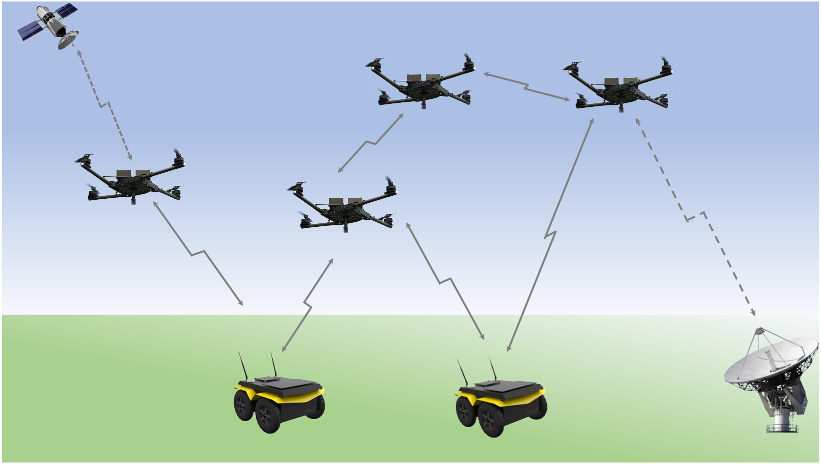
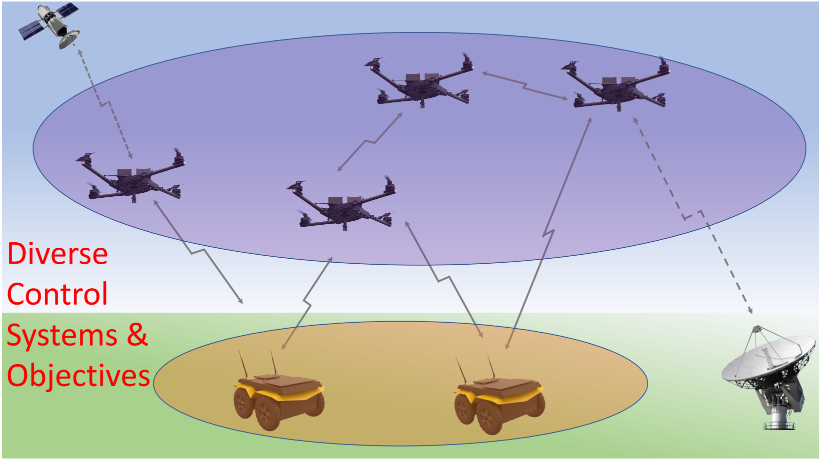
POMDPs and Reinforcement Learning for Cross-Layer Control and Network Optimization

John M. Shea and Caleb M. Bowyer

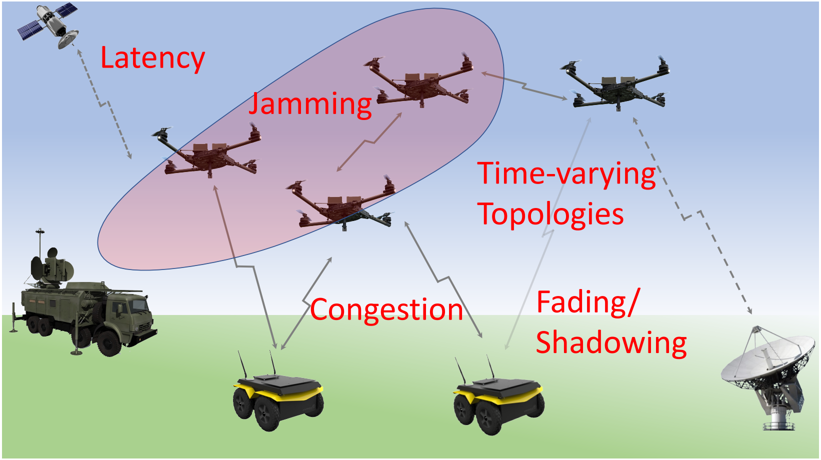
Big Picture: Joint Optimization of Control and Networks



