\mathrm{d}}) = \int p(\mathbf{x}_k^{\mathrm{R}}|\mathbf{u}_k^{\mathrm{p}}, \mathbf{x}_{k-1}^{\mathrm{R}}) p(\mathbf{x}_{k-1}^{\mathrm{R}}|\mathbf{u}_{1:k-1}^{\mathrm{p}}, \mathbf{z}_{1:k-1}^{\mathrm{d}}) d\mathbf{x}_{k-1}^{\mathrm{R}}$ **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): $z^{\mathrm{d}} = x, \quad z^{\mathrm{d}} \sim \mathcal{N}(0, r).$

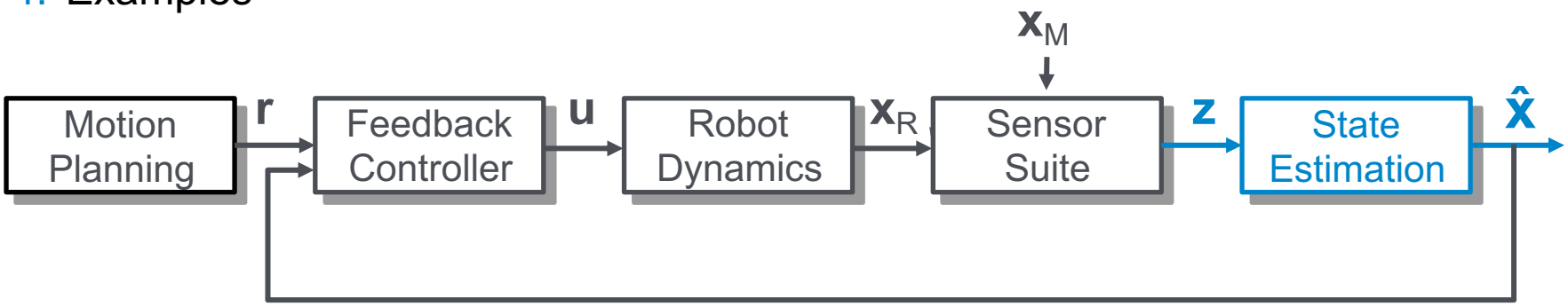


Schedule: Lectures

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. Linear Least Squares
2. Probabilistic Non-Linear Least Squares Estimation Problem
3. Optimising Non-Linear Least Squares Problems
4. Examples



Linear Least Squares – Linear Function Fit

Given:

- N data points $\tilde{z}_i$ at t_i.
- Assumption that the data can be explained with a model $z = (a_1 t + a_0)$.

Find:

$\mathbf{x}^* = [a_0^*, a_1^*]^T$ minimising the sum of square differences

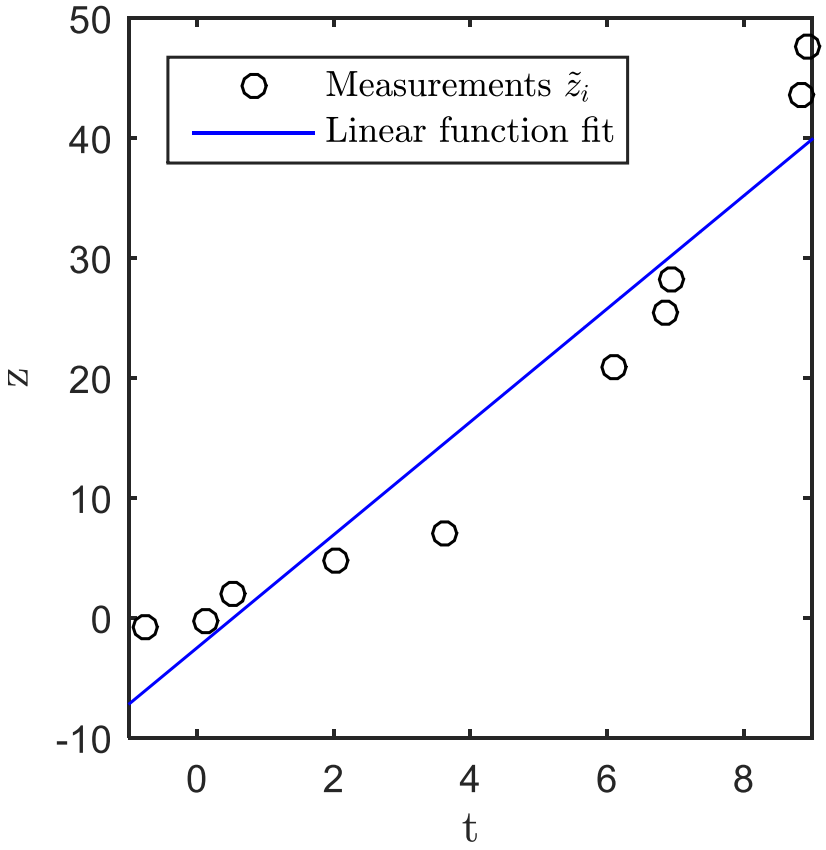
$$c(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{N} \|\tilde{z}_i - (a_1 t_i + a_0)\|_2^2.$$

Solution:

Re-write as $c(\mathbf{x}) = \frac{1}{2}(\tilde{\mathbf{z}} - \mathbf{H}\mathbf{x})^T(\tilde{\mathbf{z}} - \mathbf{H}\mathbf{x})$,

$$\tilde{\mathbf{z}} = \begin{bmatrix} \tilde{z}_1 \\ \vdots \\ \tilde{z}_N \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_N \end{bmatrix}.$$

Necessary condition for optimality:



Linear Least Squares – Polynomial Fit

Given:

- N data points $\tilde{z}_i$ at t_i.
- Assumption that the data can be explained with a model $z = (a_2 t^2 + a_1 t + a_0)$.

