

Autonomous Mobile Robots

CHEAT SHEET BY JANIS HUTZ
ETHZ, FS2026

1 Introduction

1.1 Probability

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

Def Prod $P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)$

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

Def Cont. Var Sums become integrals

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

Def Indep. x, y indep. iff $p(x, y) = p(x)p(y)$

Def Cond. Indep. iff $p(x, y|z) = p(x|z)p(y|z)$

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

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

Def Gauss. Dist. $x \sim \mathcal{N}(\mu, \Sigma)$ (μ mean, Σ cov.),
PDF: $p(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu))$

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.

2 Locomotion & Kinematics

2.1 Positioning

Def Pos Vec. ${}^W t_B = {}^W t_W {}^W B$, 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: $W a = R_{WB} a$

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) $R_{EB} = R_z(\psi) \cdot R_y(\theta) \cdot R_x(\varphi)$.

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

Def Rot. Vec $\alpha = \alpha n$ (n normal); Convert to rot. mat
 $R(\alpha, n) = I_3 + \sin(\alpha)[n]^\times + (1 - \cos(\alpha))([n]^\times)^2$

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")

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)

$T_{WB_n}(\theta) = T_{WB_0} T_{B_0 B_1}(\theta_1) \cdots T_{B_{n-1} B_n}(\theta_n)$.

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

With workspace (pos) W for $\theta_1, \theta_2 \in [-\pi, \pi]$

2.3 Inverse Kinematics (IK)

Option: Solve Forward Kinematics for angles.

Better: Law of cosine with polar coordinates. Compute angle using cosine rule,

$\theta_1 = \phi \pm \alpha$, $\theta_2 = \pm(\pi - \beta)$

(Positive for Elbow Down, Negative 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

For **Cont-time n.-lin. system of ODE** $\dot{x} = f_C(x(t), u(t))$, with measurements $z(t) = h(x(t)) + v(t)$.

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

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

$\delta z(t) = H \delta x(t) + v(t)$. Herein, H is measurements, F_C system, G input gain, w process noise, v measurement noise, both zero-mean **Gaussian White Noise Process**.

For **n.-lin. cont-time system**: $\dot{x}(t) = f_C(x(t), u(t), w(t))$

$z(t) = h(x(t)) = v(t)$, linearization is the same

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

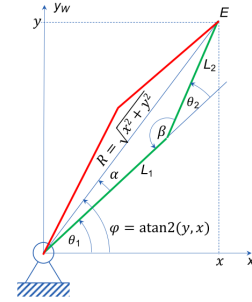
$x_k = f(x_{k-1}, u_k, w_k)$ $z_k = h(x_k) + v_k$, **linearised**:

$\delta x_k = f(\bar{x}, \bar{u}) + F_C \delta x_{k-1} + G_C \delta u_k + L_C w_k$; $\delta z_k = H_C \delta x_k$

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

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

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



2.5 Rigid body & IMU kinematics

Velocity ${}^I v_{IB} = \frac{d}{dt}({}^I t_B)$

Rot. Velocity ${}^I \omega_{IB} = \frac{d}{dt}(\alpha) {}^I t$

Velocity point ${}^B v_{IP} = {}^B v_{IB} +$

${}^B \omega_{IB} \times {}^B t_P$

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 ($\exp[\Delta \alpha]^\times = \delta R(\Delta \alpha)$)
 $R_{IB}(t + \Delta t) = \exp[\Delta \alpha]^\times R_{IB}(t)$

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.))

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

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

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. world frame

2.7 Wheeled robot Kinematics

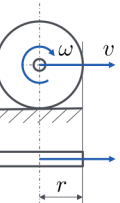
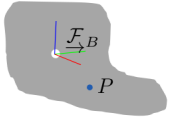
Non-holonomic systems not integrable, no inst. move in every direct.

