Robotics 2 - mobile robots

D. Beck
Apr 24

1. Introduction

Where am I? *Localization*
Where am I going? *Goal setting*
How do I get there? *Navigation*

Definition

Robot: programmed actuated mech. w/ degree of autonomy
→ perform task based on current state (no human)

Service robot: personal or professional performing tasks for humans

Mobile robot: robot able to travel under its own control

Mobile robotics: use mobile robots mainly as service robot

Types

On wheels, on legs, flying, underwater

⇒ Farms, gardening, restaurant, ocean, pers. use

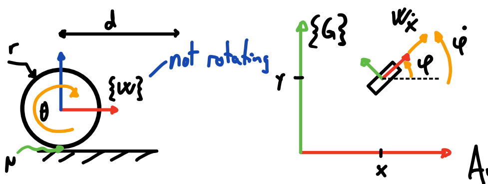
Transportation & logistics 86k sales 2022

⇒ main number of suppliers is Europe 426

2. Kinematics

Wheel kinematic

Parameters



Wheel has 3 DOF

$x \; y \; \varphi$

${}^{G}X = \begin{bmatrix} x \\ y \\ \varphi \end{bmatrix}$ ${}^{G}\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix}$

And 1 or 2 control

θ and φ (steering)

Frames

$${}^{G}_{W}T = D(x,y)\,R_z(\varphi) = \left[\begin{array}{cc|c} c_\varphi & -s_\varphi & x \\ s_\varphi & c_\varphi & y \\ \hline 0 & 0 & 1 \end{array}\right]$$

⚠ Speed more imp. for control

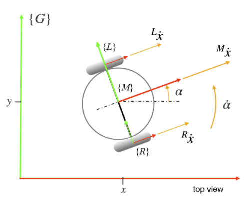
${}^{W}\dot{x} = r\dot{\theta}, \; \dot{\varphi}$

${}^{G}\dot{X} = [c_\varphi \dot{x}_w, s_\varphi \dot{x}_w, \dot{\varphi}]^T$

Non Holonomic systems

Less actuators than DOF (under-actuated)

Differential drive



${}^{M}\dot{X} = [{}^{M}\dot{x}, 0, \dot{\alpha}]^T$

Forward kin: ${}^{M}\dot{x} = \frac{1}{2}({}^{L}\dot{x} + {}^{R}\dot{x})$

$\dot{\alpha} = \frac{1}{2\rho}({}^{R}\dot{x} - {}^{L}\dot{x})$

Inverse Kin: ${}^{L}\dot{x} = {}^{M}\dot{x} - \rho\dot{\alpha}$ ${}^{R}\dot{x} = {}^{M}\dot{x} + \rho\dot{\alpha}$

Compute wheel speed

Odometry ${}^{G}\dot{X} = [c_\alpha {}^{M}\dot{x}, s_\alpha {}^{M}\dot{x}, \dot{\alpha}]^T$

Estimate global robot pose

→ Integrate velocity

${}^{G}X = \int {}^{G}\dot{X}\,dt + {}^{G}X_0$

Discrete:

$\alpha_{k+1} = \alpha_k + \dot{\alpha}_k \cdot T$

$x_{k+1} = x_k + c_{\alpha_{k+1}} {}^{M}\dot{x}_k T$

$y_{k+1} = y_k + s_{\alpha_{k+1}} {}^{M}\dot{x}_k T$

Steerable wheels

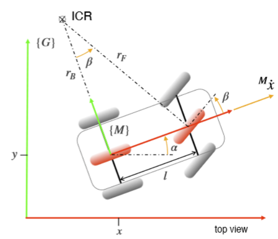
Forward ${}^{r}_{B}\dot{\alpha} = {}^{M}\dot{x}$

$\dot{\alpha} = \frac{{}^{M}\dot{x}}{\rho} \cdot \tan(\beta)$

Inverse

$\beta = \tan^{-1}\left(\frac{\rho \cdot \dot{\alpha}}{{}^{M}\dot{x}}\right)$

Same principle for odometry



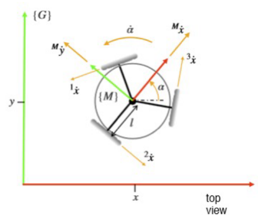
Holonomic systems

Omniwheels

→ rollers are perpendicular

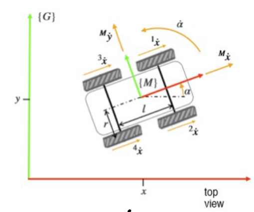
Robots w/ 3 omniwheels

Kinematic is linear!



