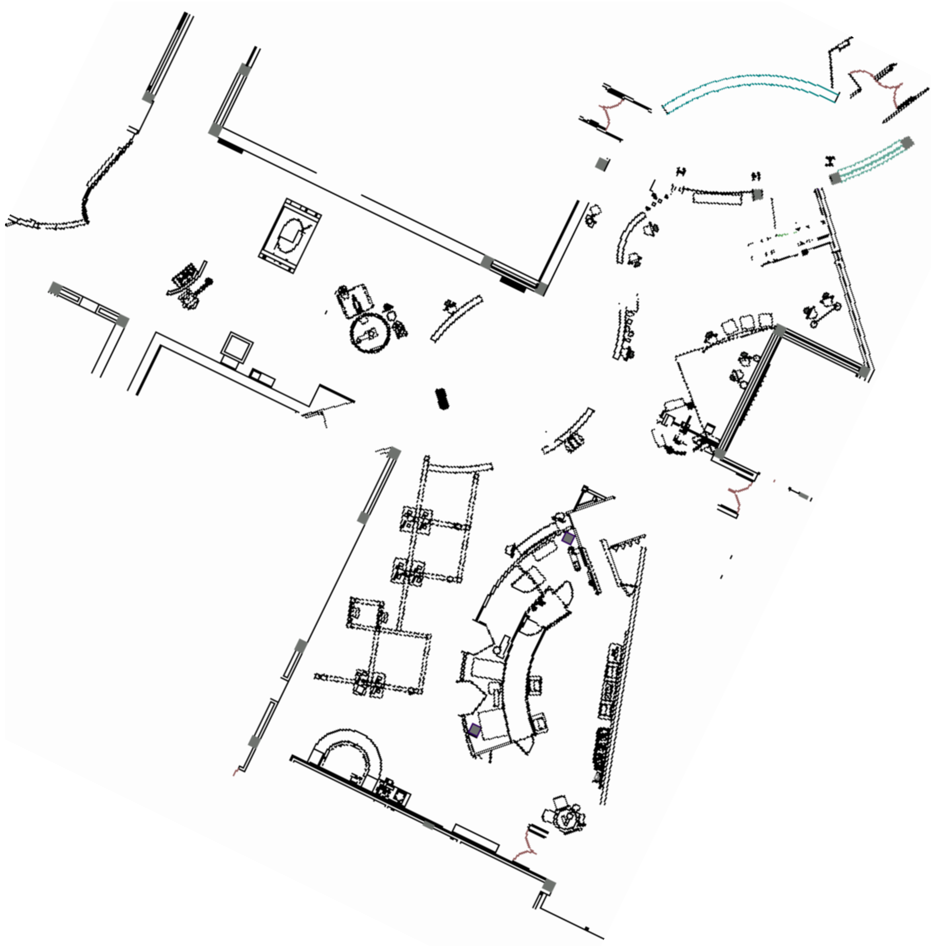
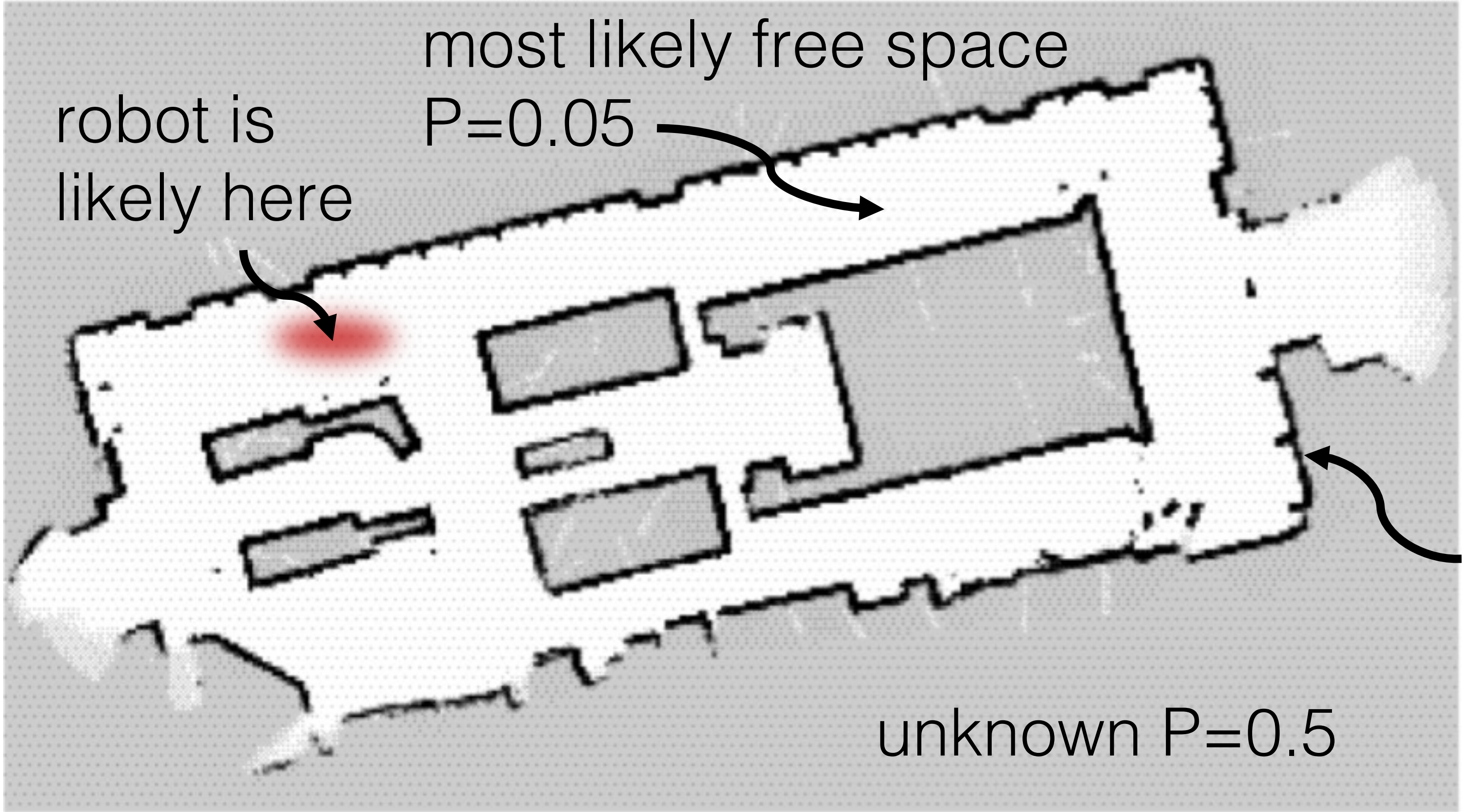
Occupancy Grid Mapping

