ETH Zürich
Chair of Software Engineering

Robotics Programming Laboratory

Bertrand Meyer
Jiwon Shin

Lecture 7: Mapping and SLAM

This lecture is based on “Probabilistic Robotics” by Thrun, Burgard, and Fox (2005) and “Introduction to Autonomous Mobile Robots” by Siegwart, Nourbakhsh, and Scaramuzza (2011).

Mapping

Map: a list of objects and their locations in an environment

- Given N objects in an environment

  $m = \{ m_1, \dots, m_N \}$

Mapping: the process of creating a map

Types of Maps

Lacation-based map

- $m = \{ m_1, \dots , m_N \}$ contains N locations
- Volumetric representation
    - A label for any location in the world
    - Knowledge of presence and absence of objects

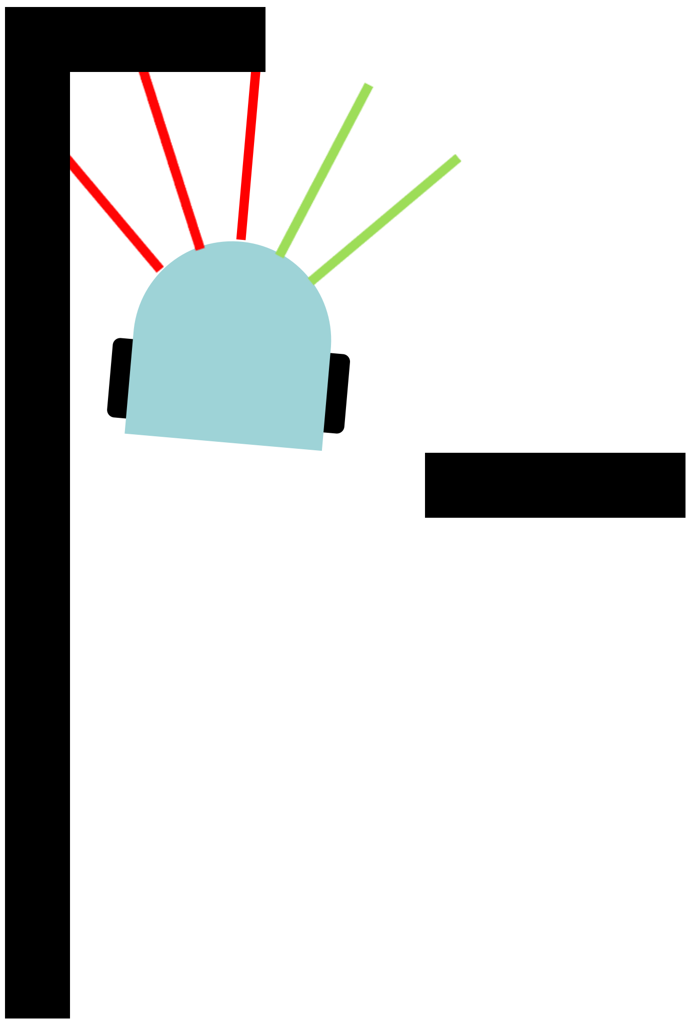
Feature-based map

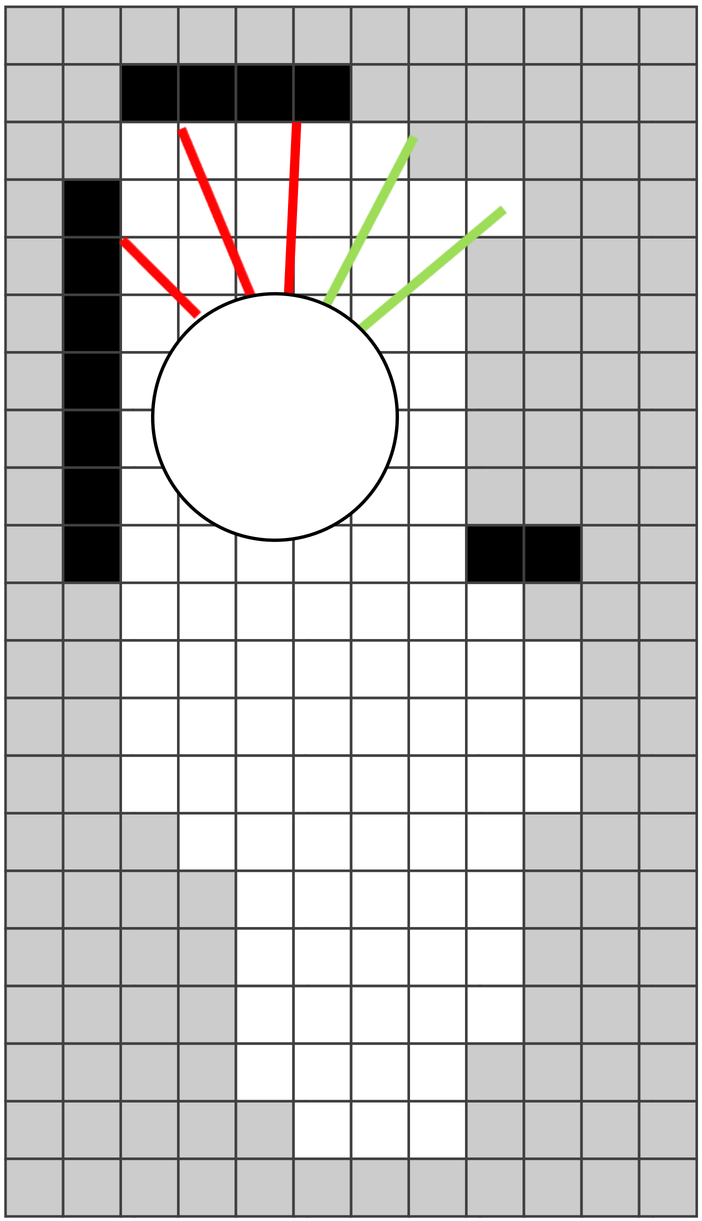
- $m = \{ m_1, \dots , m_N \}$ contains N features
- Sparse representation
    - A label for each object location
    - Easier to adjust the position of an object

Occupancy grid map

- Location-based map
- An environment as a collection of grid cells
- Each grid cell with a probability value that the cell is occupied
