

Autonomous Mobile Robots

CHEAT SHEET BY JANIS HUTZ
ETHZ, FS2026

1 Introduction

1.1 Probability

Def Sum rule $\mathbb{P}(X) = \sum \mathbb{P}(X, Y) = \sum \mathbb{P}(X \cap Y)$

Def Prod $\mathbb{P}(X \cap Y) = \mathbb{P}(X|Y)\mathbb{P}(Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$

Thm Bayes $\mathbb{P}(Y_i|X) = \frac{\mathbb{P}(X|Y_i)\mathbb{P}(Y_i)}{\sum_{j=1}^n \mathbb{P}(X|Y_j)\mathbb{P}(Y_j)}$

Def Cont. Var Sums become integrals

e.g. $\sum_X \mathbb{P}(X) = 1$ becomes $\int \mathbb{P}(x) dx = 1$

Def Indep. x, y indep. iff $\mathbb{P}(X \cap Y) = \mathbb{P}(X)\mathbb{P}(Y)$

Def Cond. Indep. iff $\mathbb{P}(X \cap Y|Z) = \mathbb{P}(X|Z)\mathbb{P}(Y|Z)$

Def $\mathbb{E}[x] = \int_{-\infty}^{\infty} x\mathbb{P}(x) dx$, also for $x = f(x)$

Def Cov $[x] = \mathbb{E}[xx^T] - \mathbb{E}[x]\mathbb{E}[x]^T = \Sigma$

Def Gauss. Dist. $x \sim \mathcal{N}(\mu, \Sigma)$ (μ mean, Σ cov.),

PDF: $f(x) = \frac{1}{\sqrt{(2\pi)^k \det(\Sigma)}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right)$

Σ^{-1} for Σ diagonal, inverse of diag els (e.g. σ^{-1})

1.2 Measurement models

$z = b_C + sM_s\omega + b + n + o$: b_C const bias, b time bias, M missal., $n \sim \mathcal{N}(0, R)$ noise, $s\omega$ corr. meas., o other infl.

Finding: Is in W -frame: may need T_{BW} or R_{BW} . Also see Sec. 3

1.3 Trigonometry & Linear Algebra

Def Rule of cosines $c^2 = a^2 + b^2 - 2ab \cos(\gamma)$

Def Orthogonal vec $v^T w = 0$

Def Determinant $ad - bc$ for mat $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Def Vec Line Eq $p + \lambda d$

1.4 Error Propagation

For functions $f(x) = Ax$, the **linear error propagation** is given by $\Sigma^f = A\Sigma^x A^T$, with Σ^x the uncertainty of x (covariance mat.)

2 Locomotion & Kinematics

2.1 Positioning

Def Pos Vec. ${}^W t_B = {}^W t_{{}^W B}$; Relative to Frame, P . in other Frame, Start P . of vec in first F., $\sin = s$, $\cos = c$

Def State vector x_R : x, v of rob in W , pos of sensors

Def Rot. Mat. $R_z(\psi)$ (Yaw), $R_y(\theta)$ (Pitch), $R_x(\varphi)$ (Roll)

$$\begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}; \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}; \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & s(\varphi) & c(\varphi) \end{bmatrix}$$

Rem Application: $W a = R_{WBB} a$

Lem $R_{BW} = R_{WB}^{-1} = R_{WB}^T$, $\det(R_{WB}) = 1$ (orth.)

Rem Cols of R_{WB} are basis vec. of Frame $\mathcal{F}_B$ in $\mathcal{F}_W$

Def Euler Ang (Tait-Brian) $R_{EB} = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\varphi)$.

$$\begin{aligned} \psi &= \arcsin\left(\frac{R_{21}}{\sqrt{1-R_{31}^2}}\right) \\ \theta &= \arcsin(-R_{31}) \\ \varphi &= \arcsin\left(\frac{R_{31}}{\sqrt{1-R_{31}^2}}\right) \end{aligned} \quad [n]^\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

For pitch axis $= \pm 90$ deg, Gimbal Lock, Jacobian singular.

Def Angle-Axis $\alpha = \alpha n$ (n normal); Convert to rot. mat

$$R(\alpha, n) = I_3 + \sin(\alpha)[n]^\times + (1 - \cos(\alpha))([n]^\times)^2$$

To quat: $q = [n, \alpha]$

Def Quaternions $q = q_w + q_x i + q_y j + q_z k$ with $i^2 = j^2 = k^2 = -1$, $(ij = -ji = k, \text{ same for } jk \text{ and } ki)$.

$q = [v(q), a(q)]^T$, $a(q) = q_w$, Add like $\mathbb{C}$ (by "groups")

$$\text{Mult } q \otimes p = \begin{bmatrix} a(q)v(p) + a(p)v(q) \times v(p) \\ a(q)a(p) - v(q)^T v(p) \end{bmatrix}$$

To Rot Mat $R_{AB} = I_3 + 2a(q_{AB})[v(q_{AB})]^\times + 2([v(q_{AB})]^\times)^2$

Def Transf. M $T_{AC} = T_{AB}T_{BC}$ R-Handed typ; Aero: Left-H

$$T_{AB} = \begin{bmatrix} R_{AB} & A^T B \\ 0_{1 \times 3} & 1 \end{bmatrix}; T_{BA} = T_{AB}^{-1} = \begin{bmatrix} R_{AB}^T & -R_{AB}^T A^T B \\ 0_{1 \times 3} & 1 \end{bmatrix}$$

2.2 Forward Kinematics (FK)

$$T_{WBn}(\theta) = T_{WB_0}T_{B_0B_1}(\theta_1) \cdots T_{B_{n-1}B_n}(\theta_n).$$

$$\text{For 2R system: } {}_W t_{WE} = \begin{bmatrix} L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \end{bmatrix}$$

Workspace W : $\theta_1, \theta_2 \in [-\pi, \pi]$. Similar for n R sys (more angles, more lengths). For 2D move in 3D space, last dim is sum of angles (or equiv). **Jacobian**: see 4.1

Def Singularity Loss of deg of Freed. $\det(J(\theta)) = 0$

2.3 Inverse Kinematics (IK)

Option: Solve Forward Kinematics for angles.