Wheel constraints $v_i = \omega_i r_i$

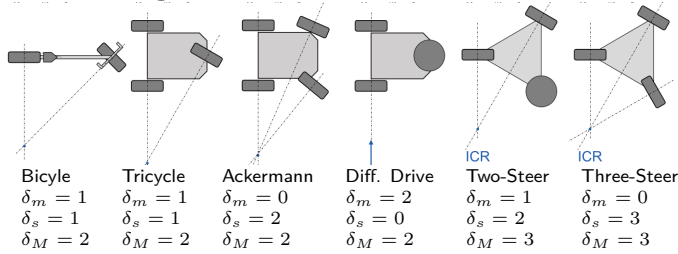
- Driving straight all v equal
- Turning Wheel axis must intersect the **Instant Centre of Rotation** (ICR), speeds: $v_i \div R_i = \Omega$ (R_i dist. wheel-ICR, Ω , vehicle body rotation rate)

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



Differential Drive Kinematics

State vec $x = [x_1, x_2, \theta]^\top$, **Inputs** $u = [\omega_l, \omega_r]^\top$, r_r radius of right wheel, 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: $v = \omega_l r_l = \omega_r r_r$, $\Omega = 0$, $D = v \Delta t$.

$$b_s = \begin{bmatrix} D \cos(\theta) \\ D \sin(\theta) \\ 0 \end{bmatrix} \quad b_t = \begin{bmatrix} R(\sin(\Delta\theta + \theta) - \sin(\theta)) \\ -R(\cos(\Delta\theta + \theta) - \cos(\theta)) \\ \Delta\theta \end{bmatrix}$$

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

Discretized: $x_k = x_{k-1} b_i$ with $i \in \{s, t\}$. ($\int \dots d\Delta t$)

3 Sensors & Actuators

Meas. Model: $z = h(x) + v + o$, with $h(x)$ deterministic mean, v zero-mean noise, o unmodelled effects, x true state

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

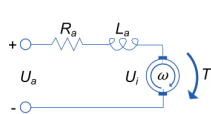
(Kirchoff) $U_a = L_a \dot{I}_a + R_a I_a + U_i$

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

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

(Mech. pow. eq. el. pow)

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



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 full distance

$u = c_u + f \cdot x'$ 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, $c = [c_u, c_v]^\top$ is optical centre of cam in pixel coords, f scale factor, and $c t_P = [t_x, t_y, t_z]^\top$

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

If p. in diff frame ${}^w t_P$, then $u = K[R_{CW} c t_{CW}] = {}^w t_P$

3.3.1 Pinhole Camera Projection with distortion

Def Model: $u = k(d(p(c t_P)))$, with:

(Projection) $x' = p(c t_P) = t_z^{-1} \cdot [t_x, t_y]^\top$

(Distortion model, $r^2 = x'^2 + y'^2$, $x'' = d(x')$)

$$x'' = \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} x' + \left[\frac{2p_1 x' y' + p_2 (r^2 + 2x'^2)}{p_1 (r^2 + 2y'^2) + 2p_2 x' y'} \right]$$

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

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

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

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

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

3.3.2 Undistorting a whole image

$u_i = k(d(k_{\text{new}}^{-1}(u_{i,\text{new}})))$ where $u_{i,\text{new}}$ is the place of pixel in output, 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

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, multi-path leading to larger distances sensed

$$\text{Proj. } z = \begin{bmatrix} u \\ d \end{bmatrix} = \begin{bmatrix} k(d(p(c t_P))) \\ [0, 0, 1] c t_P \end{bmatrix} \quad \text{Back: } c t_P = \begin{bmatrix} d x' \\ d \end{bmatrix}$$

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

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

Def Jac. J_f rows for eq of f cols for vars of each eq.

Approximation using finite differences $\frac{f(\bar{x}+h) - f(\bar{x})}{h}$,

or central differences (vector of $\frac{f(\bar{x})+h_i e_i - f(\bar{x})}{h_i}$, with e_i unit vec)

4.2 Linear Least Squares

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

Man. sol.: comp. $M = A^\top A$, $b' = A^\top b$, then $Mx = b'$.

Prob. sol.: $\text{argmax}_p(x | z)$ with

- **Max. Like** $p(x|z) \propto p(z|x) = \prod_{i=1}^N p(z_i|x)$
- **M a Post** $p(x|z) \propto p(z|x)p(x) = p(x) \prod_{i=1}^N p(z_i|x)$

4.3 Non-Linear Least Squares

Find $x^* = \text{argmax}_p(x|z) = \text{argmin}(-\log(p(x|z)))$

Gauss-Newton

Levenberg-Marquardt

Local Param.