Mecanum wheels (Swedish wheels)

→ rollers are inclined



Others: robots on a sphere, articulated vehicles

3. Sensor

Standard Filters

Why filters: prevent noise amplification and precise control

Analog filters (Lowpass, Hp, notch)

→ remove gravity
└ filter power sup.

e.g. Lowpass

1st order $G(s) = \frac{w_c}{s + w_c}$ $w_c = 2\pi f_c$

2nd $G(s) = \frac{w_c^2}{s^2 + 2\zeta w_c s + w_c^2}$

→ slope is 40 dB/dec

Digital imp. for e.g.

$y_k = S_f x_k + (1 - S_f) y_{k-1}$

$S_f = \frac{2\pi f_c T}{1 + 2\pi f_c T}$ — control period

Digital filters

Pure digital filters often deliver better results

Median → very good against outliers

MA → same as LP, as window ↑ amp ↓ + phase offset ↑

FIR → Generalized MA $y_k = \sum_{i=0}^{n-1} b_i x_{k-i}$

→ Restrictions: Big noise implies significant amp. loss and offset

Alternative → use a model based filter

Model based filters

State observers, Kalman filter

Kalman filters

Discrete-time filters for linear systems

Algorithm in two steps:

1. Prediction → estimate next value based on model
2. Update → correct estimation with measurements

! Take into account statistic effect of model and sensors

EKF: extend Kalman filter → used for non-lin systems

Applications

Sensor fusion for better precision, state observers (estimate process variable) ...

System model

State-space linear model

x_k state vect. y_k measur.

u_k control variable v_k meas. noise

w_k process noise

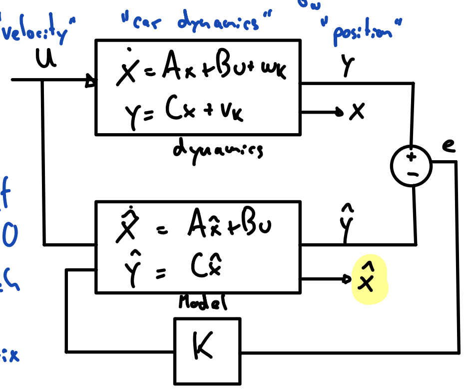
$$x_k = Ax_{k-1} + Bu_k + w_k$$

$$y_k = Cx_k + v_k$$

$$w_k = N(0, Q_{\sigma_w^2}) \quad v = N(0, R^{\sigma_v^2})$$

Concept

"State observer"



We want $\hat{x}$ to be optimal, for that e must decay to 0

→ We choose K such that e is faster

→ K is Kalman matrix

1. Prediction

Based on previous estimated state $\hat{x}_{k-1}$, predict a state $\hat{x}_k^-$

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

Based on previous state covariance matrix P_{k-1}^-, propagate to the new predicted matrix. (measure of how reliable the state is)

$$P_k^- = AP_{k-1}A^T + Q$$

2. Update

$$K_k = \frac{P_k^- C^T}{CP_k^- C^T + R}$$ → minimize error covariance

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

3. Repeat ↻ $P_k = (I - K_k C)P_k^-$

Extended Kalman Filter

Non-linear systems

→ For locally linear systems

→ Use Jacobian matrix to estimate next state

$$\Delta x_k \approx F\Delta x_{k-1} + w_k \quad \Delta y_k \approx G\Delta x_k + v_k$$

Sensor Fusion

Use Kalman filter approach to use different types of sensor for the same physical measur. and fuse their value based on smallest uncertainty

4. Position control

Holonomic robots

→ Simple as they can move in all directions

1. Determine global velocity
2. Convert global vel. to local
3. Determine wheel speed w/ inv. kin.

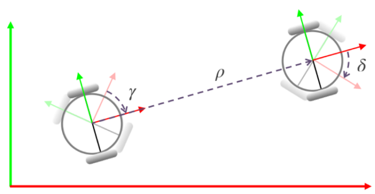
Non-Holonomic robots

Determine γ, ρ and δ and do a step-wise maneuvers.

→ only if robot can rot.

