Mobile Robots

Taekwon Ga

연세대학교
YONSEI UNIVERSITY

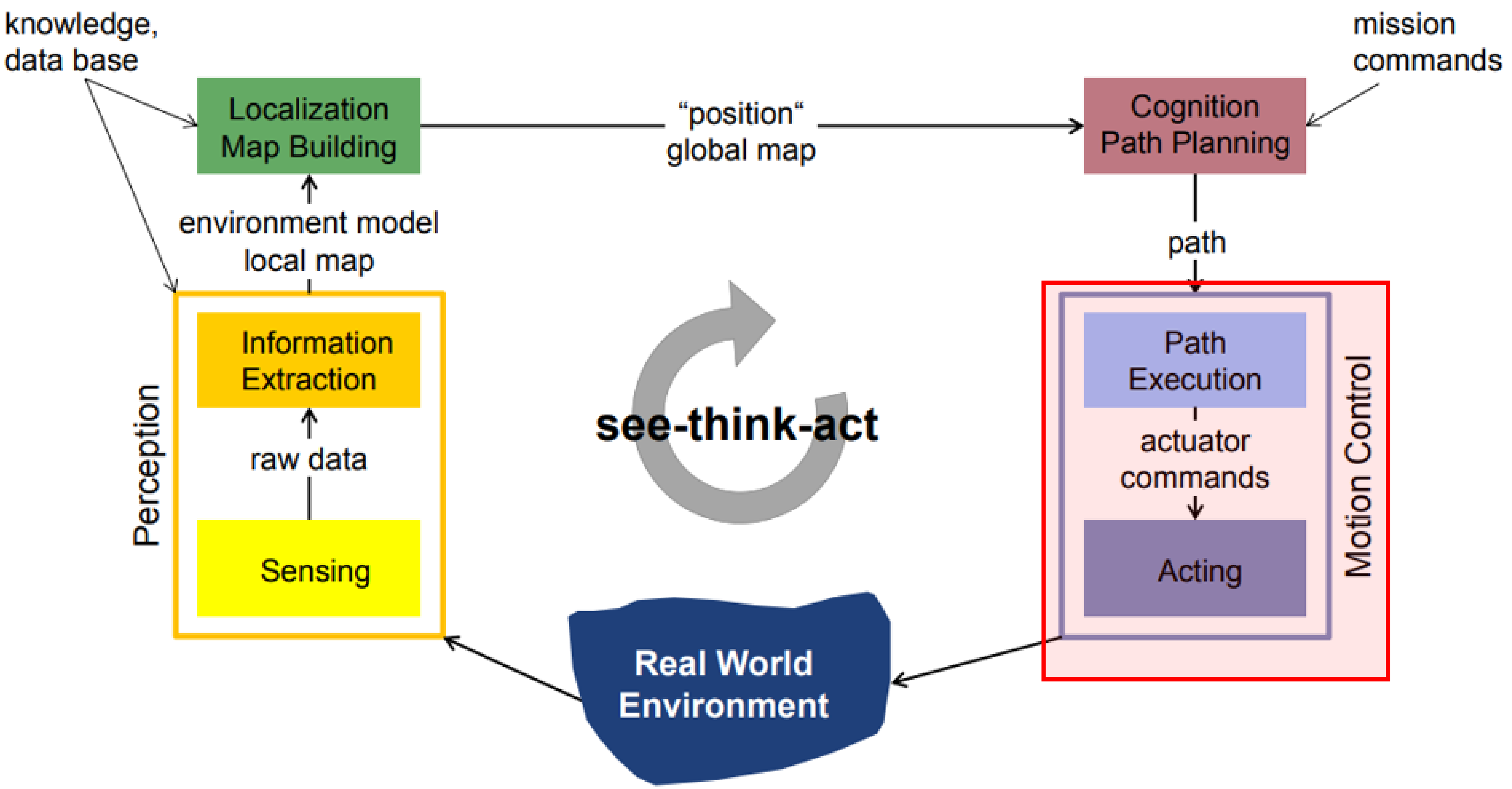
The Key questions in Autonomous Mobile Robotics

- Where am I ? → Localization
- Where am I going ?
- How do I get there ? → Path Planning

To answer these questions the robot has to

- have a model of the environment (given or autonomously built)
- perceive and analyze the environment
- find its position/situation within the environment
- plan and execute the movement

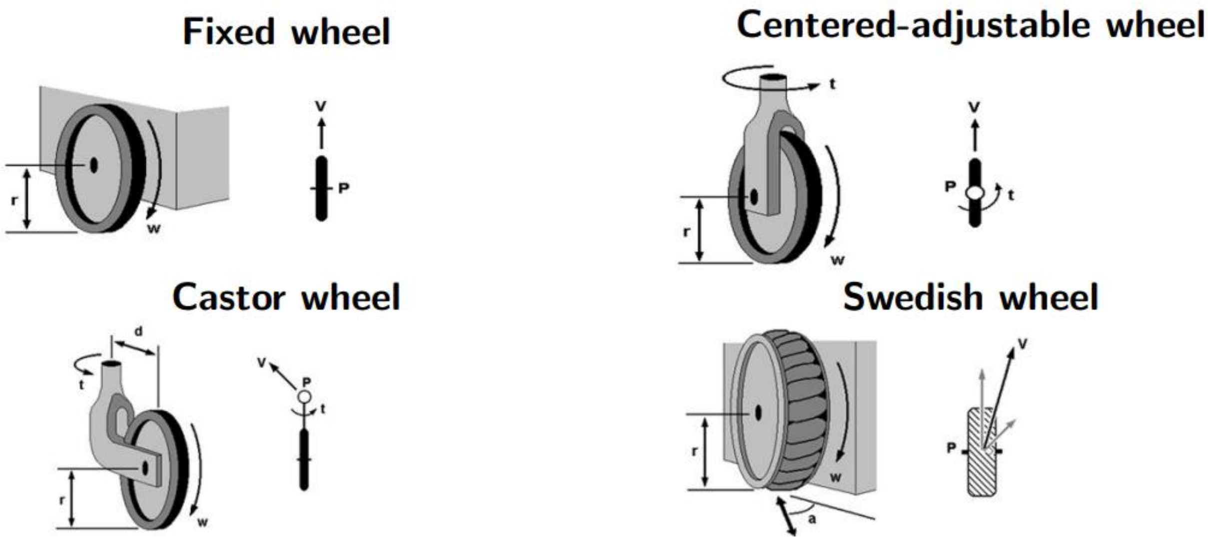
Kinematics and motion control



Kinematics and motion control

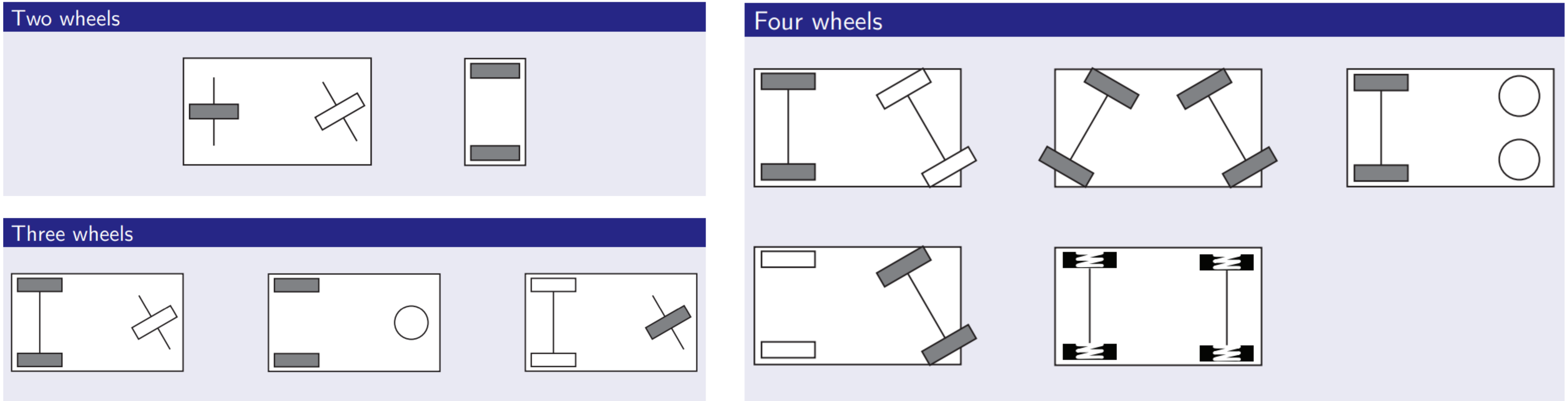
Wheel Types

- Three wheels are sufficient to guarantee the static stability of the vehicle
- rolling constraint, no-sliding constraint (lateral)
- What wheels to use? How many wheels to use?