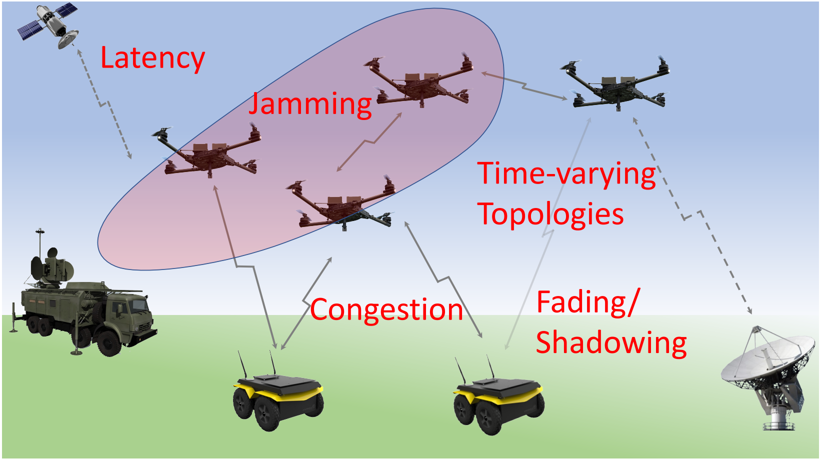
Challenges



Challenges


Latency
Jamming
Time-varying
Topologies
Congestion
Fading/
Shadowing

Challenges



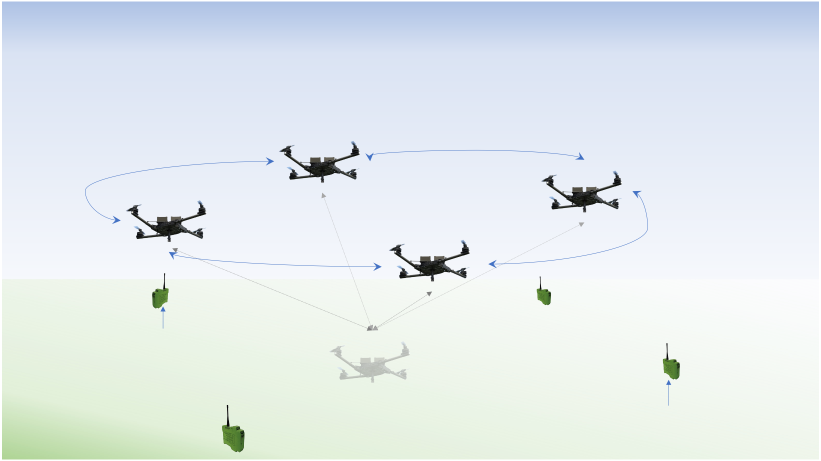
Multi-objective, distributed, partially observable
⇒ Partially observable stochastic game

Constrained Problem: Distributed Sensing

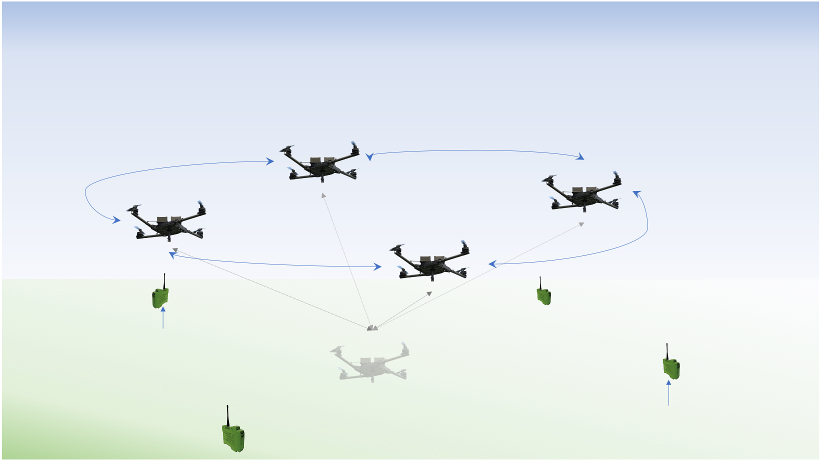
Constrained Problem: Distributed Sensing

Distributed Sensing and Coordination: Who senses and transmits? $\Rightarrow$ Partially observable, multi-agent MDP

Constrained Problem: Distributed RF Localization



Constrained Problem: Distributed RF Localization



Coordinated sensing and localization: When to sense? When to synchronize? $\Rightarrow$ Partially Observable MDP

Application: Distributed Localization in GPS-Denied Environments

- Desire low cost, low complexity, robust, high-performance solutions to tracking/RADAR in GPS-denied environments

