Mapping

Joey Wilson, Maani Ghaffari

March 8, 2024

Overview

1. Why mapping
2. Review mapping
3. Mapping notebook
4. Modern questions

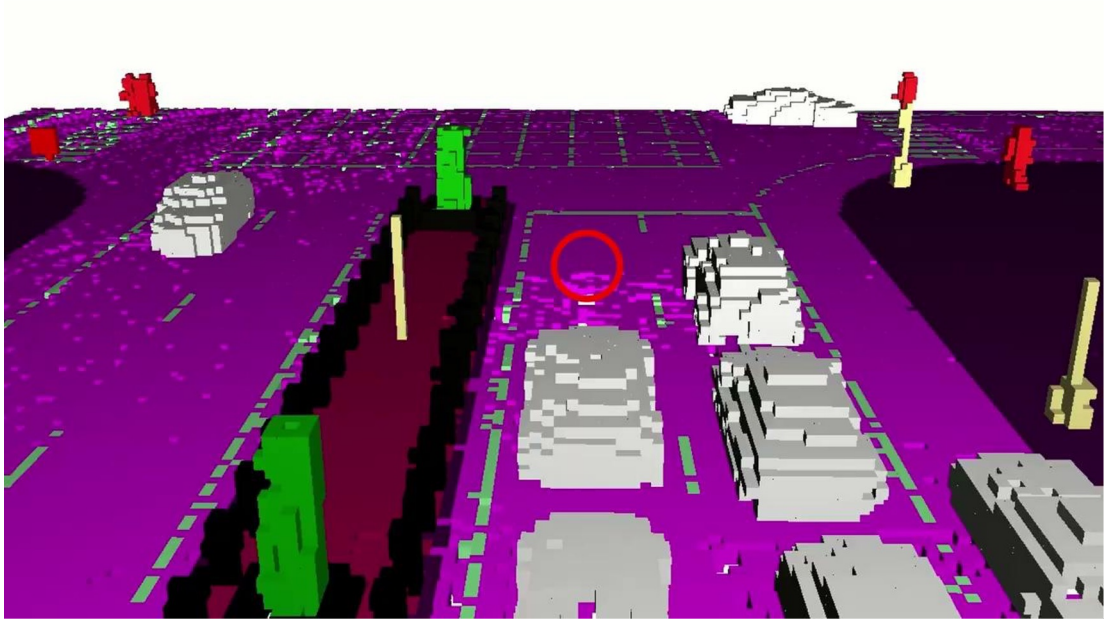
Motivation

Why Mapping?

- Now that you've solved localization, how do you decide where to go?

What to map?

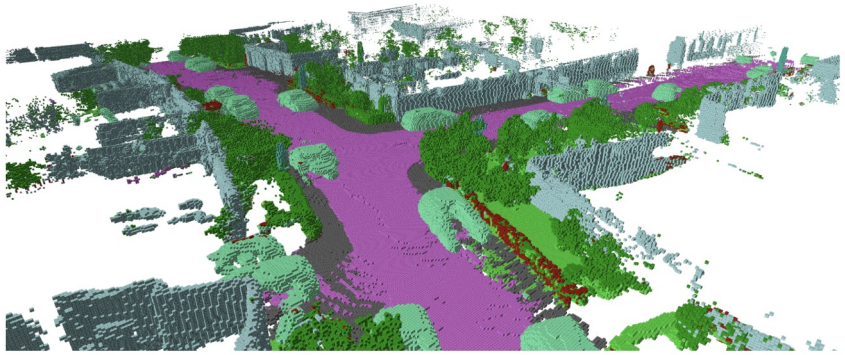
- Where can you drive? How difficult is the terrain to navigate?
- Where are other actors around you?



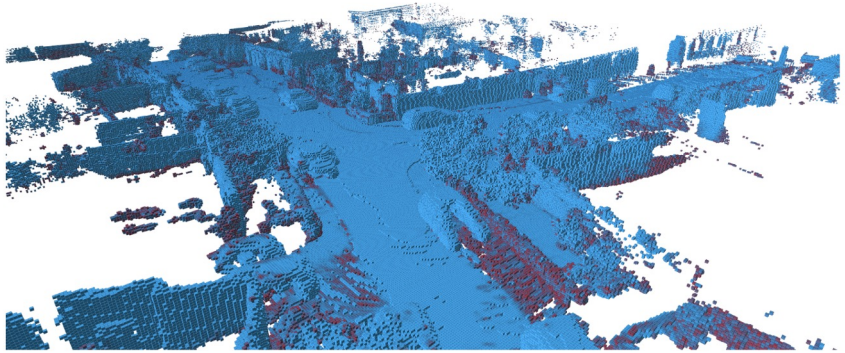
J. Wilson et al, "MotionSC: Data Set and Network for Real-Time Semantic Mapping in Dynamic Environments," IEEE Robot. Autom. Letter., vol. 7, no. 3, pp. 8439–8446, 2022.