- Easy to combine different sensor scans and different sensor modalities
- No assumption about type of features

Occupancy grid mapping





Occupancy grid cells

$\mathbf{m}_i$: the grid cell with index i

z_t: the measurement at time t

x_t: the robot's pose (x, y, θ) at time t

$p(\mathbf{m}_i \mid z_t, x_t)$: probability of occupancy

$$\frac{p(\mathbf{m}_i \mid zt, xt)}{p(\neg \mathbf{m}_i \mid zt, xt)} = \frac{p(\mathbf{m}_i \mid z_t, xt)}{1 - p(\mathbf{m}_i \mid z_t, xt)}$$: odds of occupancy

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid z_t, xt)}{1 - p(\mathbf{m}_i \mid z_t, xt)}$$: log odds of occupancy

$$p(\mathbf{m}_i \mid z_t, x_t) = 1 - \frac{1}{1 + \exp(lt_{,i})}$$

Bayes' law using log odds

$$p(A|B) = \frac{p(B|A)\, p(A)}{p(B)}$$

$$p(\neg A|B) = \frac{p(B|\neg A)\, p(\neg A)}{p(B)}$$

$$o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A)\, p(A)}{p(B|\neg A)\, p(\neg A)} = \lambda(B|A)\, o(A)$$

$$\log(\, o(A|B)\,) = \log(\, \lambda(B|A)\,) + \log(\, o(A)\,)$$

- Ranges between $-\infty$ and ∞
- Avoids truncation problem around probabilities near 0 and 1

