Multimodal Sensor Fusion with Differentiable Filters

Authors: Michelle A. Lee, Brent Yi, Roberto Martín-Martín, Silvio Savarese, Jeannette Bohg

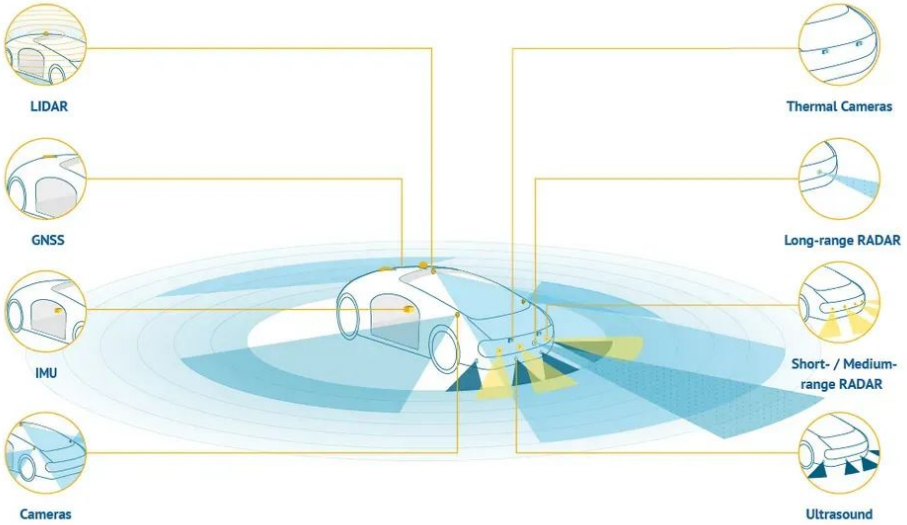
Presenter: Di Yang (Steven) Shi

9/15/2022

Note: Citations reflect ones in the paper.

Problem

Given multiple sensors with heterogeneous sensor data, we'd like to meaningfully combine, or "fuse", their data.



credit: https://www.wevolver.com/article/sensor-fusion-everything-you-need-to-know-

Motivation

Significance

- Unify inputs, extract joint feature predictions, have redundancy (e.g. Lidar filling in occlusion).
- Growing ubiquity and availability of multimodal sensors.
- Don’t underestimate other non-primary sensor inputs.

“For however many things have a plurality of parts and are not merely a complete aggregate but instead some kind of a whole beyond its parts…” – rough translated quote from Metaphysics by Aristotle

Related Work

- Sensor Fusion
  - Classical (Statistically Motivated) (Where's the learning?)
    - Sensor fusion for mobile robot navigation (1998) Can't access because IEEE but this is, to me, a surprisingly old problem.
    - An Introduction to Sensor Fusion
  - Deep (developing trend, maybe circa 2018?) (Here's the learning)
    - A Deep Learning-based Radar and Camera Sensor Fusion Architecture for Object Detection
    - PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation
- Dempster-Schafer Theory
  - Sensor Fusion Using Dempster-Shafer Theory
  - Dempster-Shafer Theory for Sensor Fusion in Autonomous Mobile Robots
- Filter Specific
  - Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors
  - How to Train Your Differentiable Filter

Context – (Recursive) State Estimation, Bayes Filter

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t \mid \mathbf{u}_{1:t}, \mathbf{z}_{1:t})$$

Motion Model (State Transition)	Post Previous Action Pre Sensor Observation,	Post Sensor Observation (normalization factor eta)
$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1})$	$\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_{t-1}) bel(\mathbf{x}_{t-1}) d\mathbf{x}_{t-1}$ (1)	$bel(\mathbf{x}_t) = \eta\, p(\mathbf{z}_t \mid \mathbf{x}_t) \overline{bel}(\mathbf{x}_t)$ (2)

