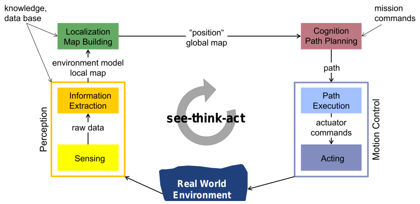
0 General

- $\text{cov}(X,Y) = E((X - E(X))(Y - E(Y)))$
- **Covariance C:** $C_{X_i,X_j} = \text{cov}(X_i, X_j)$
  – Square – Diagonal is variance
- **Cond. P.:** $P[B|A] := \frac{P[B \cap A]}{P[A]}$
- **Tot. P.:** $P[B] = \sum_{i=1}^{n} P[B \mid A_i] \cdot P[A_i]$
- **Bayes:** $P[A \mid B] = \frac{P[B|A] \cdot P[A]}{P[B]}$
- **Indep.:** $P[A \cap B] = P[A] \cdot P[B]$
- $E[X] = \int_{-\infty}^{\infty} x f_X(x) \mathrm{d}x$
- $Var[X] = E[(X - E[X])^2]$
- Cartesian $(x, y) \Leftrightarrow$ Polar (r, φ)
  – $*\ x = r\cos(\varphi)$ $*\ y = r\sin(\varphi)$
  – $*\ r = \sqrt{x^2 + y^2}$ $*\ \cos(\varphi) = \frac{x}{r}$ $*\ \sin(\varphi) = \frac{y}{r}$

1 Introduction



2 Locomotion

- Physical interaction between vehicle and environment
- **Stability:** characterized by – Number of contact pts – CoG – Static/dynamic stabilization – Inclination of terrain
- **Contact:** characterized by – Contact pt size and shape – Angle of contact – Friction
- **Environment:** characterized by – Structure – Medium
- **Implementation Aspects:** – Number of actuators – Structural complexity – control complexity – Energy consumption
- **Cost of transportation C_{mt}:**
  $= \frac{E}{m \cdot g \cdot d} = \frac{P}{m \cdot g \cdot v}$, – **E:** Energy – **P:** Power – **m:** Mass – **d:** Distance – **v:** Speed

2.1 Legged Locomotion

+ Mobility + Adaptability + Ability to manipulate environment - Mechanical complexity - Control complexity - Energy Consumption

- **Gait:** Distinct sequence of lift and release events of individual legs
- # possible events for k legged robot is $N = (2k - 1)!$
- **Static Gaits:** – System is statically sable – Requires ≥ 4 legs + Safe - Slow - Energetically inefficient
- **Dynamic Gaits:** – System is stabilized on a limited cycle – Falls over if stopped + Fast + Energetically efficient - Demanding for actuators - Demanding for control
- **Locomotion Control**
  1) Stepping sequence defined by gait pattern
  2) Stepping location
  3) Contact forces

2.2 Wheeled Locomotion

+ Highly efficient on flat surface

2.2.1 Wheel Types

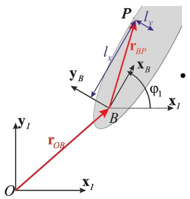
- **Standard Wheel:** – 2 DoF ($*$ wheel axle $*$ wheel contact point) – Can be steered or fixed – Steering without side effects on the body
- **Castor Wheel:** – 3 DoF ($*$ wheel axle $*$ wheel contact point $*$ offset steering joint) – Steering must be compensated by the body
- **Swedish Wheel:** – 3 DoF ($*$ Rotation around the wheel axle $*$ Rotation around the contact point $*$ Rotation around the rollers) – Similar to standard wheel but provides low resistance in another direction
- **Spherical Wheel:** – 3 DoF - Difficult to realize

2.2.2 Wheel Arrangements

- Influences manoeuvrability, controllability and stability
- No optimal arrangement
- **Stability:** – ≥ 3 wheels required for static stability – Improved by adding more wheels
- **Manoeuvrability:** – Number of DoF
- **Controllability:** – Inverse correlation between controllability and manoeuvrability – Less manoeuvrability robots can be controlled more accurately

3 Kinematics

3.1 Rigid Body Kinematics

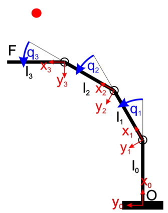


- **Translation:** All pts of the body move parallele
  – Vector from O to P in I: ${}_I r_{OP} = {}_I r_{OB} + {}_I r_{BP}$
  – Velocity of of pt P in I: ${}_I v_P = {}_I \dot{r}_{OP}$
- **Rotation:** One pt is fixed an all other move around it
  – Vector r in B rotated to I: ${}_I r = R_{IB}\, {}_B r$
  – Rotation R_{IB} along x, y, z:
  $$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) \\ 0 & 1 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix}$$
  $$\begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
  – **Inverse:** $R_{BI} = R_{IB}^{-1} = R_{IB}^{\mathsf{T}}$
  – **Composition:** $R_{AI} = R_{AB} \cdot R_{BI}$
  – Angular velocity of B in I along z: ${}_I \omega_{IB} = (0, 0, \dot{\varphi})^{\mathsf{T}}$
    $*$ Equal for every point in the body
  – **Transformation:** ${}_I \omega_{IB} = R_{IB}\, {}_B \omega_{IB}$
  – **Composition:** ${}_I \omega_{IC} = {}_I \omega_{IB} + {}_I \omega_{BC}$
  – ${}_I \hat{\omega}_{IB} = \dot{R}_{IB} R_{IB}^{\mathsf{T}} =$
  $$\begin{bmatrix} 0 & -{}_I\omega_{IB}^z & {}_I\omega_{IB}^y \\ {}_I\omega_{IB}^z & 0 & -{}_I\omega_{IB}^x \\ -{}_I\omega_{IB}^y & {}_I\omega_{IB}^x & 0 \end{bmatrix}, {}_I \omega_{IB} = \begin{bmatrix} {}_I\omega_{IB}^x \\ {}_I\omega_{IB}^y \\ {}_I\omega_{IB}^z \end{bmatrix}$$
- **Homogeneous Transformation:** Rotation + Translation
  – Vector from O to P in I: ${}_I r_{OP} = {}_I r_{OP} + R_{IB}\, {}_B r_{BP}$
  – $\begin{pmatrix} {}_I r_{OP} \\ 1 \end{pmatrix} = \begin{bmatrix} R_{IB} & {}_I r_{OB} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} {}_B r_{BP} \\ 1 \end{pmatrix}$
  – Full motion: ${}_I v_P = {}_I \dot{r}_{OP} = {}_I \dot{r}_{OB} + {}_I \omega_{IB} \times {}_I r_{BP}$
- **Vector Differentiation**
  – **Non-moving system:** ${}_I r \implies {}_I \dot{r} = \frac{d\, {}_I r}{dt}$
  – **Moving (translating or rotating) system:** ${}_B r \implies {}_B \dot{r} = \frac{d\, {}_B r}{dt} + {}_B \omega_{IB} \times {}_B r$

- **Generalized Coordinated**
  – Set of independent variables that uniquely describe the robot's configuration
  – $q = (q_1, q_2, \ldots q_n)^{\mathsf{T}}$
  – ${}_I r_{OP} = {}_I r_{OP}(q)$
- **Jacobians**
  – $J_P = \frac{\partial r_{OP}(q)}{\partial q} = \begin{bmatrix} \frac{\partial r_1}{\partial q_1} & \cdots & \frac{\partial r_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial q_1} & \cdots & \frac{\partial r_1}{\partial q_n} \end{bmatrix}$
  – Cartesian velocity to generalized velocity: $\dot{r}_P = J\dot{q}$
  – Change in generalized coordinates to Cartesian space: $\triangle r_P = J \triangle q$

3.2 Inverse Kinematics

- Given desired endeffector position $r_{OF}(q) = r_{OF}^{\text{goal}}$, determine generalized coordinates q
- $r_{OF}(q)$ not easily invertible