Are You Sure?

- Data is noisy, leading to uncertainty
- To safely act, must be able to quantify uncertainty



(a) Semantic Categories



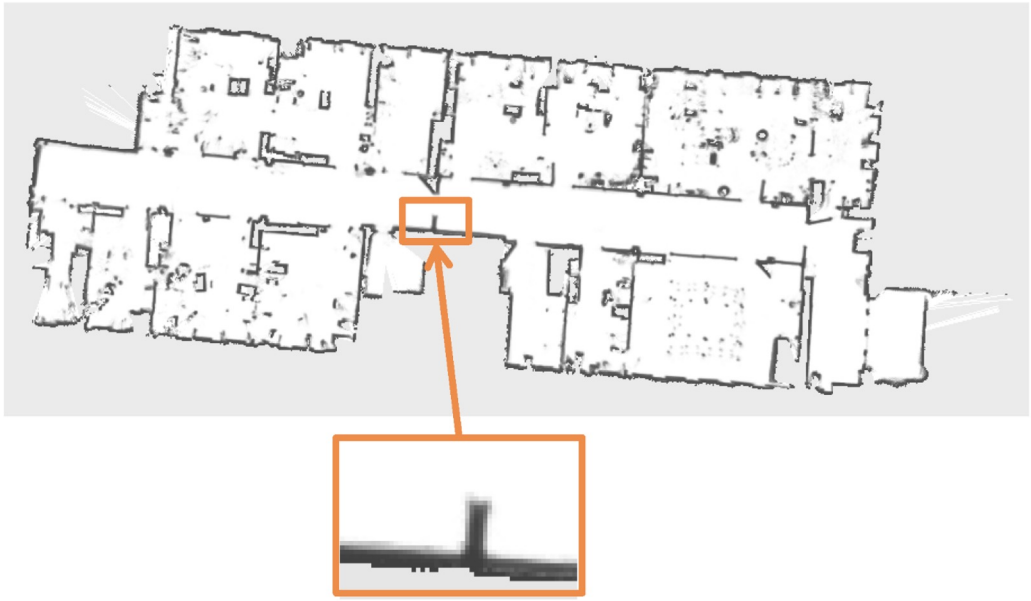
(b) Variance

J. Wilson, Y. Fu, A. Zhang, J. Song, A. Capodieci, P. Jayakumar, K. Barton, and M. Ghaffari, "Convolutional Bayesian Kernel Inference for 3D Semantic Mapping," in Proc. IEEE Int. Conf. Robot. And Automation, 2023, pp. 8364–8370.

Occupancy Grid Mapping

Occupancy Mapping

- Goal: How likely that each cell is occupied or free space?



Courtesy: C. Stachniss

Update

- Every time we are given a measurement, update the log odds

$$l(x) = \log \frac{p(x)}{1 - p(x)}$$

  - If p(x) = 0.5, *l(x)* = 0, so we usually set the prior to 0
- Posterior is a sum of inverse sensor model, recursive term, prior

$$l_{t,i} = \text{inv\_sensor\_model}(m_i, \mathbf{x}_t, \mathbf{z}_t) + l_{t-1,i} - l_0$$

```
self.map['logodd'][i] = self.inverse_sensor_model(z) + self.map['logodd'][i] - self.l_prior
```

Crude Inverse Sensor Model

- Find closest beam (by angle) to cell i

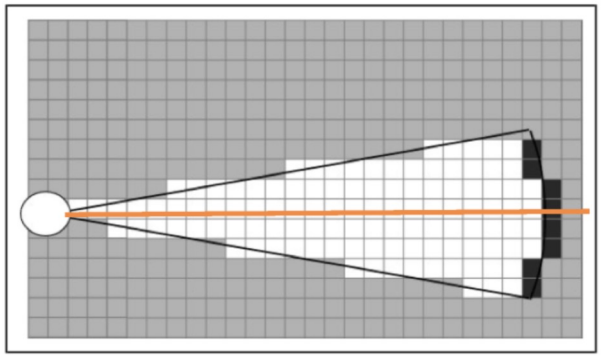
Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$