Lecture 8

Occupancy Grid Map



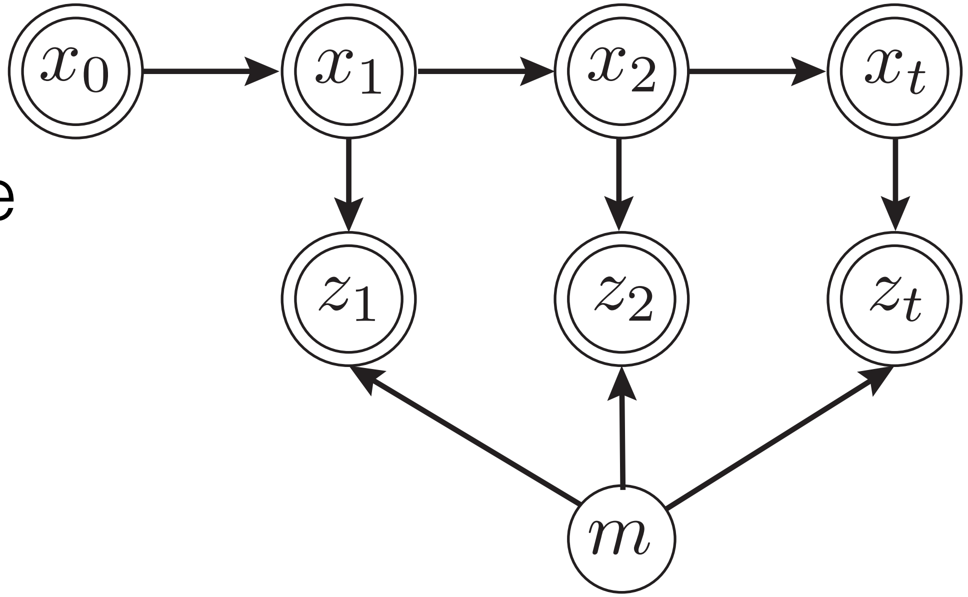
Localization & Mapping with Uncertainty



most likely a wall
P=0.95

Using Conditional Probability to make a map

- Given robot poses $x_{1:t}$ and laser rangefinder measurements $z_{1:t}$ infer a map of the environment
- Expressed probabilistically as $p(m|z_{1:t}, x_{1:t})$



- Double circle is known variable
- Arrow indicate generation
- Single circle is latent variable

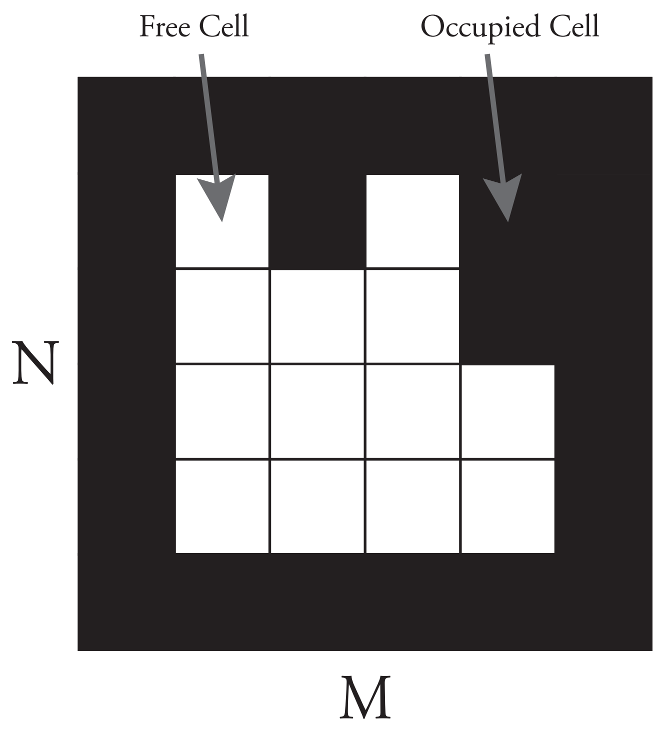
Occupancy Grid Mapping

- Map is a $M \times N$ matrix of cells
- Cell is either occupied or unoccupied
- Probability $p(m_i)$ is probability cell is occupied

$$p(m|x_{1:t}, z_{1:t}) = \prod_i p(m_i|x_{1:t}, z_{1:t})$$

- Map can be inferred from a Bayes filter with a static state

$$m = \{m_i\}_{M \times N}$$



Odds Ratio & Log Odds

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

- A is binary state $occ(i,j)$ and B is sensor reading $r = D$
- **Probability** a cell is occupied $p(occ(i,j)) = p(A)$ has range $[0,1]$
- Probability a cell is free $p(\neg A)$
- **Odds** of being occupied $o(occ(i,j)) = p(A)/p(\neg A)$ has range $[0,\infty]$
- **Log odds** $\log o(occ(i,j))$ has range $[-\infty,\infty]$
- Each cell $C(i,j)$ holds the value of log $o(occ(i,j))$
- $C(i,j) = 0$ corresponds to $p(occ(i,j)) = 0.5$