Better: Law of cosine with polar coordinates. Compute angle using cosine rule, $\theta_1 = \varphi \pm \alpha$, $\theta_2 = \pm(\pi - \beta)$ (Positive for **Elbow Down**, Neg. for **Elbow Up**)

Extension to 6R: 1. Waist: spherical coords (2 sol.)

2. 2 sols from 2R for shoulder + elbow

3. Solve for wrist joints (no influence on pos)

2.4 Temporal Models

Model **robot dyn** as **Cont-time n-lin. system of ODE** $\dot{x} = f_C(x(t), u(t), w(t))$, with meas. $z(t) = h(x(t)) + v(t)$.

Linearize around $f_C(\bar{x}, \bar{u}) = 0$, at **equilibrium**:

$$\delta \dot{x}(t) = f_C(\bar{x}, \bar{u}) + F_C \delta x(t) + G_C \delta u(t) + L_C w(t)$$

$$\delta z(t) = H \delta x(t) + v(t). \quad H \text{ is meas., } F_C \text{ system, } G \text{ input gain,}$$

w process noise, v measurement noise, both zero-mean **Gaussian White Noise Process**.

To **discretize**, integrate from t_{k-1} to t_k :

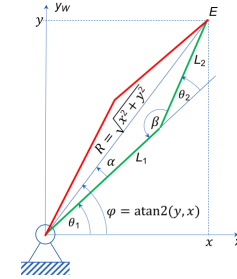
$$x_k = f(x_{k-1}, u_k, w_k); z_k = h(x_k) + v_k, \text{ linearised:}$$

$$\delta x_k = f(\bar{x}, \bar{u}) + F \delta x_{k-1} + G \delta u_k + L w_k; \delta z_k = H \delta x_k$$

Def Trapezoidal num. int $\Delta x_1 = \Delta t f_C(x_{k-1}, u_{k-1}, t_{k-1})$

$$\Delta x_2 = \Delta t f_C(x_{k-1} + \Delta x_1, u_k, t_k), \text{ then:}$$

$$x_k = x_{k-1} + 0.5 \cdot (\Delta x_1 + \Delta x_2)$$



2.5 Rigid body & IMU kinematics

Velocity ${}_I v_{IB} = {}_I \dot{t}_B$

Rot. Velocity ${}_I \omega_{IB} = \dot{\alpha} \cdot {}_I n$

Velocity point P ${}_B v_{IP} = {}_B v_{IB} +$

$${}_B \omega_{IB} \times {}_B t_{IP}$$

Rotation Matrices

- For left perturbing $\dot{R}_{IB} = [{}_I \omega_{IB}]^\times R_{IB}$
- For right perturbing $\dot{R}_{IB} = R_{IB} [{}_I \omega_{IB}]^\times$
- Constant angular velocity $(\exp[\Delta \alpha]^\times = \delta R(\Delta \alpha))$
 $R_{IB}(t + \Delta t) = \exp[\Delta \alpha]^\times R_{IB}(t) \quad \alpha = {}_I \omega_{IB} \delta t$

Quaternions

- For left perturbing $\dot{q}_{IB} = \frac{1}{2} \begin{bmatrix} {}_I \omega_{IB} \\ 0 \end{bmatrix} \otimes q_{IB}$
- For right perturbing $\dot{q}_{IB} = \frac{1}{2} q_{IB} \otimes \begin{bmatrix} {}_B \omega_{IB} \\ 0 \end{bmatrix}$

IMU (Outputs $s\tilde{a}$ (accel.), $s\tilde{\omega}$ (rot. accel.))

$${}_W \dot{t}_S = {}_W v; \quad \dot{q}_{WS} = \frac{1}{2} q_{WS} \otimes \begin{bmatrix} s\tilde{\omega} + w_g - b_g \\ 0 \end{bmatrix}$$

${}_W \dot{v} = R_{WS} (s\tilde{a} + w_a - b_a) + {}_W g$ where gray parts only IRL (in theor. models, leave out), with $\tilde{b}_g = w_{b_g}$ and $\tilde{b}_a = w_{b_a}$

IMU Sensor Model: $\tilde{z} = b_C + sMz + b + n + o$ where bias b and scale s often modelled time-varying $\dot{b}(t) = \sigma_C n(t)$. b_C const. calib; M Misalignment; n noise; o other infl.

2.6 Rigid Body Dynamics

Def **Newton II** For fin. body w/ mass m and inertia mat. I , with force F and torque T on **Centre of Mass** (CoM), expressed in body frame:

$${}_B F = \sum {}_B F_i = m({}_B \dot{v}_{CoM}) + m {}_B \omega \times {}_B v_{CoM}$$

$${}_B T = \sum {}_B T_i = I({}_B \dot{\omega}) + {}_B \omega \times I {}_B \omega$$

${}_B v_{CoM}$ vel. of CoM, ${}_B \omega$ rot. speed; both w.r.t. inert. frame

Legged robots don't need to maintain stability!