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

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

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

```
bearing_diff = np.abs(wrapToPI(z[:, 1] - self.m_i['phi']))
k = np.nanargmin(bearing_diff)
```



In the following, consider the cells along the optical axis (orange line)

Courtesy: Thrun, Burgard, Fox

Three Cases

- Beam terminates before cell (uninformative)

$$\textit{if } r > \min(z_{\max}, z_t^k + \alpha/2) \textit{ or } |\phi - \theta_{k,\text{sens}}| > \beta/2 \textit{ then}$$
$$\textit{return } l_0$$

```
if (self.m_i['range'] > min(self.z_max, z[k, 0] + self.alpha / 2)) or (bearing_min > self.beta / 2):
    l_inv = self.l_prior
```

- Beam terminates at cell (occupied)

$$\textit{if } z_t^k < z_{\max} \textit{ and } |r - z_t^k| < \alpha/2$$
$$\textit{return } l_{\text{occ}}$$

```
elif (z[k, 0] < self.z_max) and (np.abs(self.m_i['range'] - z[k, 0]) < self.alpha / 2):
    l_inv = self.l_occ
```

- Beam passes through (free)

$$\textit{if } r \leq z_t^k$$
$$\textit{return } l_{\text{free}}$$

```
elif (self.m_i['range'] < z[k, 0]) and (z[k, 0] < self.z_max):
    l_inv = self.l_free
```

Perceptual Field

- For each cell, measurement combination
  - Update cell with measurement if the measurement is in range/view of sensor
  - Filter

```
def is_in_perceptual_field(self, m, p):
    # check if the map cell m is within the perception field of the
    # robot located at pose p
    d = m - p[0:2].reshape(-1)
    self.m_i['range'] = np.sqrt(np.sum(np.power(d, 2)))
    self.m_i['phi'] = wrapToPI(np.arctan2(d[1], d[0]) - p[2])
    # check if the range is within the feasible interval
    if (self.m_i['range'] > 0) and (self.m_i['range'] < self.z_max):
        # here sensor covers -pi to pi
        if (self.m_i['phi'] > -np.pi) and (self.m_i['phi'] < np.pi):
            return True
    return False
```

Full Structure

- Each time a new measurement comes in,
  - Check which cells are in range
  - And update their posterior

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

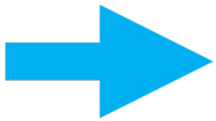
Notebook

Semantic Mapping

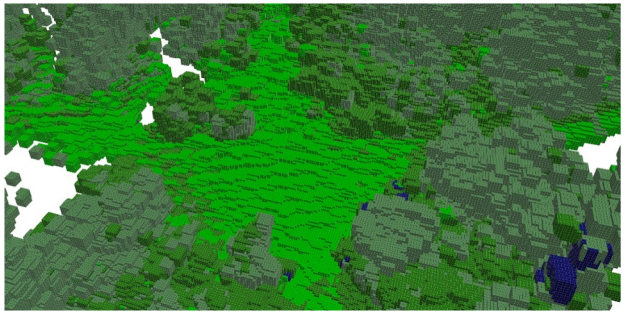
Semantic Mapping

Input: 3D Data

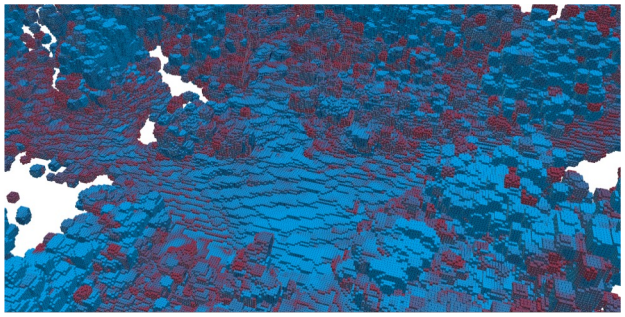
Labeled by a Semantic Segmentation Network



Output: 3D Semantic Map



With Quantifiable Uncertainty



Semantic Mapping Extension (CSM)

- Input: Points x_i with semantic predictions y_i
  - The semantic predictions are probabilities, e.g. $y_i^{grass} = p(x_i = grass) = 10\%$