Kinematics and motion control

Possible Configurations

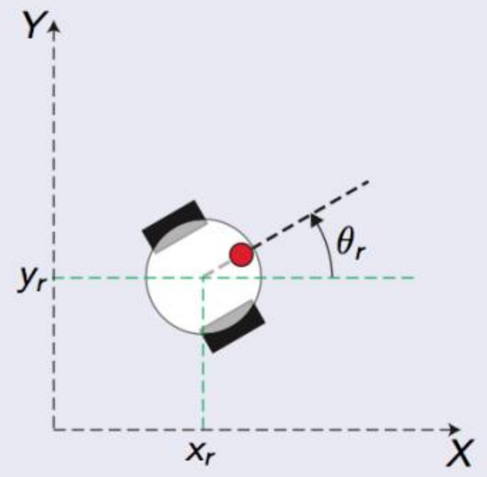


Kinematics and motion control

Configuration Space

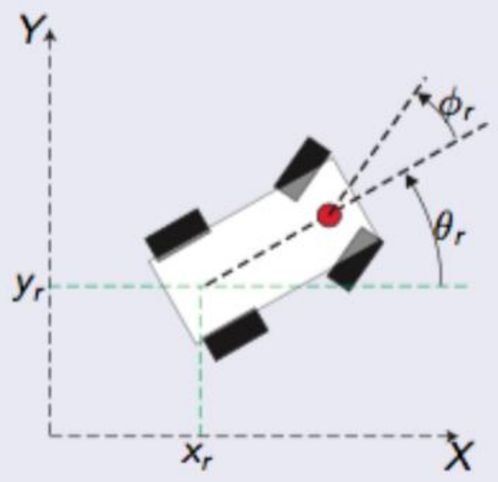
- It has dimensions equal to the number of parameters needed to uniquely describe the configuration of a mobile robot.
- Equivalent to the Joint Space for manipulators.

Unicycle



$$q = [x_r, y_r, \theta_r]^T \in \mathbb{R}^3$$

Bicycle



$$q = [x_r, y_r, \theta_r, \phi_r]^T \in \mathbb{R}^4$$

Kinematics and motion control

Kinematic model of a WMR(wheeled mobile robot)

General formulation

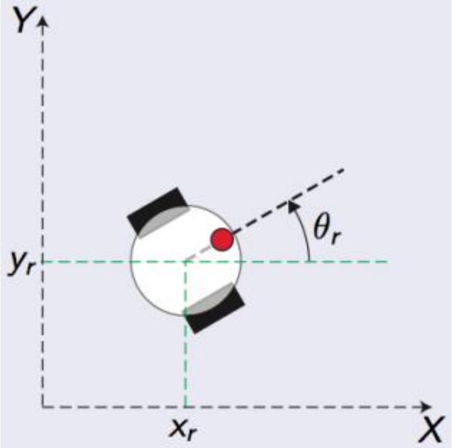
$$\dot{q} = G(q)v$$

- It represents the allowable directions of motion in the configuration space (allowable velocities)
- It is required to deal with common problems of mobile robotics (Navigation, Localization, etc.)

Kinematics and motion control

Kinematic model of the unicycle model

* Unicycle model : A motorcycle is a vehicle with a single adjustable wheel



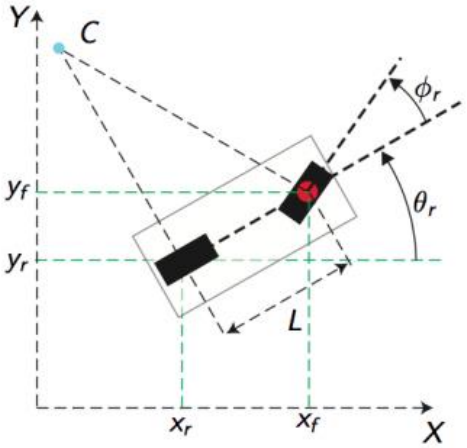
$$\dot{q} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

- The configuration is described by $q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$
- v is the linear velocity of the contact point, ω is the angular velocity of the robot
- Differential drive is the most popular unicycle type.

Kinematics and motion control

Kinematic model of the bicycle model

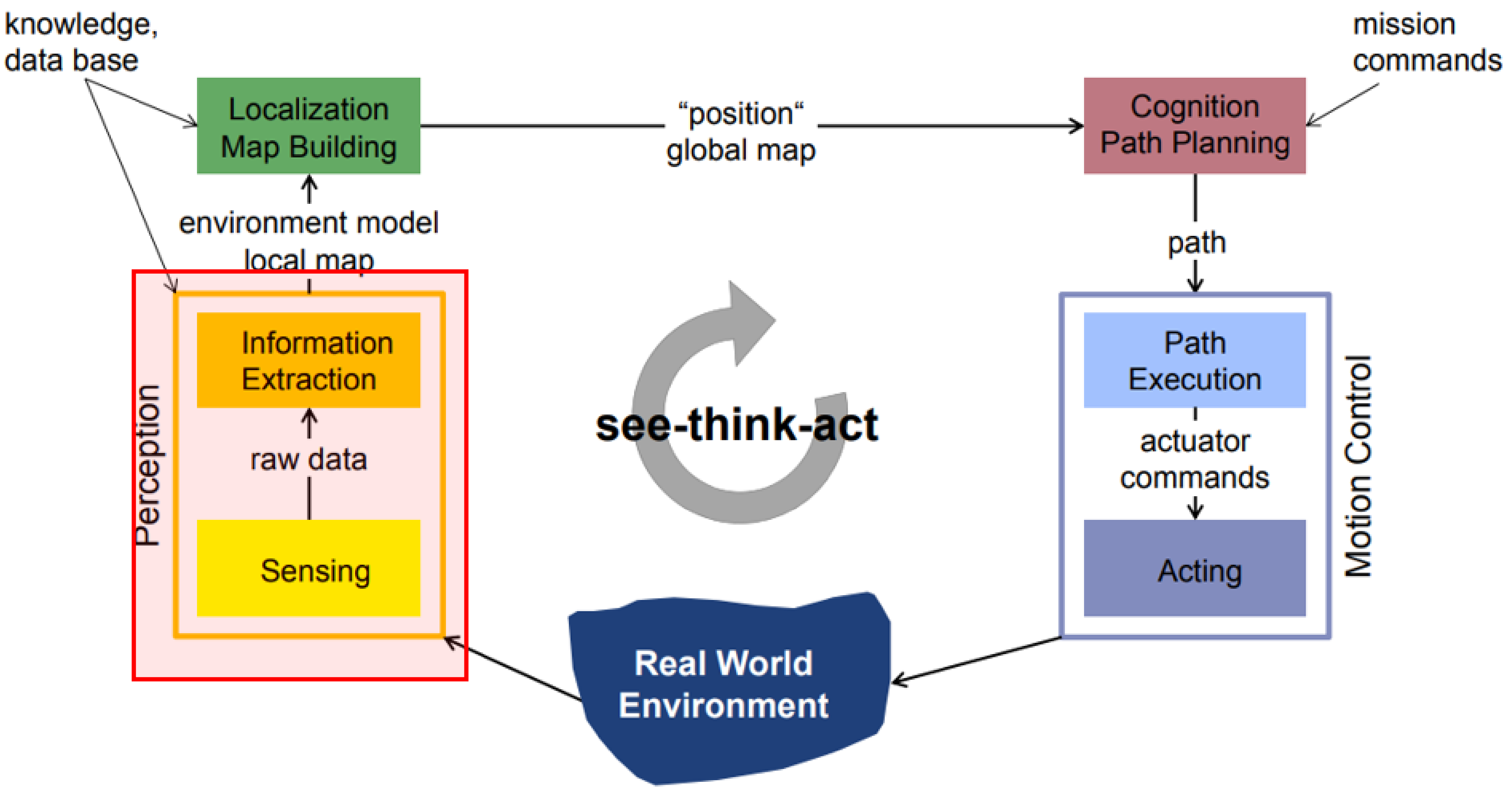
* Bicycle model : A bicycle is a vehicle having a caster (adjustable wheel) and a fixed wheel with their rotation axes perpendicular to the longitudinal plane.



