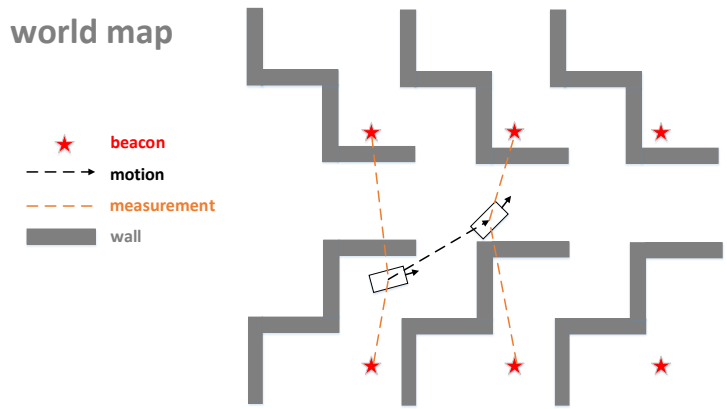
Object Localization Problem in complex and symmetric environments

Roman Korkin, Alexandr Katrutsa

August 4, 2022

Localization problem statement

- Given sequences of motion $\{\boldsymbol{u}_t\}_{t=1}^N$ and measurements $\{\boldsymbol{z}_t\}_{t=1}^N$
- Assume there exists some motion law that generates sequence of object states $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t, \eta)$, where η is a noise.
- Measurement function maps state to measurement $\boldsymbol{z}_{t+1} = g(\boldsymbol{x}_t, \zeta)$, where ζ is a noise.



Localization problem visualization

- The goal is to generate sequence of estimates $\mathbf{x}_t$ for the ground-truth $\mathbf{x}_t^*$ such that

$$L_T = \frac{1}{T}\sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x}_t^*\|_2^2 \to \min$$

Kalman filter: assumptions[1]

- Noise distributions $\zeta \sim \mathcal{N}(0, \mathbf{R})$ and $\eta \sim \mathcal{N}(0, \mathbf{Q})$
- Linear map $\mathbf{F}$ between sequential states from the motion law
- Linear map $\mathbf{H}$ from state to measurement from the measurement law
- State vector is from $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ where $\boldsymbol{x}_t \sim \boldsymbol{\mu}_t$ and $\mathbf{P}_t \sim \boldsymbol{\Sigma}_t$

[1] Rudolph Emil Kalman. "A new approach to linear filtering and prediction problems". In: *Transaction of the ASME - Journal of Basic Engineering* (1960).

Kalman filter

Predict stage

- Estimate state $\boldsymbol{x}_{t+1}^{-} = \mathbf{F}\boldsymbol{x}_t + \boldsymbol{u}_t$, where $\boldsymbol{u}_t$ is external force
- Update state covariance matrix $\mathbf{P}_{t+1}^{-} = \mathbf{F}\mathbf{P}_t\mathbf{F}^{\top} + \mathbf{Q}$

Update stage

- Estimate next measurement $\boldsymbol{z}_{t+1}^{-} = \mathbf{H}\boldsymbol{x}_{t+1}^{-}$
- Compute $\boldsymbol{y}_{t+1} = \boldsymbol{z}_{t+1} - \boldsymbol{z}_{t+1}^{-}$, where $\boldsymbol{z}_{t+1}$ is from sensors
- Compute Kalman factor $\mathbf{K} = \mathbf{P}_{t+1}^{-}\mathbf{H}^{\top}(\mathbf{H}\mathbf{P}_{t+1}^{-}\mathbf{H}^{\top} + \mathbf{R})^{-1}$
- Correct state estimate $\boldsymbol{x}_{t+1} = \boldsymbol{x}_{t+1}^{-} + \mathbf{K}\boldsymbol{y}_{t+1}$
- Update state covariance matrix $\mathbf{P}_{t+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_{t+1}^{-}$

Issues of Kalman filter

- Requires **linear** motion and measurement laws
- Motion and measurement noises must be **Gaussian**
- **Initial state** mean and variance required

Extended Kalman filter[2]

Assumptions

- Noise distributions $\zeta \sim \mathcal{N}(0, \mathbf{R})$ and $\eta \sim \mathcal{N}(0, \mathbf{Q})$
- Nonlinear map between sequential states from the motion law: $\boldsymbol{x}_{t+1}^{-} = f(\boldsymbol{x}_t)$
- Nonlinear map from state to measurement from the measurement law: $\boldsymbol{z}_{t+1} = g(\boldsymbol{x}_{t+1}^{-})$
- State vector is from $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ where $\boldsymbol{x}_t \sim \boldsymbol{\mu}_t$ and $\mathbf{P}_t \sim \boldsymbol{\Sigma}_t$

[2] Simon J Julier and Jeffrey K Uhlmann. "New extension of the Kalman filter to nonlinear systems". In: *Signal processing, sensor fusion, and target recognition VI*. vol. 3068. 1997, pp. 182–193.

Extended Kalman filter: main steps

- Prediction stage: $\boldsymbol{x}_{t+1}^{-} = f(\boldsymbol{x}_t), \quad \mathbf{F} = \frac{\partial f(\boldsymbol{x}_{t,i})}{\partial \boldsymbol{x}_{t,j}}$
- Update of covariance matrix estimation: $\mathbf{P}_{t+1}^{-} = \mathbf{F}\mathbf{P}_t\mathbf{F}^\top + \mathbf{Q}_{t+1}$
- Measurement update stage: $\boldsymbol{z}_{t+1}^{-} = g(\boldsymbol{x}_{t+1}^{-}), \quad \mathbf{H} = \frac{\partial g(\partial \boldsymbol{x}_{t+1,i}^{-})}{\partial \boldsymbol{x}_{t+1,j}^{-}}$
- Kalman gain computation and state correction

$$\mathbf{K} = \mathbf{P}_{t+1}^{-}\mathbf{H}^\top(\mathbf{H}\mathbf{P}_{t+1}^{-}\mathbf{H}^\top + \mathbf{R})^{-1}$$
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_{t+1}^{-} + \mathbf{K}(\boldsymbol{z}_{t+1} - \boldsymbol{z}_{t+1}^{-})$$
$$\mathbf{P}_{t+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_{t+1}^{-}$$

Issues of extended Kalman filter

- Noise distributions $\zeta \sim \mathcal{N}(0, \mathbf{R})$ and $\eta \sim \mathcal{N}(0, \mathbf{Q})$
- Initial state mean and variance required

Particle filter[3]

Distribution simulated by N particles with states x_t^i and weights w_t^i

Predict stage

- Estimate states distribution $\mathbf{x}_{t+1}^i = f(\mathbf{x}_t^i, \mathbf{u}_t, \eta^i)$

Update stage

- Estimate next measurement for all particles $\boldsymbol{z}_{t+1}^i = g(\boldsymbol{x}_{t+1}^i, \zeta^i)$
- Get measurements from beacons $\boldsymbol{z}_{t+1}$
- Compute likelihoods $\mathcal{L}^i(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}^i)$. For example for gaussian measurements error:
  $\mathcal{L}^i(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}^i) \sim \exp(-\sum_{j=1}^K \frac{(z_{j,t+1}-z_{j,t+1}^i)^2}{2\sigma^2})$
- Update states weights: $w_{t+1}^i \sim w_t^i \mathcal{L}^i(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}^i)$
- Resampling: $w_{t+1}^j = 1/N, \mathbf{x}_{t+1}^j = \mathbf{x}_{t+1}^{i_j}$

[3] Pierre Del Moral. "Nonlinear filtering: Interacting particle resolution". In: *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics* (1997).

Resampling

- To avoid degeneracy (too low weights of most particles) resampling required
- Goal: eliminate particles with low weights and replicate those with high weights
- Multinomial (stochastic) resampling: sampling with repetitions N states from N states with probability proportional to state weight:

$$
\begin{aligned}
& i_1, \ldots, i_N \sim \mathit{Multinomial}(w_{t+1}^1, \ldots, w_{t+1}^N) \\
& \mathbf{x}_{t+1}^1, \ldots, \mathbf{x}_{t+1}^N \leftarrow \mathbf{x}_{t+1}^{i_1}, \ldots, \mathbf{x}_{t+1}^{i_N} \\
& w_{t+1}^j = \frac{1}{N}.
\end{aligned}
$$