- **Iterative Method:**
  1) Initialize $q = q^0$ to initial guess and set $r = r(q)$
  2) Evaluate Jacobian $J_P = \left.\frac{\partial r_{OF}}{\partial q}\right|_{q=q^i}$
  3) Invert Jacobian to obtain $\triangle q = J^+ \triangle r_{OF}$
  4) Update generalized coordinates $q^{i+1} = q^i + J^+(r_{OF}^{\text{goal}} - r_{OF}^j)$
  5) Repeat from 2 till converge

3.3 Inverse Differential Kinematics

- Given desired endeffector velocity $\dot{r}_F = J_P \dot{q} = r_F^{\text{goal}}$, determine generalized velocity $q = J_P^+ \dot{r}_F^{\text{goal}}$

3.4 Redundancy and Singularity

- **Redundancy:** Jacobian is column-rank deficient
  – Pseudo inverse minimizes $\|\dot{q}\|_2$
  – Can have multiple solutions $\dot{q} = J^+ \dot{r}^{\text{goal}} + \underbrace{(I - J^+ J)}_{N} \dot{q}_0$
  – q_0 is chosen solution
- **Singularity:** Jacobian is row-rand deficient
  – Pseude inverse minimizes error $\left\|\dot{r}^{\text{goal}} - \dot{r}\right\|_2$

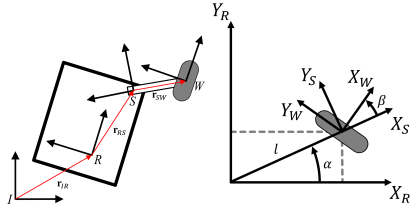
3.5 Mobile Robot Kinematics

- **Legged Robot:** Quadropad
  – $q = \begin{bmatrix} q_B \\ q_r \end{bmatrix}$ Un-actuated base / Actuated joints
  – **Contact Constraint:** $\dot{r}_F = J_F \dot{q} = 0 \Rightarrow \dot{q} = \cancel{J_F^+ \dot{r}_F}^{=0} + N_F \dot{q}_0$
  – **Body Move:** $\dot{r}_B = \cancel{J_B \dot{q}} = J_B N_F \dot{q}_0$
  – $\dot{q} = \cancel{J_F^+ \dot{r}_F}^{=0} + N_F \dot{q}_0 = N_F (J_B N_F)^+ \dot{r}_B$
- **Wheeled Robot:** Planar car
  – $q = \begin{bmatrix} x \\ \varphi \end{bmatrix}$
  – Point on wheel: ${}_I r_{OP} = \begin{bmatrix} x + r\sin(\varphi) \\ r + r\cos(\varphi) \\ 0 \end{bmatrix}$
  – ${}_I J_P = \begin{bmatrix} 1 & r\cos(\varphi) \\ 0 & -r\sin(\varphi) \\ 0 & 0 \end{bmatrix}$
  – **Contact Constraint:**
  ${}_I \dot{r}_P|_{\varphi=\pi} = {}_I J_P|_{\varphi=\pi}\, \dot{q} = \begin{bmatrix} 1 & -r \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{\varphi} \end{bmatrix} = 0$
  – **Rolling Constraint:** $\dot{x} - r\dot{\varphi} = 0$

3.6 Mobile Robots

- **Holonomic:** – Can move instantaneously in any direction of its DoF – Differential constraints are integrable
- **Non-Holonomic:** – Cannot move instantaneously in any direction of its DoF – Differential constraints are not integrable
- **Differential Forward Kinematics:** Given set of actuator speeds, determine its corresponding velocity
- **Differential Inverse Kinematics:** given a desired velocity, determine the corresponding actuator speeds

3.6.1 Wheel Kinematic



- Robot state: $\xi_I = [x, y, \theta]^{\mathsf{T}}$ $\bullet$ $\dot{\xi}_I = [\dot{x}, \dot{y}, \dot{\theta}]^{\mathsf{T}}$
- $\dot{\xi}_R = R(\theta)\dot{\xi}_I$ $\bullet$ $R(\theta) = R_z^{\mathsf{T}}(\theta)$
- ${}_W v_{IW} = \begin{bmatrix} 0 \\ -r\dot{\varphi} \\ 0 \end{bmatrix}$ no-sliding constraint / rolling constraint / planar assumption
- **Rolling Constraint:** $J_1(\beta_s)R(\theta)\dot{\xi}_I - \dot{\varphi}r = 0$
- **No-Sliding Constraint:** $C_1(\beta_s)R(\theta)\dot{\xi}_I = 0$
- **General Wheel Equation:** $v_{IW} = v_{IR} + \omega_{IR} \times r_{RS} + \omega_{IR} \times r_{SW} + \omega_{RS} \times r_{SW}$
- **Standard (Steered & Fixes) Wheel:** $r_{SW} = 0$
  – **Wheel Equation:** $v_{IW} = v_{IR} + \omega_{IR} \times r_{RS}$
    $*$ ${}_W v_{IR} = R(\alpha + \beta)R(\theta)[\dot{x}, \dot{y}, 0]^{\mathsf{T}}$
    $*$ ${}_W \omega_{IR} \times {}_W r_{RS} = [l\dot{\theta}\sin(\beta), l\dot{\theta}\cos(\beta), 0]^{\mathsf{T}}$
  – $J_1(\beta_s) = [\sin(\alpha + \beta_s), -\cos(\alpha + \beta_s), -l\cos(\beta_s)]$
  – $C_1(\beta_s) = [\cos(\alpha + \beta_s), \sin(\alpha + \beta_s), l\sin(\beta_s)]$
  – If fixed, $J_1(\beta_s) = J_1, C_1(\beta_s) = C_1$
  – Only wheel which imposes constraints
- **Castor Wheel:**
  – $J_1(\beta_s)$ same as standard wheel
  – $C_1(\beta_s) = [\cos(\alpha+\beta_s), \sin(\alpha+\beta_s), d + l\sin(\beta)]$
- **Swedish Wheel:**
  – $J_1(\beta_s) = [\sin(\alpha + \beta_s + \gamma), -\cos(\alpha + \beta_s + \gamma), -l\cos(\beta_s + \gamma)]$
  – $C_1(\beta_s) = [\cos(\alpha + \beta_s + \gamma), \sin(\alpha + \beta_s + \gamma), l\sin(\beta_s + \gamma)]$

3.6.2 Mobile Robot Kinematic

- $N = (N_f + N_s)$ wheeled robot
- $J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1S}(\beta_s) \end{bmatrix}$ $\bullet$ $C_1(\beta_s) = \begin{bmatrix} C_{1f} \\ C_{1S}(\beta_s) \end{bmatrix}$
- $J_2 = \text{diag}(r_1, \ldots, r_N)$ $\bullet$ $\dot{\varphi} = \begin{bmatrix} \dot{\varphi}_f \\ \dot{\varphi}_s \end{bmatrix}$
- **Rolling Constraint:** $J_1(\beta_s)R(\theta)\dot{\xi}_I - J_2\dot{\varphi} = 0$
- **No-Sliding Constraint:** $C_1(\beta_s)R(\theta)\dot{\xi}_I = 0$
- $\underbrace{\begin{bmatrix} J_1(\beta_s) \\ C_1(\beta_s) \end{bmatrix}}_{A} \underbrace{R(\theta)\dot{\xi}_I}_{\dot{\xi}_R} = \underbrace{\begin{bmatrix} J_2 \\ 0 \end{bmatrix}}_{B} \dot{\varphi}$

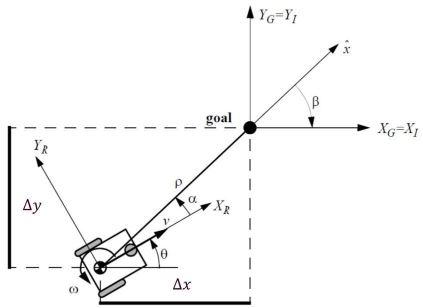
- **Forward Kinematics:** $\dot{\xi}_R = (A^{\mathsf{T}}A)^{-1}A^{\mathsf{T}}B\dot{\varphi} = A^+B\dot{\varphi}$
- **Inverse Kinematics:** $\dot{\varphi} = (B^{\mathsf{T}}B)^{-1}B^{\mathsf{T}}A\dot{\xi}_R = B^+A\dot{\varphi}$
- **Differential Drive:** – $\alpha_r = -\pi/2$ – $\beta_r = \pi$
  – $l_r = b$ – $r_r = r$ – $\alpha_r = \pi/2$ – $\beta_r = 0$
  – $l_l = -b$ – $r_l = r$
  – $\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_R = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ r/2b & -r/2b \end{bmatrix} \begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix}$
  – $\begin{bmatrix} \dot{\varphi}_r \\ \dot{\varphi}_l \end{bmatrix} = \begin{bmatrix} 1/r & 0 & b/r \\ 1/r & 0 & -b/r \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_R$