Application: Distributed Localization in GPS-Denied Environments

- Desire low cost, low complexity, robust, high-performance solutions to tracking/RADAR in GPS-denied environments
  - **Low cost, low complexity:** sensors have unreliable clocks and noisy RF
  - **Robust:** no single point of failure $\Rightarrow$ distributed sensors with robustness to failure of individual sensors
  - **High-performance:** need to generate reliable localization estimates using noisy ToF measurements from noisy clocks

Application: Distributed Localization in GPS-Denied Environments

Application: Distributed Localization in GPS-Denied Environments

Need for Synchronization

- Given accurate sensor locations and tightly synchronized clocks, distributed sensor networks can produce accurate location estimates
- With unreliable clocks, timing drifts between synchronization times reduces localization accuracy
- Clock synchronization requires communication among sensors and localization may not be possible during the synchronization times

Need for Synchronization

- Given accurate sensor locations and tightly synchronized clocks, distributed sensor networks can produce accurate location estimates
- With unreliable clocks, timing drifts between synchronization times reduces localization accuracy
- Clock synchronization requires communication among sensors and localization may not be possible during the synchronization times

Need to optimize between localization and synchronization

System Model

- Network of m stationary sensing agents
- Single asset to be tracked:
  - Asset transmits beacon signal at known times to agents to facilitate tracking in GPS-denied environment
  - Asset moves according to known Markov model

System Model – cont.

- Sensors measure time-of-flights (ToFs) of beacon signal and reference leader agent localizes (LOC) asset once measurements are fused
- Each agent's clock drifts independently and variance of clock signals increase with time
- Agents can synchronize (SYNCH) clocks at expense of not being able to measure ToFs during that time

Model-Free Localization for general $3D$ space

- Let coordinates of asset and sensor i in interval k be $(x_{k,a}, y_{k,a}, z_{k,a})$ and (x_i, y_i, z_i)

Model-Free Localization for general $3D$ space

- Let coordinates of asset and sensor i in interval k be $(x_{k,a}, y_{k,a}, z_{k,a})$ and (x_i, y_i, z_i)
- Using sensor $m-1$ as a reference, form linear equations $\mathbf{A} \cdot \mathbf{v}_k = \boldsymbol{\beta}_k$
- Here $\mathbf{v}_k = [x_{k,a}, y_{k,a}, z_{k,a}]^T$, $\mathbf{A}$ is a matrix with row i given by

$$\mathbf{A}_i = [2(x_i - x_{m-1}),\ 2(y_i - y_{m-1}),\ 2(z_i - z_{m-1})], \\ i \in \{0, 1, \ldots, m-2\},$$

and $\boldsymbol{\beta}_k$ is a column vector with component

$$\beta_i = c^2\left(\widehat{\tau}_{k,i}^2 - \widehat{\tau}_{k,m-1}^2\right) - \left(x_i^2 - x_{m-1}^2\right) - \left(y_i^2 - y_{m-1}^2\right) \\ - \left(z_i^2 - z_{m-1}^2\right), \qquad i \in \{0, 1, \ldots, m-2\}$$

Localization Solution

- The linear least squares solution is $[\widehat{x}_{k,a}, \widehat{y}_{k,a}, \widehat{z}_{k,a}]^T = \mathbf{A}^\dagger \boldsymbol{\beta}_k$ where $\mathbf{A}^\dagger$ is the Moore-Penrose pseduo-inverse of $\mathbf{A}$
- Not always solvable, depending on sensor topology – can also solve constrained least squares problem

Optimal Coordination of Localization and Synchronization

- Pure localization generally not good enough because of noisy clocks
- Does not inform system of when SYNC is needed

- Resolve both problems by treating tracking problem as HMM and treating choice of SYNC/LOC as control problem

Optimal Coordination of Localization and Synchronization

- Pure localization generally not good enough because of noisy clocks
- Does not inform system of when SYNC is needed

- Resolve both problems by treating tracking problem as HMM and treating choice of SYNC/LOC as control problem
  - Since true state of asset not directly observable, this results in a **Partially Observable Markov Decision Process (POMDP)**

POMDP

1. State space $\mathcal{M} \times \mathcal{T}$, tuples of movement state $\mathcal{M}$ and time since last sync $\mathcal{T} = \mathbb{Z}^+$
2. Two controls $\mathcal{U} = \{u_l, u_s\}$, corresponding to *localize* or *synchronize*
3. Continuous set of noisy observations from ToF measurements, $\mathcal{O}$
4. State-to-state transition function: $p_{ij}(u) = \Pr(M_{k+1} = j | M_k = i, U_k = u) \; \forall \; i, j \in \mathcal{M}$ based on Markov movement model
   (Note time since last sync is deterministic given previous state and control

POMDP – cont.