Bayes' Law using Odds

- Bayes' Law: $$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$
- Likewise: $$p(\neg A|B) = \frac{p(B|\neg A)p(\neg A)}{p(B)}$$
- So: $$o(A|B) = \frac{p(A|B)}{p(\neg A|B)} = \frac{p(B|A)p(A)}{p(B|\neg A)p(\neg A)}$$ $$= \lambda(B|A)o(A)$$
- Where: $$\lambda(B|A) = \frac{p(B|A)}{p(B|\neg A)}$$

Updating the map using Bayes' Law

- Bayes' Law can be written as

$$o(A|B) = \lambda(B|A)o(A)$$

posterior   Sensor update   prior

- Take log odds to make multiplication into addition

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

- For each cell add the evidence $\log \lambda(B|A)$ into the cells log odds

Prior and Sensor Update

- Prior
- Initially 0 if map unknown

$$l_0 = \log \frac{p(\mathbf{m}_i = 1)}{p(\mathbf{m}_i = 0)} = \log \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)}$$

- Sensor Update

$$l_{t,i} = \log \frac{p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})}{1 - p(\mathbf{m}_i \mid z_{1:t}, x_{1:t})}$$

Map Update Algorithm

Algorithm occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$)**:**

    for all cells $\mathbf{m}_i$ *do*

        if $\mathbf{m}_i$ *in perceptual field of* z_t *then*

            $l_{t,i} = l_{t-1,i} + \textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$

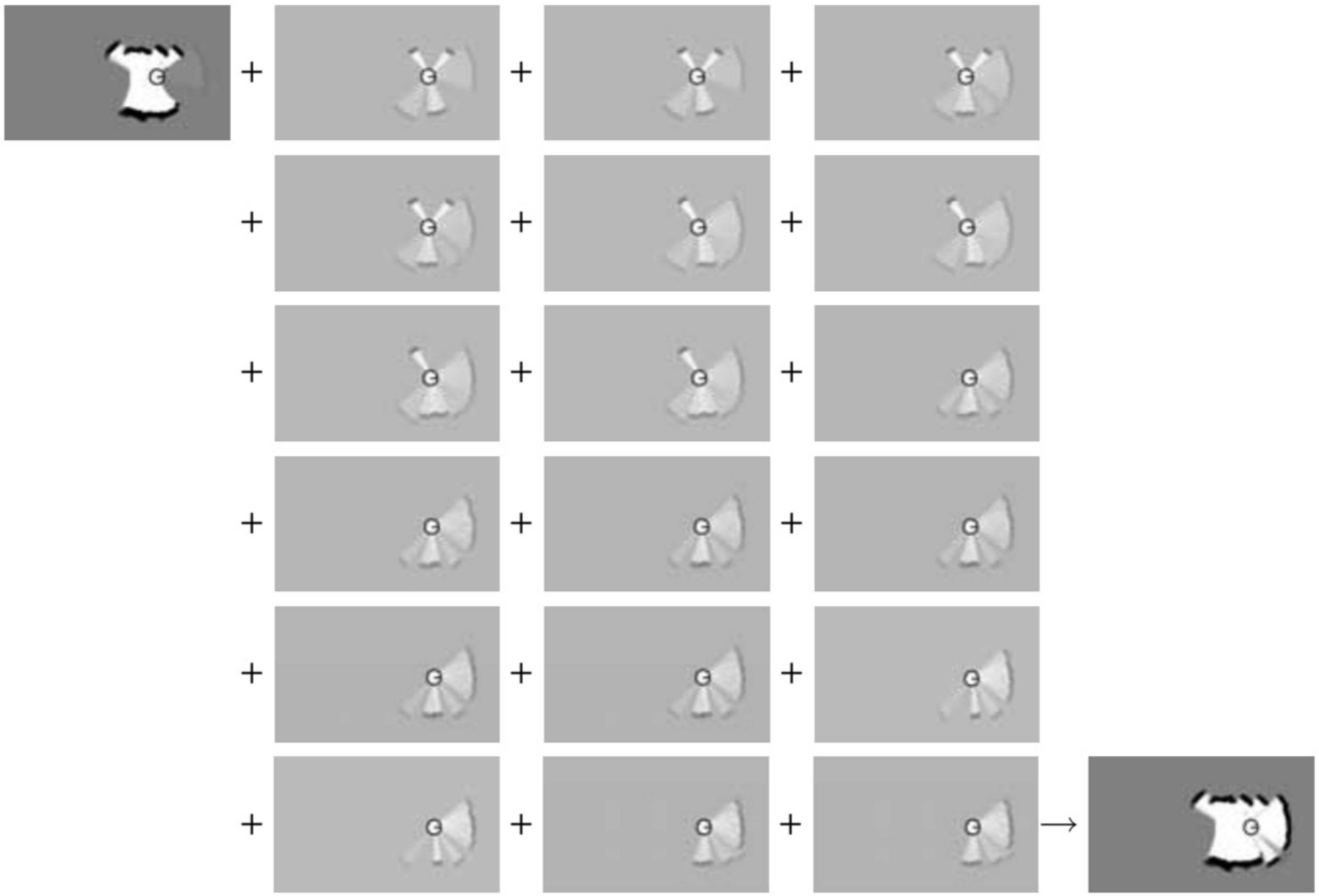
        else

            $l_{t,i} = l_{t-1,i}$

        endif

    endfor

    return $\{l_{t,i}\}$



Inverse Sensor Model

$$\textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) = \log \frac{p(\mathbf{m}_i \mid z_t, x_t)}{1 - p(\mathbf{m}_i \mid z_t, x_t)}$$