3.6.3 Maneuverability

- **Degree of Maneuverability** $\delta_{\mathbf{M}} = \delta_m + \delta_s$
- **Degree of Mobility** $\delta_{\mathbf{m}}$**:** $= \dim N(C_1(\beta_S)) = 3 - \text{rank}(C_1(\beta_s))$
  – $0 \leq \delta_m \leq 3$ – $\delta_m = 0 \Rightarrow N_f = N_s = 0$
  – $\delta_m = 3 \Rightarrow$ motion not possible
- **Degree of Steerability** $\delta_{\mathbf{s}}$**:** $= \text{rank}(C_{1s}(\beta_s))$
  – $0 \leq \delta_s \leq 2$ – $\delta_s = 0 \Rightarrow N_s = 0$
  – $\delta_s = 1 \Rightarrow N_s \geq 1$ – $\delta_s = 2 \Rightarrow N_f = 0$
- **Examples:**

	Omnidirectional	Differential	Omni-Steer	Tricyle	Two-Steer	Ackerman Steer	Bicycle	Synchro Drive	Onmi Drive
δ_M	3	2	3	2	3	2	2	2	3
δ_m	3	2	2	1	1	1	1	1	3
δ_s	0	0	1	1	2	1	1	1	0

3.7 Motion Control - Feedback Control



- Drive robot from ${}_I[x, y, \delta]$ to ${}_I[0, 0, 0]$
- **Error:** $e = [x, y, \delta]^{\mathsf{T}}$
- Get control inputs as $\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = K \cdot e = K \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$
- Find K such that $lim_{t \to \infty} e(t) = 0$ $\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$
- Polar Coordinate Transformation:
  – $\rho = \sqrt{\triangle x^2 + \triangle y^2}$
  – $\alpha = -\theta + \text{atan2}(\triangle y, \triangle x)$ – $\beta = -\theta - \alpha$
  – $\alpha, \beta \in [-\pi, \pi]$
- If $\alpha \in (-\frac{\pi}{2}, \frac{\pi}{2}]$: $\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \\ -\frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$
- Otherwise: $\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 \\ -\frac{\sin\alpha}{\rho} & 1 \\ \frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$
- Control Law:

$$\begin{bmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos(\alpha) \\ k_\rho \sin(\alpha - k_\alpha \alpha - k_\beta \beta) \\ -k_\rho \sin(\alpha) \end{bmatrix}$$

- v has a constant sign for one run
- Stable if $k_\rho > 0, k_\beta < 0, k_\alpha - k_\rho > 0$

4 Perception

- Raw Data $\implies$ Features $\implies$ Objects $\implies$ Places/Situations

4.1 Sensors

4.1.1 Types of Sensors

- **Proprioceptive (PC):** Measure values internally to the system
- **Exteroceptive (EC):** Acquire information about the robots environment
- **Passive (P):** Measure ambient environmental energy
- **Active (A):** Emit energy into the environment and measure the environments reaction to it + More accurate - May interfere - May affect the characteristic one wants to measure
- **Common Sensors**

Sensor Type	System	Class
Tactile	Bumpers	EC, P
Wheel/motor	Brush encoders	PC, P
	Optical encoders	PC, A
Heading	Compass	EC, P
	Gyroscope	PC, P
	Inclinometer	EC, A/P
Acceleration	Accelerometer	PC, P
Beacons	GPS	EC, A
	Radio, ultrasonic, Reflective Beacons	EC, A
Motion/speed	Doppler: radar/sound	EC, A
Range	Ultrasound, laser, struct. light, ToF	EC, A
Vision	CCD/CMOS	EC, P

- **Encoders:** – Convert linear/angular motion of a shaft to digital – For control of motor-driven joints
- **Heading Sensors:** – Determine orientation and inclination w.r.t some reference – **Gyroscope:** * Absolute measure for heading * **Mechanical:** Fast spinning wheel * **Optical:** Laser beam in fiber coil; Measure phase shift
- **Range Sensors:** – Traveled distance d of a sound or electromagnetic wave after a **time of flight** t – $d = ct$ – Sound: $c = 0.3\,\text{m/ms}$ – Light: $c = 0.3\,\text{m/ns}$
- **GPS:** – Satellites send their position and time to the device – Device computes location based on ToF – Time synchronisation and measurement in ToF is critical – **RTK:** Measure of carrier wave phase - up to centimeter-level accuracy TODO: More Sensors?

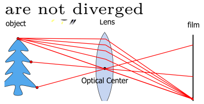
4.1.2 Uncertainty Representation

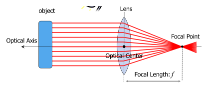
- **Systematic Error:** – Deterministic – Can be modelled and calibrated
- **Random Error:** – Cannot be predicted – Described statistically
- Density function identifies for each possible x of RV X a probability $f(x)$ – $\int_{-\infty}^{\infty} f(x)\mathrm{d}x = 1$
  – $\mu = E[X] = \int_{-\infty}^{\infty} x f(x)\mathrm{d}x$
  – $\sigma^2 = E[(X - E[X])^2]$
- **Error Propagation Law** – Error propagation in system with n inputs and m outputs $f : \mathbb{R}^n \to \mathbb{R}^m$, $f : X_1, \dots X_n \mapsto f(X_1, \dots, X_n) = Y_j$ – Uncertainty of X_i is known – Mapping between input covariance C_X and output covariance C_Y: $C_Y = F_X C_X F_X^\top$
  – $F_X = \triangledown f_X = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_n} \end{bmatrix}$ – $\mathbf{C_{X/Y}}$: Covariance of input uncertainties/propagated uncertainties of the output

4.2 Computer Vision

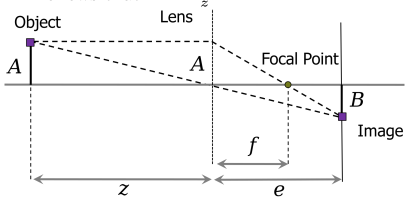
4.2.1 Camera Model

- **Pinhole Model:** – Black box with single hole – Beam of rays enter through the **Optical Center/Center of Projection C** – Inverted image is projected onto the **Image Plane**
- **Converging Lens:** – Focuses multiple rays from the same point on the object to the same point on the image plane – All rays parallel to the **optical axis** converge at the **focal point** – Rays passing through the **optical center** are not diverged



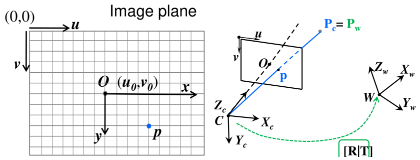


- **Thin Lens equation:** $\frac{1}{f} = \frac{1}{z} + \frac{1}{e}$
- **Pinhole Approximation:** $z \gg f \implies f \approx e$
  – I.e. lens is approximated as pinhole at distance f from image plane
  – If follows that $B' = \frac{f}{z} B$



4.2.2 Perspective Camera

- Image plane represented in front of C
- Camera measures angles and not distances
- Project pt $P_W = (x, y, z)$ onto $p_C = (u, v)$ on the image plane
- **Perspective Equations:**
  – $x = f\frac{X_C}{Z_C}$ – $y = f\frac{Y_C}{Z_C}$
  – $u = u_0 + kx$ – $v = v_0 + ky$ – **Scale factor** k [pixels/meter] – **Focal length in pixels:** $kf = \alpha$
  – $\implies \tilde{p} = \begin{bmatrix}\lambda u\\ \lambda v\\ \lambda\end{bmatrix} = \underbrace{\begin{bmatrix} kf & 0 & u_0 \\ 0 & kf & v_0 \\ 0 & 0 & 1\end{bmatrix}}_{K} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix}$
- **Intrinsic Parameter** K: depending on camera
- Rigid Body transformation $\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{R} \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + \underbrace{\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}}_{T} = [R \mid T] \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$