$$\dot{q} = \begin{bmatrix} \cos\theta_r \cos\phi_r \\ \sin\theta_r \cos\phi_r \\ \frac{1}{L}\sin\phi_r \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \omega = \begin{bmatrix} \cos\theta_r \cos\phi_r & 0 \\ \sin\theta_r \cos\phi_r & 0 \\ \frac{1}{L}\sin\phi_r & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

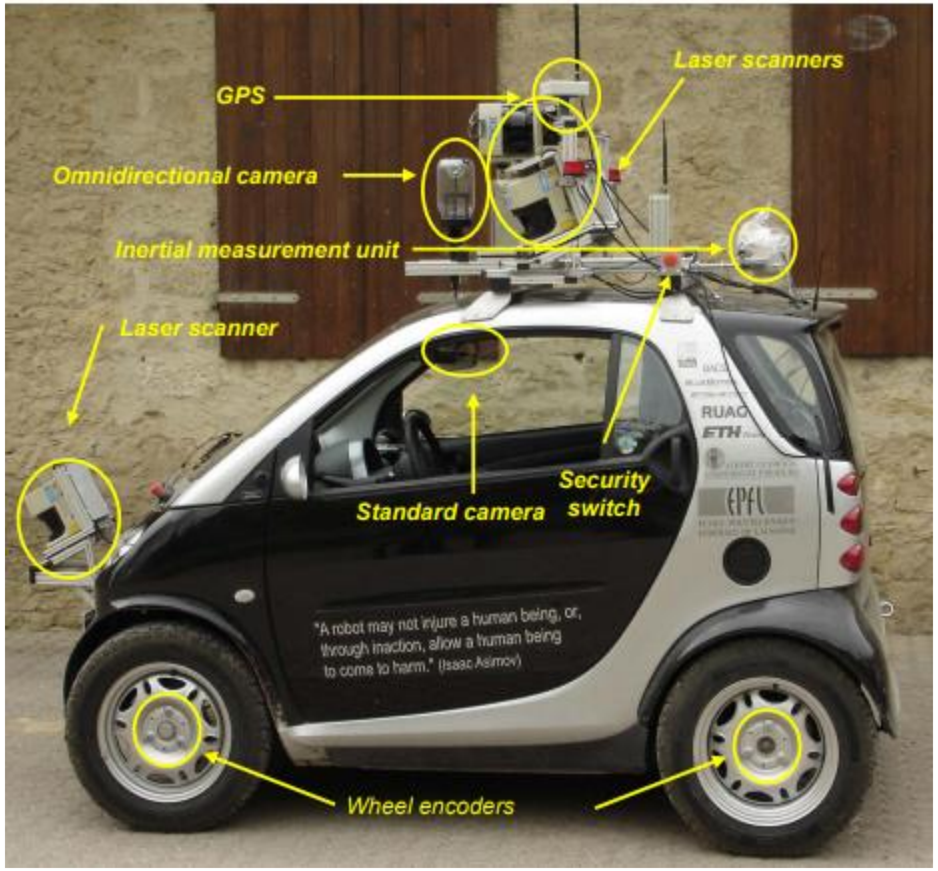
- The configuration is described by $q = \begin{bmatrix} x \\ y \\ \theta \\ \phi \end{bmatrix}$
- The model vary depending on the reference point
- In practice kinematically equivalent structures but more stable from a mechanical point of view are used.(e.g. Car-like model, Tricycle model)

Perception

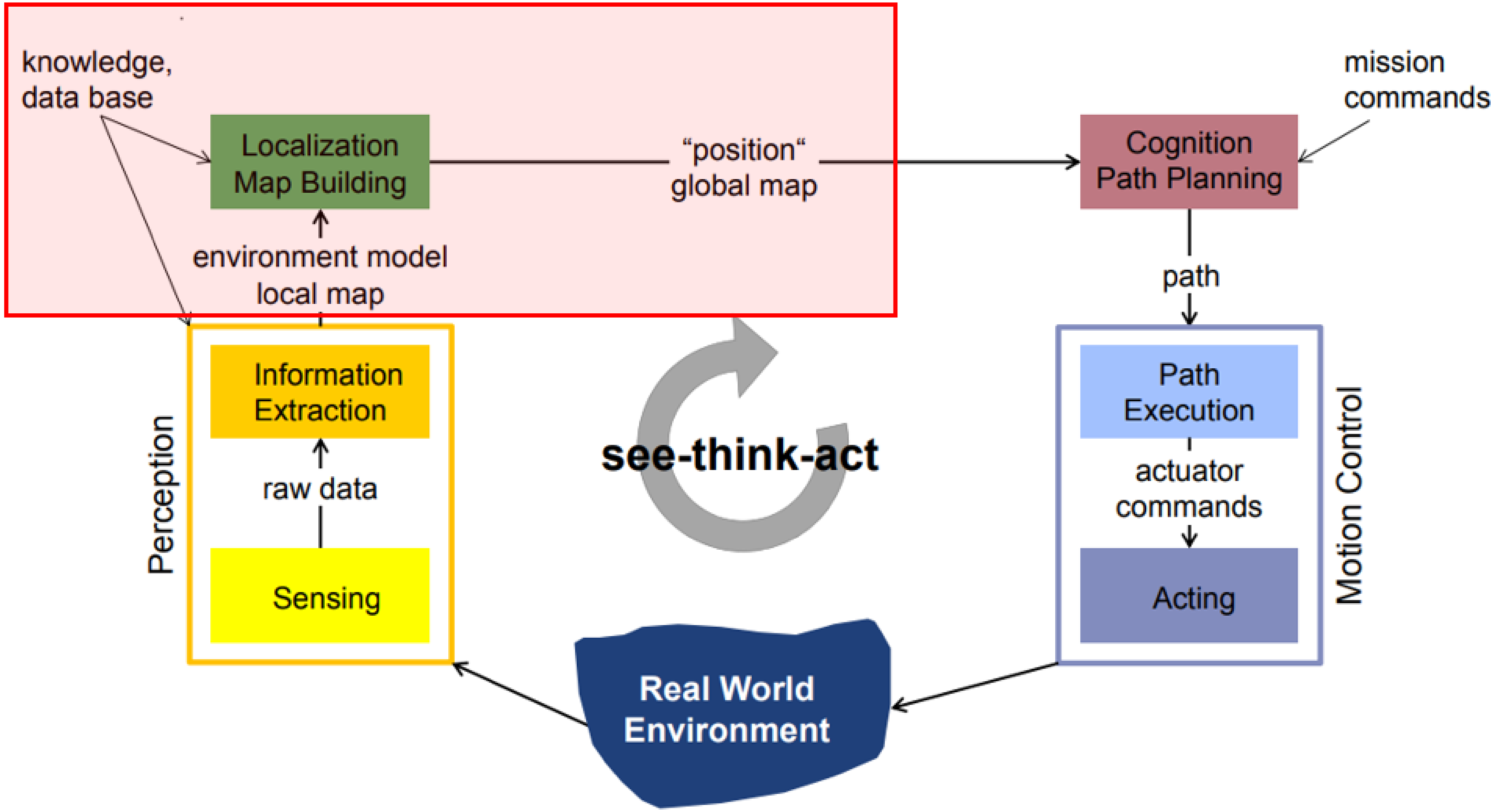

knowledge,
data base
Localization
Map Building
"position"
global map
Cognition
Path Planning
mission
commands
environment model
local map
path
Perception
Information
Extraction
raw data
Sensing
see-think-act
Path
Execution
actuator
commands
Acting
Motion Control
Real World
Environment

Sensors for Perception

- **Tactile sensors or bumpers**
  - ✓ Detection of physical contact, security switches
- **GPS**
  - ✓ Global localization and navigation
- **Inertial Measurement Unit (IMU)**
  - ✓ Orientation and acceleration of the robot
- **Wheel encoders**
  - ✓ Local motion estimation (odometry)
- **Laser scanners**
  - ✓ Obstacle avoidance, motion estimation, scene interpretation (road detection, pedestrians)
- **Cameras**
  - ✓ Texture information, motion estimation, scene interpretation