5. State-to-observation density function: $q_{jo}(u) = f(O_{k+1} = o | M_{k+1} = j, U_k = u) \; \forall \; j \in \mathcal{M}, \; \forall \; o \in \mathcal{O}$ from ToF noise
6. Cost function: MMSE of position estimate

Belief States, Observation Sequences and Control Sequences

- Given:
  - $\mathbf{o}_k$: vector of observations up to interval k
  - $\mathbf{u}_{k-1}$: vector of controls leading up to interval $k-1$

Belief States, Observation Sequences and Control Sequences

- Given:
  - $\mathbf{o}_k$: vector of observations up to interval k
  - $\mathbf{u}_{k-1}$: vector of controls leading up to interval $k-1$
- Belief state at interval k is $\mathbf{b}_k$:

$$b_k(m) = \Pr\left(M_k = m \,|\mathbf{o}_k, \mathbf{u}_{k-1}\right)$$

- the maximum a posteriori (MAP) estimate of the asset state is

$$\widehat{M}_k = \arg\max_{m \in \mathcal{M}} \ \mathbf{b}_k\left(m\right).$$

Belief Update

- Continuous observation space (localization results) – most papers consider only a finite observation space

$$b_{k+1}(m_{k+1}) = \frac{f\left(\mathbf{o}_{k+1}, m_{k+1} \left| \mathbf{u}_k \right.\right)}{f\left(\mathbf{o}_{k+1} \left| \mathbf{u}_k \right.\right)} \tag{1}$$

where

$$f(\mathbf{o}_{k+1}, m_{k+1} \left| \mathbf{u}_k \right.) = f(o_{k+1} \left| m_{k+1} \right.) \sum_{m_k \in \mathcal{M}} \Pr(m_{k+1} \left| m_k, u_k \right.) \cdot f(\mathbf{o}_k, m_k \left| \mathbf{u}_{k-1} \right.)$$

Localization Density in Belief Update

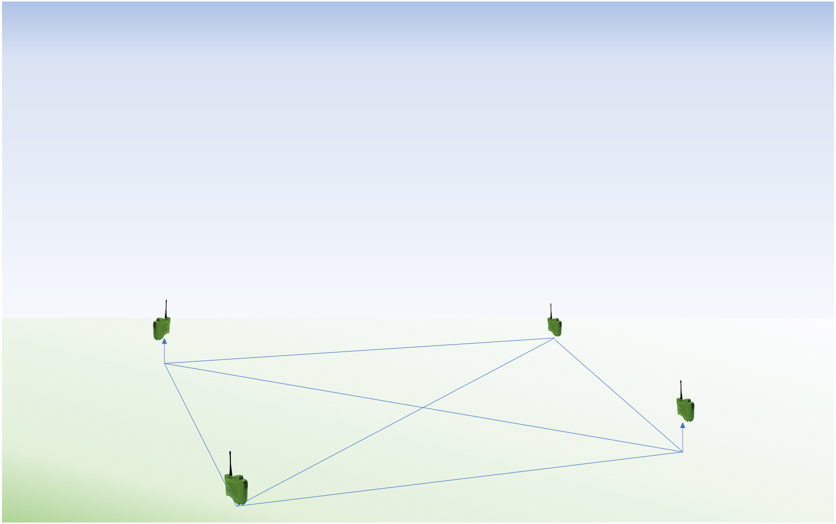
- the conditional distribution of o_k given m_k and $T_k^{(s)}$ is modeled as multi-variate Gaussian with mean determined by M_k and covariances determined by ToF variance, σ_N
- Relation between σ_N and covariances is determined empirically:
  - position estimate variances are proportional to σ_N^2 (input ToF Variance)
  - different coordinates are approximately uncorrelated and are thus treated as independent