- **Extrinsic Parameter** $[R \mid T]$: depending on transformation
- **Perspective Projection:** $K[R \mid T]$
- $\implies \tilde{p} = \begin{bmatrix}\lambda u\\ \lambda v\\ \lambda\end{bmatrix} = K[R \mid T] \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$
- **Barrel Distortion**
  – Straight lines get bend
  – **Barrel Distortion:** Image gets "round"
  – **Pincusion Distortion:** Corners pulled out
  – Model: $\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + k_1 r^2)\begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$
    * **Distortion Parameter** $\mathbf{k_1}$: Intrinsic parameter
    * $r^2 = (u - u_0)^2 + (v - v_0)^2$

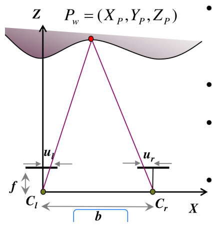
4.2.3 Omnidirectional Camera

- **Dioptric:** – System of lenses – $\sim 180°$ FOV
- **Catadioptric:** – Combination of lens and mirrors – $> 180°$ FOV – Greatly distorted (can be removed depending on mirror)
  – **Central Camera:** Mirror shaped that all incoming rays have same *single effective viewpoint*
    * Correct mirror placement is important
    + Unwarping into perspective image possible
    + Can convert points in the image to spherical vectors
    + Can apply standard algorithms
- **Polydioptric:** – Multiple cameras – $\sim 360°$ FOV

4.2.4 Camera Calibration

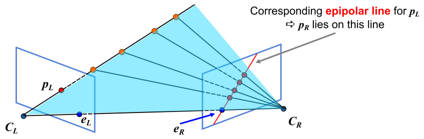
- Determine intrinsic (and extrinsic) camera parameters • $\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|T]\begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \overset{!}{=} \begin{bmatrix} m_{11} & \dots & m_{14} \\ \vdots & \ddots & \vdots \\ m_{31} & \dots & m_{34} \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix}$ • Find all m_{ij} • Requires ≥ 6 pts • Done using least-square method • Need to decompose the found matrix into K, R, T • QR factorize 3×3 submatrix $m_{11} : m_{33} \Rightarrow K := R, R := R$ • Calculate $T = K^{-1}[m_14, m_24, m_34]^\top$

4.2.5 Stereo Vision



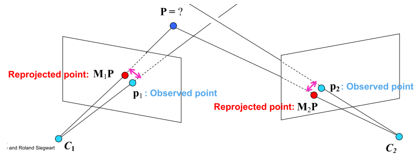
- Obtain depth information from two cameras with know relative position
- Cameras need to be calibrated
- Calculating depth Z_P of point P_W (distance to P_W)
- **Baseline b:** Distance between the two cameras
- **Disparity:** Difference in image location of the projection of a 3D point on the two image panes: $u_l - u_r$
- Two identical cameras aligned on x
  – $Z_P = \frac{bf}{u_l - u_r}$
- Two identical cameras in different Coordinate Systems
  – C_r is a rigid body transformation of C_l
  – Triangulation * $\tilde{p}_l = \lambda_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = K_l \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix}$
    * $\tilde{p}_r = \lambda_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = K_r R \begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} + T$
- **Disparity Map**
  – Compute disparity for corresponding points of every pixel
  – Can compute the depth from it and reconstruct 3D scene

4.2.6 Correspondence Search



- Identify corresponding points in two images of the same scene
- Measure similarity of two pts using a similarity measurement
- **Epipolar Plane:** Spanned by C_l, C_r and some pt P_W
- **Epipolar Line:** Projection of the ray from one camera through points in the other camera
- **Epipole:** Projection of other camera in image plane – All epipolar lines go through it
- **Epipolar Constraint:** Correspond of pt of left images lies on the epipolar line of the right image
- **Epipolar Rectification:** Transform both images to make epipolar lines collinear and parallel: 1) Remove radial distortion 2) **Warping:** Reprojection of both images to same plane

4.2.7 Triangulation



- Find 3D coordinate of a point given the projections on multiple image planes
- R_i, T_i, K_i are known
  – $\lambda_1 \underbrace{\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}}_{p_1} = \underbrace{K_1[I \mid 0]}_{M_1} \underbrace{\begin{bmatrix} X_x \\ Y_w \\ Z_w \\ 1 \end{bmatrix}}_{P}$
  – $\lambda_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = K_2[R \mid T] \begin{bmatrix} X_x \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$
- No unique solution du to noise and errors
- **Linear Approach:** Solved using SVD
- **Reprojection Approach:** Minimize Sum of Square Reprojection Error $\text{SSRE} = \|P_1 - M_1 P\|^2 + \|p_2 - M_2 P\|^2$
- Baseline is important:
  – **Too Small:** Large depth error
  – **Too Large:** * Objects may be visible only from one camera * Difficult search problem for close objects

4.2.8 Structure from Motion (SfM)

- Given corresponding image points $\{(u_1^i, v_1^i), (u_2^i, v_2^i)\}$, recover: – 3D location P_i of all n pts – Relative pose of right camera R, T – Camera intrinsic K (optionally)
- Assume K is known
- **Knowns:** $4n$: n correspondences
- **Unknowns:** $5 + 3n$: rotation (3), translation (2, since scale cannot be recovered), n 3D pts ($3n$)
- Solution exists iff $4n \geq 5 + 3n \implies n \geq 5$
- **Crossp.:** $a \times b = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [a]_\times b$
- **Epipolar Constraint:** $p_2^T E p_1 = 0$
- **Essential Matrix:** $E = [T]_\times R$
  – Computed with ≥ 5 correspondences
  – Can be decomposed into R and T
- **Normalized Img Coordinates:** $p = [\bar{u}, \bar{v}, 1]^\top = K^{-1}[u, v, 1]^\top$
- **8-pt Algorithm** – Algorithm to compute essential matrix – Uses ≥ 8 pts
  – $p_2^\top E p_1 = [\bar{u}_2, \bar{v}_2, 1] \begin{bmatrix} e_11 & e_12 & e_13 \\ e_21 & e_22 & e_23 \\ e_31 & e_32 & e_33 \end{bmatrix} \begin{bmatrix} \bar{u}_1 \\ \bar{v}_1 \\ 1 \end{bmatrix} = 0$
  – $\underbrace{[u_2u_1, u_2v_1, u_2, v_2u_1, v_2v_1, v_2, u_1, v_1, 1]}_{Q\text{:known}} \cdot \underbrace{[e_{11}, e_{12}, \dots, e_{33}]}_{\text{unknown}} = 0$
  – For 8 non-coplanar pts, unique solution is given by the EV of Q corresponding to the smallest Eigenvalue