Algorithm inverse_range_sensor_model($\mathbf{m}_i, x_t, z_t$):

Let x_i, y_i *be the center-of-mass of* $\mathbf{m}_i$

$r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$

$\phi = \text{atan2}(y_i - y, x_i - x) - \theta$

$k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$

if $r > \min(z_{\max}, z_t^k + \alpha/2)$

  return l_0

if $z_t^k < z_{\max}$ *and* $|r - z_t^k| < \alpha/2$

  return l_{occ}

if $r \leq z_t^k$

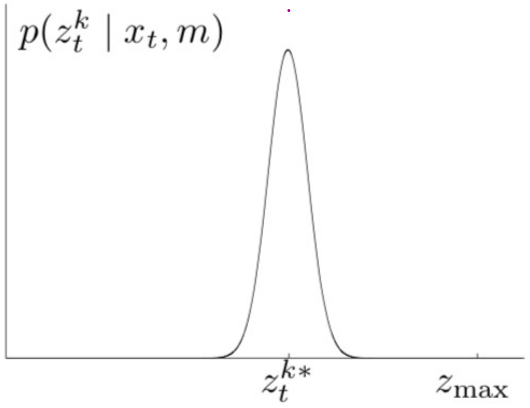
  return l_{free}

endif

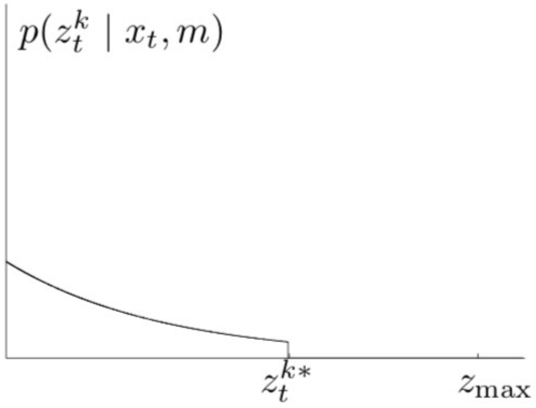
r – distance to cell
ϕ – angle to cell
α – dimension of cell
k – beam index
z_t^k - reading k from scan at time t