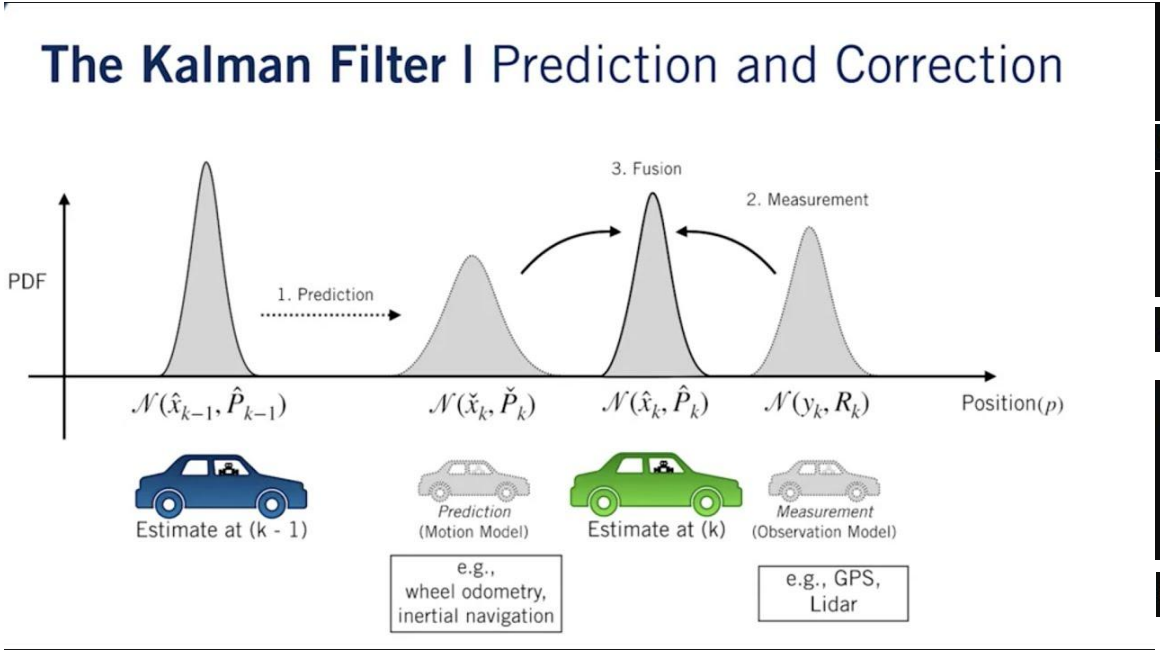
Key Terms

- Differentiable Filter (DF) – differentiable versions of bayes filters
- Kalman Filter (KF) – Gaussian distributions, linear transformations
- Extended Kalman Filter (EKF) – Still Gaussian distributions, non-linear transformations but linearizable (Taylor expansion high school calculus magic)
- Particle Filter (PF) – any distribution, more expensive as it's simulation based

Previous Limitations → Why Differentiable Filters?

- Enable end-to-end learning
- Provide interpretability and explainability
- Reduce error and training time
  - “DPFs reduce the error rate by ~80% or require 87% less training data for the same error rate.” [15]
- Improve generalization
  - “Algorithmic prior improves generalization: while LSTMs fail when tested with a different policy than used for training, DPFs are robust to changing the policy.” [15]
- Multipurpose tool
  - “But how can we find more architectures like the convolutional network for robotics?” [15]

Background – EKF



$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{q}_t) \quad (3)$$
$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{r}_t) \quad (4)$$

$$\overline{bel}(\mathbf{x}_t) \sim N(\hat{\mu}_t, \hat{\Sigma}_t)$$

$$\hat{\mu}_t = f(\mu_{t-1}, \mathbf{u}_{t-1}, 0) \quad (5)$$
$$\hat{\Sigma}_t = \mathbf{A}_{t-1}\Sigma_{t-1}\mathbf{A}_{t-1}^T + \mathbf{Q}_{t-1} \quad (6)$$

$\mathbf{A}_t$ is the Jacobian $\frac{\partial f(\mu_t, \mathbf{u}_t, 0)}{\partial \mu_t}$

$$\mathbf{K}_t = \hat{\Sigma}_t\mathbf{H}_t^T(\mathbf{H}_t\hat{\Sigma}_t\mathbf{H}_t^T + \mathbf{R}_t)^{-1} \quad (7)$$
$$\mu_t = \hat{\mu}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}_t\hat{\mu}_t) \quad (8)$$
$$\Sigma_t = (\mathbf{I}_n - \mathbf{K}_t\mathbf{H}_t)\hat{\Sigma}_t \quad (9)$$