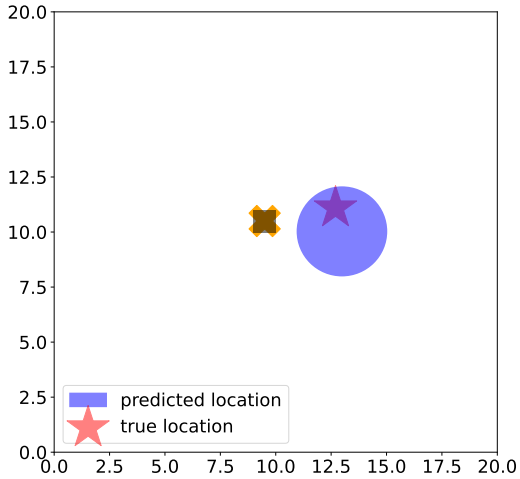
Standard filters overview

Stage	Extended Kalman filter	Particle filter
Predict	$\mathbf{x}^-_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ $\boldsymbol{P}^-_{t+1} = \boldsymbol{F}\boldsymbol{P}_t\boldsymbol{F}^T + \boldsymbol{Q}_{t+1}$ $\boldsymbol{F} = \frac{\partial f(\mathbf{x}, \mathbf{u}_{t+1})}{\partial \mathbf{x}}\Big\|_{\mathbf{x}=\mathbf{x}_t}$	$\mathbf{x}^i_{t+1} = f(\mathbf{x}^i_t, \mathbf{u}_t, \eta^i)$
Update	$\mathbf{z}_{t+1} = g(\mathbf{x}^-_{t+1})$ $\boldsymbol{H}_{t+1} = \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}\|_{\mathbf{x}=\mathbf{x}^-_{t+1}}$ $\boldsymbol{S} = \boldsymbol{H}_{t+1}\boldsymbol{P}^-_{t+1}\boldsymbol{H}^T_{t+1} + \boldsymbol{R}$ $\boldsymbol{K}_{t+1} = \boldsymbol{P}^-_{t+1}\boldsymbol{H}^T_{t+1}\boldsymbol{S}^{-1}$ $\mathbf{x}_{t+1} = \mathbf{x}^-_{t+1} + \boldsymbol{K}_{t+1}(\mathbf{z}_{t+1} - \tilde{\mathbf{z}}^i_{t+1})$ $\boldsymbol{P}_{t+1} = (\boldsymbol{I} - \boldsymbol{K}_{t+1}\boldsymbol{H}_{t+1})\boldsymbol{P}^-_{t+1}$	$\mathbf{z}^i_{t+1} = g(\mathbf{x}^i_{t+1}) + \zeta^i$ $w^i_{t+1} \sim w^i_t \mathcal{L}(\mathbf{z}^i_{t+1} \| \mathbf{x}^i_{t+1})$ $i_1..i_N \sim Multinomial(w^1_{t+1}..w^N_{t+1})$ $\mathbf{x}^1_{t+1}..\mathbf{x}^N_{t+1} \leftarrow \mathbf{x}^{i_1}_{t+1}..\mathbf{x}^{i_N}_{t+1}$ $w^i_t = \frac{1}{N}$

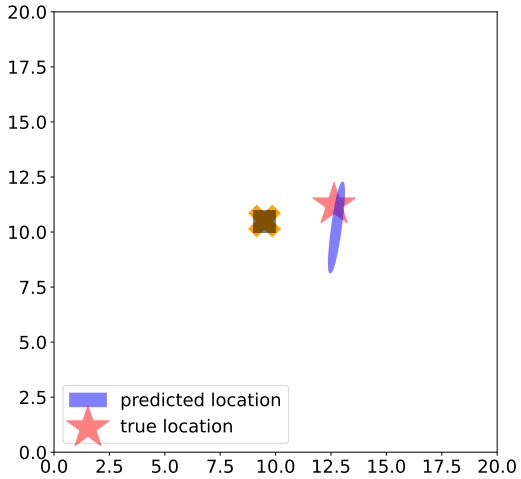
- **fast, guaranteed convergence**
- **Gaussian distribution of motion, measurement and state**
- **initial state required**

- **slow, asymptotical convergence**
- **applicable at any distributions**
- **initial state may be unknown**

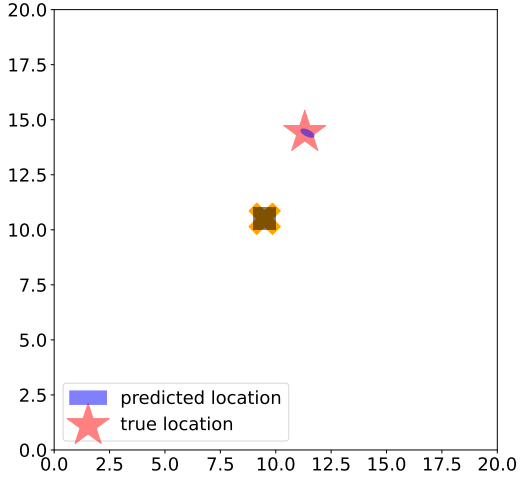
KF filter performance example



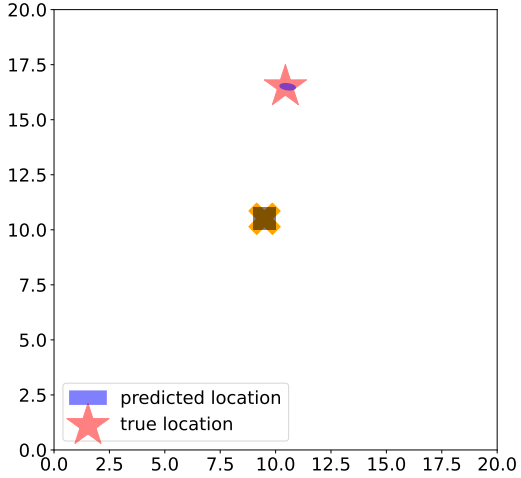
KF step 1



KF step 2

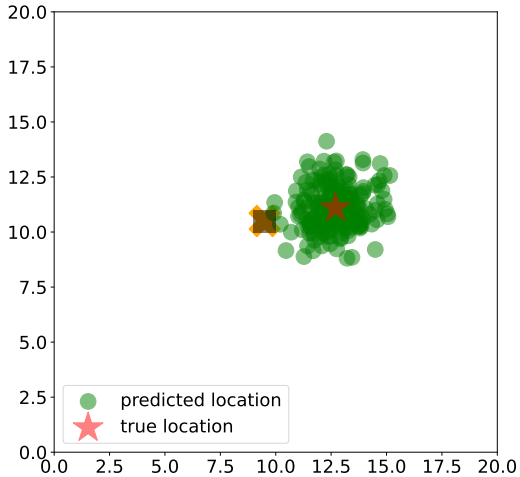


KF step 25

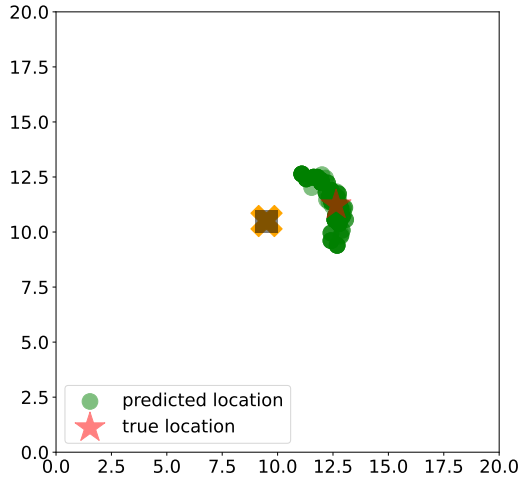


KF step 40

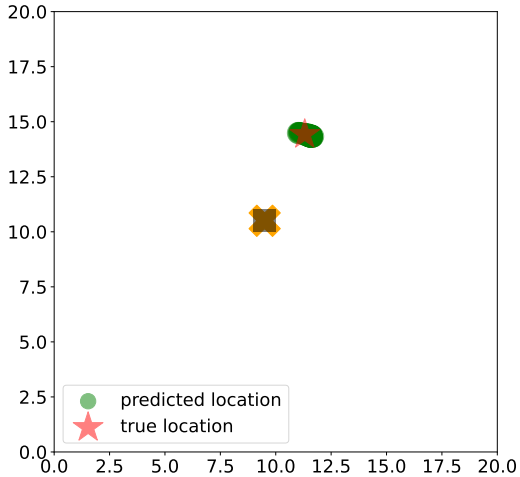
PF filter performance example



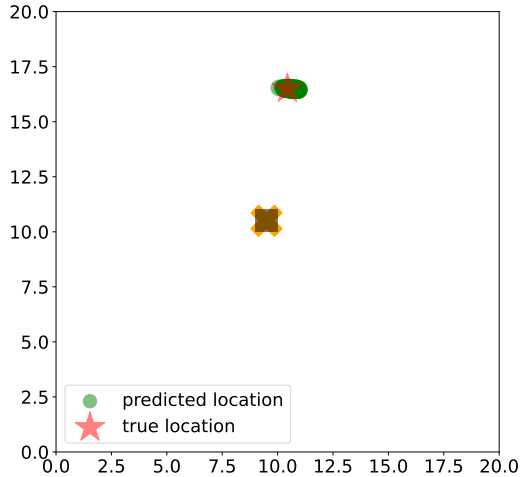
PF step 1



PF step 2



PF step 25



PF step 40

Kalman particle filter (our approach)

- Kalman part. For $i = 1..N$ particles
  - Kalman predict stage

$$\mathbf{x}_{t+1}^{i-} = f(\mathbf{x}_t^i, \mathbf{u}_{t+1}),\ \boldsymbol{P}_{t+1}^{i-} = \boldsymbol{F}^i \boldsymbol{P}_t^i \boldsymbol{F}^{i\top} + \boldsymbol{Q}_{t+1}^i$$