Find:

$\mathbf{x}^* = [a_0^*, a_1^*, a_2^*]^T$ minimising the sum of square differences

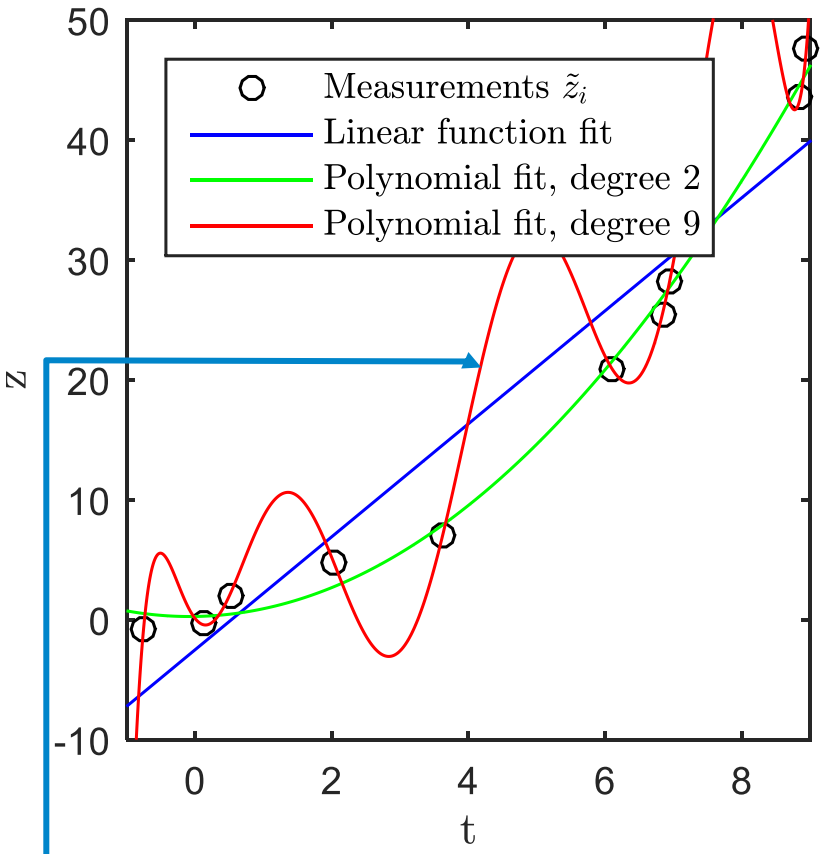
$$c(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^{N} \| \tilde{z}_i - (a_2 t_i^2 + a_1 t_i + a_0) \|_2^2.$$

Solution:

Analogous to linear function fit, but with

$$\mathbf{H} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ 1 & t_N & t_N^2 \end{bmatrix}.$$

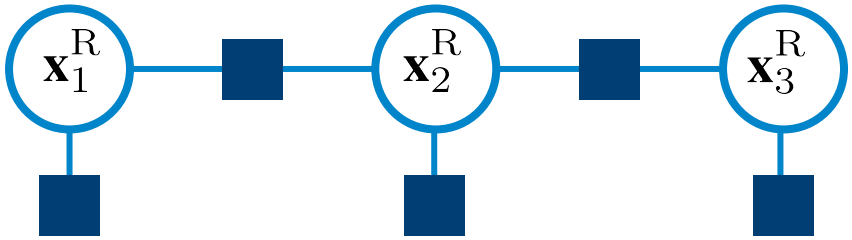
Note: This is a nonlinear function to fit, but still a **linear** regression problem!



Overfitting: the problem of explaining too little (noisy) data with too many parameters…

Factor Graph – Differential Drive Robot Example

- All states of the trajectory are being estimated



■ Measurements: error terms

Measurements are of the form $p(\mathbf{z}_i|\mathbf{x}_j)$.
This will allow us to formulate measurement errors $\mathbf{e}_i = \tilde{\mathbf{z}}_i - \mathbf{f}(\mathbf{x}_j)$.

The Probabilistic View on Estimation

Given:

- Variables to be estimated $\mathbf{x}$.
- N independent measurements $\tilde{\mathbf{z}}_i$ and associated measurement likelihood functions $p(\mathbf{z}_i|\mathbf{x})$.
- Optionally: a prior distribution for $\mathbf{x}$, $p(\mathbf{x})$.