4.4 Bayes Filter

x_k^R state at time k , z_k^p dist. meas., 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 $w'_{k,s} = \mathbb{P}[z_i | x_{k,s}] w_{k-1,s}$
- 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 $x_0 \sim \mathcal{N}(\hat{x}, P_0)$, P_0 previous covariance;

Prediction With linear state transition model (u_k odometry, w_k noise (covariance Q_k)):

$x_k = F x_{k-1} + G u_k + L w_k$ with $w_k \sim \mathcal{N}(0, Q_k)$:

- **Mean** $\hat{x}_{k|k-1} = F \hat{x}_{k-1} + G u_k$
- **Covariance** $P_{k|k-1} = F P_{k-1|k-1} F^\top + L Q_k L^\top$

Update Lin. meas.: $\tilde{z}_k = H x_k + v_k$ with $v_k \sim \mathcal{N}(0, R_k)$:

- **Meas. residual:** $y_k = \tilde{z}_k - H\hat{x}_{k|k-1}$
- **Resid. Cov:** $S_k = HP_{k|k-1}H^\top S_k^{-1}$
- **Kalman gain:** $K_k = P_{k|k-1}H^\top S_k^{-1}$
- **Updated mean:** $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k$
- **Updated Cov.:** $P_{k|k} = (I - K_k H)P_{k|k-1}$

4.7 Extended Kalman Filater (EKF)

Non-l. state trans. model $x_k = f(x_{k-1}, u_k, w_k)$ as above:

- **Mean:** $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$
- **Cov.:** $P_{k|k-1} = F_k P_{k-1|k-1} F_k^\top + L_k Q_k L_k^\top$
With F_k linearisation $\frac{\partial f}{\partial x}$ and L_k lin. $\frac{\partial f}{\partial w}$

Update N-Lin. meas.: $\tilde{z}_k = h(x_k) + v_k$:

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

Difference to above: H becomes H_k , and $H^\top$ is $H_k^\top$

5 SLAM to Spatial AI

5.1 Keypoints

Corner det. $SSD(\Delta_x, \Delta_y) \approx [\Delta_x \ \Delta_y] M [\Delta_x \ \Delta_y]^\top$ with $M = R^\top \text{diag}(\lambda_1, \lambda_2) R$; λ_i E.V. of M ; $R = \det(M) - \kappa$.
trace(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 Laplacian Operator $\nabla_{\text{norm}}^2 L = t \left(\frac{\partial^2 L}{\partial x^2} + \frac{\partial^2 L}{\partial y^2} \right)$

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)

$\lambda = [\lambda_1 \ \lambda_2]^\top = \text{argmin} |(w t C_2 + \lambda_2 w e_2) - (w t C_1 + \lambda_1 w e_2)|^2$

2 Solve normal equation $A\lambda = b$ with $q = -w e_1^\top w e_2$:

$$A = \begin{bmatrix} 1 & q \\ q & 1 \end{bmatrix} \quad b = \begin{bmatrix} e_1^\top \cdot (w t C_2 - w t C_1) \\ -e_2^\top \cdot (w t C_2 - w t C_1) \end{bmatrix}$$

3 Pick midp. $w t P = 0.5(\tau_1 + \tau_2)$; $\tau_n = w t C_n + \lambda_n w e_n$

5.3 Place Recognition

Idea: Build vocab from “visual words” (in Training). **Run-time** 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):

$x_{R,k}^* = \text{argmax} \mathbb{P}(x_{R,k} | x_M, z_{1:k}, u_{1:k})$ (recursive)

$x_{R,1:k}^* = \text{argmax} \mathbb{P}(x_{R,1:k} | x_M, z_{1:k}, u_{1:k})$ (batch)

SLAM $\{x_{R,k}^*, x_M^*\} = \text{argmax} \mathbb{P}(x_R, x_M | z_{1:k}, u_{1:k})$ (as $\uparrow$)

Mapp.: $x_M^* = \text{argmax} \mathbb{P}(x_M | x_{R,1:k}^*, z_{1:k}, u_{1:k})$ with given poses $x_{R,1:k}^*$. Thus temp. model $u_{1:k}$ 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 $\mathbb{P}(o_j | x_{R,1:k}) =$

$$\frac{\mathbb{P}(o_j | x_{R,k}, z_k) \mathbb{P}(z_k | x_{R,k}) \mathbb{P}(o_j | x_{R,1:k-1}, z_{1:k-1})}{\mathbb{P}(o_j) \mathbb{P}(z_k | x_{R,1:k}, z_{1:k-1})}$$

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): $l(o_j | x_{R,1:k}, z_{1:k}) =$

$$l(o_j | x_{R,k}, z_k) + l(o_j | x_{R,1:k-1}, z_{1:k-1}) - l(o_j)$$

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

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

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

6.3 Linear Quadratic Regulator (LQR)

For lin. cont.-time dyn. $\dot{x}(t) = F_c x(t) + G_c u(t)$.