Occupancy grid mapping

```
occupancy_grid_mapping ( x: ROBOT_POSE;
                         z: SENSOR_MEASUREMENT;
                         m: MAP )
    do
        from  i := m.cell.lower  until  i > m.cell.upper loop
            if  m.cell[i].is_in_perceptiual_field(z) then
                m.log_odds[i] := m.log_odds[i] +
                                 inverse_sensor_model (m.cell[i], x, z ) - l0
            end
        end
    end
```

$$m.log\_odds[i] := \log \frac{p(\ m.cell[i] \mid x_{i:t}, z_{i:t}\ )}{1 - p(\ m.cell[i] \mid x_{i:t}, z_{i:t}\ )}$$

$$l_0 := \log \frac{p(\ m.cell[i] = 1\ )}{p(\ m.cell[i] = 0\ )} := \log \frac{p(\ m.cell[i]\ )}{1 - p(\ m.cell[i]\ )}$$

Occupancy grid mapping

```
inverse_range_sensor_model ( x: ROBOT_POSE;
                             z: SENSOR_MEASUREMENT;
                             g: GRID_CELL) : LOG_ODDS_OCCUPANCY
    local
        xi, yi, r, φ: REAL_64
    do
        xi := g.center_of_mass.x
        yi := g.center_of_mass.y
        r := √( (xi - x.x)^2 + (yi - x.y)^2 )                          grid range
        φ := atan2(yi – x.y, xi – x.x) – x.θ                           grid angle
        k := argmin_j | φ – z.beam[j].θ |                              beam index
        if r > min( zmax, z.beam[k].range + α/2 ) or | φ – z.beam[k].θ | > β/2 then
                Result := l0                  grid out of range or behind an obstacle
        elseif z.beam[k].range < zmax and | r - z.beam[k].range | < α/2 then
                Result := locc                                   grid in the obstacle
        else -- r <= z.beam[k]
                Result := lfree                                  grid unaccupied
        end
    end
```

α: thickness of the obstacle
β: opening angle of the beam
z_{max}: max range of the beam

But what about drift?

Localization

- If we have a map, we can localize

Mapping

- If we know the robot's pose, we can map

Do both!

- Estimate a map
- Localize itself relative to the map

Simultaneous Localization and Mapping (SLAM)

Simultaneous Localization and Mapping

Localization: $p(x \mid m, z, u)$

Mapping: $p(m \mid x, z)$

SLAM: $p(x, m \mid z, u)$

- The map depends on the robot's pose during the measurement
- If the pose is known, mapping is easy

Rao-Blackwellization

$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}) = p(x_{1:t} \mid z_{1:t}, u_{0:t-1})\, p(m \mid x_{1:t}, z_{0:t-1})$

SLAM posterior = robot path posterior * mapping with known poses

$p(x_{1:t} \mid z_{1:t}, u_{0:t-1})$: localization

$p(m \mid x_{1:t}, z_{0:t-1})$: mapping

$x_{1:t}$: the robot's poses (x, y, θ)

m: the map

$z_{1:t}$: the measurements

$u_{0:t-1}$: the controls

Murphy, K. 1999. Bayesian map learning in dynamic environments. In NIPS '99 (Neural Info. Proc. Systems)

Rao-Blackwellized particle filter SLAM

Use a particle filter to represent potential trajectories of the robot

- Every particle carries its own map
- The probability of survival of a particle is proportional to the likelihood of the measurement with respect to the particle's own map

Problem: big map * large number of particles!