  - Kalman update stage

$$\tilde{\mathbf{z}}_{t+1}^i = g(\mathbf{x}_{t+1}^{i-}),\ \boldsymbol{K}_{t+1}^i = \boldsymbol{P}_{t+1}^{i-} \boldsymbol{H}_{t+1}^{i\top} (\boldsymbol{H}_{t+1}^i \boldsymbol{P}_{t+1}^{i-} \boldsymbol{H}_{t+1}^{i\top} + \boldsymbol{R})^{-1}$$
$$\mathbf{x}_{t+1}^{i+} = \mathbf{x}_{t+1}^{i-} + \boldsymbol{K}_{t+1}^i (\mathbf{z}_{t+1} - \tilde{\mathbf{z}}_{t+1}^i),\ \boldsymbol{P}_{t+1}^{i+} = (\boldsymbol{I} - \boldsymbol{K}_{t+1}^i \boldsymbol{H}_{t+1}^i) \boldsymbol{P}_{t+1}^{i-}$$

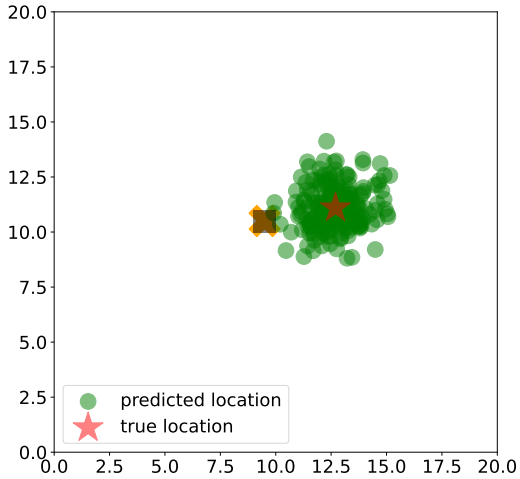
- Particle part
  - New measurements prediction: $\tilde{\mathbf{z}}_{t+1}^{i+}(\mathbf{x}_{t+1}^{i+})$
  - Likelihood computation $\mathcal{L}^i(\mathbf{z}_{t+1}|\mathbf{x}_{t+1}^{i+}) = \prod_{j=1}^K p(\mathbf{z}_{t+1}^j|\mathbf{x}_{t+1}^{i+})$
  - Weights update $w_{t+1}^i \sim w_t^i \mathcal{L}(\mathbf{z}_{t+1}^i|\mathbf{x}_{t+1}^i)$
  - Resampling $i_1, \ldots, i_N \sim \mathit{Multinomial}(w_{t+1}^1, \ldots, w_{t+1}^N)$
  - $\mathbf{x}_{t+1}^{1+}, \ldots, \mathbf{x}_{t+1}^{N+} \leftarrow \mathbf{x}_{t+1}^{i_1+}, \ldots, \mathbf{x}_{t+1}^{i_N+}$
  - $\boldsymbol{P}_{t+1}^1, \ldots, \boldsymbol{P}_{t+1}^N \leftarrow \boldsymbol{P}_{t+1}^{i_1+}, \ldots, \boldsymbol{P}_{t+1}^{i_N+}$
  - Add motion noise $\mathbf{x}_{t+1}^i = \mathbf{x}_{t+1}^{i+} + \eta^i$

Extended Kalman particle filter[4]

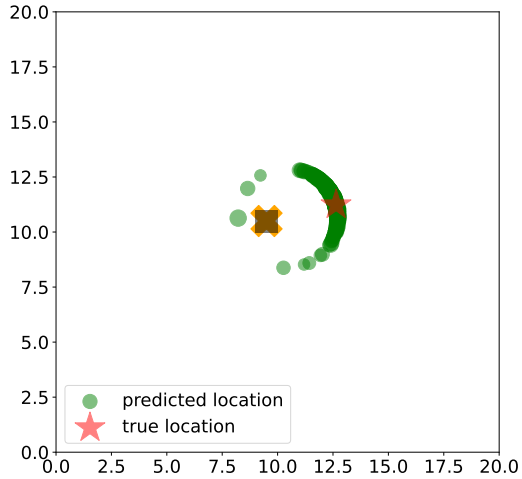
- Alternativce approach: other weights updates
  $w^i_{t+1} = \frac{p^i_{t+1}\mathcal{L}^i(\mathbf{z}_{t+1}|\mathbf{x}^{i+}_{t+1})w^i_t}{d^i_{t+1}}$,
- $d^i_{t+1} = \mathbb{P}(\xi = \mathbf{x}^i_{t+1} - \mathbf{x}^{i+}_{t+1})$, $\xi \sim \mathcal{N}(0, \boldsymbol{P}^i_{t+1})$
  $p^i_{t+1} = \mathbb{P}(\eta = \mathbf{x}^i_{t+1} - \mathbf{x}^{i+}_{t+1})$, $\eta \sim \mathcal{N}(0, \boldsymbol{Q}^i_{t+1})$
- Drawback: large weights are assigned to irrelevant particles due to inconsistency between covariance matrices $\boldsymbol{P}^i_{t+1}$ and likelihood $\mathcal{L}^i$
- $\mathbf{Q}^i$ might have lower rank, poor convergence

[4] M.S. Arulampalam et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *IEEE Transactions on Signal Processing* 50.2 (2002), pp. 174–188. DOI: 10.1109/78.978374.