4.2.9 Image Filtering

- **Linear:** Replace each pixel by a linear combination of its neighbours
- **Shift-Invariant:** Same operation is performed on every point on the image
- **Filter H:** aka. *kernel, mask, window*
- **Boundaries:** Need to handle specially:
  – Ignore – Zero padding – Pad with edge value – Mirror boundary – Wrap around from other side
- **Normalization:** Prevent img from getting brighter/darker
- **Correlation:** $J(x) = F \circ I(x, y) = \sum_{i=-N}^{N} \sum_{j=-M}^{M} F(i,j) I(x+i, y+j)$
  – Not associative
- **Convolution:** $J(x) = F * I(x, y) = \sum_{i=-N}^{N} \sum_{j=-M}^{M} F(i,j) I(x-i, y-j)$
  – Equivalent to correlation except that the filter is flipped before correlating
  – Same result to correlation if the filter is symmetric
  – Is associative: $F * (G * I) = (F * G) * I$
- **Correlation Cost: (per pixel)**
  – **2D:** * **Mult.:** $(2N+1)^2$ * **Add.:** $(2N-1)^2 - 1$
  – **2 × 1D:** * **Mult.:** $2(2N+1)$ * **Add.:** $4N$
  – **2 × 1D with const.:** * **Mult.:** 1 * **Add.:** $4N$
- **Derivation:** $J(x) = \frac{I(x+1) - I(x-1)}{2}$
- **Smoothing:**
  – For blurring or noise detection
  – **1D Gaussian:** $G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$
  – **2D Gaussian:**
    $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = g_\sigma(x) \cdot g_\sigma(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-y^2}{2\sigma^2}}$
    * Is separable
- **Similarity Measure**
  – **Sum of Square Diff.:** $\text{SSD}(x) = \sum_i (F(i) - I(x+i))^2 = \sum_i F(i)^2 + \sum_i I(x+i)^2 - 2\sum_i (F(i)I(x+i))$
  – **Normalized Cross Correlation:** $\text{NCC} = \frac{\sum_i (F(i)I(x+i))}{\sqrt{\sum_i F(i)^2}\sqrt{\sum_i I(x+i)^2}}$

- **Zero-Mean Normalized Cross Correlation (ZNCC)**: Invariant to local average intensity. Maximize: $\text{ZNCC}(x) = \frac{\sum_i (F(i)-\mu_F)(I(x+i)-\mu_{I_x})}{\sqrt{\sum_i (F(i)-\mu_F)^2}\sqrt{\sum_i (I(x+i)-\mu_{I_x})^2}}$
  * $\mu_F = \frac{\sum_i F(i)}{2N+1}$ * $\mu_{I_x} = \frac{\sum_i I(x+i)}{2N+1}$
- **Template Matching:** Use template as filter and apply similarity measure
- **Edge Detection:** 1) Noise reduction 2) Gradient $S' = (G_\sigma * I)' = G'_\sigma * I$ 3) Thresholding 4) Non-maximal suppression
  - **Derivation of Gaussian:** $G'_\sigma(x,y) = \frac{\partial G_\sigma(x,y)}{\partial x} + \frac{\partial G_\sigma(x,y)}{\partial y}$ * Edges at min/max
  - **Laplacian of Gaussian:** $\text{LoG} = G''_\sigma(x,y) = \frac{\partial^2 G_\sigma(x,y)}{\partial x^2} + \frac{\partial^2 G_\sigma(x,y)}{\partial y^2}$ * Edges at zero crossing
  - **Prewitt:** * $F_x = \left[\begin{array}{c|c|c} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{array}\right]$ * $F_x = \left[\begin{array}{c|c|c} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{array}\right]$
  - **Sobel:** * $F_x = \left[\begin{array}{c|c|c} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array}\right]$ * $F_x = \left[\begin{array}{c|c|c} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{array}\right]$
  - **Roberts:** * $F_x = \left[\begin{array}{c|c} 0 & 1 \\ 1 & 0 \end{array}\right]$ * $F_x = \left[\begin{array}{c|c} 1 & 0 \\ 0 & 1 \end{array}\right]$

4.2.10 Point Features

- **Harris Corner Detection:** Patch at (x,y) of size P has great difference to all surrounding patches $(x+\Delta x, y+\Delta y)$
  - $\text{SSD}(\Delta x, \Delta y) = \sum_{x,y\in P}(I(x,y) - I(x+\Delta x, y+\Delta y))^2$
  - $\Rightarrow \text{SSD}(\Delta x, \Delta y) \approx \sum_{x,y\in P}(I_x(x,y)x - I_y(x,y)y)^2$
    * $I_x = \frac{\partial I(x,y)}{\partial x}$ * $I_y = \frac{\partial I(x,y)}{\partial y}$ * Using first-order Taylor approx
  - $\Rightarrow \text{SSD}(\Delta x, \Delta y) \approx [\Delta x, \Delta y] M [\Delta x, \Delta y]^\mathsf{T}$
    * $M = \sum_{x,y\in P}\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} = R^{-1}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$
  - * $\boldsymbol{\lambda_1, \lambda_2}$ **small:** Flat region * $\boldsymbol{\lambda_1 >> \lambda_2}$**:** Edge * $\boldsymbol{\lambda_1 << \lambda_2}$**:** Edge * $\boldsymbol{\lambda_1, \lambda_2}$ **large:** Corner
  - **Cornerness Function:** $C = \det - \kappa \text{ trace}^2(M)$
    * Extract local minima * Used since calculating eigenvalues is expensive * κ is magic number
  - **Invariant to:** * Rotation * Linear intensity change
  - **Not invariant to:** Scale change
- **Scale Invariant Feature Transform (SIFT) Features:**
  - Detect blobs
  - Assigns each keypoint a orientation and descriptor
  - **Keypoint Detection:** 1) Downscale image multiple times 2) Gaussian blur each at different rates 3) Calculate *Difference of Gaussian (DoG)* (subtract successive smoothed images) 4) Find keypoints (local extrema in the DoG)
  - **Keypoint Orientation:** 1) Generate gradient orientation histogram for each keypoint 2) Orientation is value of highest peak in histogram
  - **Keypoint Descriptor:** 1) Divide neighbourhood into 4×4 regions 2) Compute orientation histograms along 8 orientations 3) Concatenate $4 \times 4 \times 8 = 128$ values in vector
  + High robustness
  + Invariant to rotation
- **Features from Accelerated Segment Test (FAST) Detector:** – Detects corners – Look at 16 pixels around target pixel – If they are all significantly darker/brighter, we have a FAST corner + Fast - Not robust against noise
- **Binary Robust Independent Elementary Features (BRIEF) Descriptor:**
  – Detect blobs 1) "Random" (predefined) selection of pixel pairs 2) Compare intensity of pairs and store result in descriptor 3) Matching done with Hamming Distance
  + High speed in description and matching
  - Not scale invariant
  - Not rotation invariant
- **Binary Robust Invariant Scalable Keypoints (BRISK) Detector** – Detect blobs 1) Detect corners using FAST 2) Compare intensity similar to BRIEF but in some specific geometric pattern + High speed + Rotation and scale invariant - Slower than BRIEF

4.2.11 Place Recognition

- Training set of images needed to create structure
- **Bag of features:** Bag of descriptors of a word – Image is described by this set – Sectional information is removed
- **Visual Word:** Single number representing a high-dimensional descriptor – Done by **k-means clustering** – Bag of features ⇒ bag of visual words
- **Visual Vocabulary:** DB of visual words created by many training images
- Scene is collection of visual words
- Efficiently find matches:
  – **Vocabulary Tree:** Hierarchical arrangement of visual vocabulary (I guess) * Leaf holds image which matches the feature * By feeding in features of new images, we get a bag of matching images * Number of matches determines the probability
  – **Inverted File DB:** * List of all possible visual words points to list containing all images containing this feature * Query DB for all features of a given list and count occurrences of a certain image (using voting array)
- **Term Frequency-Inverse Document Frequency (tf-idf):** Not all words are equally important
  – Measure importance of visual word inside image
  – Used for voting array
  – For word w_i in img j: * $\text{tf-idf}_{ij} = \text{tf}_{ij} \cdot \text{idf}_i$
    * $\text{tf}_{ij} = \frac{\#\text{occur. of } w_i \text{in } j}{\#\text{words in } j}$
    * $\text{idf}_i = \log \frac{|\text{Img}|}{|\{\text{img}|w_i \in \text{img}\}|}$
- **FABMAP** – Place recognition – Assume: worlds is set of discrete places – **Place:** vector of visual words at that scene – Calculate probability of being at a know location – Frequent occurring objects are suppressed + High performance