Improve pose estimate

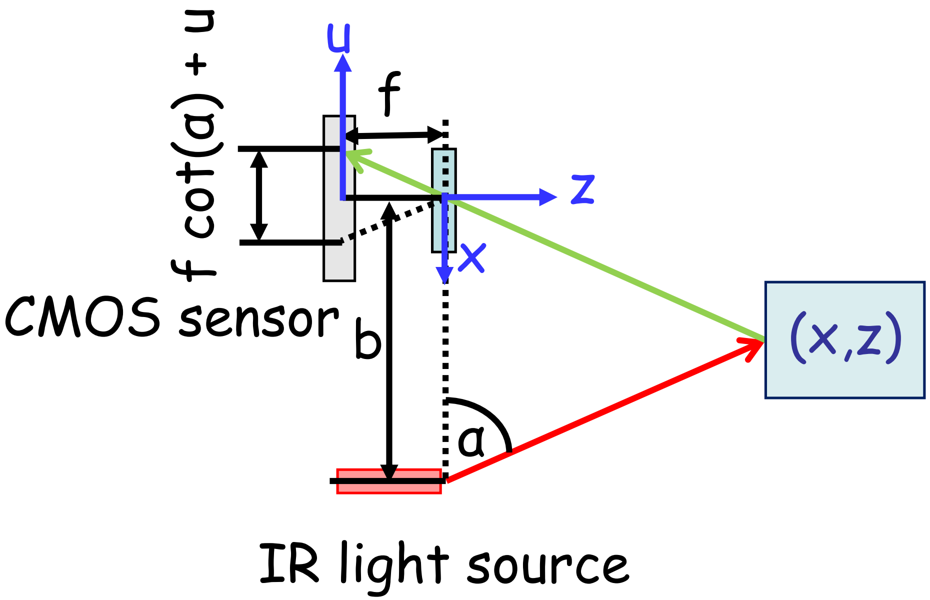
- Use scan matching to compute locally consistent pose correction
- Smaller error -> fewer particles necessary

Robot perception

How do we compute $p(z \mid x, m)$?

- Compare raw data to the map directly
- Compare features extracted from raw data to the map

Sensor model: structured light



$$x = \frac{b \cdot u}{f\cot(\alpha) + u} \qquad z = \frac{b \cdot f}{f\cot(\alpha) + u}$$

$$\frac{\partial u}{\partial z} = G_p = \frac{b \cdot f}{z^2}$$

$$\frac{\partial \alpha}{\partial z} = G_\alpha = \frac{b \sin(\alpha)^2}{z^2}$$

PrimeSense

- Operating range: 0.35 m – 1.4 m
- Spatial resolution: 0.9 mm at 0.5m
- Depth resolution: 0.1 cm at 0.5m

Likelihood fields

Project the end points of a sensor scan z_t into the map

- Measurement noise: Zero-centered Gaussian distribution
  - $p_{hit}(z^k_t \mid x_t, m) = \varepsilon_\sigma(dist)$
  - dist: distance between the measurement and the nearest obstacle in the map m
- Failures: Point-mass distribution
  - $$p_{max}(z^k_t \mid x_t, m) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases}$$
- Unexplained random measurements: Uniform distribution
  - $$p_{rand}(z^k_t \mid x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z^k_t \leq z_{max} \\ 0 & \text{otherwise} \end{cases}$$

$p(z^k_t \mid x_t, m) = z_{hit}\, p_{hit} + z_{rand}\, p_{rand} + z_{max}\, p_{max}$

$z_{hit}, z_{rand}, z_{max}$: mixing weights

Likelihood fields

```
likelihood_field_range_finder ( x: ROBOT_POSE;
                                z: SENSOR_MEASUREMENT;
                                m: MAP ) : REAL_64
    local
        xi, yi, d, q: REAL_64
    do
        q := 1.0
        from i := z.beam.lower until i > z.beam.upper loop
            if z.beam[i].range < zmax then
                xi := x.x + z.beam[i].x * cos(x.θ) – z.beam[i].y * sin(x.θ) +
                          z.beam[i].range * cos(x.θ + z.beam[i].θ)
                yi := x.y + z.beam[i].y * cos(x.θ) + z.beam[i].x * sin θ +
                          z.beam[i].range * sin(x.θ + z.beam[i].θ)
                d := m.compute_distance_to_the_nearest_obstacle(xi ,yi)
                q := q · ( zhit · prob(d, σhit) + zrand / zmax )
            end
        end
        Result := q
    end
```

Measurement coordinate

Likelihood fields

Advantages

- Smooth
  - Small changes in the robot's pose result in small changes of the resulting distribution
- Computationally more efficient than ray casting

Disadvantages

- No modeling of dynamic objects
- Sensors can see through the wall
  - Nearest neighbor cannot determine if a path is obstructed by an obstacle
- No map uncertainty considered
  - Can change occupancy to occupied, free, and unknown