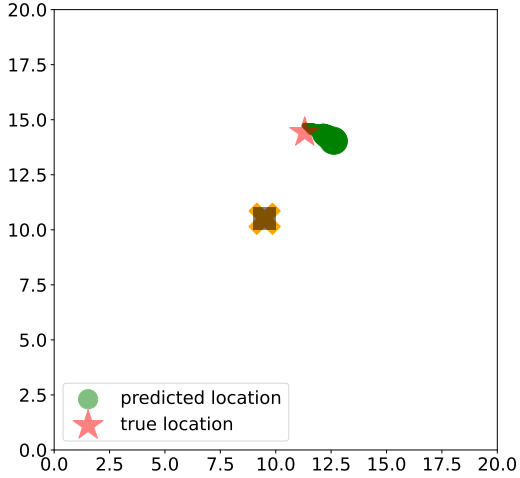
KPF filter performance example



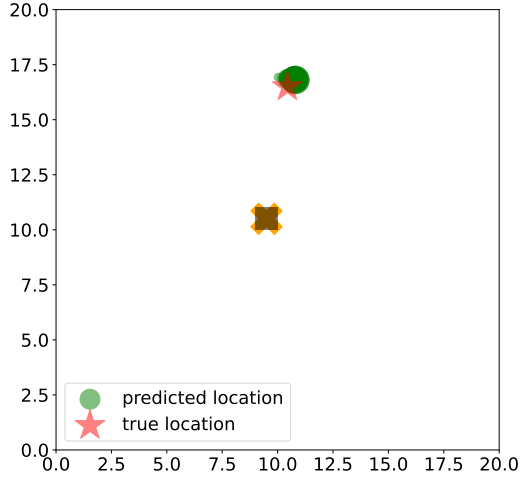
KPF step 1



KPF step 2



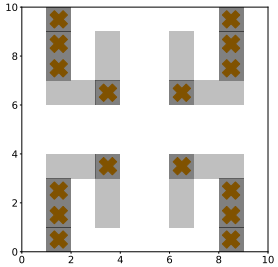
KPF step 25



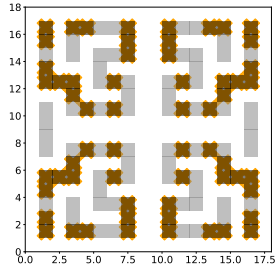
KPF step 40

Worlds for tests

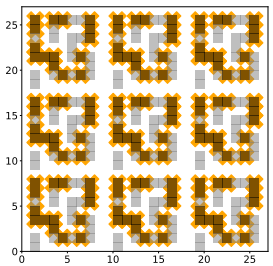
- Environments for tests



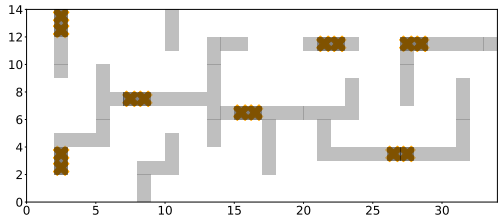
(a) world 10×10



(b) World 18×18



(c) WORLD 27×27

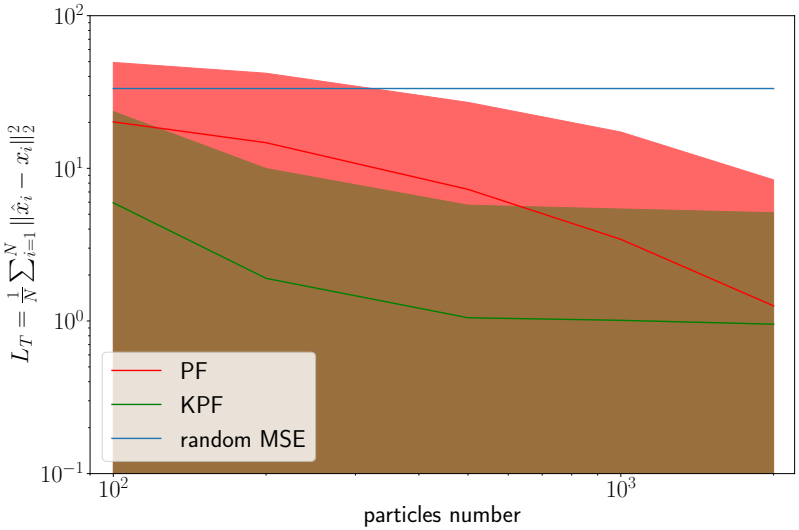


(d) *Labyrinth*

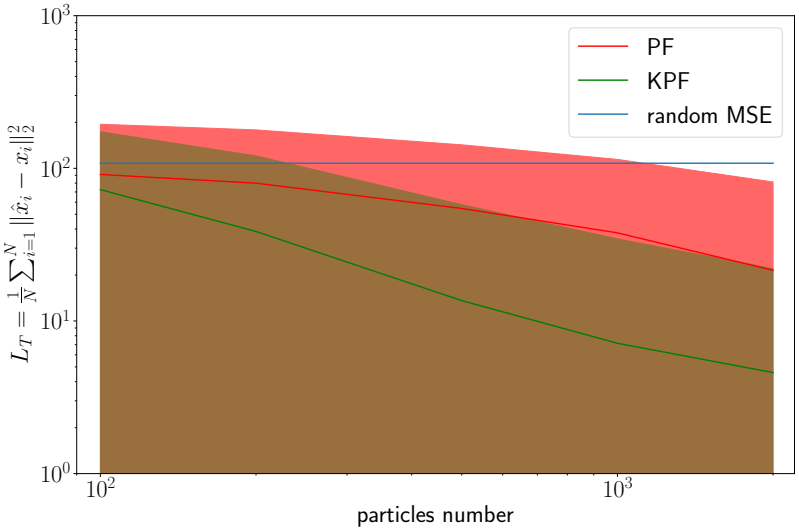
Test environments. Gray – walls, orange – beacons.

- Dataset
  - start at random point
  - input: motion x, y, ϕ and measurements $d_1..d_5$
  - random sampling of ϕ if object hits the wall
  - 8000/1000/1000 trajectories for train, val, test

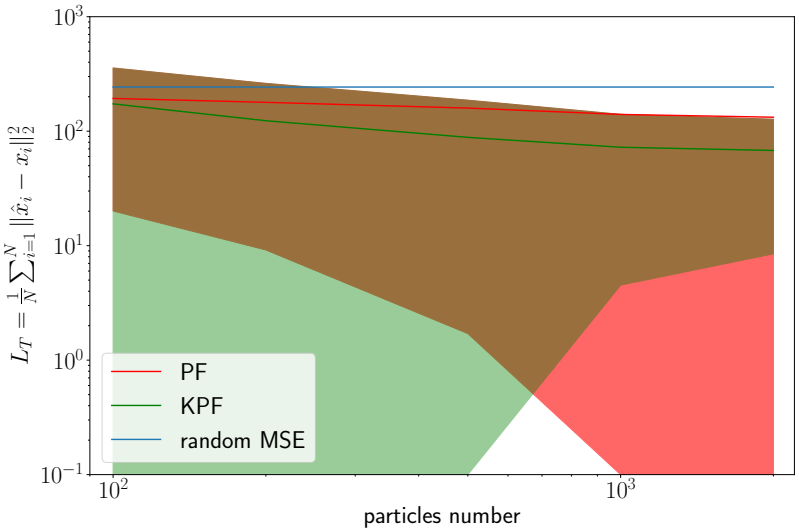
MSE dependence on the number of particles



(a) world 10×10 cells

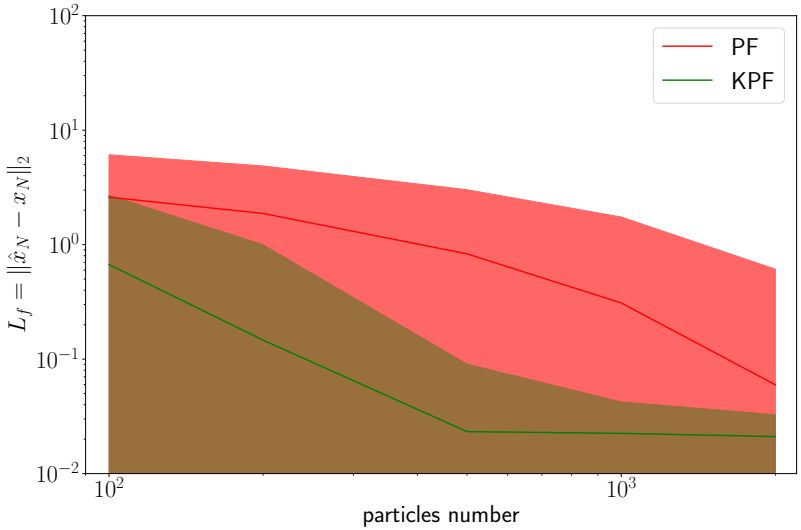


(b) World 18×18 cells

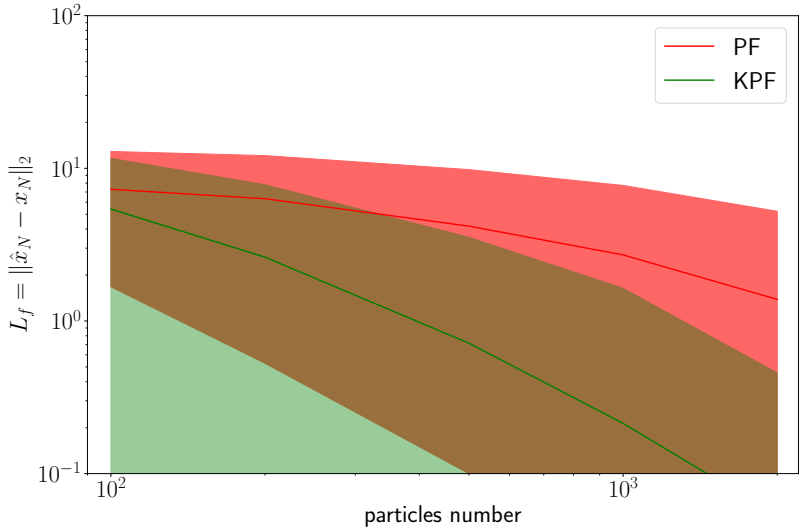


(c) WORLD 27×27 cells

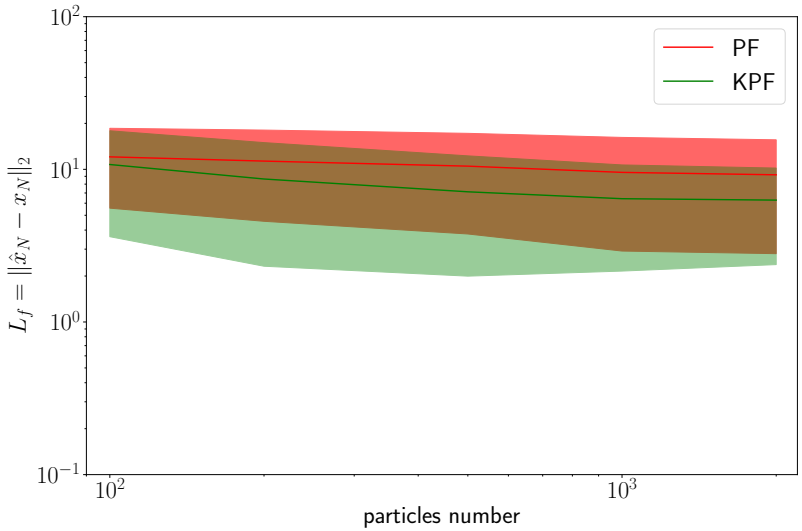
Dependence of final state error on the number of particles



(a) world 10×10 cells



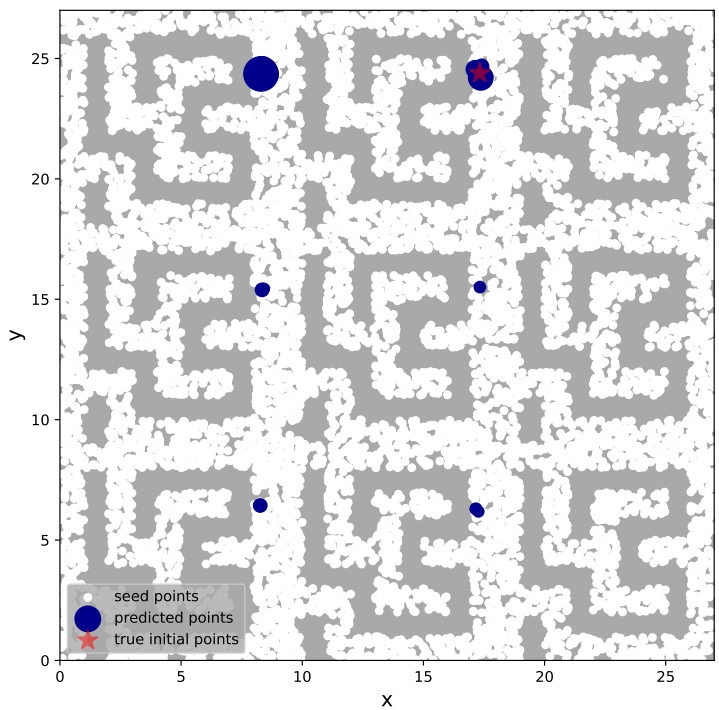
(b) World 18×18 cells



(c) WORLD 27×27 cells

Theoretical limits of filter performance

- Likelihood $\mathcal{L}^i \sim \prod_{t=1}^{T_{\max}} \exp\left(-\frac{\sum_{k=1}^{k_{\max}}(|\mathbf{x}_t^i - \mathbf{s}_k| - \mathbf{m}_{kt})^2}{2\sigma^2}\right)$, where $\mathbf{x}_t^i$ – i-th particle coordinates at t, s_k – k-th beacon coordinate, $\mathbf{m}_{kt}$ – measured distance till k-th beacon at t
- In WORLD 27×27: few "suboptimal" initial points with (almost) same likelihoods
- Grey points – walls and beacons, white points – random initial coordinates blue points – the most probable coordinates
- Distance bet