2.7 Wheeled robot Kinematics

Non-holonomic systems not integrable, no inst. move in every direct.

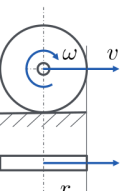
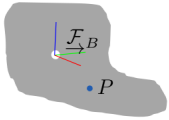
Wheel constraints $v_i = \omega_i r_i$ (r_i constraints)

- Driving straight** all v equal
- Turning** Wheel axis must intersect the **Instant Centre of Rotation** (ICR) of vehicle, speeds: $v_i \div R_i = \Omega$ ($R_i = \text{dist. wheel-ICR}$; Ω : vehicle rotation rate (around ICR))

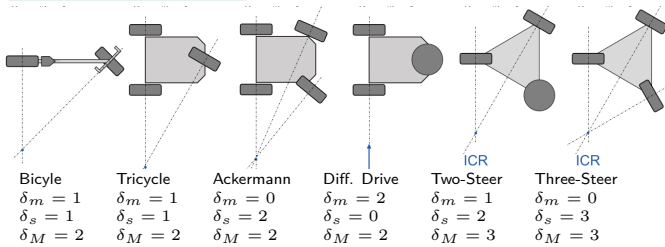
Below: α, l pos in frame, β rot at that pos (z -ax). To compute c in $c \cdot {}_B v_{WB} = \omega$ (For multiple wheels, construct mat. from this)
 $c_i = [\sin(\alpha + \beta) \div r, -\cos(\alpha + \beta) \div r, -\cos(\beta) \cdot l \div r]$

Maneuverability

- Deg. of Mobility: $\delta_m = 3 - \# \text{constrained directions}$
- Deg. of Steerability: $\delta_s = \# \text{steerable wheels}$
- Deg. of Maneuverability: $\delta_M = \delta_m + \delta_s$



Wheel Configurations



Def Differential Drive Kinematics

State vec $\mathbf{x} = [x_1, x_2, \theta]^\top$, **Inputs** $\mathbf{u} = [\omega_l, \omega_r]^\top$, radius of right (left) wheel r_r (r_l), w width of robot

Gen. eq. of Motion $\dot{x}_1 = v \cos(\theta)$, $\dot{x}_2 = v \sin(\theta)$, $\dot{\theta} = \Omega$, with

$$v = 0.5 \cdot (\omega_l r_l + \omega_r r_r), \Omega = \frac{\omega_r r_r - \omega_l r_l}{w}$$

Straight: $v = \omega_l r_l = \omega_r r_r$, $\Omega = 0$, $D = v \Delta t$.

$$\mathbf{b}_s = \begin{bmatrix} D \cos(\theta) \\ D \sin(\theta) \\ 0 \end{bmatrix}, \mathbf{b}_t = \begin{bmatrix} R(\sin(\Delta\theta + \theta) - \sin(\theta)) \\ -R(\cos(\Delta\theta + \theta) - \cos(\theta)) \\ \Delta\theta \end{bmatrix}$$

Turning: $\Omega = (\omega_l r_l) / R_l = (\omega_r r_r) / R_r$, $R = v / \Omega$, $\Delta\theta = \Omega \Delta t$

Discretized: $\mathbf{x}_k = \mathbf{x}_{k-1} + \mathbf{b}_i$ with $i \in \{s, t\}$, respectively

3 Sensors & Actuators

Meas. Model: $\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{v} + \mathbf{o}$, with $\mathbf{h}(\mathbf{x})$ determ. mean, $\mathbf{v}$ zero-mean noise, $\mathbf{o}$ unmodelled effects, $\mathbf{x}$ true state. $\mathbf{h}(\mathbf{x})$ describes how to compute $\mathbf{x}$ from known values (e.g. Intercept-Theorem). Probabilistic M.M: use rand var in def, and s. 1.4

Motor encoders Typ. 64-2048 incrm. per rev; Estim. rot

Rolling-Shutter Most CMOS sensors don't take full image at once, need time stamp for each row

3.1 GNSS

Need ultra-precise time sync ($c \approx 0.3 \text{ m/ns}$). **Errors**

- Multipath problem (signal bounce) (0.5 - 100m)
- Ionosphere delays (10m)
- Satellite pos. err, trop. delay (1m)

3.2 Actuators

Hydraulic acc., easy control, power; maint., speed, price

Pneumatic price, shock abs., speed; acc., loud, maint.

3.2.1 DC Motor

(Kirchoff) $U_a = L_a \dot{I}_a + R_a I_a + U_i$

(Torque, Lorenz Force) $T = k_T I_a$

(Induced V, Faraday) $U_i = k_i \omega$

(Mech. pow. = electric power)

$$U_i I_a = k_i \omega I_a = T \omega = k_T I_a \omega \Rightarrow k_i = k_T =: k$$

3.3 Cameras

Def Pinhole projection $\begin{bmatrix} u & v \end{bmatrix}^\top = \frac{f}{z} \begin{bmatrix} x & y \end{bmatrix}^\top$ with f the distance to the lens and z the distance from object

$u = c_u + f \cdot x'$ and $v = c_v + f \cdot y'$ where $x' = t_x \div t_z$ and $y' = t_y \div t_z$ where u, v are the pixel x, y coords, $\mathbf{c} = [c_u, c_v]^\top$ is optical centre of cam in pixel coords, f scale factor.

$$\text{The full proj: } \mathbf{u} = \begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \mathbf{K} \mathbf{c} \mathbf{t}_P$$