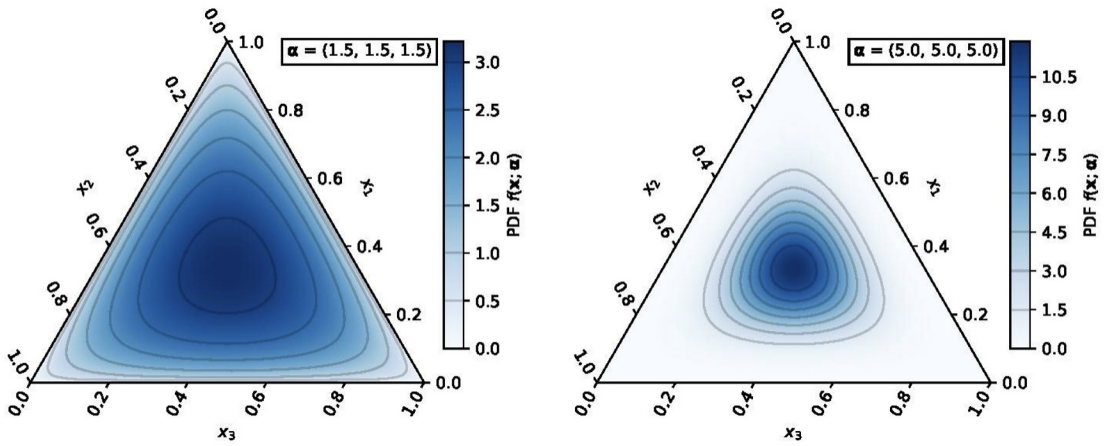
- Now, use Dirichlet concentration parameters to keep track of each cell

$$\alpha_k^{new} = \alpha_k + y_k$$

Dirichlet Distribution

- From the concentration parameters, we can obtain the expectation (a probability) and variance

$$\eta_*^c = \sum_{j=1}^{C} \alpha_*^j, \quad \mathbb{E}[\alpha_*^c] = \frac{\alpha_*^c}{\eta_*^c}, \quad \mathbb{V}[\alpha_*^c] = \frac{\frac{\alpha_*^c}{\eta_*^c}(1 - \frac{\alpha_*^c}{\eta_*^c})}{1 + \eta_*^c}$$

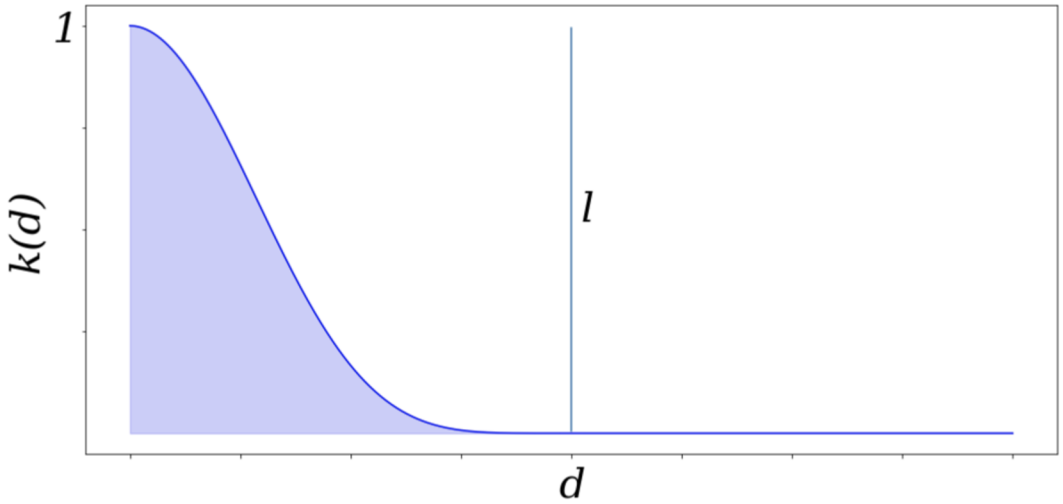


https://en.wikipedia.org/wiki/Dirichlet_distribution

Continuous Mapping (BKI)

- Points don't just inform us about the cells they fall in
- They also tell us about nearby cells
  - The weight can be approximated through a kernel

$$\alpha^c_{*,t} = \alpha^c_{*,t-1} + \sum_{i=1}^{N_t} k(x_*, x_i) y^c_i$$

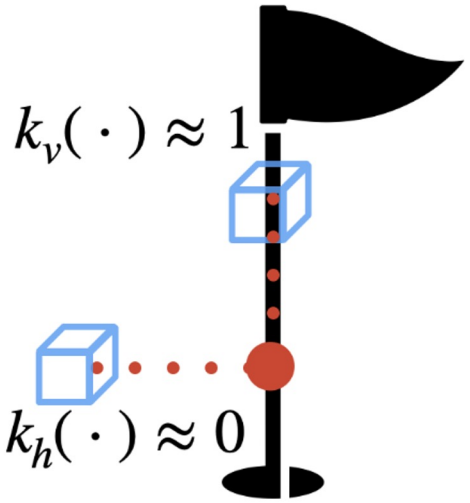


Kernel Shapes (ConvBKI)