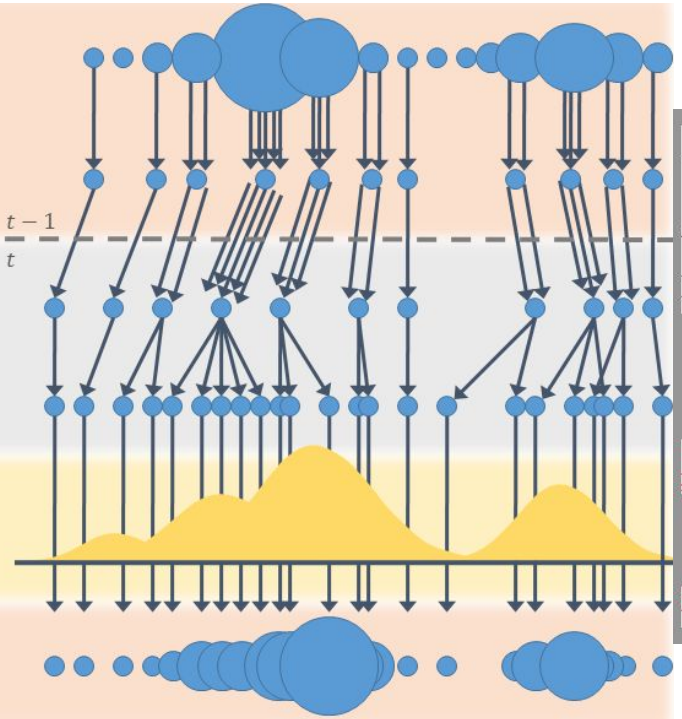
$\mathbf{H}_t$ is the Jacobian $\frac{\partial h(\mu_t, 0)}{\partial \mu_t}$

credit: https://www.youtube.com/watch?v=LioOvUZ1MiM

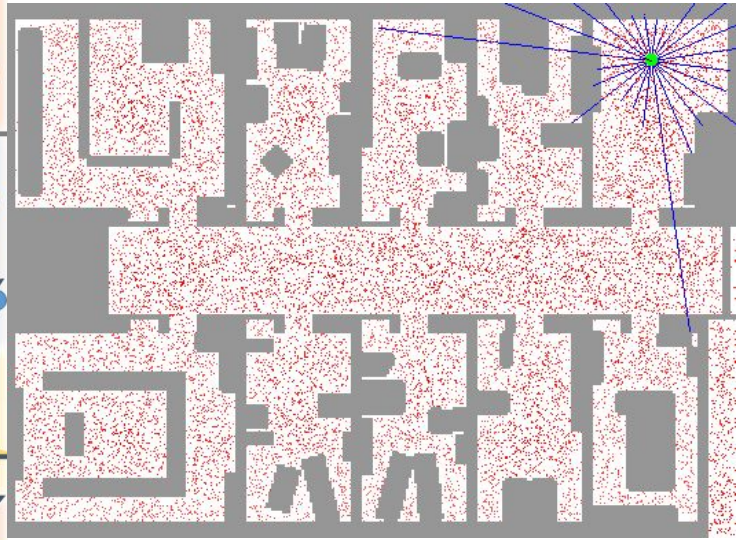
Derivation of 3-9 in Probabilistic Robotics, linked in the Extended Readings Slide. Read at your own risk.

Background – Particle Filtering

- **Begin** with weighted samples from t-1
- **Resample:** draw samples according to $\{w_{t-1}\}^{n=1:N}$
- **Drift:** apply motion model (no noise)
- **Diffuse:** apply noise to spread particles
- **Measure:** weights are assigned by likelihood response
- **Finish:** density estimate



$$\forall_i : \mathbf{x}_t^{[i]} \sim p(\mathbf{x}_t \mid \mathbf{u}_{t-1}, \mathbf{x}_{t-1}^{[i]}) \quad (10)$$



$$\forall_i : w_t^{[i]} \sim p(\mathbf{z}_t \mid \mathbf{x}_t^{[i]}) \quad (11)$$