Correlation-based measurement model

Map matching

1. Compute a local map m_{robot} from the scans z_t in robot frame
2. Transform the local map m_{robot} to the global coordinate frame m_{local}
3. Compare the local map m_{local} and the map m

$$\rho = \frac{\sum_{x,y}(m_{x,y} - \bar{m}) \cdot (m_{x,y,local}(x_t) - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m})^2 \sum_{x,y}(m_{x,y,local}(xt) - \bar{m})^2}} \text{ : correlation}$$

$$\bar{m} = \frac{1}{2N}\sum_{x,y}(m_{x,y} + m_{x,y,local}) \text{ : average map value}$$

$$p(m_{local} \mid x_t, m) = \max\{\rho, 0\}$$

Correlation-based measurement model

Advantages

- Easy to compute
- Explicitly considers free-space

Disadvantages

- Does not yield smooth probability in pose x_t
    - May convolve the map m with a Gaussian kernel first
- Can incorporate inappropriate local map information
    - May contain areas beyond the maximum sensor range
- Does not include the noise characteristic of range sensors

Feature extraction

feature: compact representation of raw data

- Range scans: lines, corners, local minima in range scans, etc.
- Camera images: edges, corners, distinct patterns, etc.
- High level features in robotics: places

Advantages of using features

- Reduction of computational complexity
    - Increase in feature extraction
    - Decrease in feature matching

Feature extraction: split and merge

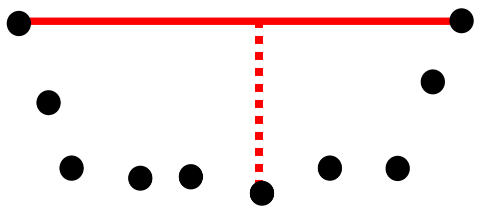
```
split( s: POINT_SET ) :  LINE_SET -- sorted points
    local
        pmax: POINT
        l: LINE
        lines: LINE_SET
    do
        create l.make_from_points( s )
        create lines.make_empty
        pmax := l.compute_farthest_point
        if l.compute_distance( pmax ) > dmax then
            lines.add_set( split( s.split_set(1, pmax) ) )
            lines.add_set( split( s.split_set(1, pmax) ) )
        else
            lines.add( l )
        end
        Result := lines
    end
```

Feature extraction: split and merge

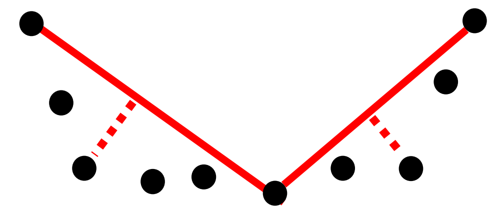
```
merge( lines: LINE_SET ) : LINE_SET
    local
        l: LINE
        out_lines: LINE_SET
    do
        create l.make_empty
        create lines.make_empty
        from  until not lines.is_next_pair_collinear loop
            l.merge_lines( lines.left_line , lines.right_line )
            if l.compute_distance( l.compute_farthest_point ) < d_max then
                out_lines.add(l)
                lines.mark_current_pair_as_used
            end
            lines.increment_next_pair
        end
        out_lines.add_set( lines.get_all_unmarked_lines )
        Result := out_lines
    end
```