Localization

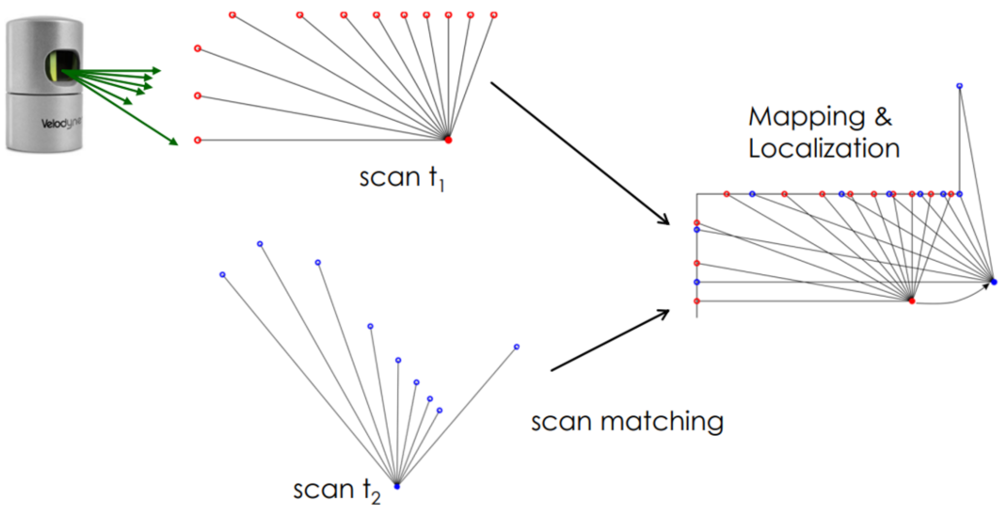


Localization

- Map–based localization
  - ✓ The robot estimates its position using perceived information and a map
  - ✓ The map
    - might be known (**localization**)
    - Might be built in parallel (**simultaneous localization and mapping – SLAM**)

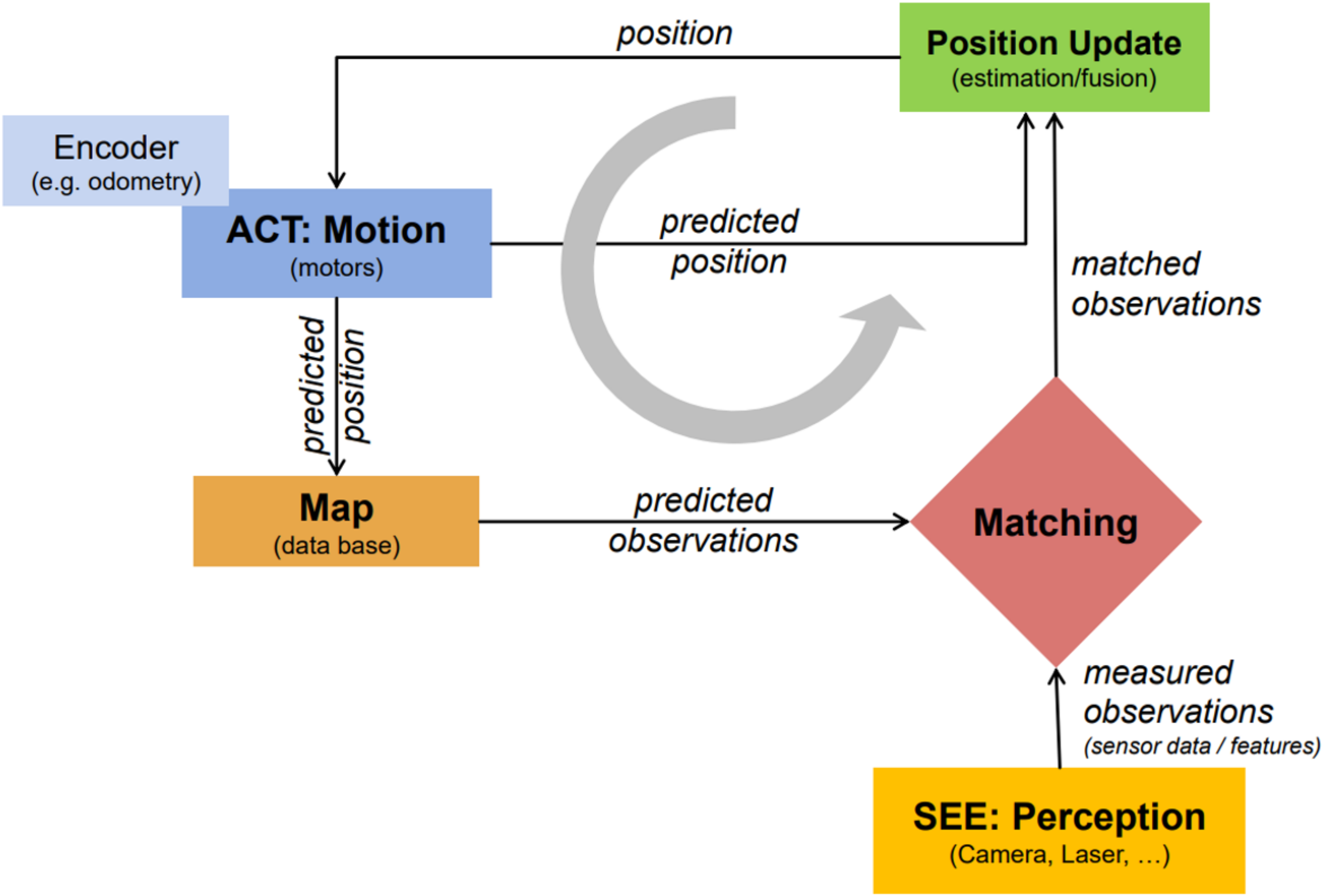
- Challenges
  - ✓ Measurements and the map are inherently error prone
  - ✓ Thus, the robot has to deal with uncertain information
    - → **Probabilistic map–based localization**

- Approach
  - ✓ The robot estimates the belief state about its position through an SEE and ACT cycle



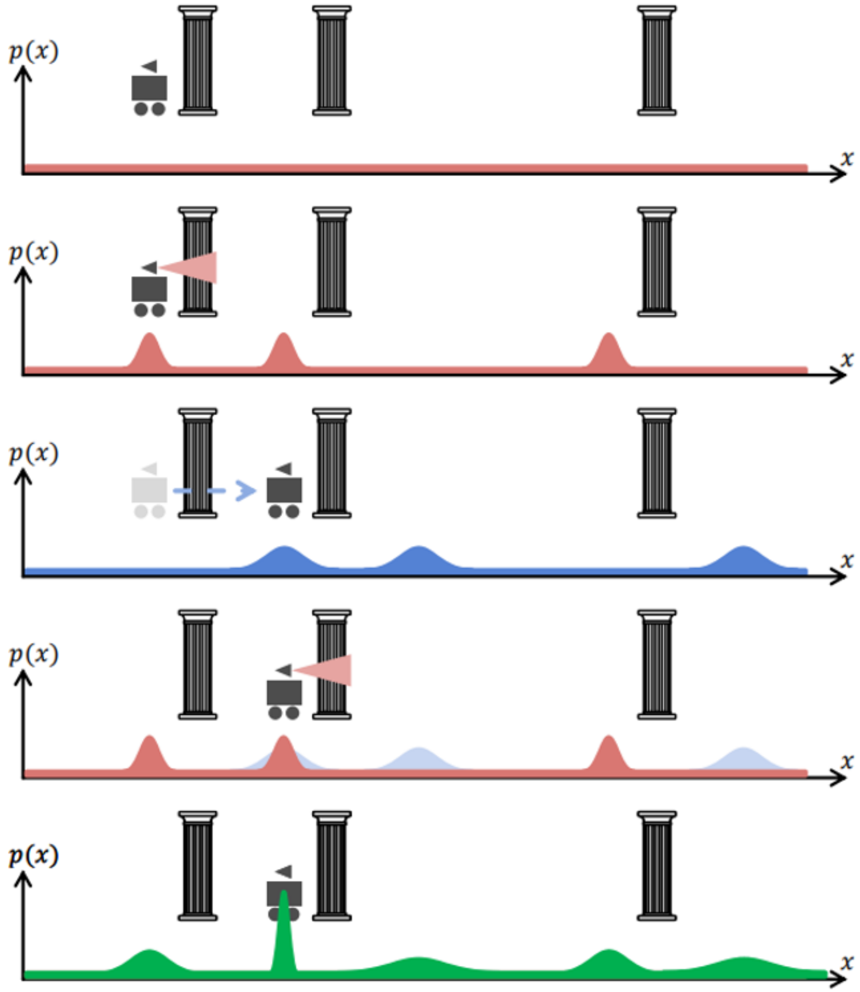
scan matching process of SLAM

Localization : the estimation cycle (ACT-SEE)



Localization : the estimation cycle (ACT-SEE)

- SEE: The robot queries its sensors
  → finds itself next to a pillar
- ACT: Robot moves one meter forward
  - ✓ motion estimated by wheel encoders
  - ✓ accumulation of uncertainty
- SEE: The robot queries its sensors again
  → finds itself next to a pillar
- Belief update (information fusion)



Bayes Filter

- ACT(motion model) : probabilistic estimation of the robot's new belief state $\overline{bel}(x_t)$based on the previous location $\mathrm{bel}(x_{t-1})$ and the probabilistic motion model $p(x_t|u_t, x_{t-1})$ with action u_t (control input).