Sensor Model: Beam Model

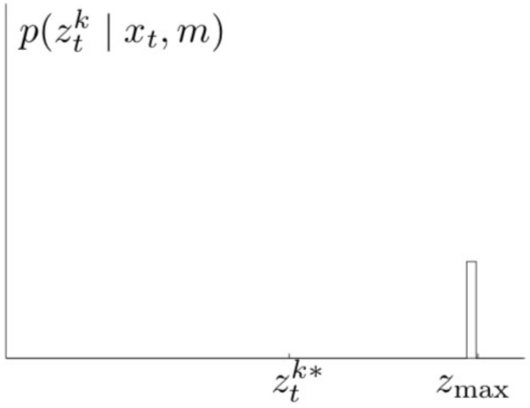
(a) Gaussian distribution p_{hit}



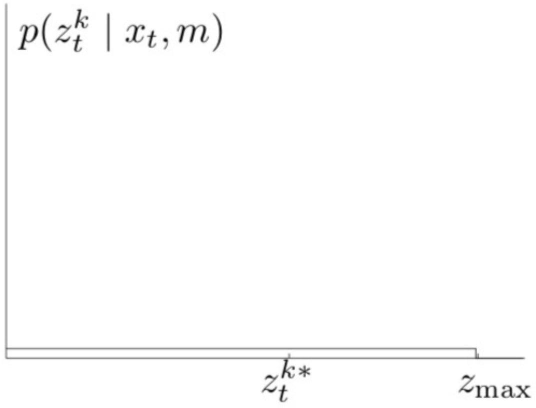
(b) Exponential distribution p_{short}



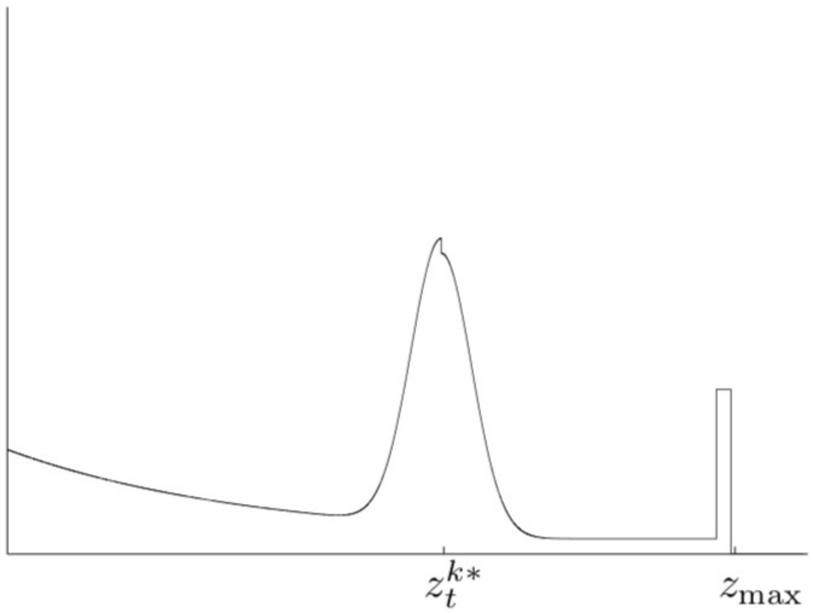
(c) Uniform distribution $p_{\max}$



(d) Uniform distribution p_{rand}

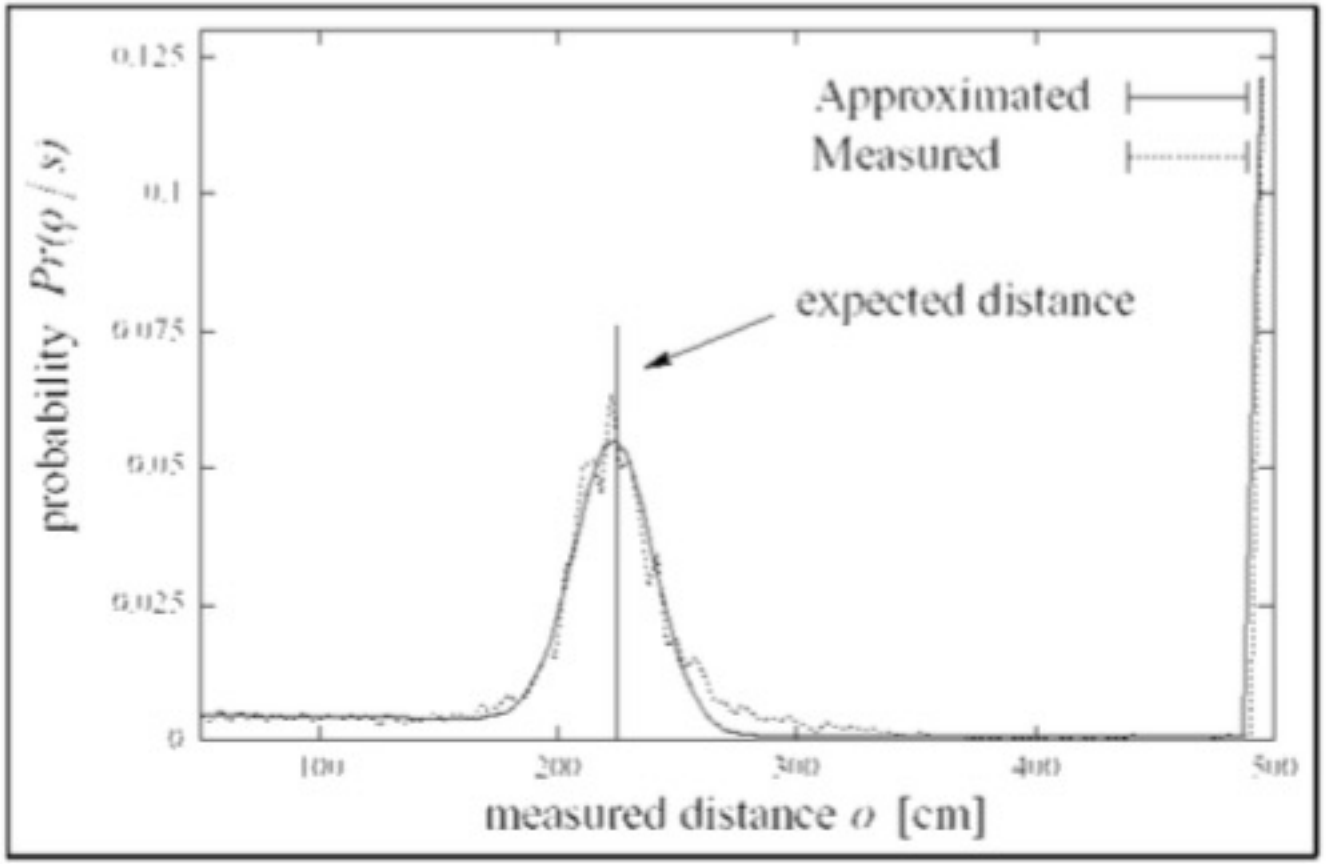


$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}$$



Sensor Model

Probability of reading a range given known occupancy at known distance



Inverse Sensor Model

- If laser terminates at C_{ij} at distance D

$$\lambda(z = D | occ(i,j)) = \frac{p(z = D | occ(i,j))}{p(z = D | \neg occ(i,j))} \approx \frac{.06}{.005} = 12$$

$$\log_2 \lambda \approx +3.5$$

- If the laser passes through C_{ij}

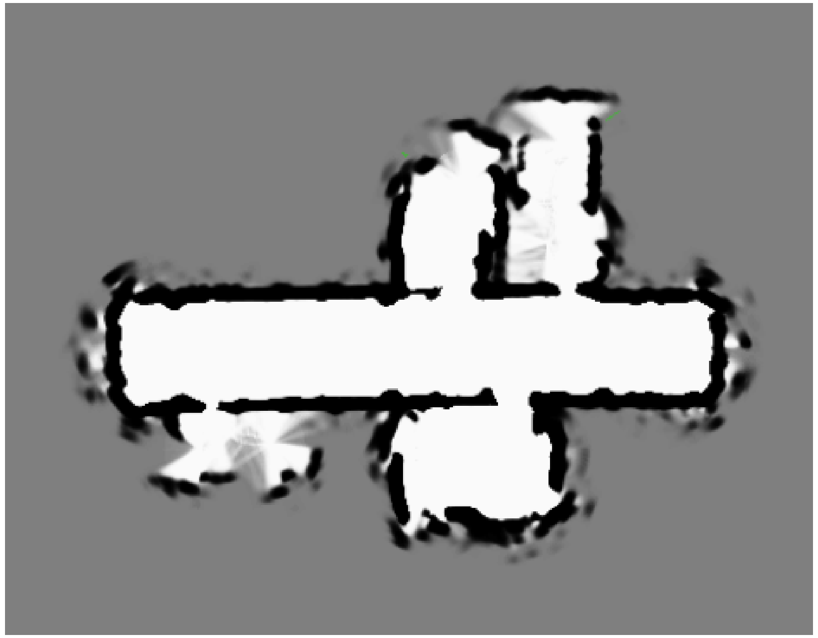
$$\lambda(z > D | occ(i,j)) = \frac{p(z > D | occ(i,j))}{p(z > D | \neg occ(i,j))} \approx \frac{.45}{.9} = 0.5$$