Find:

$\mathbf{x}^* = \operatorname{argmax} p(\mathbf{x}|\mathbf{z})$ with

$$p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{N} p(\mathbf{z}_i|\mathbf{x})$$ **: Maximum Likelihood, ML**

$$p(\mathbf{x}|\mathbf{z}) \propto p(\mathbf{z}|\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}) \prod_{i=1}^{N} p(\mathbf{z}_i|\mathbf{x})$$ **: Maximum a Posteriori, MAP**

The Probabilistic View on Least Squares

Now assume:

- Gaussian measurement likelihood $p(\mathbf{z}_i|\mathbf{x}) = \mathcal{N}(\mathbf{h}_i(\mathbf{x}), \mathbf{R}_i)$.
- Optionally: Gaussian prior $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}_0, \mathbf{P})$.

Find: $\mathbf{x}^* = \operatorname{argmax} p(\mathbf{x}|\mathbf{z}) = \operatorname{argmin}(-\log(p(\mathbf{x}|\mathbf{z})))$

ML Estimator: minimise the weighted sum of least squares

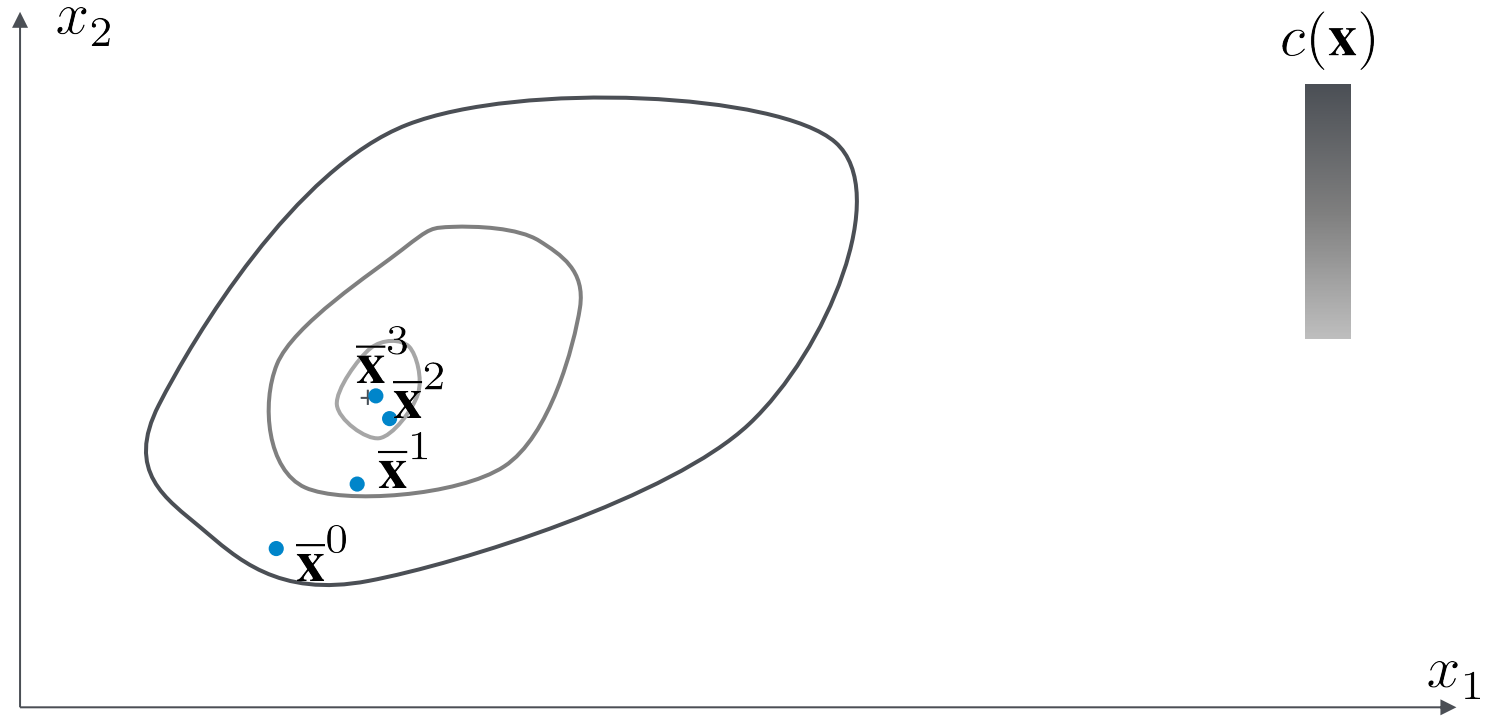
MAP Estimator: minimise the sum of least squares plus prior cost

How To Solve Nonlinear Least Squares Problems [8,9]

Linear case $\mathbf{h}_i(\mathbf{x}) = \mathbf{H}_i\mathbf{x}$

The solution is directly obtained using the normal equations.