If p. in diff frame, e.g. W -frame then $\mathbf{u} = \mathbf{K}[\mathbf{R}_{CW} \mathbf{c} \mathbf{t}_{CW}]_W \mathbf{t}_P$

3.3.1 Pinhole Camera Projection with distortion

Def Model: $\mathbf{u} = \mathbf{k}(d(\mathbf{p}(\mathbf{c} \mathbf{t}_P)))$, with ($\mathbf{c}$ as above):

(Projection) $\mathbf{x}' = \mathbf{p}(\mathbf{c} \mathbf{t}_P) = t_z^{-1} \cdot [t_x, t_y]^\top$

(Distortion model) $r^2 = x'^2 + y'^2$, $\mathbf{x}'' = d(\mathbf{x}')$

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

(Scale and Centre) $\mathbf{u} = \mathbf{k}(\mathbf{x}'') = \text{diag}([f_u, f_v]) \cdot \mathbf{x}'' + \mathbf{c}$

All with k_i radial distortion params, optional for $i > 2$, p_i tang. dist. param, f_u, f_v focal length in pixels

Inverse $\mathbf{c} \mathbf{r} = [d^{-1}(\mathbf{k}^{-1}(\mathbf{u})), 1]^\top$

(To unit plane) $\mathbf{x}'' = \mathbf{k}^{-1}(\mathbf{u}) = [f_u^{-1}, f_v^{-1}]^\top (\mathbf{u} - \mathbf{c})$

(Un-distort) $\mathbf{x}' = d^{-1}(\mathbf{x}'')$ (usually comp. numerically)

(Compute ray) $\mathbf{c} \mathbf{r} = [\mathbf{x}', 1]^\top$

3.3.2 Undistorting a whole image

$\mathbf{u}_i = \mathbf{k}(d(\mathbf{k}_{\text{new}}^{-1}(\mathbf{u}_{i, \text{new}})))$ where $\mathbf{u}_{i, \text{new}}$ is the place of pixel in output, $\mathbf{u}_i$ is the input

Omnidir. Cam undistortion model with $f(u, v) = \sum_{i=0}^N a_i \rho^i$ with $\rho = \sqrt{(u - c_u)^2 + (v - c_v)^2}$, $N = 4$ accurately describes it for most fisheye and catadioptric cameras

3.4 Depth and Range sensing

Typ. derive D.M. with sensor dir param (as vec line eq)

3.4.1 Triangulation-based

Struct. Light Single cam, single projector: **Spatial acc, no worky in bright light, interference with other IR depth cams**

Active Stereo 2 cams, 1 proj: **worky in bright light, need stereo matching, less accurate, error grows with distance**

3.4.2 Classic Stereo

Both images: same plane, focal length, centre, x -axis. Given corresponding pixels $[u_l, v]$ and $[u_r, v]$, $z = \frac{b \cdot f}{u_r - u_l}$ with $u_l = f \cdot \frac{x}{z} + c_u$ and $u_r = f \cdot \frac{x-b}{z} + c_u$

3.4.3 Time of Flight, Projection

No occlusions/shadows, Interference with other dev, multipath leading to larger distances sensed

$$\text{Proj. } \mathbf{z} = \begin{bmatrix} u \\ d \end{bmatrix} = \begin{bmatrix} \mathbf{k}(d(\mathbf{p}(\mathbf{c} \mathbf{t}_P))) \\ [0, 0, 1]_{\mathbf{c} \mathbf{t}_P} \end{bmatrix} \text{ Back: } \mathbf{c} \mathbf{t}_P = \begin{bmatrix} d \mathbf{x}' \\ d \end{bmatrix}$$

3.4.4 Range Sensors

Ultrasonic Typ. freq: 40kHz - 180kHz, Range: 12cm - 5m, Acc: $\approx 2\text{cm}$, rel error $\approx 2\%$ **meas. for transp. surf., cheap(ish), Cone wider, reflect. angle dep, wind / currents**

LiDAR Time-of-Flight-based, **Accuracy, range, works in bright light, Complex, expensive, one timestamp per measurement**. Typical Ranges: up to 100m

4 Multi-Sensor Estimation

4.1 Linearization

$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\bar{\mathbf{x}}) + \mathbf{J}_f|_{\mathbf{x}=\bar{\mathbf{x}}}(\mathbf{x} - \bar{\mathbf{x}})$, f' , no vec in 1D; $\bar{\mathbf{x}}$ lin. p.

Def Jac. $\mathbf{J}_f$ rows for eq of $\mathbf{f}$; cols for vars of each eq.

Part. diff; Approx. using finite differences $\frac{f(\bar{\mathbf{x}} + h) - f(\bar{\mathbf{x}})}{h}$, or central differences (vector of $\frac{f(\bar{\mathbf{x}} + h_i \mathbf{e}_i) - f(\bar{\mathbf{x}})}{h_i}$, with $\mathbf{e}_i$ unit vec)

4.2 Linear Least Squares

Goal: $\text{argmin}_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A} \mathbf{x} - \mathbf{b}\|_2^2$, $\mathbf{A}$: rows i -th datap. col c : t_i^{c-1} .

Man. sol.: comp. $\mathbf{M} = \mathbf{A}^\top \mathbf{A}$, $\mathbf{b}' = \mathbf{A}^\top \mathbf{b}$, then $\mathbf{M} \mathbf{x} = \mathbf{b}'$.

Prob. sol.: $\text{argmax} \mathbb{P}(\mathbf{x} | \mathbf{z})$ with Maximum ...