Classical filters performance

Methods	# particles	MSE mean	MSE std	Fin mean	Fin std	Inference time, sec/particle
PF	100	20.6	29.3	2.6	3.5	9
PF	200	14.7	27.3	1.9	3.00	17
PF	500	7.3	19.8	0.8	2.19	42
PF	1000	3.42	13.9	0.3	1.43	80
PF	2000	1.26	7.15	0.06	0.55	155
KPF	100	5.9	17.6	0.67	2.0	24
KPF	200	1.9	8.1	0.15	0.85	47
KPF	500	1.1	4.7	0.023	0.07	115
KPF	1000	1.0	4.4	0.22	0.02	220
KPF	2000	0.96	4.2	0.021	0.01	430

Table: PF and KPF performance in world 10×10

Methods	# particles	MSE mean	MSE std	Fin mean	Fin std	Inference time, sec/particle
PF	100	91,2	103.3	7.3	5.6	11
PF	200	79.9	98.7	6.3	5.8	20
PF	500	54.6	88.0	4.2	5.6	50
PF	1000	37.8	76.7	2.7	5.0	96
PF	2000	21.5	60.6	1.4	3.9	214
KPF	100	72.3	101.0	5.4	6.2	27
KPF	200	38.9	82.1	2.6	5.2	53
KPF	500	13.6	44.2	0.7	2.8	126
KPF	1000	7.1	27.4	0.2	1.4	250
KPF	2000	4.6	17.4	0.06	0.4	509

Table: PF and KPF performance in World 18×18

Classical filters performance

Methods	# particles	MSE mean	MSE std	Fin mean	Fin std	Inference time, sec/particle
PF	100	193.2	173.1.3	12.1	6.5	13
PF	200	178.7	169.5	11.3	6.8	25
PF	500	159.1	157.4	10.5	6.7	61
PF	1000	140.1	141.5	9.6	6.6	118
PF	2000	132.4	133.1	9.2	6.4	220
KPF	100	173.2	183.6	10.8	7.1	27
KPF	200	123.4	138.2	8.6	6.3	63
KPF	500	88.3	98.3	7.1	5.1	153
KPF	1000	72.4	67.9	6.4	4.3	316
KPF	2000	67.9	59.4	6.3	3.9	630

Table: PF and KPF performance in WORLD 27×27

Are the classical filters efficient?

Results discussion

- KPF outperforms standard PF
- Accurate prediction for simple worlds
- Large error in WORLD 27×27 caused by high symmetry
- The classical filters perform well if the number of particle is large $\rightarrow$ too long computational time
- Few particles may result in poor convergence.

Motivation for neural net-based methods

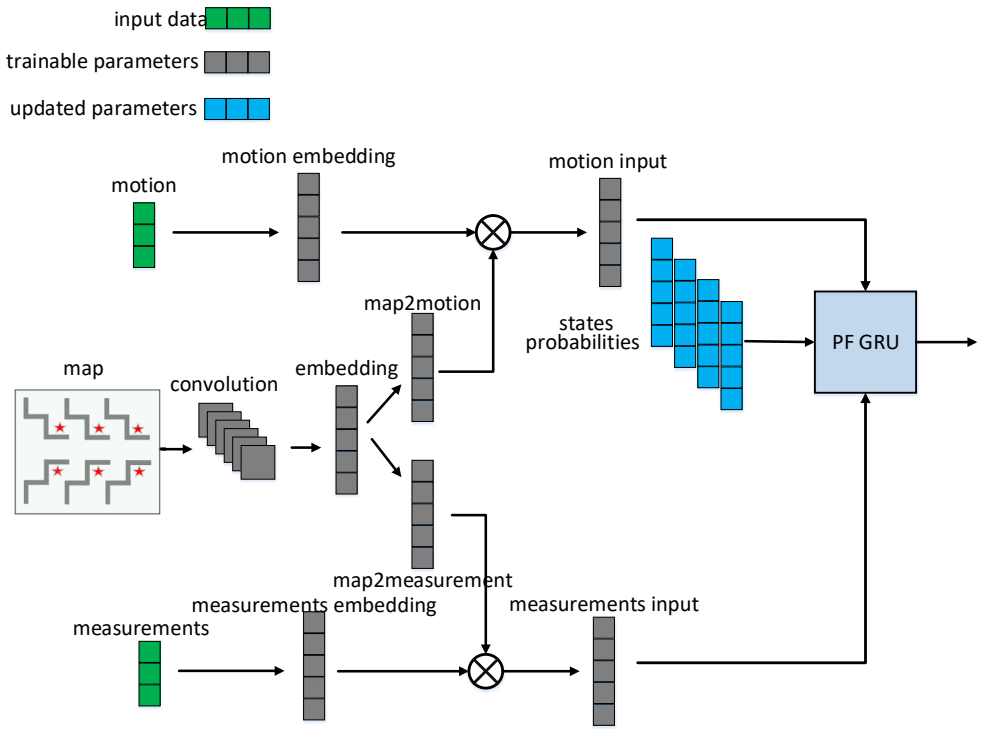
- Memorization in latent space reduces particles number
- Noise level can be trained
- Measurements roughening to reduce degeneracy
- High dimensional latent space to avoid degeneracy

RNN cell approach: $\boldsymbol{h}_{t+1} = RNN(\boldsymbol{h}_t, \boldsymbol{u}_t, \boldsymbol{z}_t), \mathbf{x} = f(\mathbf{h})$

- Standard GRU cell
- Deep particle filter
- Neural Kalman filter
- Model-based particle filter network

Deep particle filter[5]

- Embed motion, measurement and environment data in $\mathbb{R}^d$
- Combine motion embedding and measurement embedding with environment embedding separately
- Feed the result as input to PF GRU cell

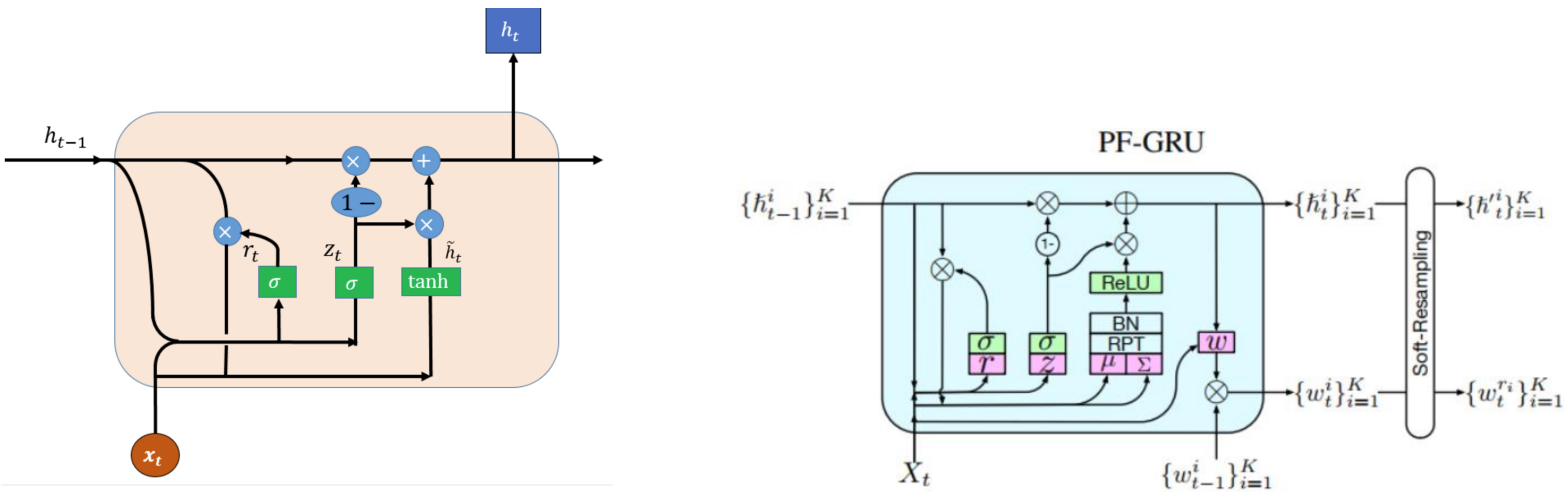


One step of deep particle filter RNN

[5] Xiao Ma et al. *Particle Filter Recurrent Neural Networks*. 2019. arXiv: 1905.12885 [cs.LG].