→ small error lead to big misalignment

→ angles in range $-\pi$ to π



Direct Maneuver

Position control law from Siegwart 3.6.2

! Distance and orient to goal computed at every control cycle.

→ speed must be saturated.

Differential drive

Current pose ${}^G X = \begin{bmatrix} x \\ y \\ \alpha \end{bmatrix}$ Desired pose ${}^G X_d = \begin{bmatrix} x_d \\ y_d \\ \alpha_d \end{bmatrix}$

$$\rho = \sqrt{(x_d - x)^2 + (y_d - y)^2}$$

$$\gamma = \operatorname{atan2}(y_d - y, x_d - x) - \alpha$$

$$\delta = \gamma + \alpha - \alpha_d$$

Position control laws

$${}^M\dot{x} = k_\rho \rho c_\gamma$$

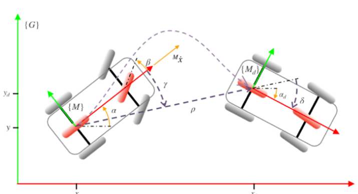
$$\dot{\alpha} = k_\gamma \gamma + k_\rho s_\gamma c_\gamma \left(1 + k_\delta \frac{\delta}{\gamma}\right)$$

! $\rho = 0$ $\gamma = 0$ → undefined $\gamma, \delta = [-\pi, \pi]$

Steerable robots

ρ, γ, δ same as for differential drive

$k_\rho > 0$ $k_\gamma > k_\rho$ $k_\delta < 0$



Position control law

$${}^M\dot{x} = k_\rho \rho \quad \beta = k_\gamma \gamma + k_\delta \delta$$

5. Localization

Position tracking robots thinks it knows where it is
Global localization robot has no idea where it is
Kidnapped robot think it knows but it is somew. else

estimate robot pose base on speed + time + previous est.

Incremental estimation

→ dead reckoning
e.g odometry

Types

Encoder counter also able to detect direction w/ 2 chan.
DC tachometer $V \propto$ angular speed
IMU {
- Accelerometer : m/s^2
- Gyroscope : rad/s
- Magnetometer : "compass" ! perturbations

→ Cheap + fast but drift
! systematic errors , Non systematic errors
→ physical model , Slippage, skidding

Absolute estimation

GPS 32 satellites in orbit, measure time of travel from ≥ 4 Acc : 1 - 10 m
LIDAR infrared laser, deliver distance & quality grade

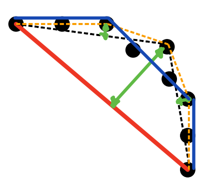
2D, 3D cameras

→ Usually slow or not available

LIDAR and Landmarks

Split and merge

1. Read points, create line connecting end points
2. Find point with max dist to the line
3. Add new line, repeat



Rule : if distance > threshold
Split repeat
done else
mark points as one line
↳ Merge : 2 lines are colinear < threshold → merge them

Segmentation In indoor environment with walls if dist between 2 points is small → keep points as segment

Landmarks lines reveal corner information, hence some landmarks ⌐ ⌙ , doors...
w/ camera use markers or vision algorithm like Harris corner, SIFT, SURF to detect features

Sensor fusion for localization

Combine relative sensors for relative pose and absolute sensors for absolute pose correction (landmarks)
→ Kalman Filter mostly used

a. Non-linear SSP model w/ noise

$X_k = f(X_{k-1}, \delta_{k-1}, w_{k-1})$ $\quad \delta_{k-1} = {}^{n}\dot{x}_{k-1} T$

1. State estimation (assumed 0)

$X_k = f(X_{k-1}, \delta_{k-1}, 0)$

2. Prediction

$P_k^- = F_{X_{k-1}} P_{k-1} F_{X_{k-1}}^T + F_{w_{k-1}} Q F_{w_{k-1}}^T$

$F_x = \frac{df}{dX}\Big|_{w=0}$ $\quad F_w = \frac{df}{dw}\Big|_{w=0}$

→ need corrections of P_k^- or it grows up ↑↑

3. Update, with landmark position!

$\hat{X}_k = \hat{X}_k^- + K_k Y_k$ "innovation" Y_k = "diff between measur. and estimation"

$P_k = (I - K_k H_{X_k}) P_k^-$ → smaller cov. matrix

$K_k = P_k H_{X_k}^T \big(\underbrace{H_{X_k} P_k H_{X_k}^T + H_{V_k} R H_{V_k}^T}_{\text{Innovation cov. matrix}}\big)^{-1}$

Jacobi of landmark measur.