- **Likelihood** $\mathbb{P}(\mathbf{x} | \mathbf{z}) \propto \mathbb{P}(\mathbf{z} | \mathbf{x}) = \prod_{i=1}^N \mathbb{P}(z_i | \mathbf{x})$
- **a Post** $\mathbb{P}(\mathbf{x} | \mathbf{z}) \propto \mathbb{P}(\mathbf{z} | \mathbf{x}) \mathbb{P}(\mathbf{x}) = \mathbb{P}(\mathbf{x}) \prod_{i=1}^N \mathbb{P}(z_i | \mathbf{x})$

4.3 Non-Linear Least Squares

Find $\mathbf{x}^* = \text{argmax} \mathbb{P}(\mathbf{x} | \mathbf{z}) = \text{argmin}(-\log(\mathbb{P}(\mathbf{x} | \mathbf{z})))$

Gauss-Newton

Levenberg-Marquardt

Local Param.

4.4 Bayes Filter (in DAG)

$\mathbf{x}_k^R$ state at time k, $\mathbf{z}_k^p$ dist. meas., $\mathbf{u}_k^p$ wheel odometry (= meas.). Typ. care ab. curr. state: altern. pred. & update.

4.5 Particle Filter

Is a bayes filter approximating the state distribution with a set of random samples. Update step:

- Apply Bayes rule $w'_{k,s} = \mathbb{P}[z_k | \mathbf{x}_{k,s}] w_{k-1,s}$
- Renormalize: $w_{k,s} = w'_{k,s} \div \sum_s w'_{k,s}$
- Resample: rand. sel. S particles acc. to weights and $w_{k,s} = S^{-1}$

4.6 Kalman Filtear (KF)

Bayes Filter for Gauss. dist of R.V. & linear meas. model. Initial state $\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}, \mathbf{P}_0)$, $\mathbf{P}_0$ previous covariance;

Prediction With linear state transition model ($\mathbf{u}_k$ odometry, $\mathbf{w}_k$ noise (covariance $\mathbf{Q}_k$)):

$$\mathbf{x}_k = \mathbf{F} \mathbf{x}_{k-1} + \mathbf{G} \mathbf{u}_k + \mathbf{L} \mathbf{w}_k \text{ with } \mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k):$$

- **Mean** $\hat{\mathbf{x}}_{k|k-1} = \mathbf{F} \hat{\mathbf{x}}_{k-1} + \mathbf{G} \mathbf{u}_k$
- **Covariance** $\mathbf{P}_{k|k-1} = \mathbf{F} \mathbf{P}_{k-1|k-1} \mathbf{F}^\top + \mathbf{L} \mathbf{Q}_k \mathbf{L}^\top$

Update Lin. meas.: $\tilde{\mathbf{z}}_k = \mathbf{H} \mathbf{x}_k + \mathbf{v}_k$ with $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$:

- **Meas. residual:** $\mathbf{y}_k = \tilde{\mathbf{z}}_k - \mathbf{H} \hat{\mathbf{x}}_{k|k-1}$
- **Resid. Cov:** $\mathbf{S}_k = \mathbf{H} \mathbf{P}_{k|k-1} \mathbf{H}^\top + \mathbf{R}_k$
- **Kalman gain:** $\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}^\top \mathbf{S}_k^{-1}$
- **Updated mean:** $\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k$
- **Updated Cov.:** $\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}) \mathbf{P}_{k|k-1}$

4.7 Extended Kalman Filater (EKF)

Non-l. state trans. model $\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k)$ as above:

- **Mean:** $\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k)$
- **Cov.:** $\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{L}_k \mathbf{Q}_k \mathbf{L}_k^\top$

With $\mathbf{F}_k$ linearisation $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ and $\mathbf{L}_k$ lin. $\frac{\partial \mathbf{f}}{\partial \mathbf{w}}$

Update N-Lin. meas.: $\tilde{\mathbf{z}}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k$:

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

Difference to above: $\mathbf{H}$ becomes $\mathbf{H}_k$, and $\mathbf{H}^\top$ is $\mathbf{H}_k^\top$

5 SLAM to Spatial AI

5.1 Keypoints

Corner det. $SSD(\Delta x, \Delta y) \approx [\Delta x \ \Delta y] \mathbf{M} [\Delta x \ \Delta y]^\top$ with $\mathbf{M} = \mathbf{R}^\top \text{diag}(\lambda_1, \lambda_2) \mathbf{R}$; λ_i E.V. of $\mathbf{M}$; κ const 0.04-0.15;

$$R = \det(M) - \kappa \cdot \text{TR}(M)^2 = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2;$$

$$M = \sum_{x,y \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Blob Detection (I is the image)

Laplacian of Gaussian (LoG): $L = g(x, y, t) \cdot I(x, y)$. Then apply

$$\text{Laplacian Operator } \nabla_{\text{norm}}^2 L = t \left(\frac{\partial^2 L}{\partial x^2} + \frac{\partial L}{\partial y^2} \right)$$

Diff. of Gaussians (DoG): $\Delta L = L(x, y, t) - L(x, y, kt)$

SIFT Detector (1) Subsample + Blur **(2)** DoG on each res. image **(3)** Keypoints extrema in DoG pyramid

5.2 Bootstrapping

PnP Problem Persp. n -P. Find sol. for camera pose *directly*

RANSAC RANdom SAMpling Consensus for find. outliers & correct

Stereo Triang. Given two rays (known poses for points in 2D).