Deep particle filter: PF GRU cell details

- Replace one hidden state by N – particles number
- Add resample after cell
- Tanh replaced by BatchNorm followed by ReLU for longer truncated BPTT[6]

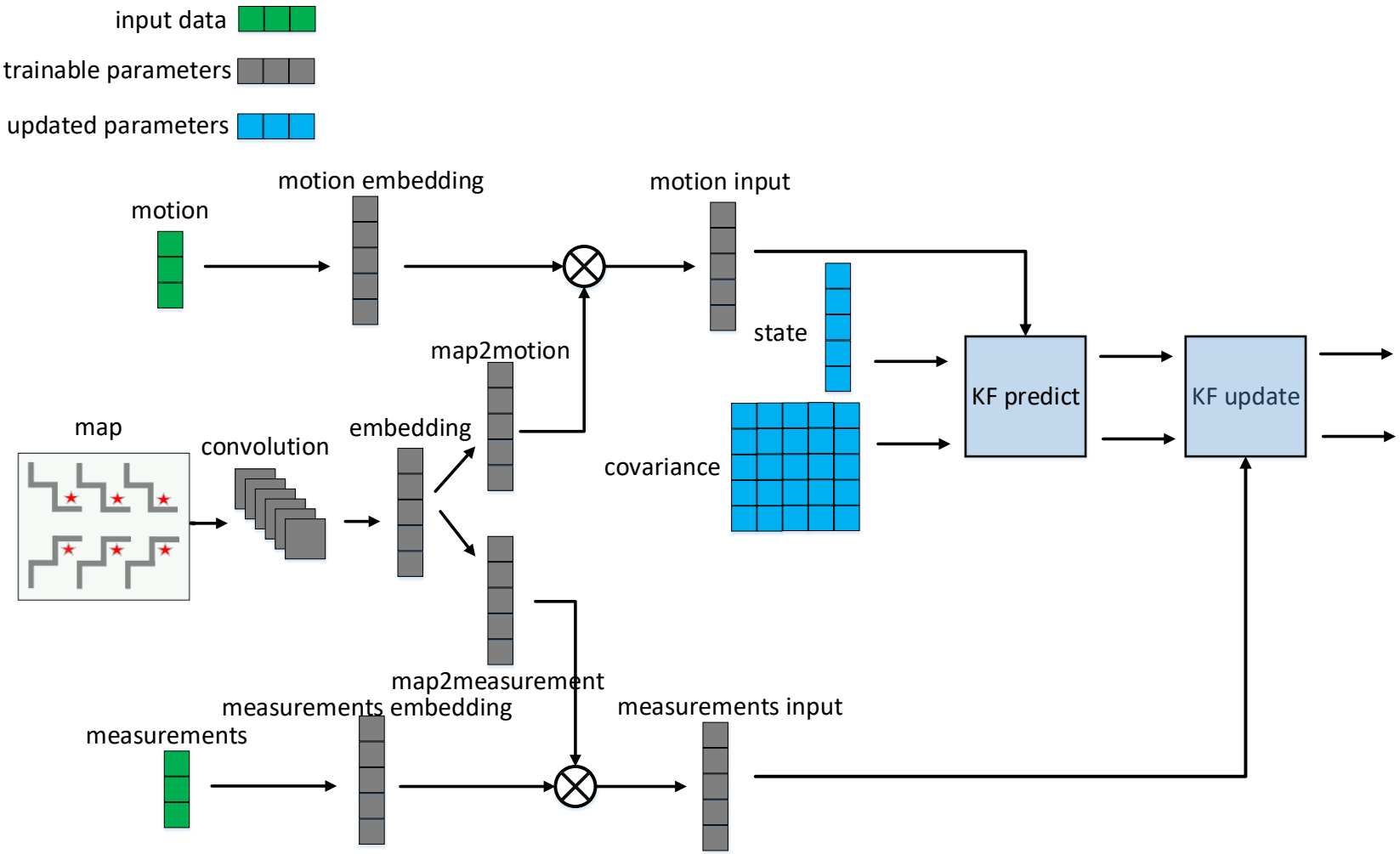


(a) Standard GRU cell (b) PF GRU cell

PF GRU cell vs the standard one

[6] Mirco Ravanelli et al. "Light Gated Recurrent Units for Speech Recognition". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* (2018). URL: https://doi.org/10.1109%2Ftetci.2017.2762739.

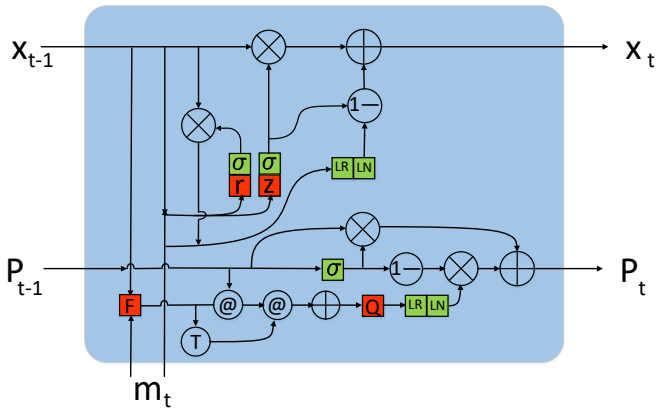
Neural Kalman filter (our approach)



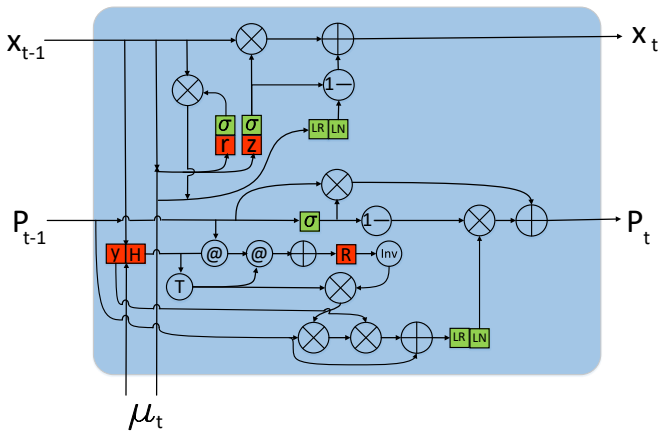
One step of neural Kalman filter RNN

Neural Kalman (our approach)

- RNN-trainable parameters:
  - same architecture as above
  - Kalman GRU predict and update cells
  - Motion and measurement embedding are inputs
  - Sigmoid used as in standard GRU for grad decay/explosion prevention
  - Tanh constraint for a covariance matrix
  - Process and measurement covariance matrices are trainable



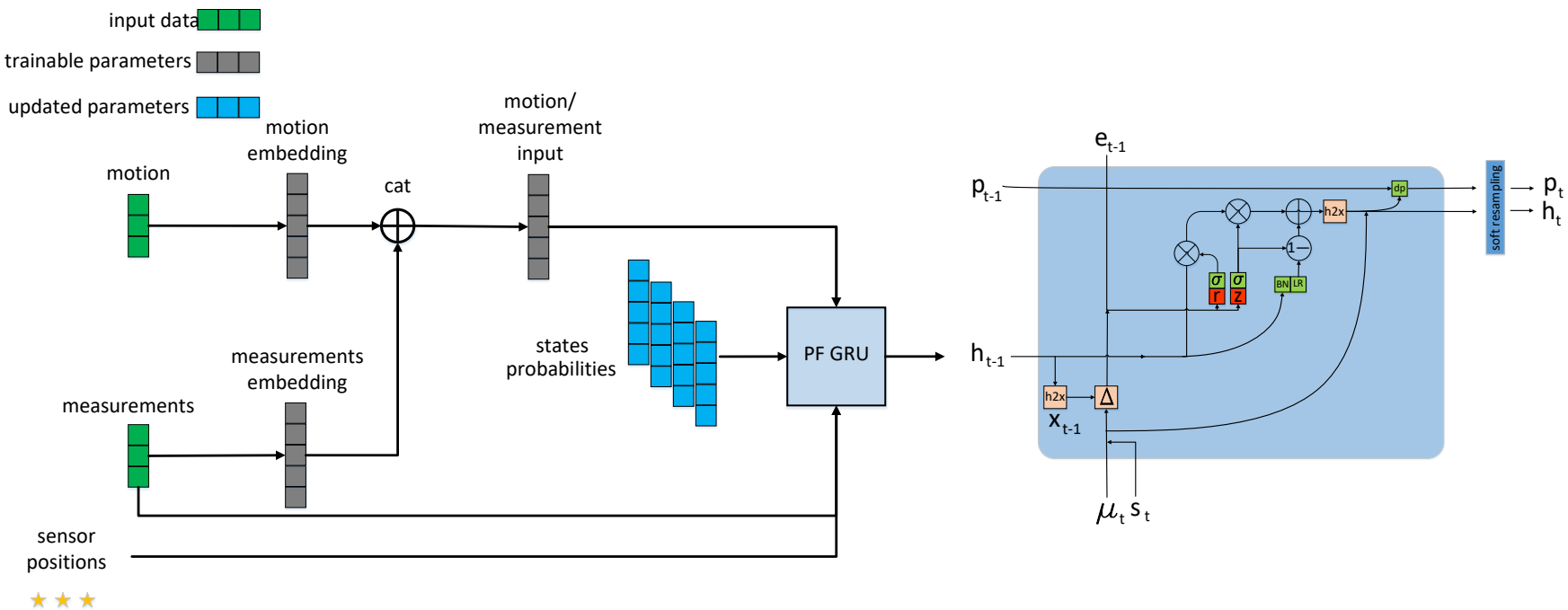
KF GRU predict cell



KF GRU update cell

Model-based particle filter network (our approach)