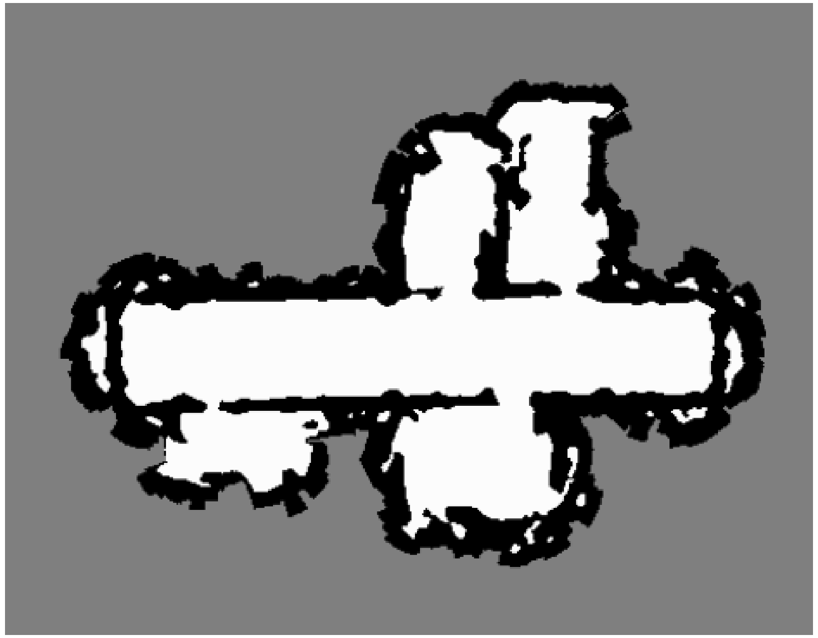
$$\log_2 \lambda \approx -1.0$$

Implementation

- Find endpoint of each ray on map grid and update
- Rasterize each laser ray into the map to determine cells that are currently visible and free or occupied
- Convert known pose (x, y, θ) to start cell and reading (θ,d) to end cell in the map
- Compute new log odds for each cell the ray touches
- Can divide ray into steps and check each cell along the ray
- Can use Breshenham's algorithm to update cells along the ray

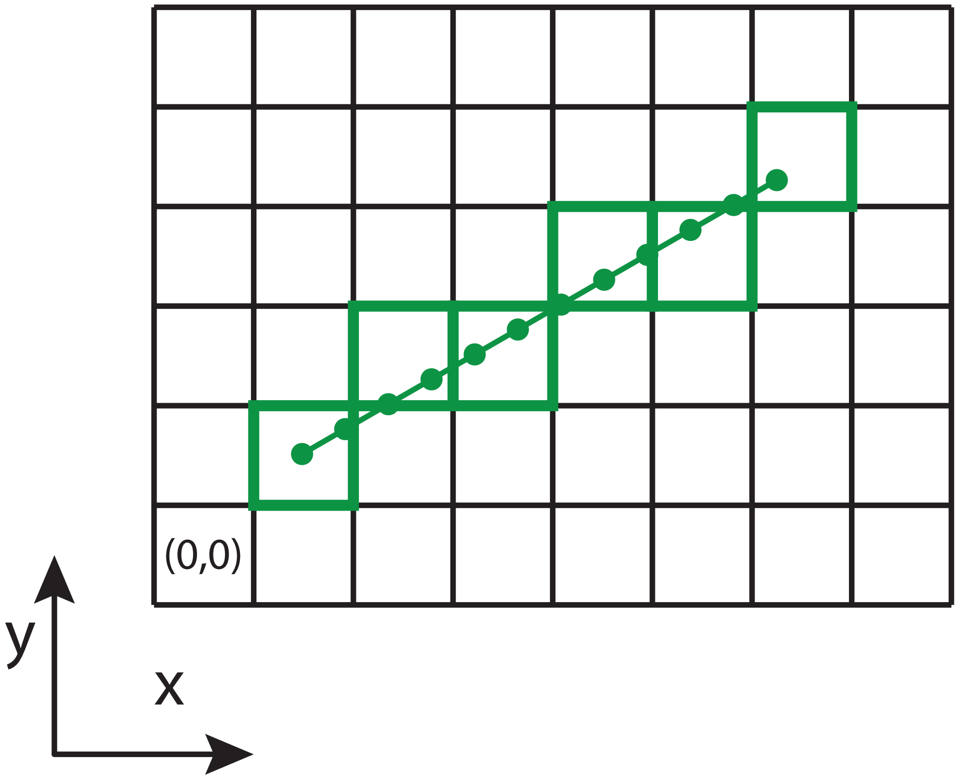
Maximum Likliehood Map





- Any cells p>0.5 are occupied, cells p<0.5 are free and cells p=0.5 are unknown.