Find good point in 3D. Fast sol: **Midpoint Method**:

1 Find p. along ray w/ min. dist (Lin. Least Squares)

$$\lambda = [\lambda_1 \lambda_2]^T = \text{argmin} \| (w \mathbf{t}_{C_2} + \lambda_2 w \mathbf{e}_2) - (w \mathbf{t}_{C_1} + \lambda_1 w \mathbf{e}_2) \|^2$$

2 Solve normal equation $A\lambda = \mathbf{b}$ with $\mathbf{q} = -w \mathbf{e}_1^T w \mathbf{e}_2$:

$$A = \begin{bmatrix} 1 & q \\ q & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \mathbf{e}_1^T \cdot (w \mathbf{t}_{C_2} - w \mathbf{t}_{C_1}) \\ -\mathbf{e}_2^T \cdot (w \mathbf{t}_{C_2} - w \mathbf{t}_{C_1}) \end{bmatrix} \quad C_i \text{ cam}$$

3 Pick midp. $w \mathbf{t}_P = 0.5(\tau_1 + \tau_2)$; $\tau_n = w \mathbf{t}_{C_n} + \lambda_n w \mathbf{e}_n$

5.3 Place Recognition

Idea: Build vocab from “visual words” (in Training). **Runtime**

Detect keypoints; Extract descriptors; Build histogram; Query DB for similarity; If no match, insert into DB, else use to compute with e.g. RANSAC

5.4 Mapping

Problem Formulations Localisation (always static given map):

$\mathbf{x}_{R,k}^* = \text{argmax } \mathbb{P}(\mathbf{x}_{R,k} \mid \mathbf{x}_M, \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$ (recursive)

$\mathbf{x}_{R,1:k}^* = \text{argmax } \mathbb{P}(\mathbf{x}_{R,1:k} \mid \mathbf{x}_M, \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$ (batch)

SLAM $\{\mathbf{x}_{R,k}^*, \mathbf{x}_M^*\} = \text{argmax } \mathbb{P}(\mathbf{x}_R, \mathbf{x}_M \mid \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$ (as $\uparrow$)

Mapp.: $\mathbf{x}_M^* = \text{argmax } \mathbb{P}(\mathbf{x}_M \mid \mathbf{x}_{R,1:k}^*, \mathbf{z}_{1:k}, \mathbf{u}_{1:k})$ with given poses $\mathbf{x}_{R,1:k}^*$. Thus temp. model $\mathbf{u}_{1:k}$ doesn't matter.

Prob. Occ. Grid $\mathbb{P}(f_j) = \mathbb{P}(\neg o_j) = 1 - \mathbb{P}(o_j)$, with $\mathbb{P}(o_j)$ prob. cell j occupied; pairwise independent.

Occ. Map. w/ depth sensor $\mathbb{P}(o_j \mid \mathbf{x}_{R,1:k}) =$

$$\frac{\mathbb{P}(o_j \mid \mathbf{x}_{R,k}, \mathbf{z}_k) \mathbb{P}(\mathbf{z}_k \mid \mathbf{x}_{R,k}) \mathbb{P}(o_j \mid \mathbf{x}_{R,1:k-1}, \mathbf{z}_{1:k-1})}{\mathbb{P}(o_j) \mathbb{P}(\mathbf{z}_k \mid \mathbf{x}_{R,1:k}, \mathbf{z}_{1:k-1})}$$

map prior; Prev. occ. est.; Occ. based on curr. range meas.;

Update function: $l(a) := \log(\text{Odds}(a))$ with $\text{Odds}(a) = \frac{\mathbb{P}(a)}{1 - \mathbb{P}(a)}$ (inv. sensor model): $l(o_j \mid \mathbf{x}_{R,1:k}, \mathbf{z}_{1:k}) =$

$$l(o_j \mid \mathbf{x}_{R,k}, \mathbf{z}_k) + l(o_j \mid \mathbf{x}_{R,1:k-1}, \mathbf{z}_{1:k-1}) - l(o_j)$$

In 3D 3D voxel j as signed dist. s and weight w , update: $s_k =$

$$\frac{w_{k-1} s_{k-1} + \tilde{s}_k}{w_{k-1} + 1} \text{ with } w_k = \min(w_{\max}, w_{k-1} + 1)$$

Impl. Using HashTables or octree

5.5 Dense Tracking and Loop Closure

Idea: Min. sum of sq. errors over all pixels w/ Gauss-Newton.

6 Planning & Control

6.1 SISO & MIMO

Single-Input-Single-Output: r Reference (e.g. speed), u Sys. Input (e.g. gas pedal pos), z Sys. Out. (e.g. speed of car)

Multiple-Input-Multiple-Output: r Reference (typ: trajectory), u Sys. Input (e.g. 4 rotor speeds), x internal states (pos, orient, speed, rot. speed), z Sys. Output (e.g. speed of car)

6.2 Proportional-Integral-Differential (PID)

$u(t) = k_p e(t) + k_i \int_{t_0}^t e(\tau) d\tau + k_d \frac{de(t)}{dt}$, where params k_p (curr), k_i (long-term), k_d (trend) reduce corresp. errors

Drone Control ${}_B \mathbf{F}$ and ${}_B \mathbf{M}$ as in sect. 2.6, use $\mathbf{u}' = {}_B \mathbf{M}$.

6.3 Linear Quadratic Regulator (LQR)

For lin. cont.-time dyn. $\dot{\mathbf{x}}(t) = \mathbf{F}_c \mathbf{x}(t) + \mathbf{G}_c \mathbf{u}(t)$.