Markov localization

P(x|y) → probability of x after using data y

Notation : Global robot pose $X_t = [x, y, \alpha]$
robot path $X_T = \{X_0, \dots, X_t\}$
control input U_t
Observation Z_t

Bayes theorem $P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}$

→ update probability based on new evidence

Concept

2D is a grid

← (R) → robot

1D

C	S	S	C	S	C	C	S	C
	0.1	0.2	0.4	0.2	0.1			

Kernel

unfolded map of a labyrinth

→ Prediction based on model of movement
Uptade → bayes → odometry X + landmarks Z

6. Navigation

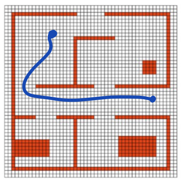
Reactive navigation without a map

Braitenberg vehicles
→ Direct connection between sensors and motors
→ Works but w/ not obstacles on the way
e.g. Robot follows a light source

Bug algorithm (need memory → state machine)
1. Create straight line between start and goal
2. Move along line in small inc.
3. If obstacle, turn right
4. Follow obstacle until line is crossed again

→ need Localization, holonomic

Occupancy grid
→ World into discrete cells
→ need memory



Distance transformation
1. Start at goal and calculate distance to neighbors
2. Repeat for neighbors
3. Take path of steepest gradient from start to finish

$2\sqrt{2}$	$1+\sqrt{2}$	2	$1+\sqrt{2}$	$2\sqrt{2}$
$1+\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$	$1+\sqrt{2}$
2	1	Goal	1	2
$1+\sqrt{2}$	$\sqrt{2}$	1	$\sqrt{2}$	$1+\sqrt{2}$
$2\sqrt{2}$	$1+\sqrt{2}$	2	$1+\sqrt{2}$	$2\sqrt{2}$

Variant : D^* → use a cost map to minimise time or energy

Similar : Fast marching, Dijkstra's

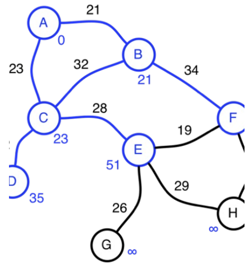
High cost to calculate path, only once

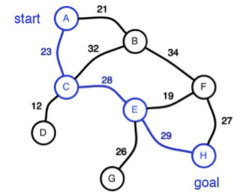
With a map
2 phases
1. Planning → analyze map
2. Query phase → find a path to goal

Roadmap methods
Probabilistic roadmap method PRM
→ Assign nodes on the occupancy grid
→ Determine shortest path along node lines
→ Often suboptimal

Rapidly exploring random tree RRT
→ Take into account robot kinematic
→ Create trees as node must keep orientation data

Topological map
Used for roadmap methods
→ use Dijkstra's algorithm
1. Start node is assigned 0, all others ∞
2. Calc. dist from A to neighbors
   → assign value to neighbors
3. Extend neighbor with lowest value
4. Update node value if shorter path is available
5. Expand like in 3. until all nodes have value





7. Mapping

With Kalman Filter
Assume robot is perfectly localized
State vector comprises the estimated pose of the M landmarks $\hat{X} = [{}^{G}L_{1x}, {}^{G}L_{1y}, \ldots, {}^{G}L_{Mx}, {}^{G}L_{My}]$

P of size $2M \times 2M$

1. Prediction (stationary) $\hat{X}_k^- = \hat{X}_{k-1}$ $\hat{P}_k^- = \hat{P}_{k-1}$
2. Update with new landmarks
   → extend state vector
   ⇒ insertion Jacobian

SLAM
Solve problem of Localization and mapping at the same time. "Chicken and egg problem"
→ State vector contain robot pose and Landmarks!
→ But not all errors are Gaussian!

FastSLAM : hold hypothesis of trajectories

Pose Graph SLAM : use nodes and a front end, back end software for robot

Sequential Monte-Carlo localization "Particle filter"
Estimator create randomly distributed particles
→ Resample to get the best state vector to explain reality

Multicopters, are cool!
z linear : thrust } altitude
z angular : yaw } Heading
x angular : roll
y angular : pitch } Attitude