Beleif Updates During Sync

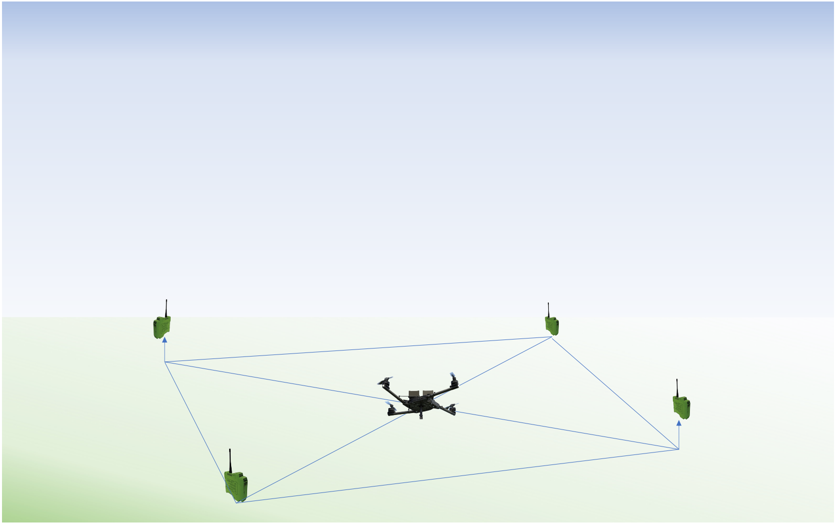
- If the control is **sync**, then no measurement o_{k+1} is available from localization;
- Update the belief by applying the Markov model transitions probabilities

$$\mathbf{b}_{k+1} = \mathbf{P} \cdot \mathbf{b}_k$$

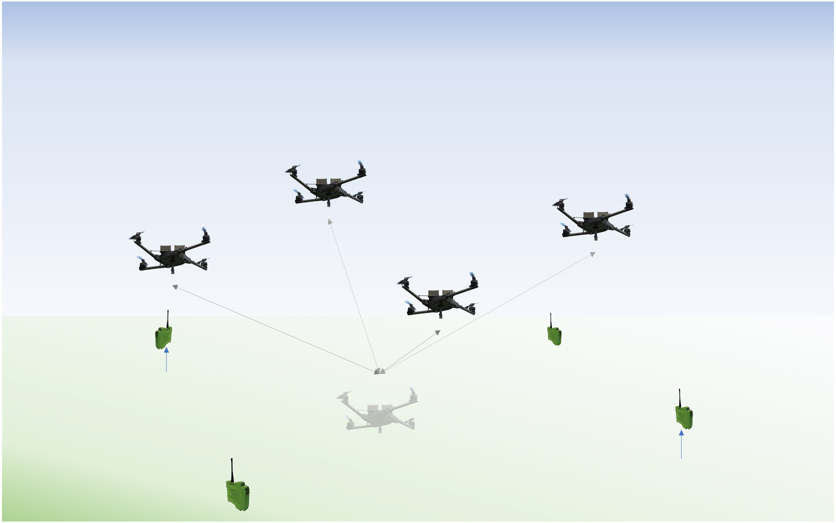
Simulation Model: 3D Sensing Model



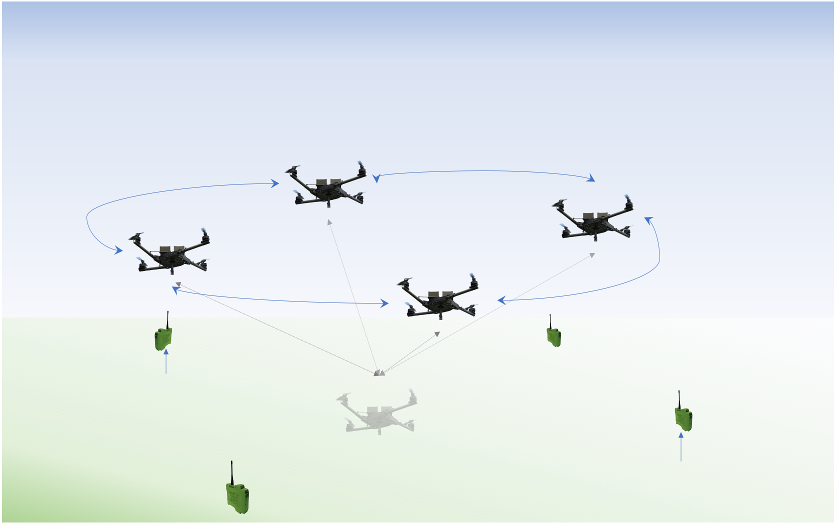
3D Sensing Model and Asset Ground Station