We try to minimise cost functional (for $\mathbf{u}(t) = -\mathbf{K} \mathbf{x}(t)$)

$$J = \int_t^\infty \mathbf{x}(\tau)^\top \mathbf{Q} \mathbf{x}(\tau) + \mathbf{u}(\tau)^\top \mathbf{R} \mathbf{u}(\tau) d\tau$$

Solution: $\mathbf{K} = \mathbf{R}^{-1}(\mathbf{G}_c^\top \mathbf{P})$, with $\mathbf{P}$ found from

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

Finding $\mathbf{K}$ is expensive, but *offline*, at runtime only $\mathbf{u}(t)$.

Non-Lin: Approx, $\delta \dot{\mathbf{x}}(t) = \mathbf{F}_c \delta \mathbf{x}(t) + \mathbf{G}_c \delta \mathbf{u}(t)$

6.4 MPC

Cost function ($p(\mathbf{x}_N)$ terminal cost, sum the stage cost)

$$J_{0 \rightarrow N}(\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{u}_{N-1}) = p(\mathbf{x}_N) + \sum_{k=0}^{N-1} q(\mathbf{x}_k, \mathbf{u}_k)$$

We minimize the above s.t. for $k \in \{0, \dots, N-1\}$ we have

$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, $g(\mathbf{x}_k, \mathbf{u}_k) \leq 0$ and $\mathbf{x}_N \in \mathcal{X}_f$ and $\mathbf{x}_0 = \mathbf{x}(0)$

Finite-Horizon Lin-Quad Control Quad. Cost:

$$J_{0 \rightarrow N}(\mathbf{x}_0, \mathbf{u}_0, \dots) = \mathbf{x}_N^\top \mathbf{P} \mathbf{x}_N + \sum_{k=0}^{N-1} \mathbf{x}_k^\top \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^\top \mathbf{R} \mathbf{u}_k$$

without constraints, **State Feedback Law** $\mathbf{u}_0^* = -\mathbf{K} \mathbf{x}(0)$. With constraints, minimize as above, $\mathbf{x}_{k+1} = \mathbf{F} \mathbf{x}_k + \mathbf{G} \mathbf{u}_k$

6.5 Motion Planning

Config. Space: $\mathcal{C}$. Subset of robot states for planning.

Free C-Space: $\mathcal{C}_{\text{free}}$, **C-Space Obstacles** $\mathcal{C}_{\text{obst}}$ (occupied)

Collision checker: $c(x) : \mathcal{C} \rightarrow \{0, 1\}$

Visibility graph: Connect corners, goal outside obstacles

Voronoi Diagram: Edges at max. dist. from obst. (benefit: safer paths). Also tends to be faster than Dijkstra.

6.5.1 A* Algorithm

Function $h(\dots)$ is a lower bound of optimal cost.

```

1 procedure ASTAR(Graph, start, goal)
2   for each v in Graph.Verticies do
3     dist[v] ← ∞
4     totDistEst[v] ← ∞
5   prev[v] ← undef
6   insert start into openSet
7   dist[start] ← 0
8   totDistEst[start] ← h(start, goal)
9   while openSet not empty do
10    u ← vert in openSet w/ min totDistEst
11    remove u from openSet
12    if u == goal then
13      return dist[u], prev
14    for each neighbour v of u do
15      alt ← dist[u] + G.Edges(u, v).dist
16      if alt < dist[v] then
17        dist[v] ← alt
18        totDistEst[v] ← alt + h(v, goal)
19        insert v into openSet
20        prev[v] ← u
21  return ∞, prev

```

6.5.2 Rapidly-Exploring Random Tree (RRT)

```

1 procedure RRT(start, goal)
2   INSERTVERTEX(start, Graph)
3   for k < MAX_ITER do
4     s ← scal. unif. rand. sample on [0, 1]
5     if s < GOAL_CONNECT_PROB then
6       x ← goal
7     else
8       x ← random coll.-free config
9       x_n ← NEARESTCONFIGURATION(Graph, x)
10      x_f ← STOPPINGCONFIGURATION(x_n, x)
11      if x_f ≠ x_n then
12        insertVertexGraph, x_f
13        ▷ Extension of RRT* goes here
14        insertEdgeGraph, x_n, x_f
15      if x_f == x_n then
16        return SUCCESS, Graph
17  return FAILURE, Graph

```

Returns a collision-free path as graph. Need nearest neighbour search. Extension to RRT* to make path better:

Informed RRT* extension: Once conn. betw. start and goal found, restrict sampling to (hyper)ellipsoid.

6.5.3 Exploration

Find action sequence by finding goals, planning collision-free paths there, choose goal/path with maximum utility (typically max map information)

6.5.4 Collision Avoidance

Dynamic Window Approach Assume: Robot moves inst. on circ. arcs (v, ω) . Compute arcs with coll. Accounts for Kino-Dyn, Cost func prone to loc. min, assumes static obj

```

1  $X_{\text{near}} \leftarrow \text{NEIGHBOURS}(\text{Graph}, x_f, R)$ 
2  $x_{\text{min}} \leftarrow \text{NEIGHBOURS}(\text{Graph}, x_f, R)$ 
3  $c_{\text{min}} \leftarrow \text{COST}(x_n) + \text{EDGE\_COST}(x_n, x_f)$ 
4 for each  $x_{\text{near}}$  in  $X_{\text{near}}$  do
5   if  $\text{EDGE\_COLLISION\_FREE}(x_{\text{near}}, x_f)$  and  $\text{COST}(x_{\text{near}}) +$ 
    $\text{EDGE\_COST}(x_{\text{near}}, x_f) < c_{\text{min}}$  then
6      $\text{INSERT\_EDGE}(\text{Graph}, x_f, x_{\text{near}})$ 
7      $x_{\text{parent}} \leftarrow \text{PARENT}(\text{Graph}, x_{\text{near}})$ 
8      $\text{REMOVE\_EDGE}(\text{Graph}, x_{\text{parent}}, x_{\text{near}})$ 