credit: https://taylor.raack.info/tag/particle-filters/

Fusion

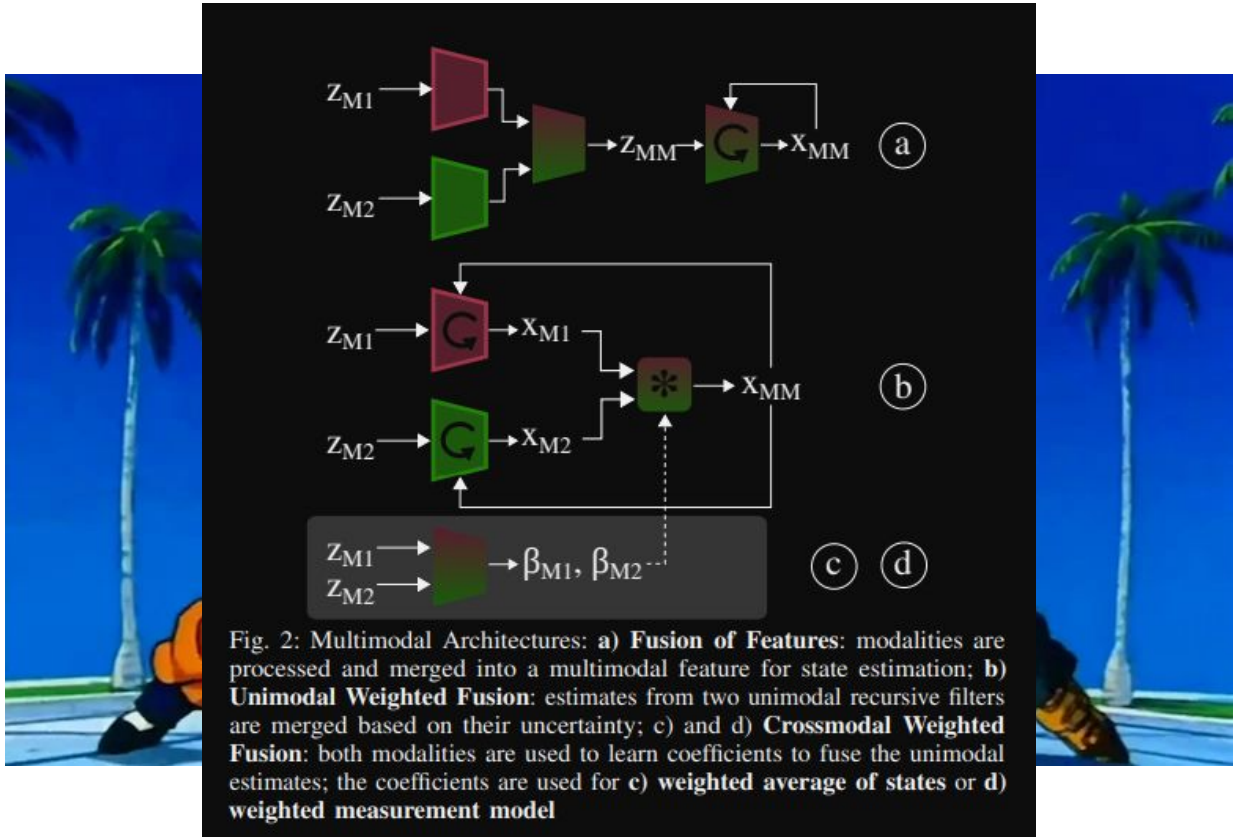
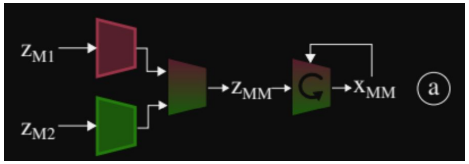


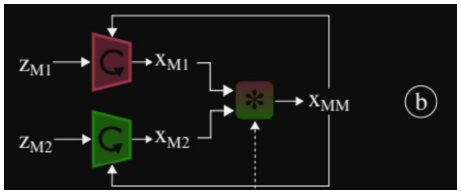
Fig. 2: Multimodal Architectures: **a) Fusion of Features**: modalities are processed and merged into a multimodal feature for state estimation; **b) Unimodal Weighted Fusion**: estimates from two unimodal recursive filters are merged based on their uncertainty; c) and d) **Crossmodal Weighted Fusion**: both modalities are used to learn coefficients to fuse the unimodal estimates; the coefficients are used for **c) weighted average of states** or **d) weighted measurement model**

Fusion Options

- **Fusion of Features (Feature Fusion EKF and Feature Fusion PF)**
  - ➢ Encoder for each sensor → fully connected → filter