Divide and Step Along Ray

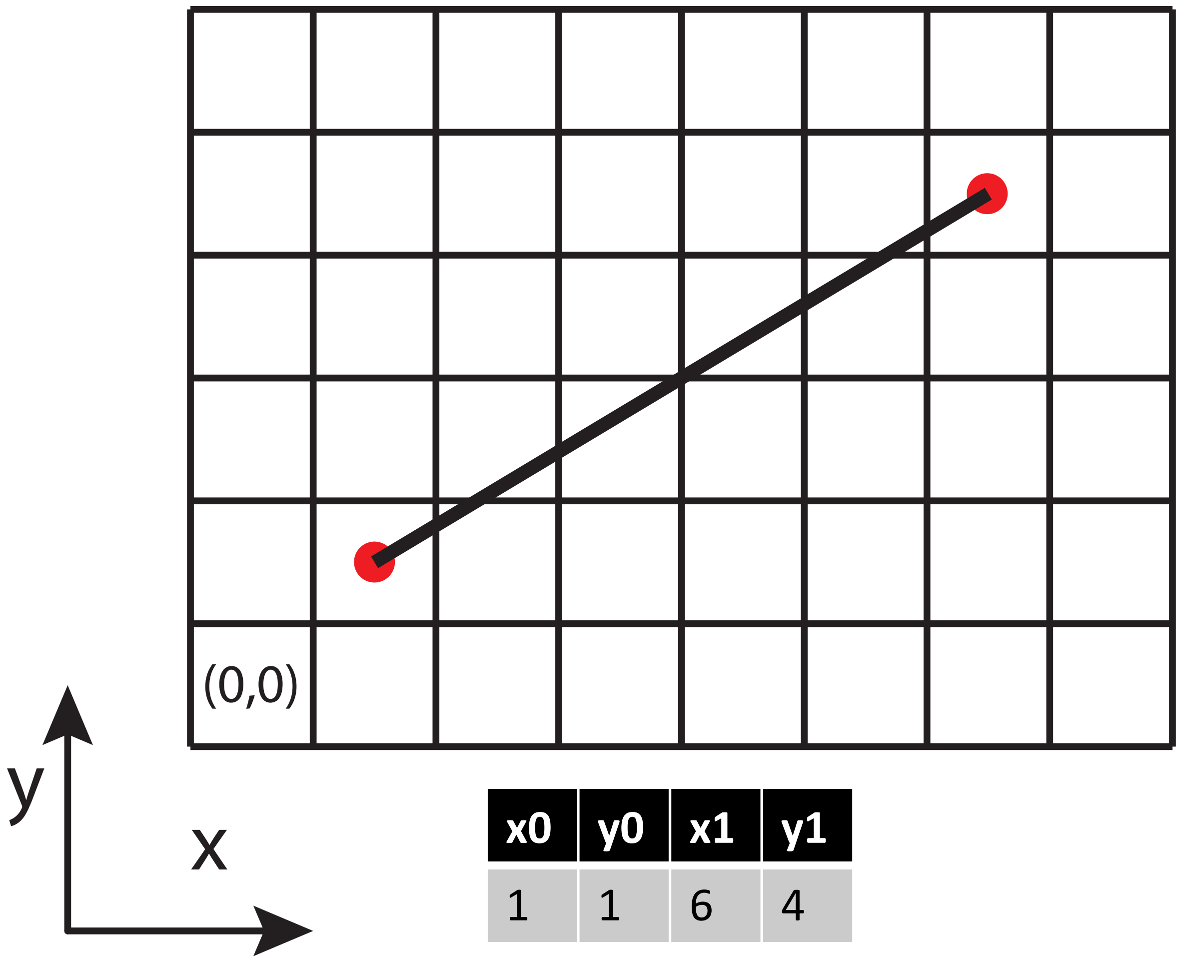


- Divide ray into ½ cell steps
- Check cell each step touches, but ensure you don't update cell twice
- In this case 12 iterations of loop
- Floating point math

Breshenham’s Algorithm – Integer Math

```
dx = abs(x1-x0);
dy = abs(y1-y0);
sx = x0<x1 ? 1 : -1;
sy = y0<y1 ? 1 : -1;
err = dx-dy;
x = x0;
y = y0;

while(x != x1 || y != y1){
    updateOdds(x,y);
    e2 = 2*err;
    if (e2 >= -dy){
           err -= dy;
           x += sx;
    }
    if (e2 <= dx){
           err += dx
           y += sy
    }
}
```