4.2.12 Line Extraction

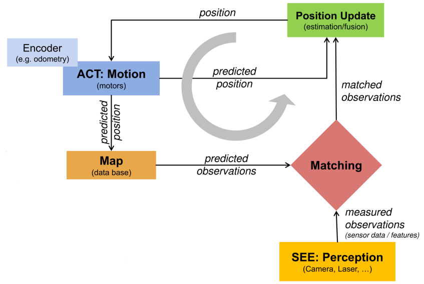
- **Task:** Given set of pts, find best fitting line – Do not have to figure out which pts belong to which line
- **Split-and-Merge:**
  1) **Split:** a) Fit line through point set (or take end-pts) b) Find most distant pt c) If distance > threshold, split set and repeat
  2) **Merge:** If two consecutive line segments are almost collinear, merge them
  - Sensitive to outliers
- **Random Sample Consensus (RANSAC):** 1) Draw line through two randomly selected pts 2) Compute distance of all pts to it 4) Repeat $k = \frac{\log(1-p)}{\log(1-w^2)}$ times * **w:** fraction of inliers * **p:** prob of finding set of pts free of outliers 5) Select line with least total error
  - Non-Deterministic
- **Hough-Transform:**
  – Each pt in image space defines a sinosoid in Hough space
  1) For each pt compute $\rho = x\cos\theta + y\sin\theta, \theta \in [0, 180]$ 2) Capture votes for parameters 3) Find (rho, θ) with most votes

5 Localization

- Problems:
  – **Position tracking:** Keep track of known location
  – **Global localization:** Localize without initial location
  – **Kidnapping robot problem:** Robot is moved to new location
- **Map Representation**
  – Known a priori or build by robot
  – **Types:** * Architecture map * Finite/infinite set of lines * Exact cell decomposition * Fixed cell decomposition * Topological map
- **Belief Representation**
  – Uncertain due to error $\implies$ probabilistic localization
  – **Continuous:** * Precision bound by sensor data * Typically single hypotheses ($\implies$ danger of getting lost)
  – **Discretized:** * Precision bound by discretization resolution * Typically multiple hypotheses * High resource demands
  – **Types:** * Continuous map with single hypotheses prob. dist. $p(x)$ * Continuous map with multiple hypotheses prob. dist. $p(x)$ * Discretized metric map (grid k) with prob. dist. $p(x)$ * Discretized topological map (nodes n) with prog. dist. $p(x)$
- **Odometry**
  – Process of calculating current position by using previous position and measured motion of last time step
  – **Differential Drive Robot:**
    $p_t = f(p_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{\Delta S_r + \Delta S_l}{2}\cos(\theta_{t-1} + \frac{\Delta S_r - \Delta S_l}{2b}) \\ \frac{\Delta S_r + \Delta S_l}{2}\sin(\theta_{t-1} + \frac{\Delta S_r - \Delta S_l}{2b}) \\ \frac{\Delta S_r - \Delta S_l}{b} \end{bmatrix}$,
    b = Distance between wheels
  – **Error Model:**
    * **Error Propagation:** $\Sigma_{p_t} = F_{p_{t-1}}\Sigma_{p_{t-1}}F_{p_{t-1}}^\mathsf{T} + F_{\Delta S}\Sigma_{\Delta S}F_{\Delta S}^\mathsf{T}$
    * **Error Model:** $\Sigma_{\Delta S} = \begin{pmatrix} k_r|\Delta S_r| & 0 \\ 0 & k_l|\Delta S_l| \end{pmatrix}$
    * with $F_{x/y} = \nabla_{x/y} f$

5.1 Map-Based Localization

- Estimate position using perceived information and a map
- **Terminology:** – $\mathbf{x_t}$**:** robot location at time t
  – $\mathbf{u_t}$**:** proprioceptive sensor reading at time t
  – $\mathbf{z_t}$**:** exteroceptive sensor reading (observation) at time t
  – **M**$= \{m_0, m_1, \dots, m_{n-1}\}$ true map of the environment with n features
  – $\overline{\textbf{bel}}(\mathbf{x_t}) = p(x_t \mid z_{1\to t}, u_{1\to t})$
  – $\overline{\textbf{bel}}(\mathbf{x_t}) = p(x_t \mid z_{1\to t-1}, u_{1\to t})$
- **Requirements**
  – Initial probability distribution $\text{bel}(x_0)$
  – Map $M = \{m_0, m_1, \dots, m_n\}$ of the environment
  – Proprioceptive and exteroceptive data u_t and z_t
  – Probabilistic motion model $p(x_t \mid u_t, M)$
    * Derived from the robots kinematics
  – Probabilistic measurement model $p(z_t \mid x_t, M)$
    * Derived from the exteroceptive sensor model
- **Steps**
  0) Robot is placed somewhere in the environment
  S1) **Prediction/Action Update (ACT):** Move and estimate position using proprioceptive sensors
    * Uncertainty grows
  S2) **Perception/Measurement Update (SEE):** Make observation using exteroceptive sensors
    * Uncertainty shrinks



5.1.1 Markov Localization

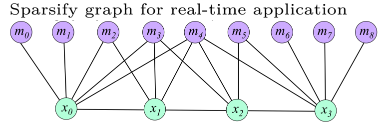
- **Solves:** – Global localization – Position tracking – Kidnapped robot
- Discretized pose representation x_t
- $\text{bel}(x_t)$ is representation by arbitrary prob. density function
  – Multiple hypotheses
- **Markov Assumption:** $\text{bel}(x_t) = p(x_t \mid x_0, u_{0\to t}, z_{0\to t}) = p(x_t \mid x_{t-1}, u_t, z_t)$
- **Steps:**
  for all x_t **do**
  S1) **ACT:** $\overline{\text{bel}}(x_t) = \sum_{x_{t-1}} p(x_t \mid u_t, x_{t-1})\text{bel}(x_{t-1})$
  S2) **SEE:** $\text{bel}(x_t) = \eta p(z_t \mid x_t, M)\overline{\text{bel}}(x_t)$ with $\eta = \frac{1}{p(y)}$
  end for
  - Huge state space
  - Inefficient
- **Improvements:** – Change of cell size – Adaptive cell size – **Randomized Sampling:** Approximation of belief state by a representative subset of possible locations. Higher density around peak points

5.1.2 Extended Kalman Filter (EKF) Localization

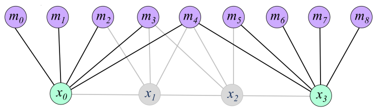
- **Solves:** – Position tracking
- $\text{bel}(x_t)$, motion model and measurement model is represented by a normal distribution
  – Single hypotheses
- **Steps:**
  S1) **Prediction Update**
    * New position: $\hat{x}_t = f(x_{t-1}, u_t)$
    * Position uncertainty: $\hat{P}_t = F_x P_{t-1} F_x^\mathsf{T} + F_u Q_t F_u^\mathsf{T}$
    * ∘ $\mathbf{P_{t-1}}$**:** Covariance of the previous state $x-1$
      ∘ $\mathbf{Q_t}$**:** Covariance of motion model noise
      ∘ $\mathbf{F_{x/u}}$**:** Jacobian w.r.t. x/u
  S2) **Measurement Update**
    1) **Observation:** Obtain set of observation z_t^i with covariance $R_t^i, i = 1 \dots n$
    2) **Measurement Prediction:** Predict feature observations $\hat{z}_t^j = h^j(\hat{x}_t, m^j)$ and compute its Jacobian H^j w.r.t $\hat{x}_t$
    3) **Matching:**
      ∘ Compute the innovation $v_t^{ij} = [z_t^i - \hat{z}_t^j]$
      ∘ Compute the innovation covariance $\Sigma_{IN_t}^{ij} = H^j \hat{P}_t {H^j}^\mathsf{T} + R_t^i$
      ∘ Find matches with a validation gate g
        ▷ Mahalanobis distance: ${v_t^{ij}}^\mathsf{T}(\Sigma_{IN_t}^{ij})^{-1}v_t^{ij} \le g^2$
    4) **Estimation:**
      ∘ Stack validated observations into z_t, corresponding innovations into v_t, measurement Jacobians into H_t and $R_t = \text{diag}(R_t^i)$
      ∘ Compute composite innovation covariance Σ_{IN_t}
      ∘ For each match, update position estimate $x_t = \hat{x}_t + K_t v_t$
      ∘ For each match, update covariance $P_t = \hat{P}_t - K_t\Sigma_{IN_t}K_t^\mathsf{T}$
        ▷ **Kalman Gain:** $K_t = \hat{P}_t H_t^\mathsf{T}(\Sigma_{IN_t})^{-1}$
  + Precise
  + Efficient
  - Robot may get lost