  Unimodal Weighted Fusion (Unimodal EKF)
  - ➢ Encoder for each sensor → filter → multiply distributions [12] → joint filter





$$bel(\mathbf{x}_t^{MM}) = \mathcal{N}(\mu_t^{MM}, \Sigma_t^{MM}) \quad (1)$$

$$\mu_t^{MM} = \frac{(\Sigma_t^{M_1})^{-1}\mu_t^{M_1} + (\Sigma_t^{M_2})^{-1}\mu_t^{M_2}}{(\Sigma_t^{M_1})^{-1} + (\Sigma_t^{M_2})^{-1}} \quad (2)$$

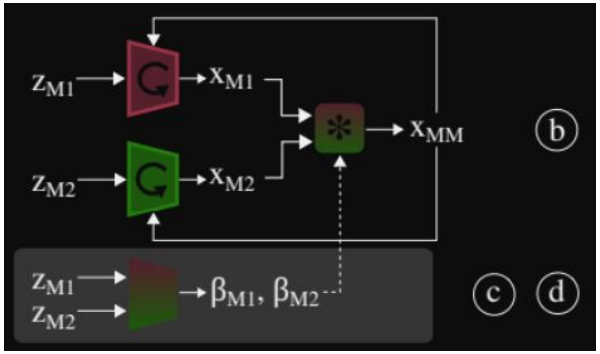
$$\Sigma_t^{MM} = ((\Sigma_t^{M_1})^{-1} + (\Sigma_t^{M_2})^{-1})^{-1} \quad (3)$$

Fusion Options Ctd.

- **Crossmodal Weighted Fusion (Crossmodal Fusion EKF)**
  - Unimodal weighted fusion base
  - Learn weights from multimodal input for multiplying

$$\mu_t^{MM} = \frac{\vec{\beta}_t^{M_1} \odot \mu_t^{M_1} + \vec{\beta}_t^{M_2} \odot \mu_t^{M_2}}{\vec{\beta}^{M_1} + \vec{\beta}_t^{M_2}} \quad (4)$$

$$\Sigma_t^{MM} = \frac{\mathbf{B}_t^{M_1} \odot \Sigma_t^{M_1} + \mathbf{B}_t^{M_2} \odot \Sigma_t^{M_2}}{\mathbf{B}_t^{M_1} + \mathbf{B}_t^{M_2}} \quad (5)$$



- **Unimodal Weighted Measurement Models (Unimodal Fusion PF)**
  - Unimodal weighted fusion-esque
  - Combine measurement models (with a mixture model) instead of the estimated states.

$$w_t^{[i]} = w_t^{M_1,[i]} + w_t^{M_2,[i]} \quad (6)$$

- **Crossmodal Fusion PF**
  - Apply and learn unimodal weights to the particle weights from above.

$$w_t^{[i]} = \beta_t^{M_1} * w_t^{M_1,[i]} + \beta_t^{M_2} * w_t^{M_2,[i]} \quad (7)$$

Experimental Setup – Pretraining & Tasks

Pretraining

- **Dynamics Models – 1,4,8,16 step prediction errors minimization**
- **PF Measurement Models – observation-conditioned**
- **EKF Virtual Sensor Models – observation-conditioned**
- **Measurement Uncertainties – Not pretrained, learned end-to-end**

Tasks

- **Pushing a planar object in simulation and in the real world**
  - **"Estimate the 2D position of the unknown object on a table surface while the robot intermittently interacts with the object"**
- **Opening a door in simulation**
  - **"Estimate the 2D position of the door's revolute joint and its joint angle while the door is being opened." For context, the revolute joint is what the door handle is connected to inside the door.**

Experimental Setup – Dataset

- **Data Provided**
  - Gray-scaled images from an RGB camera (1x32x32)
    - Randomly blacked out at rates 0.4 and 0.8 to simulate noise and sensor failure.
  - Forces from a force/torque sensor (and binary contact information)
  - 3D Position of robot end-effector (i.e. the robot hand)
- **Collected with MuJoCo [29] (Multi-Joint dynamics with Contact)**
  - Simulated planar pushing: “1000 trajectories with 250 steps at 10Hz, of a simulated Franka Panda robot arm pushing a circular puck.”
  - Simulated door opening: “600 trajectory with 800 steps at 10Hz each, of the Franka Panda robot pushing and pulling a kinematically constrained door object.”
  - Real planar pushing: “Stitch[ed] 1000 pushing trajectories with 45 steps at 18Hz” with MIT pushing dataset [30].