Nonlinear case: iterative methods starting at an initial guess for $\mathbf{x}$



The Gauss-Newton Method

Given:

- Nonlinear error terms of the form $\mathbf{e}_i(\mathbf{x}) := \tilde{\mathbf{z}}_i - \mathbf{h}(\mathbf{x}) \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_i)^*$.

Iterative solution to the associated nonlinear least squares problem:

1. Pick a suitable **starting point** $\bar{\mathbf{x}}^0$.
2. Approximate the cost employing linear error terms $\mathbf{e}_i(\mathbf{x}) \approx \mathbf{e}_i(\bar{\mathbf{x}}^k) + \mathbf{E}_i(\bar{\mathbf{x}}^k)\Delta\mathbf{x}$, with $\mathbf{E}_i(\bar{\mathbf{x}}^k) = \frac{\partial}{\partial \mathbf{x}} \mathbf{e}_i(\mathbf{x})|_{\bar{\mathbf{x}}^k}$ (note: typically sparse) and formulate the **normal equations**:

$$\underbrace{\sum_i \mathbf{E}_i(\bar{\mathbf{x}}^k)^T \mathbf{R}_i^{-1} \mathbf{E}_i(\bar{\mathbf{x}}^k)}_{=:\mathbf{A}} \Delta\mathbf{x} = \underbrace{-\sum_i \mathbf{E}_i(\bar{\mathbf{x}}^k)^T \mathbf{R}_i^{-1} \mathbf{e}_i(\bar{\mathbf{x}}^k)}_{=:\mathbf{b}}.$$

3. **Solve** the normal equations for $\Delta\mathbf{x}$ using e.g. **Cholesky Decomposition** (more on this step see Lecture 8).
4. **Update** the current estimate as $\bar{\mathbf{x}}^{k+1} = \bar{\mathbf{x}}^k + \Delta\mathbf{x}$.
5. Check **convergence**: if not OK, go to step 2.

*Note: a prior can be added to the error terms analogously.

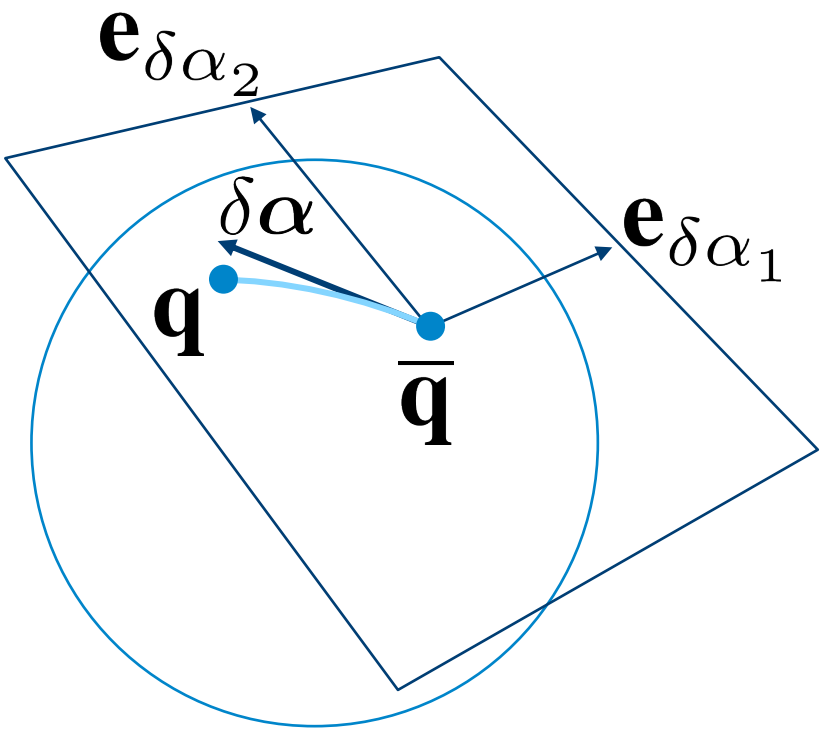
Levenberg-Marquardt

A trust-region method that interpolates between gradient-descent and Gauss-Newton.

1. Pick a **starting point** $\bar{\mathbf{x}}^0$, initial parameter $\lambda^0 = \max \mathrm{diag}(\mathbf{A})$ and ν, e.g. $\nu = 2$.
2. Build the **modified** Gauss Newton system: $(\mathbf{A} + \lambda \mathrm{diag}(\mathbf{A}))\Delta\mathbf{x} = \mathbf{b}$.
3. **Solve** it for $\Delta\mathbf{x}$.
4. **Update**
   - If cost is reduced: $\bar{\mathbf{x}}^{k+1} = \bar{\mathbf{x}}^k + \Delta\mathbf{x}$ and expand trust region $\lambda^{k+1} = \lambda^{k+1}/\nu$.
   - Else: no update on $\bar{\mathbf{x}}$, contract the trust region $\lambda^{k+1} = \lambda^{k+1}\nu$; go to Step 3
5. Check **convergence**: if not OK, go to step 2.