```

Vel. Obst. Assume: Robot in str. line (v_x, v_y) . Comp pos with coll. **Vel. of obj, prone to local optima, no kino-dyn.**

Potential Field Methods Define *repulsive* and *attractive* potential $c = c_{\text{att}} + c_{\text{rep}}$. With e.g. $c_{\text{att}} = \frac{1}{2} k_{\text{att}} \|\mathbf{x} - \mathbf{x}_{\text{goal}}\|^2$ and

$$c_{\text{rep}} = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{\rho(\mathbf{x})} + \frac{1}{\rho_{\text{lim}}} \right) & \rho \leq \rho_{\text{lim}} \\ 0 & \text{else} \end{cases}$$

Simple control laws, may trap in loc. min., no diff. const, no guar. to avoid coll

6.6 Learning To Act

Used if there is no state-transition model or cost/reward func.

6.6.1 Markov Decision Process

The goal is to maximize the reward. Along the route, get small reward, at the end large reward (good or bad).

Def. by states $\mathbf{x} \in \mathcal{X}$ (RL: s), actions $\mathbf{u} \in \mathcal{U}$ (RL: a), prob. state trans. $\mathcal{T}(\mathbf{x}, \mathbf{u}, \mathbf{x}_+) = \mathbb{P}(\mathbf{x}_+ | \mathbf{x}, \mathbf{u})$, reward func $\mathcal{R}(\mathbf{x}, \mathbf{u}, \mathbf{x}_+)$, start state $\mathbf{x}_0$, optional terminal state $\mathbf{x}_N$.

Utility Func Expected reward: $V = \sum_{k=0}^N r_k$ or *discounted* reward $V = \sum_{k=0}^{\infty} \gamma^k r_k$ with $\gamma < 1$ (if inf. runtime)

Solving (Val iter) $V_0(\mathbf{x}) = 0$ and $V_{i+1}(\mathbf{x}) = \max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$ with

$$Q(\mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}_+} \mathbb{P}(\mathbf{x}_+ | \mathbf{x}, \mathbf{u}) [\mathcal{R}(\mathbf{x}, \mathbf{u}, \mathbf{x}_+) + \gamma V_i(\mathbf{x}_+)]$$

Repeat until conv. to V^* ($\mathcal{O}(|\mathcal{U}||\mathcal{X}|^2)$ per iter). Optimal policy:

$$\pi^*(\mathbf{x}) = \operatorname{argmax}_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$$

Using policy iter:

```

1 Choose  $\pi_0(\mathbf{x})$ 
2 while policy has not converged do
3   repeat  $V_{i+1}^{\pi_j}(\mathbf{x}) = Q(\mathbf{x}, \pi(\mathbf{x})) \forall \mathbf{x}$  and fixed pol.  $\pi_j$ 
4   until values converge
5 One step:  $\pi_{j+1}(\mathbf{x}) = \operatorname{argmax}_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$  with  $V_i = V_{i+1}^{\pi_j}$ 

```

Model-based learning uses empirical models of $\mathcal{T}$ and $\mathcal{R}$

6.6.2 Reinforcement Learning

Passive Direct Evaluation Act according to policy π , store sum of discounted rewards, average them. (But too simple)

Sample-Based Use $V_{i+1}^{\pi}(\mathbf{x}) = Q(\mathbf{x}, \pi(\mathbf{x}))$ w/ $V_0^{\pi}(\mathbf{x}) = 0$. We need state trans. model, instead $\tilde{R}_j$ (approx. prob. w/ statistics) and thus

$$V_{i+1}^{\pi}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^N \tilde{R}_j(\mathbf{x}, \pi(\mathbf{x}), \mathbf{x}_+) + \gamma V_i^{\pi}(\mathbf{x}_+)$$

Active Find optimal policy π instead of state values $V(\mathbf{x})$.

Q-Learn. Here: restate Val. Iter in Q -Values ($Q_0(\mathbf{x}, \mathbf{u}) = 0$):

$$Q_{i+1}(\mathbf{x}, \mathbf{u}) = Q(\mathbf{x}, \mathbf{u}) \quad \text{with} \quad V_i(\mathbf{x}_+) = \max_{\mathbf{u}} (Q_i(\mathbf{x}_+, \mathbf{u}))$$

Compute using sample:

$$\tilde{Q}_i(\mathbf{x}, \mathbf{u}) = \tilde{R}_i(\mathbf{x}, \mathbf{u}, \mathbf{x}_+) + \gamma \max_{\mathbf{u}} (Q_i(\mathbf{x}_+, \mathbf{u}))$$

Then update: $Q_{i+1}(\mathbf{x}, \mathbf{u}) = (1 - \alpha)Q_i(\mathbf{x}, \mathbf{u}) + \alpha \tilde{Q}_i(\mathbf{x}, \mathbf{u})$. Called off-policy learning, needs exploration. Simplest is random actions (ε -greedy): ε is prob. to act randomly, $1 - \varepsilon$ is prob. to act on pol. **Space explored, still doing random stuff**

Approaches *Model-based* (estimate trans. model, e.g. Dyna), Value-based (estimate val or Q -func and extract pol., e.g. Q-Learn), Actor-Critic (estim. val or Q of curr. pol., improve pol., e.g. A3C, SAC), Policy-Gradient (diff. expect. reward w.r.t. params of policy network, e.g. REINFORCE)