Experimental Setup – Baselines & Success Metrics

- **Baseline: An unstructured LSTM model**
- **Metrics: Root Mean Squared Error (RMSE), Joint Angle error**
- **Success: LSTM Parity**

Experimental Results – Error Comparison

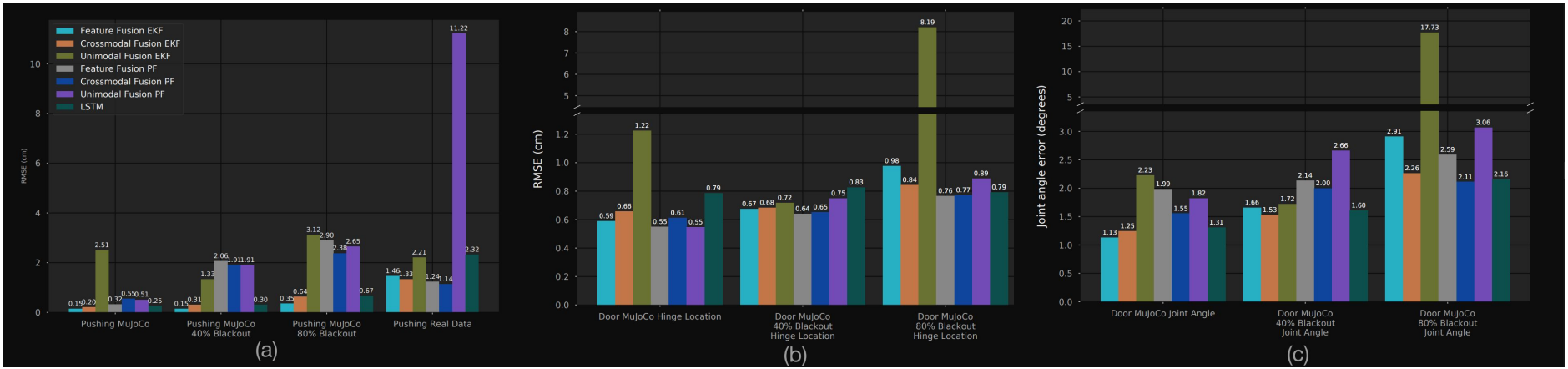


Fig. 3: We show the results of our tasks: pushing and door manipulation. In (a), RMSE error in centimeters for each filter in our validation set for MuJoCo pushing simulation data, MuJoCo pushing simulation data with 40% and 80% image blackout, and real world pushing. In (b) and (c) respectively, RMSE error in centimeters and joint angle error in degrees for each filter in our validation set for the MuJoCo door simulation data and MuJoCo simulation data with 40% and 80% image blackout.

Experimental Results – Offsets by Sensor Type

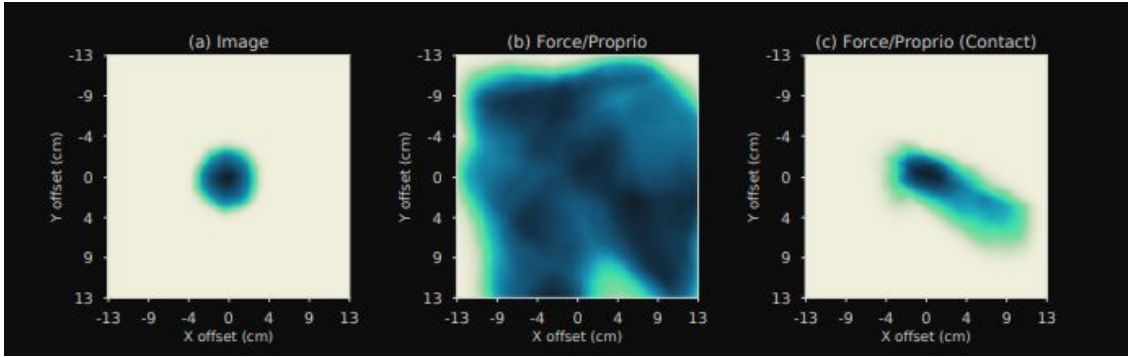


Fig. 4: Examining outputs from unimodal measurement models within the proposed fusion architectures can help us understand how our neural networks are fusing each sensing modality. Using a `Crossmodal Fusion PF` model trained end-to-end on the `Mujoco door dataset`, we plot how the likelihood of a state changes as a function of positional offsets from the ground-truth door hinge position: **(a)** using the unimodal image measurement model, **(b)** using the force/proprioception model on a timestep before the robot has made contact with the door, and **(c)** using the force/proprioception model after the robot gripper has made contact with the door.