Optimisation On Manifolds

We may have to perform optimisations on manifolds…



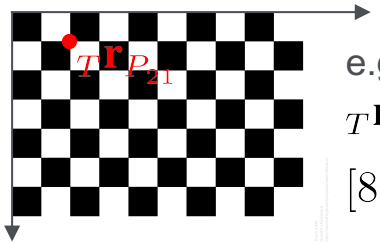
Example: quaternions:

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

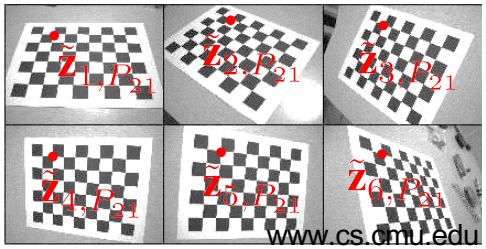
Modifications to solving optimisation problems:

- When computing Jacobians, compute them w.r.t. $\delta\boldsymbol{\alpha} \in \mathbb{R}^3$.
- Compute the update $\Delta\boldsymbol{\alpha}$ and apply it as $\overline{\mathbf{q}}^{k+1} = \delta\mathbf{q}(\Delta\boldsymbol{\alpha}) \otimes \overline{\mathbf{q}}^{k}$.

Example: Camera Calibration



e.g. ${}_T\mathbf{r}_{P_{21}} = [8, 4, 0]^T$

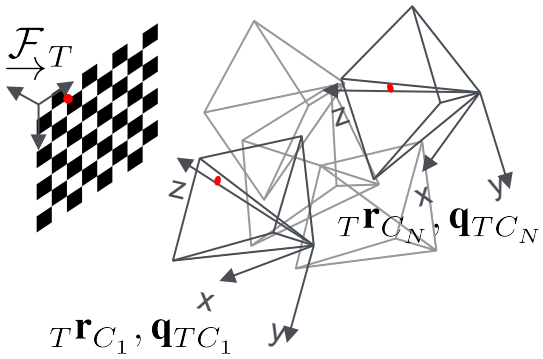


Given:

- A calibration pattern with $j = 1, \ldots, M$ observable points ${}_T\mathbf{r}_{P_j}$ defined in the target frame $\underrightarrow{\mathcal{F}}_T$.
- A set of images N of the calibration pattern taken by the same camera from different points of view. Assume the calibration points are identified in the image as $\tilde{\mathbf{z}}_{i,P_j}$ (known 3D-2D correspondences).
- The pinhole camera projection model (with radial-tangential distortion) $\mathbf{u} = \mathbf{k}(\mathbf{d}(\mathbf{p}({}_C\mathbf{r}_P)))$.

Find:

- The most likely camera poses (extrinsics) ${}_T\mathbf{r}_{C_i}, \mathbf{q}_{TC_i}$ per image.
- The most likely camera intrinsics parameters $\mathbf{x}_c$.



Solution:

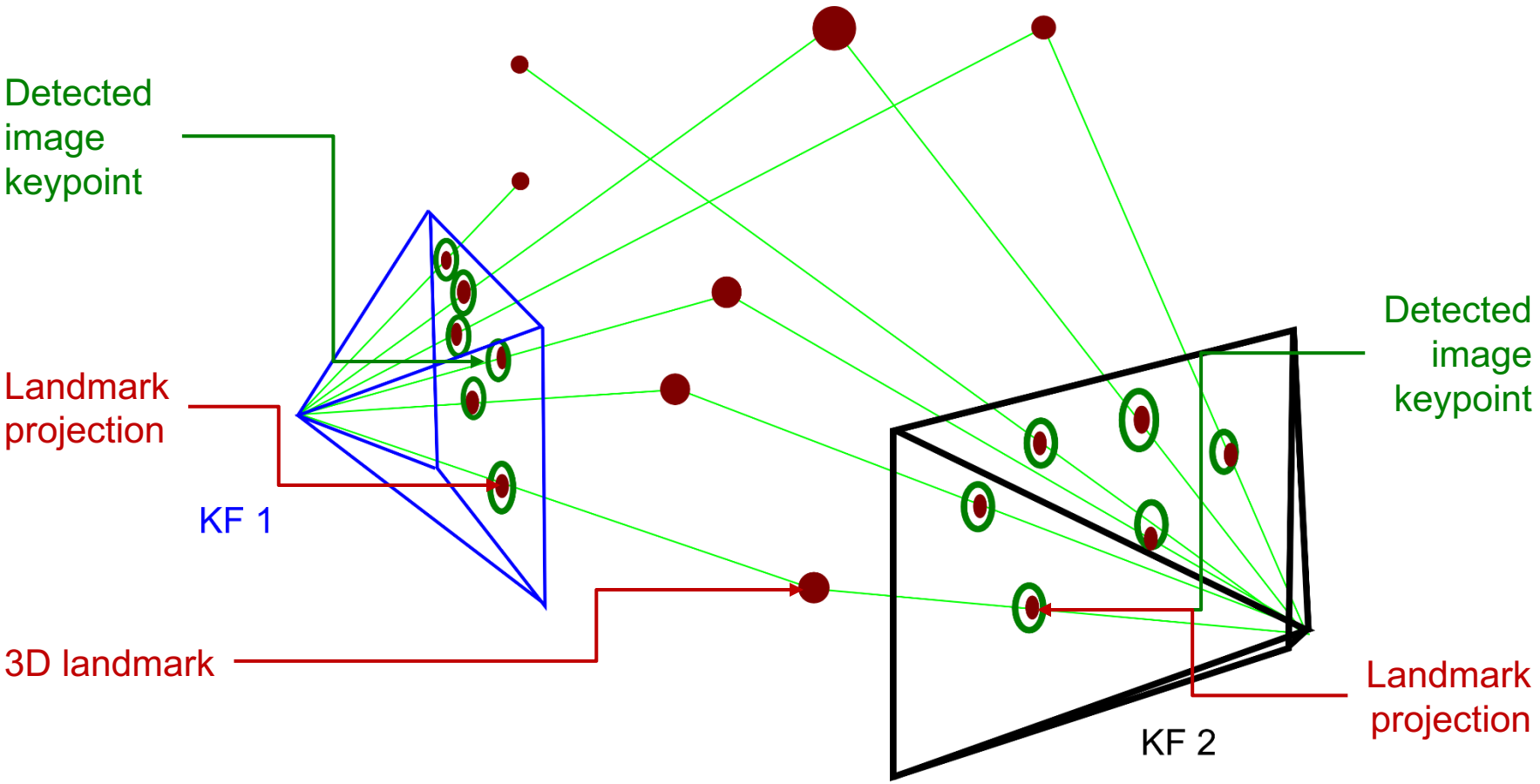
We formulate the reprojection error per image and point:

$$\mathbf{e}_{i,j} = \tilde{\mathbf{z}}_{i,P_j} - \underbrace{\mathbf{k}(\mathbf{d}(\mathbf{p}(}_{\text{Cam. projection}}\underbrace{\mathbf{C}_{C_iT}\,{}_T\mathbf{r}_{P_j} - \mathbf{C}_{C_iT}\,{}_T\mathbf{r}_{C_i}}_{\text{Transform point into camera frame}}))).$$

Minimise the overall reprojection cost $c(\mathbf{x}_c, {}_T\mathbf{r}_{C_1}, \mathbf{q}_{TC_1}, \ldots, {}_T\mathbf{r}_{C_N}, \mathbf{q}_{TC_N}) = \sum_{i=1}^{N}\sum_{j=1}^{M} \mathbf{e}_{i,j}^T \mathbf{R}^{-1} \mathbf{e}_{i,j}$,
with $\mathbf{R} = \text{diag}[\sigma^2, \sigma^2]$, where σ is the detection uncertainty in pixels.

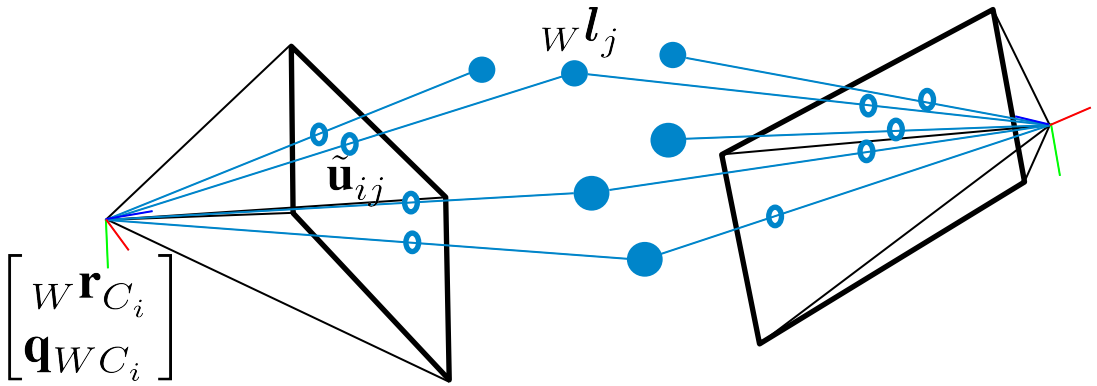
Note: we will need the Jacobians $\frac{\partial \mathbf{e}_{i,j}}{\partial \mathbf{x}_c}$, $\frac{\partial \mathbf{e}_{i,j}}{\partial {}_T\mathbf{r}_{C_i}}$ and $\frac{\partial \mathbf{e}_{i,j}}{\partial \boldsymbol{\alpha}_{TC_i}}$ (see lecture 4).

The Visual SLAM Problem



Bundle Adjustment: Visual SLAM as Optimisation Problem

Given: 2D detections $\tilde{\mathbf{u}}_{ij}$ of corresponding unknown 3D landmarks ${}_W\boldsymbol{l}_j$ in images $i = 1 \dots N$.



Find: 3D landmark positions ${}_W\boldsymbol{l}_j$ and 6D camera poses $[{}_W\mathbf{r}_{C_i}, \mathbf{q}_{WC_i}]^T$ where the images were taken.