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})\, dx_{t-1}$$ for continuous probabilities

$$\overline{bel}(x_t) = \sum_{x_{t-1}} p(x_t|u_t, x_{t-1})bel(x_{t-1})$$ for discrete probabilities

* Theorem of total probability

$$p(x) = \sum_y p(x|y)p(y)$$ for discrete probabilities

$$p(x) = \int_y p(x|y)p(y)dy$$ for continuous probabilities

Bayes Filter

- SEE(observation model) : probabilistic estimation of the robot's new belief state $\mathrm{bel}(x_t)$ as a function of its measurement data z_t and its former belief state $\overline{bel}(x_t)$.

$$bel(x_t) = \eta p(z_t|x_t, M)\overline{bel}(x_t)$$

* The Bayes rule

$$p(x|y) = \eta p(y|x)p(x)$$

$\eta = p(y)^{-1}$ normalization factor ($\int p = 1$)

Bayes Filter

- Probability theory is widely and very successfully used for mobile robot localization.
- There are many methods for probability-based localization that apply Bayes filters.
  - ✓ Kalman Filter (KF, EKF, UKF)
  - ✓ Particle Filter (MCL, AMCL)
  - ✓ Histogram Filter
  - ✓ FastSLAM
- Further reading:
  - ✓ "**Probabilistic Robotics**," Thrun, Fox, Burgard, MIT Press, 2005.
  - ✓ "Introduction to Autonomous Mobile Robots", Siegwart, Nourbakhsh, Scaramuzza, MIT Press 2011

❖ Exercise 1:

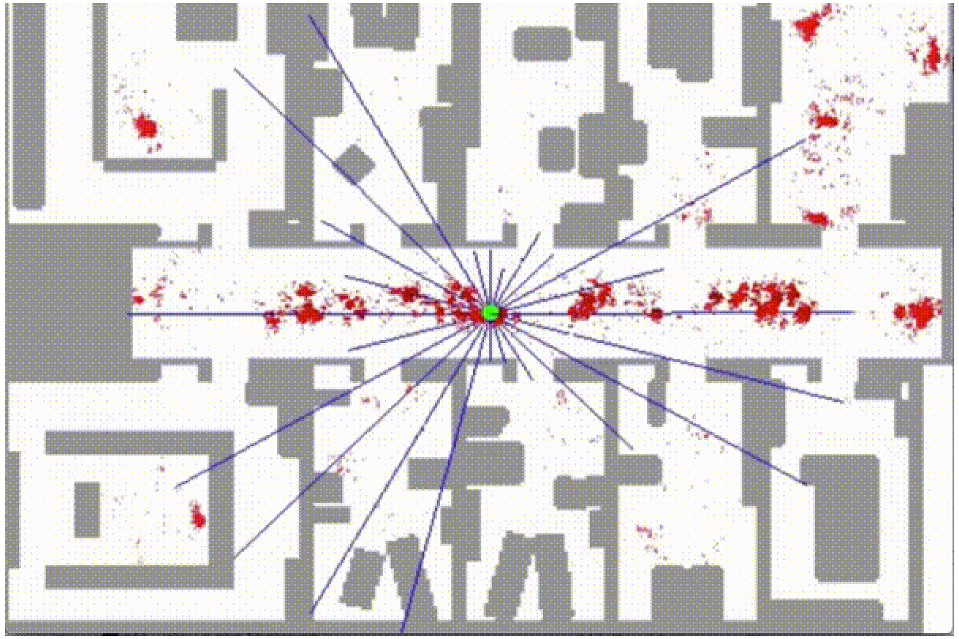
Describe 3 Bayes filter-based localization methods with their features, advantages, and disadvantages

Particle Filter

- The Kalman and Particle filters are algorithms that **recursively update an estimate of the state** and find the innovations driving a stochastic process given a sequence of observations. The Kalman filter accomplishes this goal by **linear projections**, while the Particle filter does so by a **sequential Monte Carlo method.**

- The Kalman filter relies on the linearity and normality assumptions. However, many models are non-linear and/or non-gaussian.

- Unlike Kalman filter, particle filter can handle non-linear dynamics and non-Gaussian noise distributions

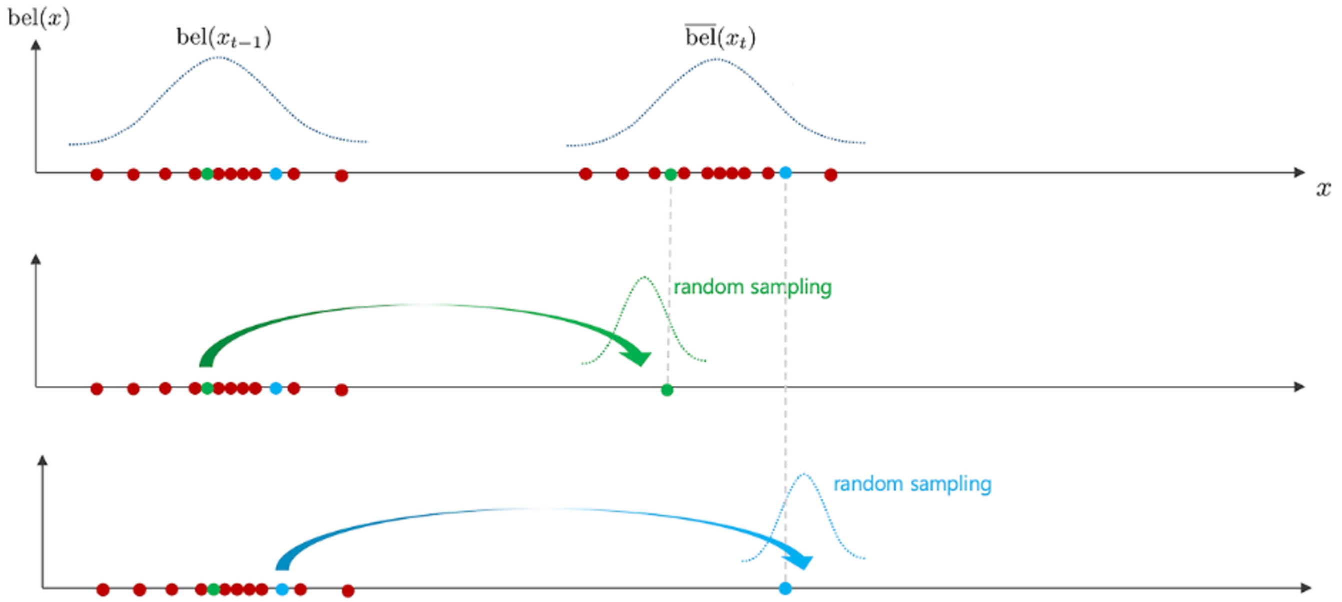


Particle Filter

1. Prediction Step

For each particle, predict the next state using the system model.