Experimental Results – Dynamic Weighting

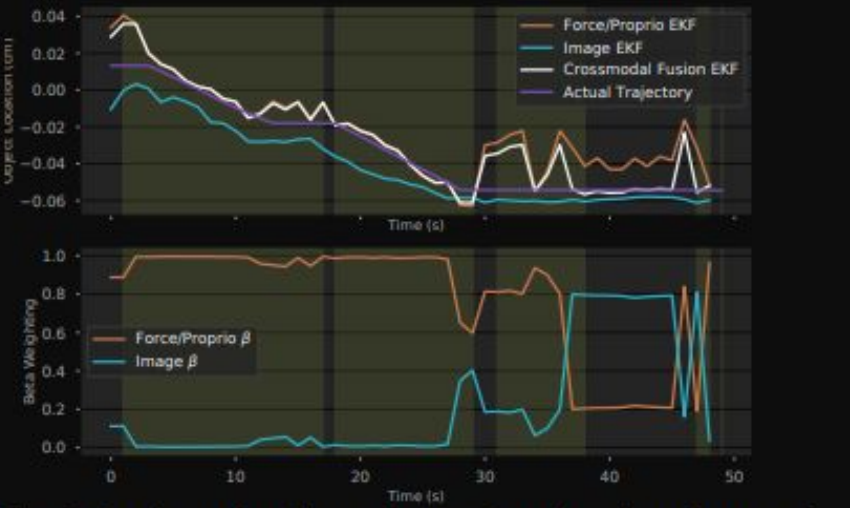


Fig. 5: For the real pushing task, we show the estimated states of the filters (top) and the weights for the unimodal filters (bottom) from the `Crossmodal Fusion EKF`. Contact is indicated by the purple background. The `Crossmodal Fusion EKF` learns to weight the force and proprioceptive unimodal EKF when the robot is touching the object, and switch to the image unimodal EKF when the robot breaks contact with the object.

Experimental Results Discussion

- Comparable Results Shown (Error Comparison)
- Piecewise Explainability (Offsets by Sensor Type), Patchability
  - Isolate both the contribution and weighting of any given source of sensor data.
  - Modularized architecture
- Robustness and Redundancy (Dynamic Weighting)
- End-to-End Deployment – successfully learned and applied intermediate models

Critique, Limitations, Open Issues, Future Work

- Critiques, Limitations
  - Tradeoff between performance and filter fit by distribution.
    - (See Real-time Particle Filters and Particle Filters for Positioning, Navigation and Tracking on trying to address this).
  - Lacking experimentation on more adversarial sensors and measurements.
    - consider two sensors that gave opposite measurements, tradeoff between recency bias and correction speed
- Open Issues, Future Work
  - Fusing > 2 sensor inputs, potential challenges in scaling
  - Unimodal learning overshadowing sensor fusion in cost/benefit perspective
    - establish a metric of “fusion gain” to measure

Extended Readings (Some Repeated)

- Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors
- Probabilistic Robotics Chapters 2,3,4 (Recursive State Estimation, Gaussian Filters, Non Parametric Filters, respectively)
- Site, PyTorch Filter Library, and Source Code
- How to Train Your Differentiable Filter

Summary

- Problem: Cleverly fusing multimodal sensor input, ideally in a deployable and interpretable manner
- Importance and Difficulty: Sensor fusion is, currently, a promising avenue for providing accuracy and confidence gains, and redundancy and robustness as a result. However, it should be computed in a way that's responsive, generalizable, scalable, and explainable.
- Limitations of Prior Work: Older state estimation models were not easily explainable, patchable, or deployable.
- Key Insights
  - We can use deep learning to learn the dynamics and measurement models to make filters differentiable, thus enable end-to-end learning and deploying.
  - A model can learn to update weightings to contextually adapt and focus on the sensor modals which are most accurate for its current state.
- Demonstrated: We can train and optimize sensor fusion end-to-end in addition to effectively explaining decision rationale at each timestep and being able to patch erroneous behaviors.