- No environment encoding, sensors **precise** location input
- Model-based measurement: distances to nearest landmarks as features
- Reduced number of parameters
- Modified GRU cell



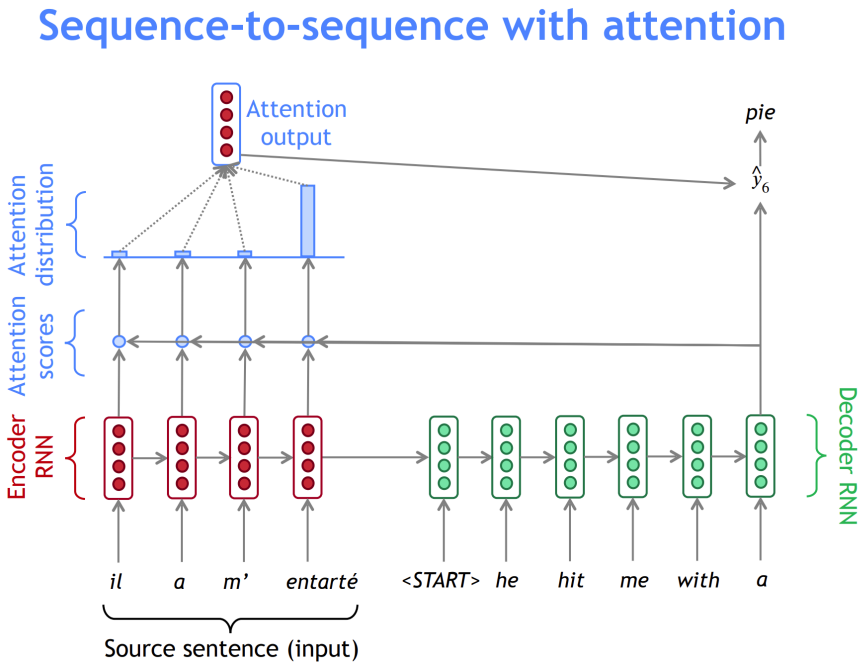
MBPFN architecture

Model-based particle filter network

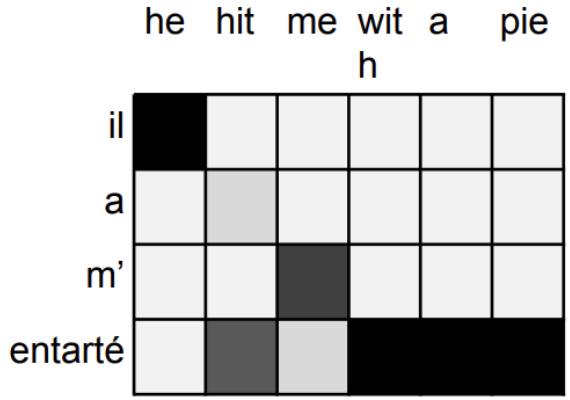
- Get true coordinates from hidden space
  $\mathbf{x} = \mathbf{f}_{h2x}(\mathbf{h}) = \sigma(\boldsymbol{W}_{h2x}(\mathbf{h}) + \mathbf{b}_{h2x})$
- Predict distances to the nearest landmarks $\mathbf{s}$ as
  $\mathbf{z} = D(\mathbf{f}_{h2x}(\mathbf{h}) - \mathbf{s})$
- Convert motion to hidden space:
  $\mathbf{h}_m = \boldsymbol{W}_h[\mathit{leakyrelu}(\mathit{relu}(\boldsymbol{W}_z\mathbf{z}^*) + \mathit{relu}(\boldsymbol{W}_m, \mathbf{m}))]$
- Concat hidden state, predicted, and measured distances as
  $\mathbf{a} = \mathit{cat}(\mathbf{h}_m, \mathbf{z}^*, \mathbf{z})$
- Use $\mathbf{a}$ as input for GRU-like cell, get new $\mathbf{h}_1$
- Reparametrization trick for noise addition
- Get new distances to sensors $\mathbf{z}_1 = D(\mathbf{f}_{h2x}(\mathbf{h}_1) - \mathbf{s})$
- Log likelihood for pf-step
  $\log \mathcal{L} = \boldsymbol{W}_1(\mathit{leakyrelu}(W_{z^*}\mathbf{z}^* + W_{z1}\mathbf{z}_1))$
- Perform resample

How to make better resampling:Attention in NMT

- Given: sentence in french
- Required: translated into english
- Attention determines french word correspondence to an english word (or group of words) and vice-versa



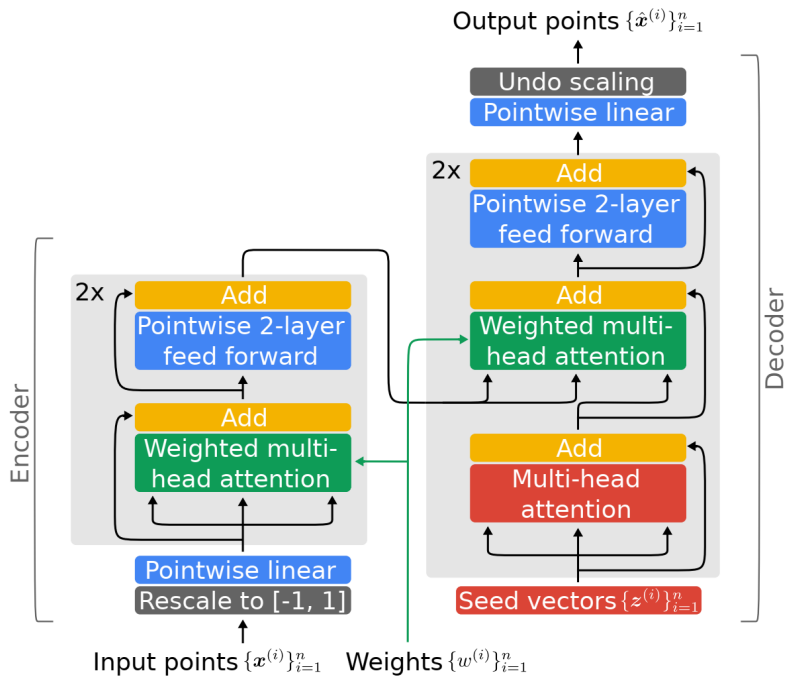
Attention in NMT



Attention matrix

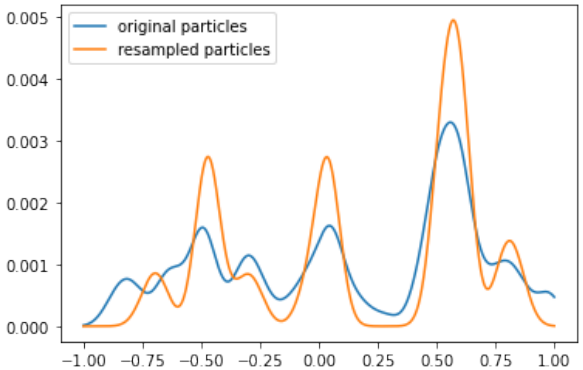
Transformed-based resampling[7]

- Stochastic resample is biased
- Deterministic as an alternative
- Differentiable to use in neural nets
- Transformer: attention is all we need!
- Transformer-based resampling: Transsampling

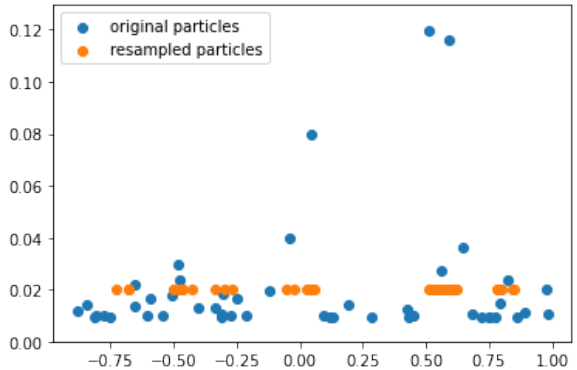


[7] Murphy K Zhu M and Jonschkowski R. *Towards Differentiable Resampling*. 2020. URL: https://arxiv.org/abs/2004.11938.