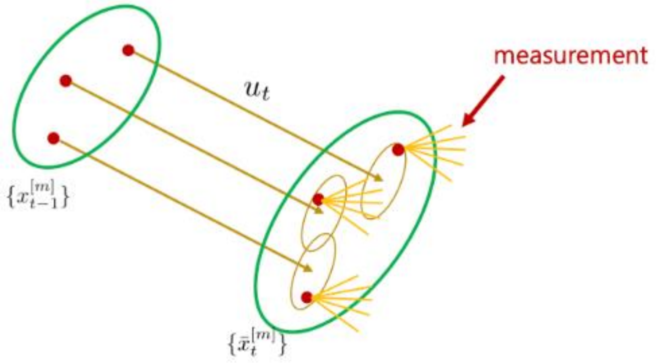
$$
\boxed{\begin{array}{l} \text{for } m = 1 \text{ to } M : \\ \quad \text{sample } \bar{x}_t^{[m]} \sim p(x_t | x_{t-1}^{[m]}, u_t) \end{array}}
$$

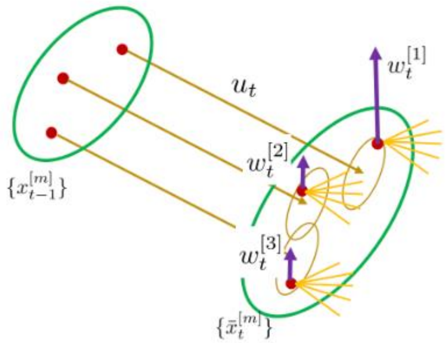


Particle Filter

2. Correction Step

$x_t = \{\}$ empty set
for $m = 1$ to M :
$\quad w_t^{[m]} = p(z_t | x_t^{[m]})$
$\quad$ resample $\bar{x}_t^{[m]}$ with probability $\propto w_t^{[m]}$
$\quad x_t = x_t \cup \{\bar{x}_t^{[m]}\}$



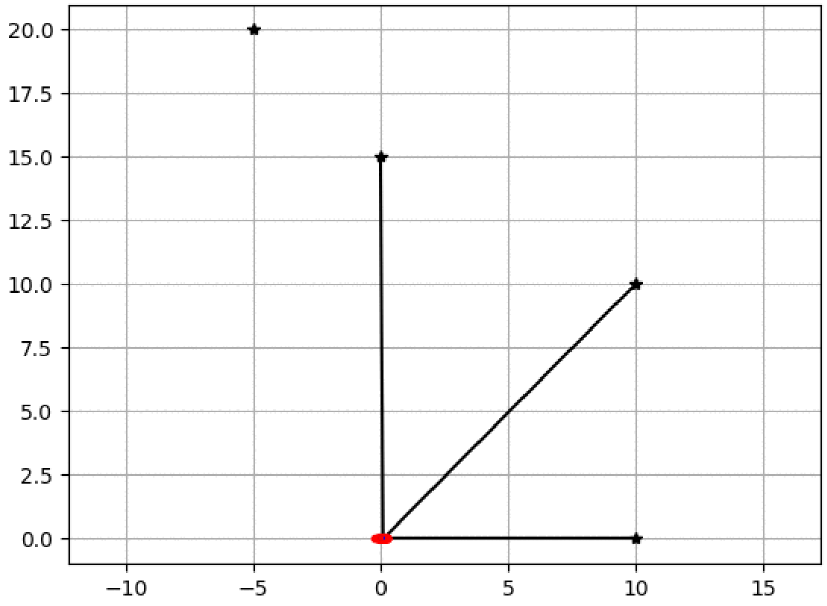


1. For each particle, calculate the weight based on the likelihood of the observed measurement z_t given the predicted state $x_t^{[m]}$

2. Resample M particles according to their weights. (SIR, Sequantial Importance Sampling)

Python Example : Particle Filter

Particle Filter example with 4 RFID sensors, the robot can measure a distances from 4 RFID sensors.



$$s = \begin{bmatrix} x \\ y \\ v \\ \psi \end{bmatrix}, \qquad z = \begin{bmatrix} d \\ x_{sensor} \\ y_{sensor} \end{bmatrix}$$

true trajectory
dead reckoning trajectory
estimated trajectory with PF

Python Example : Particle Filter

Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

for $m = 1$ to M do

  sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$

  $w_t^{[m]} = p(z_t \mid x_t^{[m]})$

  $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

endfor

for $m = 1$ to M do

  draw i with probability $\propto w_t^{[i]}$

  add $x_t^{[i]}$ to $\mathcal{X}_t$

endfor

return $\mathcal{X}_t$

```python
def pf_localization(px, pw, z, u):
    """
    Localization with Particle filter
    """

    for ip in range(NP):
        x = np.array([px[:, ip]]).T
        w = pw[0, ip]

        #  Predict with random input sampling
        ud1 = u[0, 0] + np.random.randn() * R[0, 0] ** 0.5
        ud2 = u[1, 0] + np.random.randn() * R[1, 1] ** 0.5
        ud = np.array([[ud1, ud2]]).T
        x = motion_model(x, ud)

        #  Calc Importance Weight
        for i in range(len(z[:, 0])):
            dx = x[0, 0] - z[i, 1]
            dy = x[1, 0] - z[i, 2]
            pre_z = math.hypot(dx, dy)
            dz = pre_z - z[i, 0]
            w = w * gauss_likelihood(dz, math.sqrt(Q[0, 0]))

        px[:, ip] = x[:, 0]
        pw[0, ip] = w

    pw = pw / pw.sum()  # normalize

    x_est = px.dot(pw.T)
    p_est = calc_covariance(x_est, px, pw)

    N_eff = 1.0 / (pw.dot(pw.T))[0, 0]  # Effective particle number
    if N_eff < NTh:
        px, pw = re_sampling(px, pw)
    return x_est, p_est, px, pw
```

Python Example : Particle Filter

Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):

$\bar{\mathcal{X}}_t = \emptyset$

$r = \text{rand}(0; M^{-1})$

$c = w_t^{[1]}$

$i = 1$

for $m = 1$ to M do

  $U = r + (m-1) \cdot M^{-1}$

  while $U > c$

    $i = i + 1$

    $c = c + w_t^{[i]}$

  endwhile

  add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$

endfor

return $\bar{\mathcal{X}}_t$

```python
def re_sampling(px, pw):
    """
    low variance re-sampling
    """

    w_cum = np.cumsum(pw)
    base = np.arange(0.0, 1.0, 1 / NP)
    re_sample_id = base + np.random.uniform(0, 1 / NP)
    indexes = []
    ind = 0
    for ip in range(NP):
        while re_sample_id[ip] > w_cum[ind]:
            ind += 1
        indexes.append(ind)

    px = px[:, indexes]
    pw = np.zeros((1, NP)) + 1.0 / NP  # init weight

    return px, pw
```

Python Example : Particle Filter

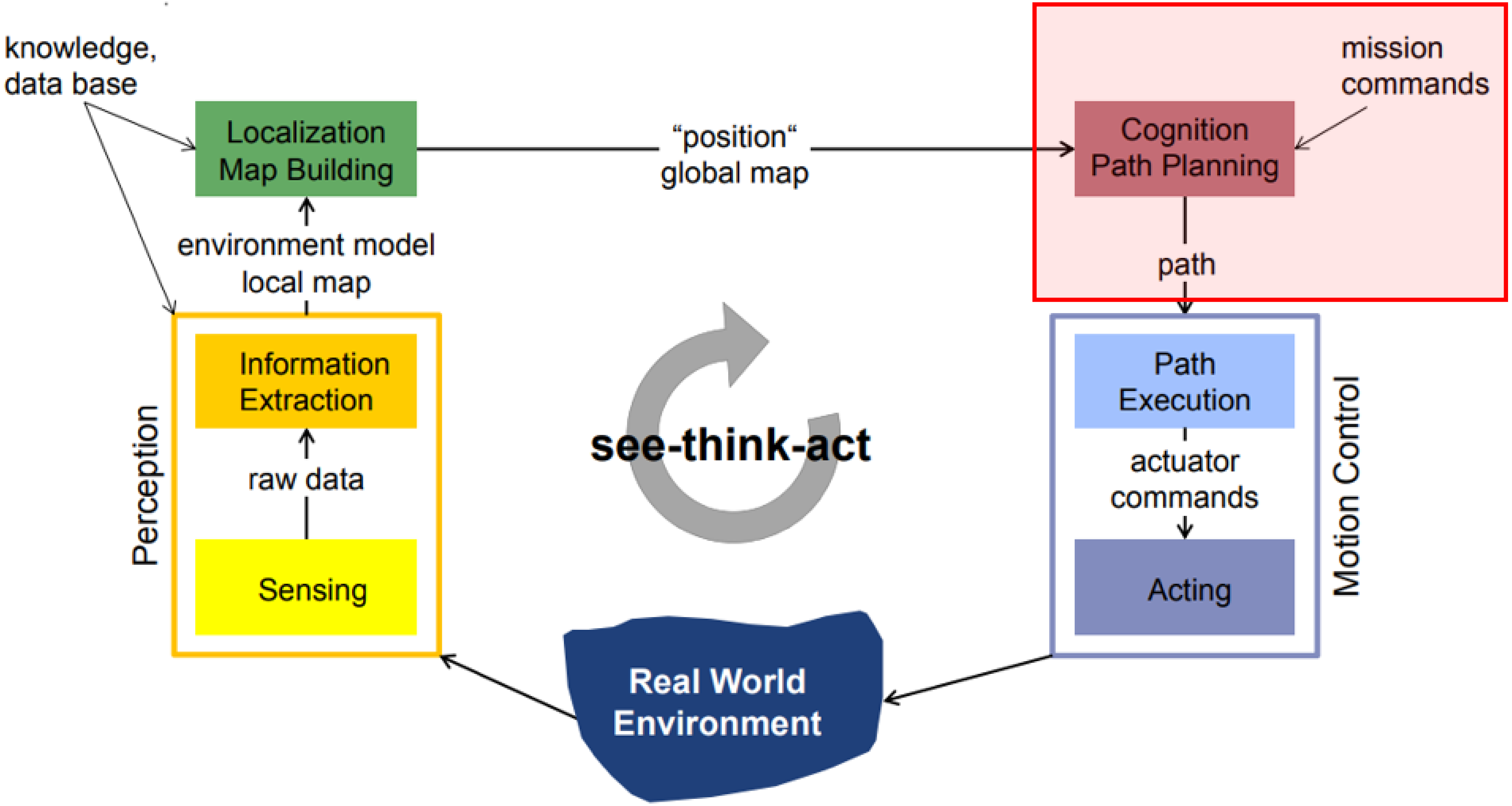
❖ Exercise 2:

Implement a Particle Filter assuming that data is received from GPS sensors instead of RFID sensors.

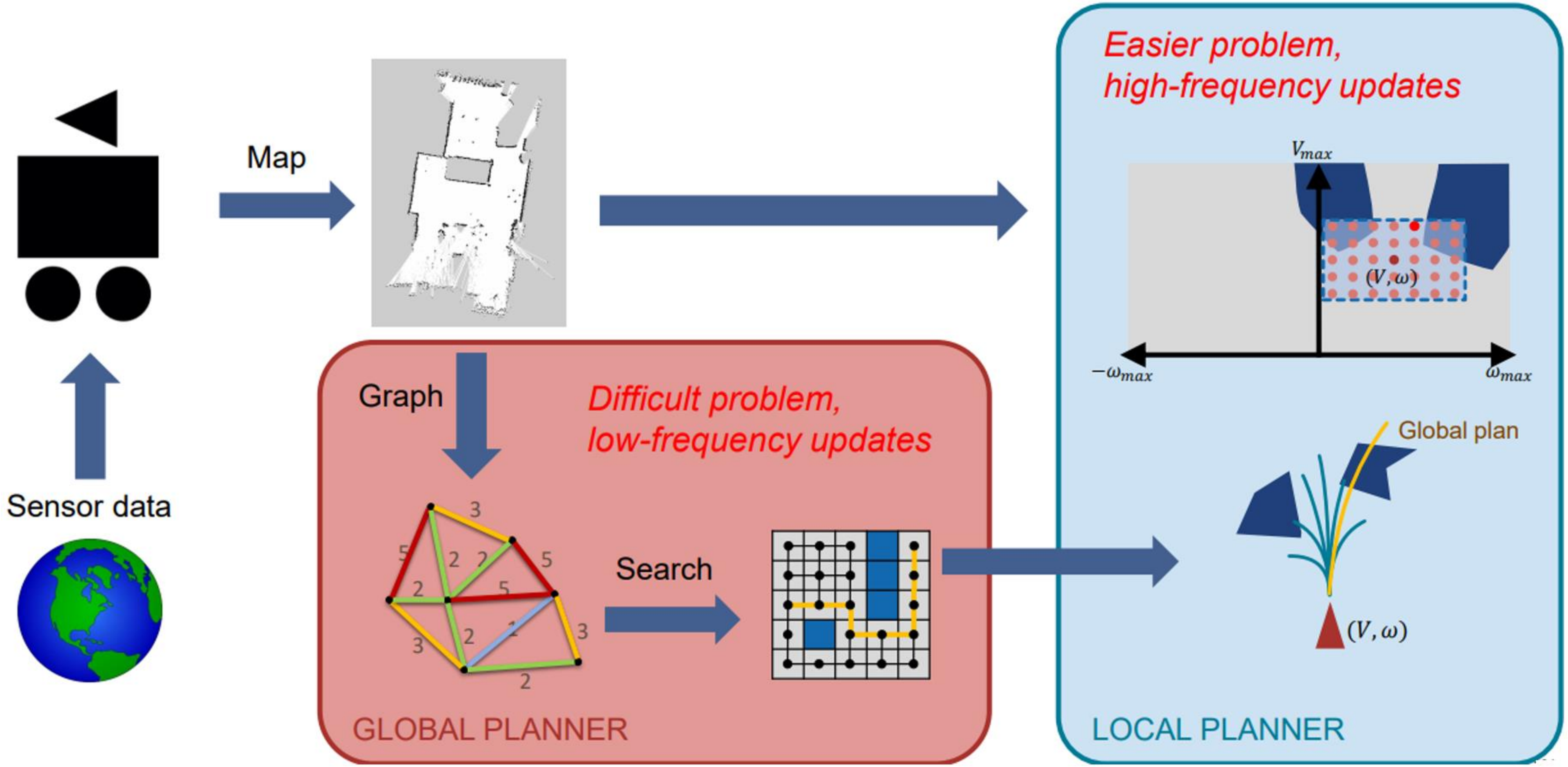
- ✓ The GPS can measure the robot's x and y coordinates.
- ✓ Covariance matrix of the GPS $\Sigma = \begin{bmatrix} 1^2 & 0 \\ 0 & 1^2 \end{bmatrix}$.
- ✓ Assume that the sensor model follows a Multivariate Gaussian Distribution.

$$p(z_t|x_t) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(z_t - h(x_t))^T \Sigma^{-1} (z_t - h(x_t))\right)$$

Path Planning



Path Planning Hierarchy for Mobile Robots

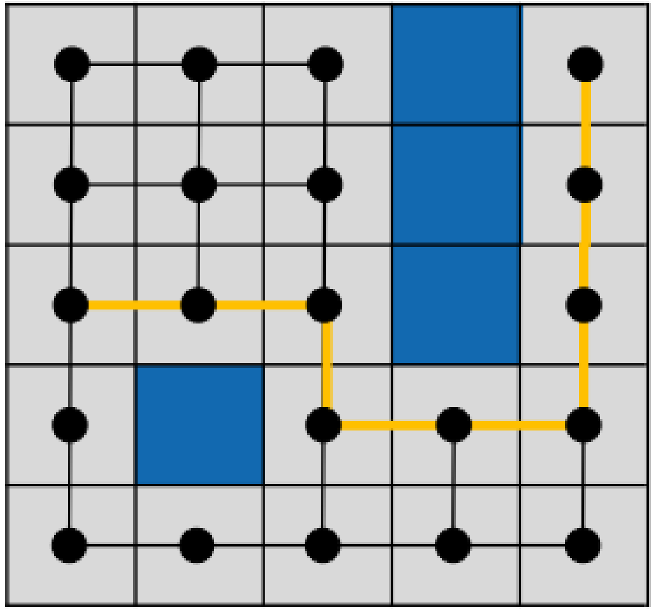


Path Planning Hierarchy for Mobile Robots

Global Path Planning	Local Path Planning
• To set the overall path from the start to the goal	• To help the robot follow the global path while avoiding obstacles and optimizing its movement in real-time
• global View	• local view
• low frequency	• high frequency
• static environment	• dynamic environment

Global Path Planning : Grid-based Planning

- Finding the shortest path can be treated as a graph search problem
- Suffer from poor scaling in higher dimensions
- Algorithms
  1. Dijkstra's algorithm
  2. A* Algorithm
  3. Breadth-first search
  4. Depth-first search



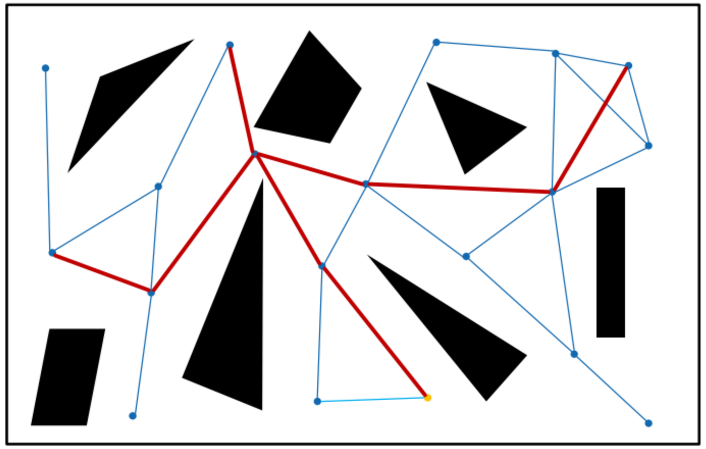
A* is very commonly used in robot planning, especially for low-dimensional state spaces.