5.2 Simultaneous Localization and Mapping (SLAM)

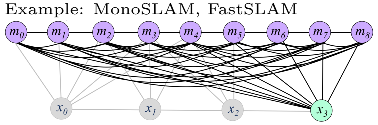
- Create map and localize in parallel only using proprioceptive and exteroceptive sensor data
- **Terminology:** – $\mathbf{x_t}$**:** robot location at time t – $\mathbf{X_t}$$= \{x_0, x_1, \dots, x_t\}$ robot path till time t – $\mathbf{u_t}$**:** motion between $t-1$ and t – $\mathbf{U_t}$$= \{u_0, u_1, \dots, u_t\}$ sequence of relative motions till time t – $\mathbf{z_t}$**:** landmark observation at time t – $\mathbf{Z_t}$$= \{z_0, z_1, \dots, z_t\}$ sequence of landmark observations till time t – $\mathbf{z_t}$**:** exteroceptive sensor reading (observation) at time t – **M**$= \{m_0, m_1, \dots, m_{n-1}\}$ true map of the environment with n landmarks
- **Full SLAM:** Reconstruct full path $p(X_t, M \mid Z_t, U_t)$
- **Online SLAM:** Reconstruct current position $p(x_t, M \mid Z_t, U_t)$
- **Approaches:**
  – **Full Graph Optimisation**
    * Eliminate observations and control inputs
    * Solve for constraints between poses and landmarks
    + Finds globally consistent solution
    + Trajectories can be very smooth
    - Sparsify graph for real-time application



  – **Key Frames**
    * Retain most representative poses and their dependency links
    * Optimize resulting graph
    * Example: PTAM, OKVIS, ORB-SLAM



- **Filtering**
  * Summarize all experience w.r.t. last pose
  * Done using state vector and covariance matrix
  * Example: MonoSLAM, FastSLAM



5.2.1 EKF SLAM

- Example of filtering
- Summarizes past experience:
  - Extended state vector
    $y_t = \left[x_t, m_1, \cdots, m_{n-1}\right]^\mathsf{T}$
  - Corresponding covariance
    $$P_{y_t} = \begin{bmatrix} P_{xx} & \cdots & P_{xm_{n-1}} \\ \vdots & \ddots & \vdots \\ P_{m_{n-1}x} & \cdots & P_{m_{n-1}m_{n-1}} \end{bmatrix}$$

S1) **Prediction:**
  - According to EKF equations
    * $\hat{y}_t = \begin{bmatrix} \hat{x}_t \\ m_i \end{bmatrix} = \begin{bmatrix} f(x_{t-1}, u_t) \\ 0 \end{bmatrix}$
    * $\hat{P}_{y_t} = F_y P_{y_{t-1}} F_y^\mathsf{T} + F_u Q_t F_u^\mathsf{T}$

S2) **Measurement:**
  - Measurement model $\hat{z}_i = h_i(\hat{x}_t, m_i)$ as in EKF localization

S3) **Update:**
  - Update the state with actual observations Z_{n-1}:
  - $y_t = \hat{y}_t + K_t(Z_{n-1} - h_{0:n-1}(\hat{x}_t, M_{n-1}))$
  - $P_{y_t} = \hat{P}_{y_t} - K_t \Sigma_{IN} K_t^\mathsf{T}$
  - Where * $\Sigma_{IN} = H \hat{P}_{y_t} H^\mathsf{T} + R$ * Kalman gain $K_t = \hat{P}_{y_t} H (\Sigma_{IN})^{-1}$

- Initially, the covariance matrix is sparse (features are uncorrolated)
- After some time the covariance matrix becomes more dense
- **MonoSLAM**
  - EKF SLAM implementation
  - Only one camera $\implies$ no proprioceptive data $\implies$ custom motion model
  - **Constant Velocity Motion Model**
    $$f_v = \begin{bmatrix} \mathbf{r}^W_{\text{new}} \\ \mathbf{q}^{WR}_{\text{new}} \\ \mathbf{v}^W_{\text{new}} \\ \omega^W_{\text{new}} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^W + (\mathbf{v}^W + \mathbf{V}^W)\Delta t \\ \mathbf{q}^{WR} \times \mathbf{q}((\omega^W + \mathbf{\Omega}^W)\Delta t) \\ \mathbf{v}^W + \mathbf{V}^W \\ \omega^W + \mathbf{\Omega}^W \end{bmatrix}$$
  - Unknown linear and angular accelerations cause a velocity impulse in each time step:
    $n = \begin{bmatrix} \mathbf{V}^W \\ \mathbf{\Omega}^W \end{bmatrix} = \begin{bmatrix} \mathbf{a}^W \Delta t \\ \alpha^W \Delta t \end{bmatrix}$
- **Challenges** – Robust local motion estimation – Mapping and loop-closure detection – Map management and optimisation – Sensitivity to incorrect data associations – Tradeoff between efficiency and precision – Dynamic objects – Collaborative exploration

6 Planning

- **Completeness:** A robot system is complete iff for all possible problems, when a solution trajectory exists, the system will find it

6.1 Path Planning

- **Problem:** Find path in the physical space from the initial position to the goal position avoiding obstacles
- **Workspace:** Physical area where the robot moves
- **Configuration Space** C: k-dimensional space representing all configurations (position, rotations etc.) of a robot: – Occupied areas O – Free areas $F = C \setminus O$

6.1.1 Graph Search

- **Idea:** Search path in connectivity graph of free space
- Consists of two steps

1) **Graph Construction**
  - **Visibility Graph**
    * ∘ Polygonal configuration space
      ∘ Robot is a point $\implies$ enlarge obstacles
      ∘ Connect corners which "see" each other
    + Gives shortest path
    - Complexity depends on number of obstacles
    - Path is close to obstacles
    - Does not consider robot motion constraints
  - **Voroni Diagram**
    * ∘ For each point in free space compute the distance to the nearest obstacle
      ∘ Points which are equidistant to two obstacles for edges
    + Maximizes distance to obstacles
    + Can do the process in reverse. Robot in unknown environment creates diagram by moving on unknown veroni edges
    - Not optimal
    - Does not consider robot motion constraints
  - **Exact Cell Decomposition**
    * ∘ Draw lines according to obstacle geometry
      ∘ Each filed is in either F or O
      ∘ Create connectivity graph from cells in F
    * Assumes: precise position in F does not matter
    - Complexity depends on number and complexity of obstacles
  - **Approximate Cell Decomposition**
    * ∘ Put grid over map
      ∘ Each cell is either in O or F
      ∘ Create connectivity graph from cells in F
    * **Fixed-Size:** Grid has fixed size
    * **Variable-Size:** Recursively split partially occupied cells into 4 new cells
    - Loss of narrow passages
  - **Lattice Graph**
    * ∘ Overlay space with custom edges/curves
      ∘ Each cell is either in O or F
      ∘ Create connectivity graph from cells in F
    * Cell decomposition is a special case of it
    + Great freedom

2) **Graph Search**
  - Expected cost from start to goal via node n:
    $f(n) = \underbrace{g(n)}_{\text{cost so far}} + \epsilon \underbrace{h(n)}_{\text{cost to go heuristic}}$
  - **Breath-First Search (BFS)**
    * Nodes expanded according to FIFO queue and closed list
    * Solution is optimal for uniform edge cost
    * $\mathcal{O}(|V| + |E|)$
    + First solution found is optional
    * Greedy backtracking of solution
    * **BF(Graph G, Node Start, Node Goal)**

```
FIFO.push(Start)
while FIFO not empty do
  Node current ← FIFO.pop()
  if current == Goal then
    return SUCCESS
  end if
  visited.push(current)
  Nodes next = getNeighbours(current)
  for all next ∉ visited do
    FIFO.push(next)
  end for
end while
return FAILURE
```