Feature extraction: split and merge

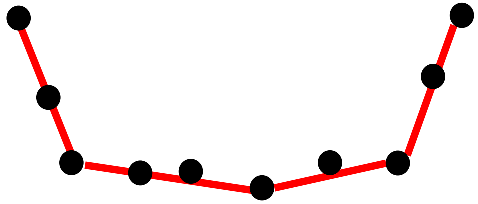
Split



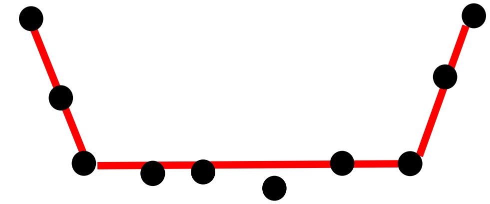
Split



Split

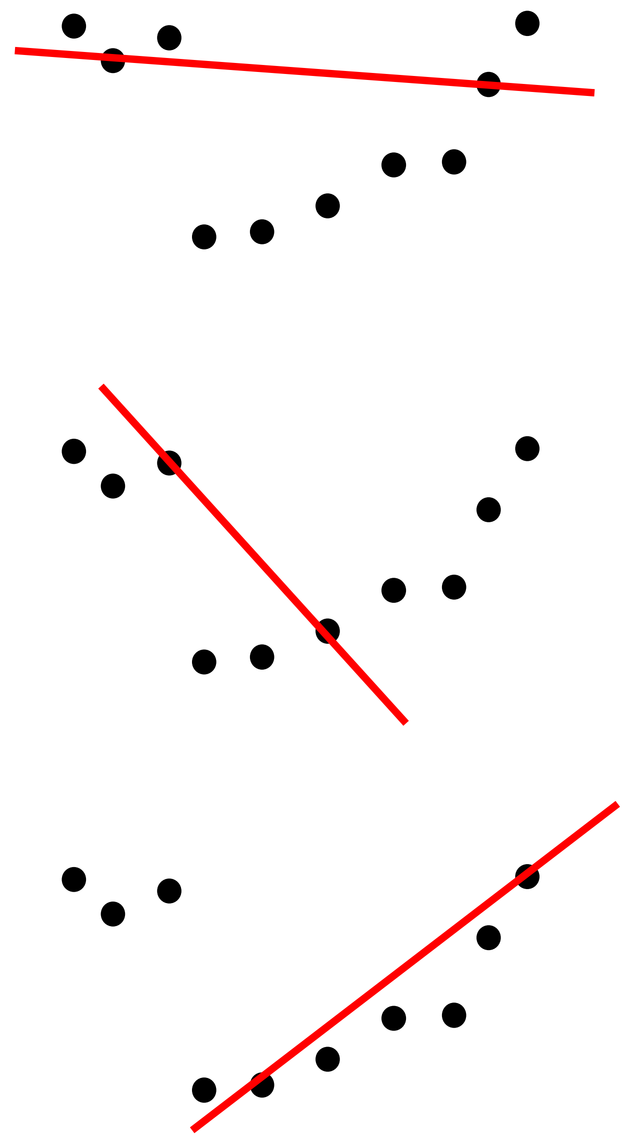


Merge



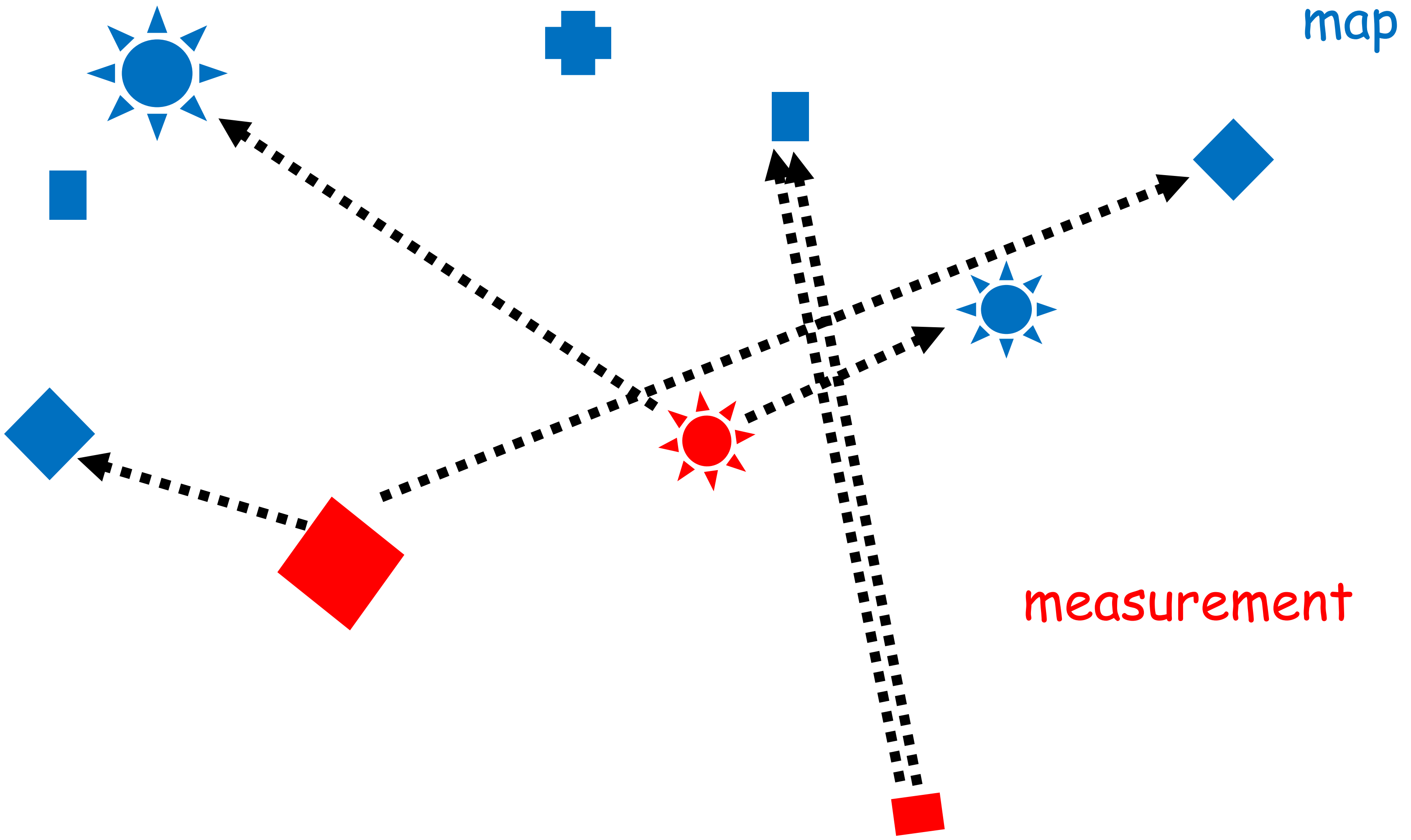
Feature extraction: RANSAC

```
RANSAC( s: POINT_SET ) :  LINE
    local
        l: LINE
        line: LINE
        num: INTEGER_16
    do
        create l.make_empty
        from c := 1 until  c > c_max  loop
            l.set_line_from_two_random_points(s)
            if l.count_inliners > num then
                num := l.count_inliners
                line := l
            end
        end
        Result := line
    end
```



Fischler, M. and Bolles, R. 1981. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". Communications of the ACM. 24(6).

Data association



Data association: Nearest Neighbor

Measurement: $F = \{f_1, \ldots, f_n\}$

Map features: $M = \{m_1, \ldots, m_l\}$

```
Nearest_neighbor (F, M)
    for i = 1 to |F|
        d_min = Mahalanobis2(f_i, m_1)
        nearest = 1
        for j = 2 to |M|
            d_j = Mahalanobis2(f_i, m_j)
            if d_j < d_min then
                nearest = j
                d_min = d_j
            endif
        endfor
        if d_min <= X^2(d_i, α) then
            H(i) = nearest
        else
            H(i) = 0
        endif
    endfor
```

α: desired confidence level

Data association: Joint Compatibility

```
JCBB(H, i) – Joint Compatibility Branch and Bound
        if i > m
                if pairings(H) > pairings(Best)
                        Best = H
                endif
        else
                for j = 1 to n
                        if individual_compatibility(Ei, Fj) and
                  joint_compatibility(H, Ei, Fj)
                                JCBB([H j], i + 1)
                endif
                endfor
                if pairings(H) + m – i >= pairings(Best) -- can do better?
                        JCBB([H 0], i + 1) -- star node: Ei not paired
                endif
        endif
```

Neira, J. Tardos, J.D. 2001. "Data association in stochastic mapping using the joint compatibility test", Robotics and Automation, IEEE Transactions on 17 (6): 890–897.