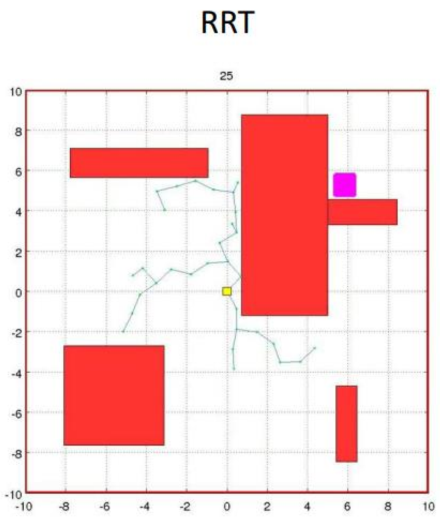
Global Path Planning : Sampling-based Planning

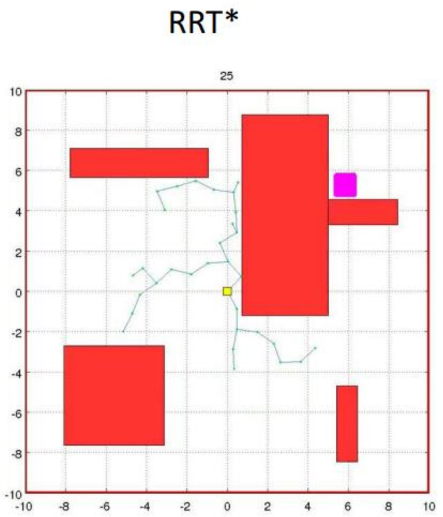
- Finding a path by randomly sampling the robot's configuration space

- efficient in finding paths in high-dimensional spaces, making them suitable for robots with many degrees of freedom

- Algorithms
  1. RRT
  2. RRT*
  3. PRM

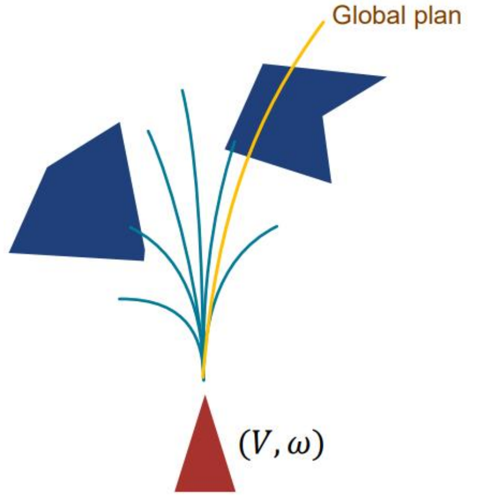


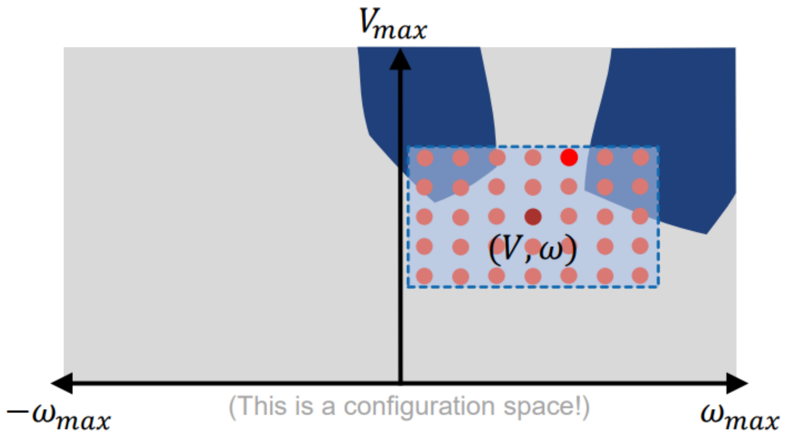




Local Path Planning : Dynamic Window Approach

- Generating full time-varying trajectories for $V(t)$ and $\omega(t)$ is still very challenging
- If we assume $(V,\ \omega)$ are constant for a fixed Δt, each local path in the future is a circular arc segment
- This can be easily considered as a type of velocity configuration space
- Maximise utility metric (usually maximise speed, minimize distance to goal, maximise distance from obstacles) across configuration samples





e.g.

Score:

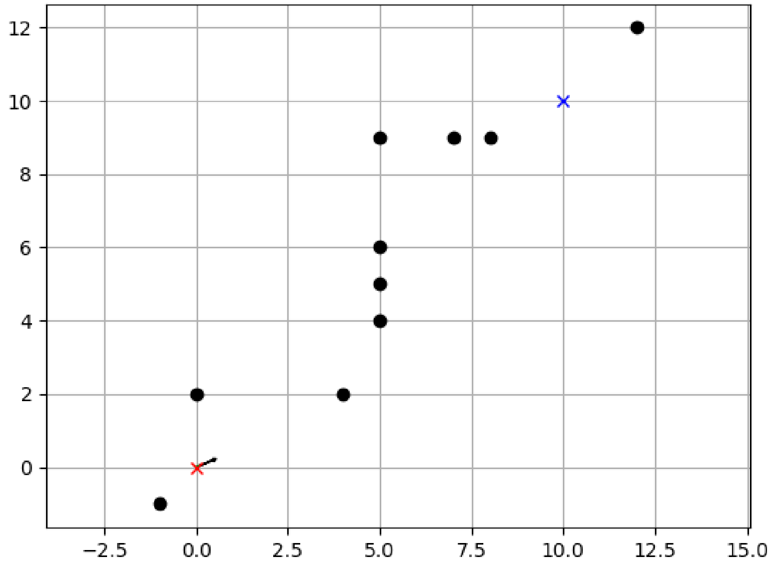
$$G(v,\omega) = \alpha\,\text{heading}(v,\omega) + \beta\,\text{dist}(v,\omega) + \gamma\,\text{velocity}(v,\omega)$$

Python Example : Dynamic Window Approach

- This is a simple 2D navigation code with Dynamic Window Approach.(No global path planning!)

- red cross is a position of the robot and blue cross is a position of the goal.

- green line is a predicted trajectory calculated by dynamic window approach controller



Python Example : Dynamic Window Approach

Algorithm:

1. Sample feasible inputs
2. For each feasible input:
    a. Simulate trajectory over horizon
    b. Score trajectory
3. Pick control input that leads to best score

```python
def calc_dynamic_window(x, config):
    """
    calculation dynamic window based on current state x
    """

    # Dynamic window from robot specification
    Vs = [config.min_speed, config.max_speed,
          -config.max_yaw_rate, config.max_yaw_rate]

    # Dynamic window from motion model
    Vd = [x[3] - config.max_accel * config.dt,
          x[3] + config.max_accel * config.dt,
          x[4] - config.max_delta_yaw_rate * config.dt,
          x[4] + config.max_delta_yaw_rate * config.dt]

    #  [v_min, v_max, yaw_rate_min, yaw_rate_max]
    dw = [max(Vs[0], Vd[0]), min(Vs[1], Vd[1]),
          max(Vs[2], Vd[2]), min(Vs[3], Vd[3])]

    return dw
```

```python
# evaluate all trajectory with sampled input in dynamic window
for v in np.arange(dw[0], dw[1], config.v_resolution):
    for y in np.arange(dw[2], dw[3], config.yaw_rate_resolution):

        trajectory = predict_trajectory(x_init, v, y, config)
        # calc cost
        to_goal_cost = config.to_goal_cost_gain * calc_to_goal_cost(trajectory, goal)
        speed_cost = config.speed_cost_gain * (config.max_speed - trajectory[-1, 3])
        ob_cost = config.obstacle_cost_gain * calc_obstacle_cost(trajectory, ob, config)

        final_cost = to_goal_cost + speed_cost + ob_cost
```

❖ Exercise 3:

Why don’t we use only a local path planner? Explain why global planning needs to be incorporated.