  - **Depth-First Search (DFS)**
    * Nodes expanded according to LIFO queue and closed list
    + Better space complexity
    * $\mathcal{O}(|V| + |E|)$
    - First solution found may not be optional
    * Greedy backtracking of solution
    * **DF:** Same as BF but using LIFO
  - **Dijkstra's Algorithm**
    * Nodes expanded according to Min-Heap (on $g(n)$) and closed list
    * Positive edge cost
    * $\mathcal{O}(|V| \log(|V|) + |E|)$
    * Often computed from the goal position to get all distances to the goal
    + First solution found is optional
    * Greedy backtracking of solution
    * **Dijkstra(Graph G, Node Start, Node Goal)**

```
MIN-HEAP.push(Start)
while MIN-HEAP not empty do
  Node current ← MIN-HEAP.pop()
  if current == Goal then
    return SUCCESS
  end if
  visited.push(current)
  Nodes next = getNeighbours(current)
  for all next ∉ visited do
    MIN-HEAP.push(next)
  end for
end while
return FAILURE
```

  - **A***
    * Nodes expanded according to Min-Heap (on $f(n)$) and closed list
    * Heuristic function guides search
      ∘ Must underestimate the distance
      ∘ Often $h(n)$ = Distance from n to goal
    * Positive edge cost
    * Good for single source shortest path
    * Runtime depends on chosen $h(n)$
    + First solution found is optional
    + Very efficient if we are ok finding a suboptimal solution
      ∘ Done if $\epsilon > 1$
    * Greedy backtracking of solution
    * **A-Star(Graph G, Heur H, Node Start, Node Goal)**

```
HEUR-MIN-HEAP.push(Start)
while HEUR-MIN-HEAP not empty do
  Node current ←
  HEUR-MIN-HEAP.pop()
  if current == Goal then
    return SUCCESS
  end if
  visited.push(current)
  Nodes next = getNeighbours(current)
  for all next ∉ visited do
    HEUR-MIN-HEAP.push(next)
  end for
end while
return FAILURE
```

  - **Rapidly Exploring Random Trees (RRTs)**
    * Grows randomized tree
    * Selects random points and connects it to closest point of the tree using a new branch
    * Most likely returns suboptimal solution
    * Probabilistically complete
    * **RRT(Node Start, Node Goal, System Sys, Environment Env)**

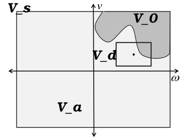
```
Graph.init(Start)
while Graph.size() ≤ threshold do
  Node rand = random()
  Node near = Graph.nearest(rand)
  try
    Node new = Sys.propagate(near, rand)
    Graph.addNode(new)
    Graph.addEdge(near, new)
  end try
end while
return FAILURE
```

6.1.2 Potential Field Planning

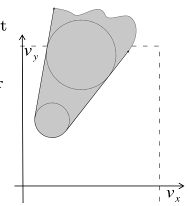
- **Idea:** Robot follows gradient of potential field
- Global
- Implicit incorporation of basic system models
- **Local Potential Fields**
  - **Potential Field:**
    $U(q) = \underbrace{U_{\text{att}}(q)}_{\text{attractive}} + \underbrace{U_{\text{rep}}(q)}_{\text{repulsive}}$
  - $U_{\text{att}}(q) = \frac{1}{2} k_{\text{att}} (q - q_{\text{goal}})^2$
  - $U_{\text{rep}}(q) =$
    $\begin{cases} \frac{1}{2} k_{\text{rep}} (\frac{1}{\rho(q)} - \frac{1}{\rho_{\text{lim}}})^2 & \text{if } \rho(q) \geq \rho_{\text{lin}} \\ 0 & \text{otherwise} \end{cases}$
  - * $\rho(\mathbf{q})$: Min distance between q and the closest obstacle * $\rho_{\mathbf{lim}}$: Distance of influence of obstacle * $\mathbf{k_{att/rep}}$: Scaling factor
  - Robot follows force $F(q) = -\bigtriangledown U(q)$
  - May get stuck in local minima
  - No incorporation of robot's dynamic constraints
- **Harmonic Potential Fields**
  - Robots follows solution of Laplace Equation
    $\triangle U = \sum \frac{\partial^2 U}{\partial^2 q_i} = 0$
  - Boundary is specified according to:
    * **Neumann:** ∘ Equidistant lines parallel to the obstacles - Potentially small distance to the obstacle ∘ $\frac{\partial U(q)}{\partial q} = g(q) = 0$
    * **Dirchlet:** ∘ Obstacle boundaries have constant high potential ∘ Goal has constant low potential ∘ Close to obstacles, robot moves perpendicularly away from obstacle ∘ $U(q) = f(q) = \text{const}$ + Safe paths - Long paths
  + Absence of local minima
  - Solved via discretization of the workspace into cells
  - Euler approximation:
    $\bigtriangledown U(q)_i \approx \frac{U(q+\delta e_i) - U(q)}{\delta}$
  - Iterative method: $U^{k+1}(q) =$
    $\frac{1}{2n} \sum_{i=1}^{n} (U^k(q + \delta e_i) + U^k(q - \delta e_i))$
    * **n:** # dimensions * $\mathbf{e_i}$: unit vector
  - Steps from iteration i to $i+1$
    1) Initialize cell values
       ∘ **Dirchlet:** Set obstacles to 1 and goal and all other cells to 0
    2) Update all cells except goal and obstacles
       a) Add up values of 4 neighbouring cells
       b) Scale sum by $2n$
       c) Update cell values
    3) Repeat 2 till convergence
    4) Backtrack solution

6.2 Obstacle Avoidance

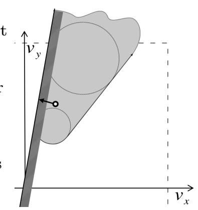
- Function of current on optionally past sensor readings, its current and goal location
- Prone to local minima
+ Incorporate system model
- **Dynamic Window Approach (DWA)**
  - Assume: robot moves on circular arcs (ν, ω)



  - **Obstacle space** $\mathbf{V_o}$: Transformation of obstacles from workspace to input space
  - **Admissible space** $\mathbf{V_a}$: complement of V_o
  - **Static Window** $\mathbf{V_s}$: Configuration which are allowed by robot limits
  - **Dynamic Window** $\mathbf{V_d}$: Configuration which can be reached in one step
    * Centred around the current configuration
  - **New configuration** $\mathbf{V_r} = V_a \cap V_s \cap V_d$
    * Selected which maximizes some objective function containing heading, distance to goal and velocity
  - Not safe when obstacles are dynamic
- **Velocity Obstacles (VO)**
  - Assume: * Robot moves on piece-wise linear curves * Robot is omnidirectional * Robot and obstacles are circular * Obstacles move at constant speed



  - Collision of robot and obstacle if $\|p_{\text{RO}} + \nu_R t\| < r_{\text{R}} + r_{\text{O}}$
  - **Velocity Obstacle:** Velocities in velocity space which lead to collisions in less than τ time
  - $\text{VO}^\tau_{\text{RO}} = \bigcup_{0 \leq t \leq \tau} D(-\frac{p_{\text{RO}}}{t}, \frac{r_{\text{RO}}}{t})$ * $\mathbf{p_{RO}}$: Relative position between robot and obstacle * $\mathbf{VO_{RO}}$: Relative velocity between robot and obstacle * $\mathbf{r_{R/O}}$: Radius of robot/obstacle * $\mathbf{D(x, y)}$: Disc centred at (x, y)
  - Multiple objects lead to multiple velocity obstacle regions
  - Robot selects any velocity outside of the complement of VOs
  - Does not model interaction effects
- **Reciprocal Velocity Obstacles (RVO)**
  - Assume: * Robot moves on piece-wise linear curves * Robot is omnidirectional * Robot and obstacles are circular * Fair behaviour of all agents



  - Identical to CO, but collision avoidance is shared between interaction agents
  - Operate in relative velocity space
  - Linear constraints decides if obstacles are avoided to the left or right
    * Tangent to the boundary of the CO which is closer to the current velocity
  - **Current relative velocity leads to collision:** Shift at least with half the velocity difference to the linear constraint
  - **Current relative velocity does not lead to collision:** Shift at most with half the velocity difference towards the linear constraint