Solution: minimise the reprojection error

$$\mathbf{e}_{ij}({}_W\mathbf{r}_{C_i}, \mathbf{q}_{WC_i}, {}_W\boldsymbol{l}_j) = \tilde{\mathbf{u}} - \mathbf{u}(\boldsymbol{T}_{C_iW}\ {}_W\boldsymbol{l}_j),$$

over all images

$$\min \sum_{i=1}^{N} \sum_{j \in \mathcal{J}(i)} \|\mathbf{e}_{ij}\|^2,$$

Set of landmark indices as visible in camera i.

applying Levenberg-Marquardt.

Jacobians needed for the Bundle Adjustment Problem

We will need **Jacobians** of $\mathbf{e}_{ij}({}_W\mathbf{r}_{C_i}, \mathbf{q}_{WC_i}, {}_W\mathbf{l}_j) = \tilde{\mathbf{u}} - \mathbf{u}(\boldsymbol{T}_{C_iW}\, {}_W\boldsymbol{l}_j)$.

Let's rewrite first: $\mathbf{e}_{ij} = \tilde{\mathbf{u}} - \mathbf{u}(\boldsymbol{T}_{WC_i}^{-1}\, {}_W\boldsymbol{l}_j) = \tilde{\mathbf{u}} - \mathbf{u}(\mathbf{C}_{WC_i}^T\, {}_W\mathbf{l}_j - \mathbf{C}_{WC_i}^T\, {}_W\mathbf{r}_{C_i})$.

We will employ the chain rule. So, we first need $\frac{\partial}{\partial {}_{C_i}\mathbf{l}}\mathbf{u}({}_{C_i}\mathbf{l}) = \frac{\partial}{\partial {}_{C_i}\mathbf{l}}\mathbf{k}(\mathbf{d}(\mathbf{p}({}_{C_i}\mathbf{l})))$.

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

Jacobian of distortion

Now:

$$\frac{\partial_C \mathbf{l}_j}{\partial_W \mathbf{r}_{C_i}} = -\overline{\mathbf{C}}_{WC_i}^T, \quad \frac{\partial_C \mathbf{l}_j}{\partial \delta\boldsymbol{\alpha}_i} = \overline{\mathbf{C}}_{WC_i}^T [{}_W\bar{\mathbf{l}}_j - {}_W\bar{\mathbf{r}}_{C_i}]^\times, \quad \frac{\partial_C \mathbf{l}_j}{\partial_W \mathbf{l}_j} = \overline{\mathbf{C}}_{WC_i}^T.$$

Now we have everything to build the Gauss-Newton System of equations:

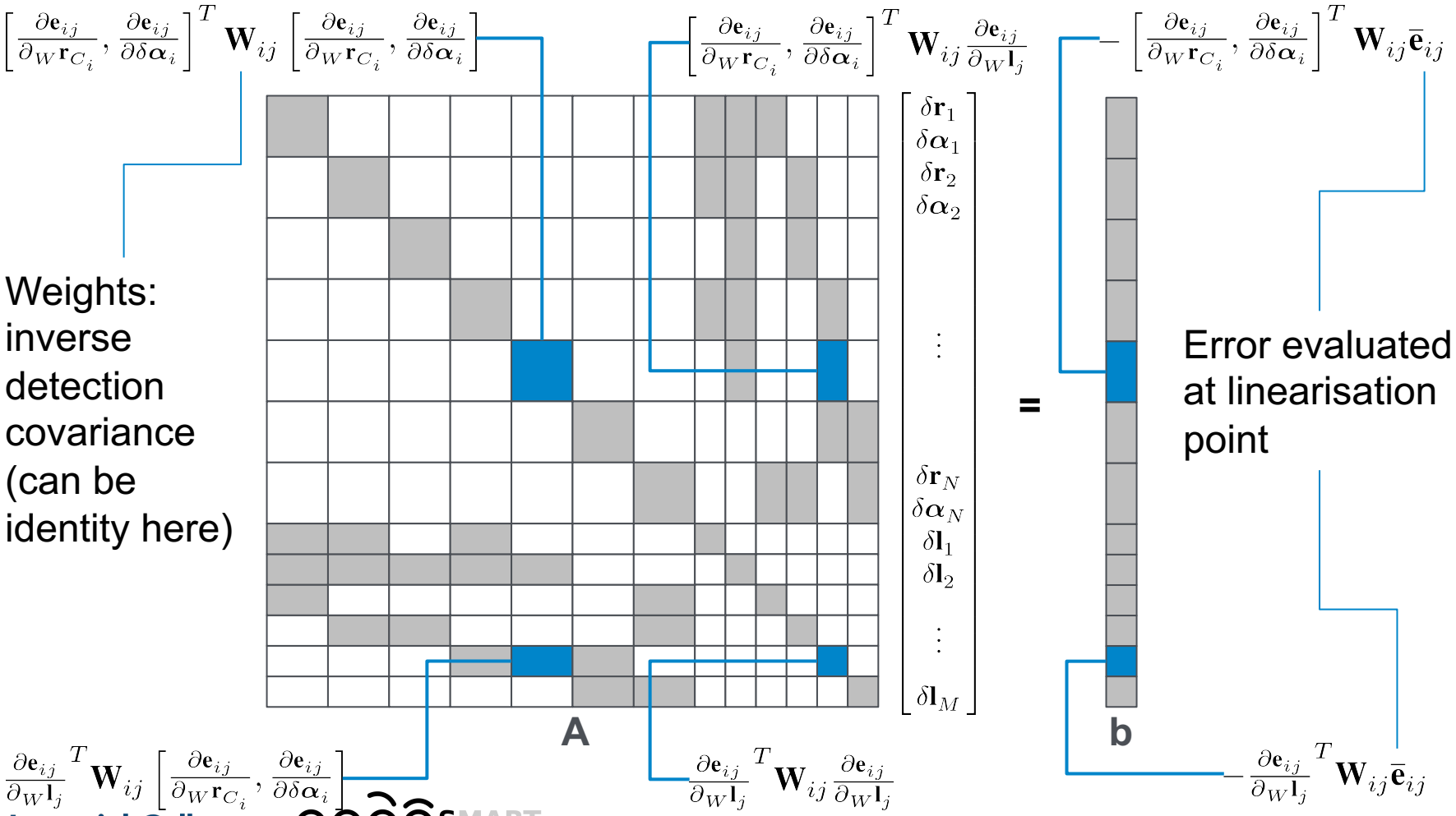
$$\underbrace{\sum_{ij} \mathbf{E}_{ij}(\bar{\mathbf{x}})^T \mathbf{R}_{ij}^{-1} \mathbf{E}_{ij}(\bar{\mathbf{x}})}_{=:\mathbf{A}} \Delta\mathbf{x} = \underbrace{-\sum_{ij} \mathbf{E}_{ij}(\bar{\mathbf{x}})^T \mathbf{R}_{ij}^{-1} \mathbf{e}_{ij}(\bar{\mathbf{x}})}_{=:\mathbf{b}},$$