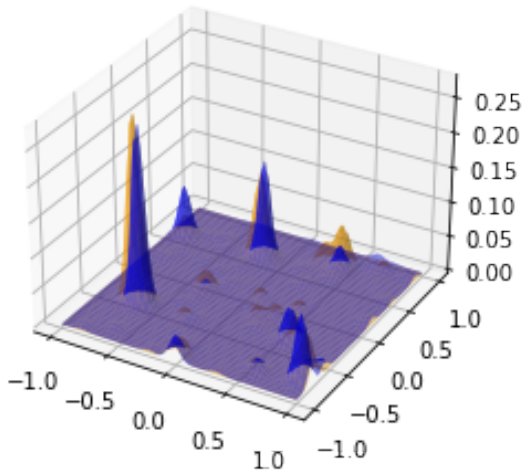
Transsampling performance examples



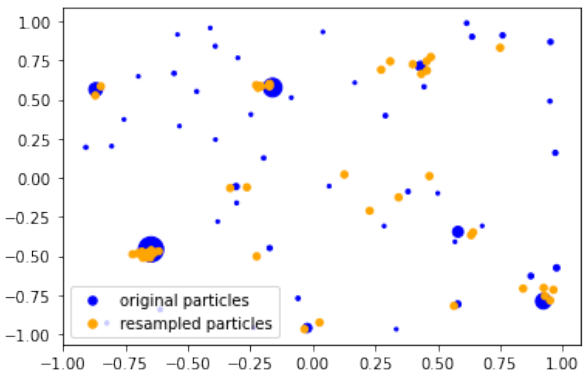
(a) $1D$ case: distributions



(b) $1D$ case: points



(c) $2D$ case: distributions



(d) $2D$ case: points

Transformed-based resampling results

- Drop-in transsampling insted hard- or soft-resampling in Labyrinth world
- Compare PF with hard- and soft-resampling, transsampling, and KPF with hard- and soft-resampling (200 particles in all experiments)

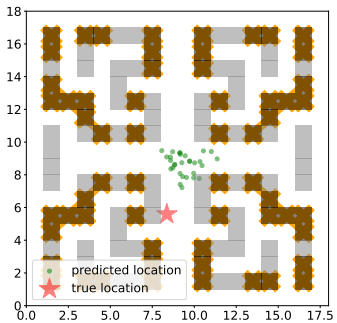
Algorithm	MSE	Final error	Calculation time
PF non-differentiable resampling	185	12	25
PF differentiable resampling	186	12	30
KPF non-differentiable resampling	131	9	90
KPF differentiable resampling	**105**	**8**	100
PF Transsampling	160	5	1000

- Transsampling showed slow performance even in inference mode
- Simultaneous training with deep PF is hardly tractable (even for lower particles number)
- Separate training on representative samples is an option
- Other architectures maybe tried (linformer, reformer)

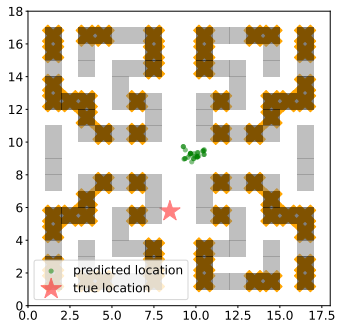
Results for three test worlds

Environment	MSE mean	MSE std	Fin mean	Fin std
world 10×10				
PF 2000	1.25	7.15	0.06	0.055
KPF 1000 (our)	**1.01**	**4.40**	**0.02**	**0.02**
PFRNN	1.80	6.82	0.1	0.07
Neural KF (our)	2.24	8.04	0.21	0.33
GRU	1.45	6.88	0.09	0.08
MBPFN (our)	1.86	6.96	0.19	0.15
world 18×18				
PF 2000	21.47	59.96	1.38	3.84
KPF 1000 (our)	**4.59**	**17.43**	**0.05**	**0.40**
PFRNN 30	12.95	32.83	0.66	1.59
Neural KF (our)	16.30	33.59	0.76	1.41
GRU	12.4	33.60	0.29	1.06
MBPFN (our)	12.5	37.5	0.97	1.63
world 27×27				
PF 2000	141.54	142.31	9.10	6.89
KPF 2000 (our)	**67.85**	**59.38**	**6.28**	**3.91**
PFRNN 30	73.89	66.95	6.51	3.66
Neural KF (our)	92.18	77.78	7.61	3.79
GRU	79.2	69.1	6.69	3.5
MBPFN (our)	70.71	58.86	6.59	3.48

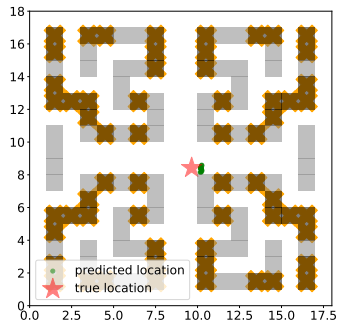
Model based particle filter network against impoverishment



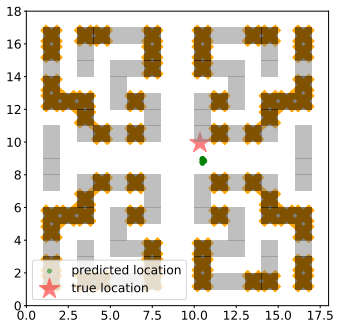
(a) $t = 0$



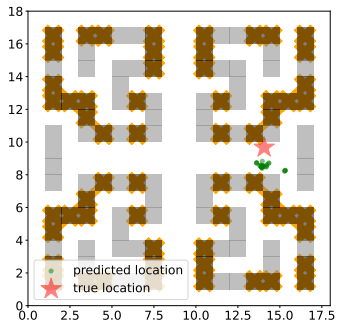
(b) $t = 1$



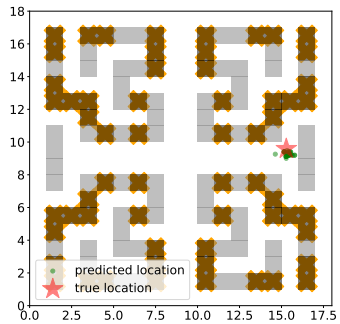
(c) $t = 16$



(d) $t = 24$



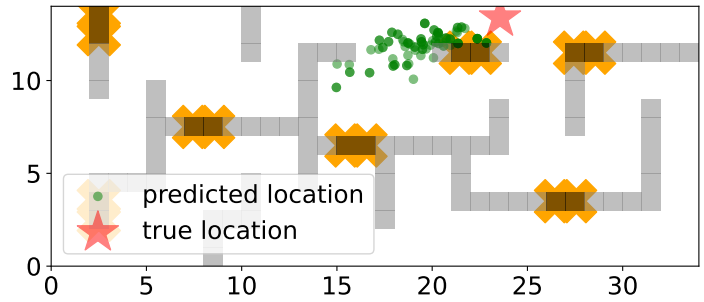
(e) $t = 43$



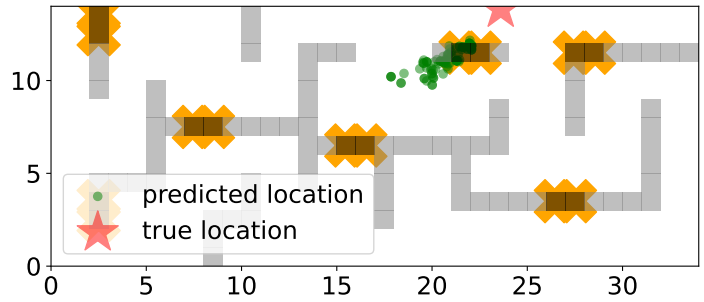
(f) $t = 49$

Model based particle filter network can avoid impoverishment. Once lost target state ($t = 1$) may be found again ($t = 16$).

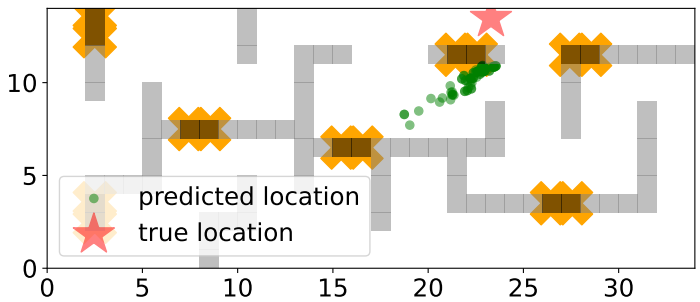
MBPFN dynamic in the *Labyrinth* environment



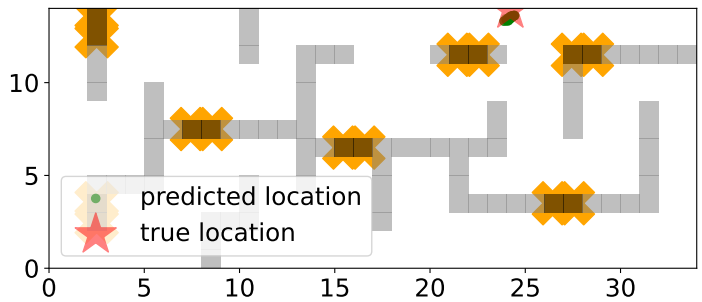
(a) step 1



(b) step 2



(c) step 3



(d) step 11

Performance comparison in *Labyrinth* environment

Methods	# particles	MSE mean	MSE std	Fin mean	Fin std	Train time, hrs	Inference time, sec/particle
PF	2000	5.67	40.22	0.35	2.21	–	0.3
KPF	2000	**0.11**	**3.07**	**0.029**	**0.02**	–	0.89
PFRNN	30	5.87	26.5	0.26	0.13	20	0.01
MBPFN (our)	30	1.45	10.67	0.23	0.13	20	**0.010**

- KPF outperforms all other methods with respect to MSE values as in previous tests
- MBPFN shows best MSE performance from neural net-based
- Neural net-based methods requires long training stage, but are faster in the inference stage

Conclusion

- Different classical and neural network-based methods tested in various environments.
- Particle filter needs too much particles and computational time in complex environments
- The new version of Kalman particle filter is proposed and outperforms considered competitors in test complex environments.
- The novel model-based particle filter network (MBPFN) outperform the considered neural net-based competitors in test complex environemnts.
- Although it gives at smaller accuracy, it is is 100 times faster than KPF.
- We show that MBPFN can avoid impoverishment issue of particle filter
- The deterministic differentiable Transformer-based resampling implemented but showed slow performance