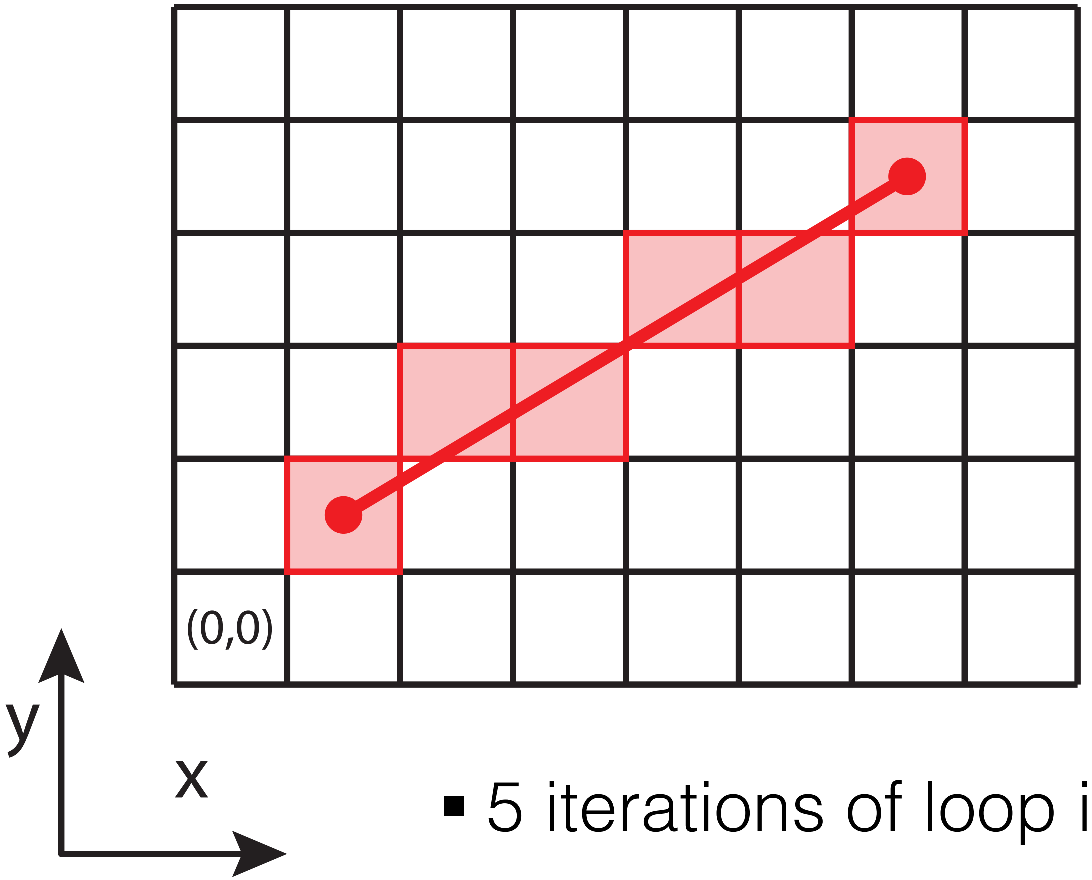


x0	y0	x1	y1
1	1	6	4

Breshenham's Algorithm

```
dx = abs(x1-x0);
dy = abs(y1-y0);
sx = x0<x1 ? 1 : -1;
sy = y0<y1 ? 1 : -1;
err = dx-dy;
x = x0;
y = y0;

while(x != x1 || y != y1){
    updateOdds(x,y);
    e2 = 2*err;
    if (e2 >= -dy){
            err -= dy;
            x += sx;
    }
    if (e2 <= dx){
            err += dx
            y += sy
    }
}
```

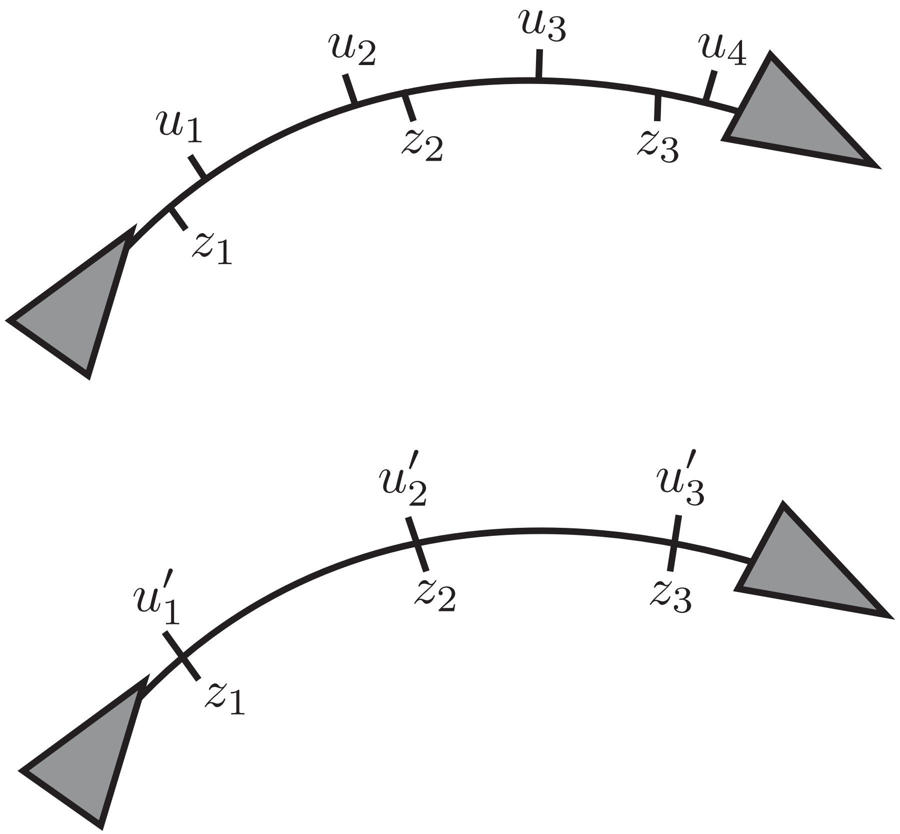


e2	x	y	err
4	2	2	4
8	3	2	1
2	4	3	3
6	5	3	0

- 5 iterations of loop in this case
- All integer math

Interpolating Observations

- Odometry and laser scans happen at different rates and arrive at different times.
- Interpolation gives odometry readings aligned with the laser scan times.



Interpolating Poses – Pose Trace

- Scans and Odometry happen at different rates…
- Already implemented in `OccupancyGridSlam::copyDataForSLAMUpdate()`
- Found in `common/pose_trace.cpp`
- Adds pose_xyt_t from odometry to a vector
- Interpolates x, y and θ linearly and assigns new timestamp
- Access this vector later to get interpolated poses that match the timestamps of the LIDAR scan

Moving Laser Scan in Mapping

```
struct adjusted_ray_t
{
    Point<float> origin;  ///< Position of the robot when the ray was measured
    float        range;   ///< Range of the measurement
    float        theta;   ///< Angle in global frame, not robot frame
};
```

```
void Mapping::updateMap(...){
        MovingLaserScan movingScan(scan, previousPose, pose);

        for(auto& ray : movingScan){
                scoreEndpoint(ray, map);
        }

        for(auto& ray : movingScan){
                scoreRay(ray, map);
        }
        previousPose_ = pose;
}
```