We try to minimise cost functional (for $u(t) = -Kx(t)$)

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

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

$$F_c^\top P + P F_c - (P G_c) R^{-1} + Q = 0$$

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

Non-Lin: Approx, $\delta \dot{x}(t) = F_c \delta x(t) + G_c \delta u(t)$

6.4 MPC

Cost function ($p(x_N)$ terminal cost, sum the stage cost)

$$J_{0 \rightarrow N}(x_0, u_0, \dots, u_{N-1}) = p(x_N) + \sum_{k=0}^{N-1} q(x_k, u_k)$$

We minimize the above s.t. for $k \in \{0, \dots, N-1\}$ we have $x_{k+1} = f(x_k, u_k)$, $g(x_k, u_k) \leq 0$ and $x_N \in \mathcal{X}_f$ and $x_0 = x(0)$

Finite-Horizon Lin-Quad Control Quad. Cost:

$$J_{0 \rightarrow N}(x_0, u_0, \dots) = x_N^\top P x_N + \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k$$

without constraints, **State Feedback Law** $u_0^* = -Kx(0)$. With constraints, minimize as above.

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.Vertices 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     xn ← NEARESTCONFIGURATION(Graph, x)
10    xf ← STOPPINGCONFIGURATION(xn, x)
11    if xf ≠ xn then
12      insertVertexGraph, xf
13      ▷ Extension of RRT* goes here
14      insertEdgeGraph, xn, xf
15    if xf == xn 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 (because this is quite bad)

```

1 Xnear ← NEIGHBOURS(Graph, xf, R)
2 xmin ← NEIGHBOURS(Graph, xf, R)
3 cmin ← COST(xn) + EDGECOST(xn, xf)
4 for each xnear in Xnear do
5   if EDGECOLLISIONFREE(xnear, xf) and COST(xnear
6     + EDGECOST(xnear, xf) < cmin then
7     INSERTEDGE(Graph, xf, xnear)
8     xparent ← PARENT(Graph, xnear)
9     REMOVEEDGE(Graph, xparent, xnear)

```

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 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. $c_{\text{att}} = \frac{1}{2} k_{\text{att}} \|x - x_{\text{goal}}\|^2$ and $c_{\text{rep}} = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{\rho(x)} + \frac{1}{\rho_{\text{lim}}} \right) & \rho \leq \rho_{\text{lim}} \\ 0 & \text{else} \end{cases}$ **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 $x \in \mathcal{X}$ (RL: s), actions $u \in \mathcal{U}$ (RL: a), prob. state trans. $\mathcal{T}(x, u, x_+) = \mathbb{P}(x_+ | x, u)$, reward func $\mathcal{R}(x, u, x_+)$, start state x_0 , optional terminal state 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 $\gamma < 1$

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

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

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

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

Using policy iter:

```

1 Choose  $\pi_0(x)$ 
2 while policy has not converged do
3   repeat  $V_{i+1}^{\pi_j}(x) = Q(x, \pi(x)) \forall x$  and fixed pol.  $\pi_j$ 
4   until values converge
5 One step:  $\pi_{j+1}(x) = \operatorname{argmax}_u Q(x, u)$  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 $V_{i+1}^{\pi}(x) = Q(x, \pi(x))$ w/ $V_0^{\pi}(x) = 0$.

We need state trans. model, instead $\tilde{R}_j$ (approx. prob. w/ statistics) and thus

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

Active Find optimal policy π instead of state values $V(x)$. *Q-Learn*. Here: restate Val. Iter in *Q-Values* ($Q_0(x, u) = 0$):

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

Compute using sample:

$$\tilde{Q}_i(x, u) = \tilde{R}_i(x, u, x_+) + \gamma \max_u (Q_i(x_+, u))$$

Then update: $Q_{i+1}(x, u) = (1 - \alpha) Q_i(x, u) + \alpha \tilde{Q}_i(x, u)$. Called off-policy learning, needs exploration. Simplest is random actions (ϵ -greedy): ϵ is prob. to act randomly, $1 - \epsilon$ 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)