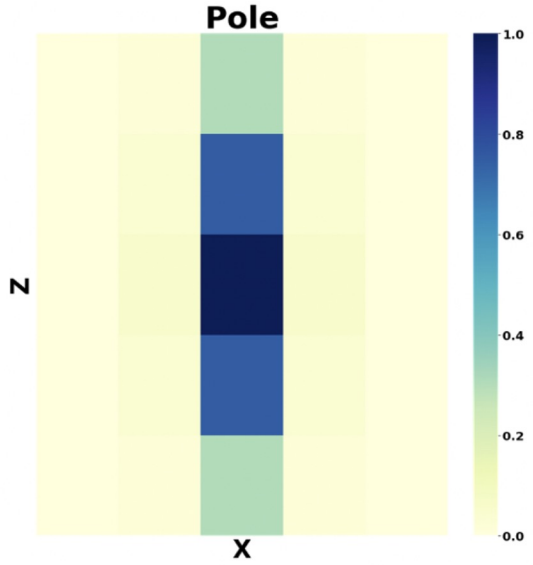
- Semantic categories have different shapes
  - Poles are tall
  - Road is flat
- Can take a product of kernels in the update

$$\alpha^c_{*,t} = \alpha^c_{*,t-1} + \sum_{i=1}^{N_t} k(x_*, x_i) y_i^c$$

$$k_h(|| \begin{bmatrix} x - x' \\ y - y' \end{bmatrix} ||) k_v(|| \begin{bmatrix} z - z' \end{bmatrix} ||)$$



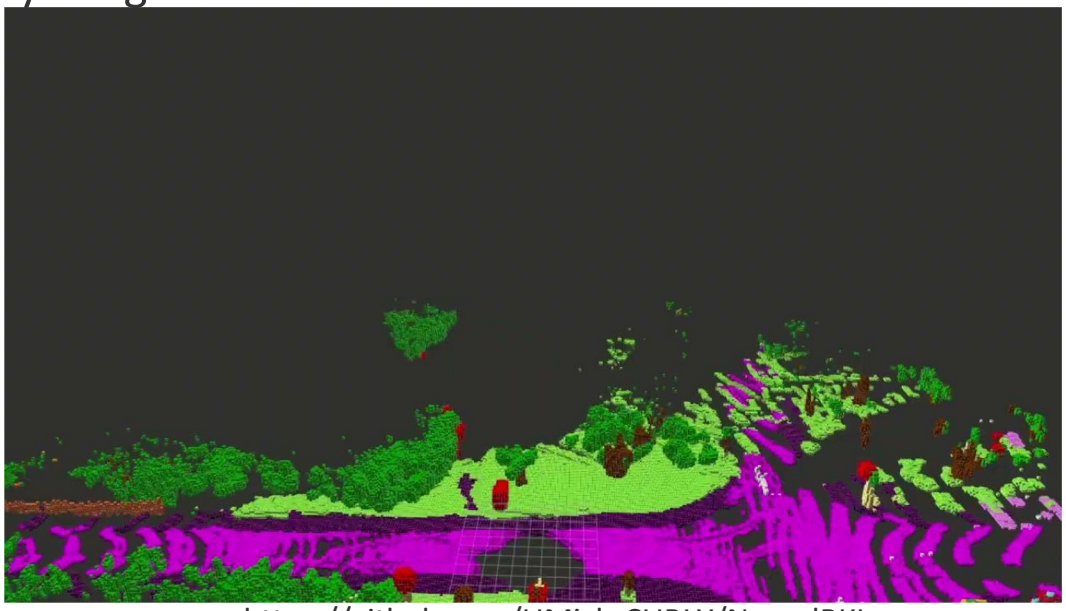
(a) Pole Intuition



(b) Learned Weights

ConvBKI

- The BKI update is a weighted sum of the input points per channel
  - If you are familiar with deep learning, this is a depthwise convolution
  - And everything is differentiable!



https://github.com/UMich-CURLY/NeuralBKI

Open Questions

- How to handle moving objects?
  - Dynamic objects leave behind traces (the update step does not remove them)
- What if we want to work in feature space?
  - This can provide us an open dictionary, not restricted to pre-specified categories
- How to fuse multi-modal input?