$\mathbf{E}_{ij}(\bar{\mathbf{x}}) = \frac{\partial}{\partial \mathbf{x}}\, \mathbf{e}_{ij}(\mathbf{x})\big|_{\bar{\mathbf{x}}}$.

Jacobian of reproj. error w.r.t. all the variables (lots of zeros)

Structure of the Bundle Adjustment Problem

The **Gauss-Newton system of equations** smartly (additive per reproj. error):



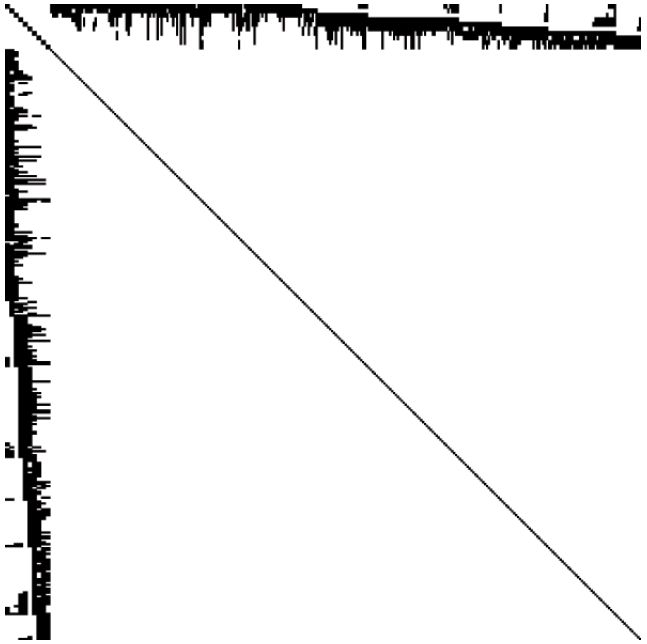
Imperial College London

SMART ROBOTICS LAB

Some Notes on Bundle Adjustment

- Typical problems contain many more points than poses
- Due to the sparsity of the linear equation system, it needs to be solved in a with some tricks, e.g.
  - Use Schur-Complement elimination of landmarks
  - Use sparse Cholesky composition
- In practise, **use an off-the-shelf solver** like
  - ceres: http://ceres-solver.org/
  - g2o: https://github.com/RainerKuemmerle/g2o

  This means, you will only have to implement the error function and Jacobians!



Example matrix A of a modestly sized problem. https://en.wikipedia.org/wiki/Bundle_adjustment

Batch Optimisation vs. Recursive Estimation (Filtering)

Batch Optimisation

- Estimate all states ever at once.
- No need for a temporal model, but can employ one.
- Enables re-linearization, leading to highest possible accuracy.
- However: when estimating a time-varying state, this means inserting variables for each time-step (i.e. the problem will grow with time)!

Recursive Estimation

- Estimate only the current state and maintain some distribution.
- Needs a temporal model for prediction, which is alternated with measurement updates.
- Constant state vector size, constant complexity.
- Re-linearization for old state is impossible (reduced accuracy).
- Delayed measurements cannot be included.

Any Questions?

See you on Thursday at 1pm for the practical