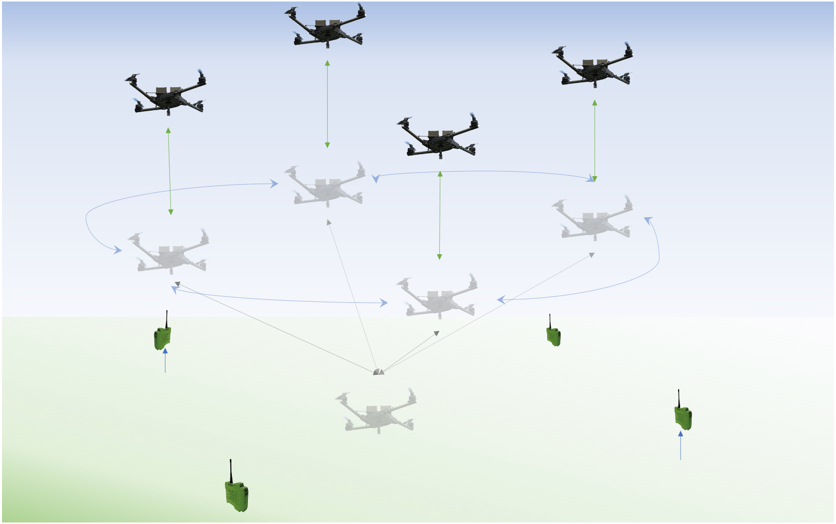
3D Sensing and Asset Movement Model



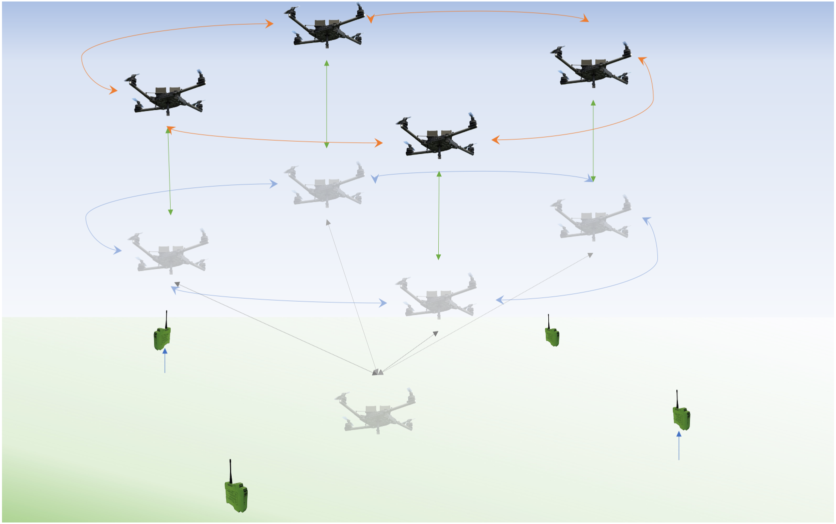
3D Sensing and Asset Movement Model



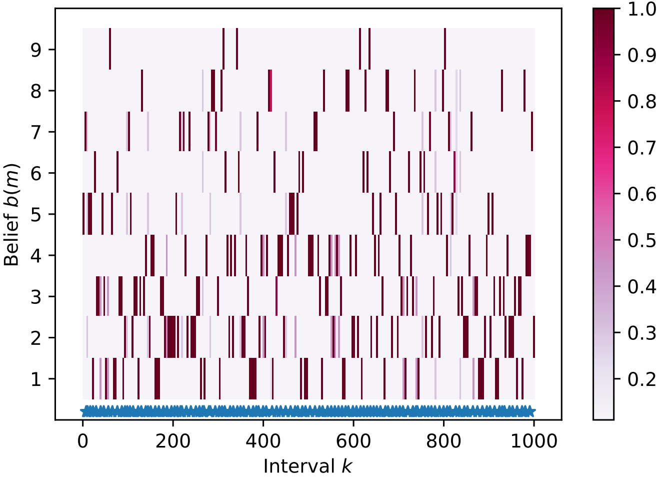
3D Sensing and Asset Movement Model



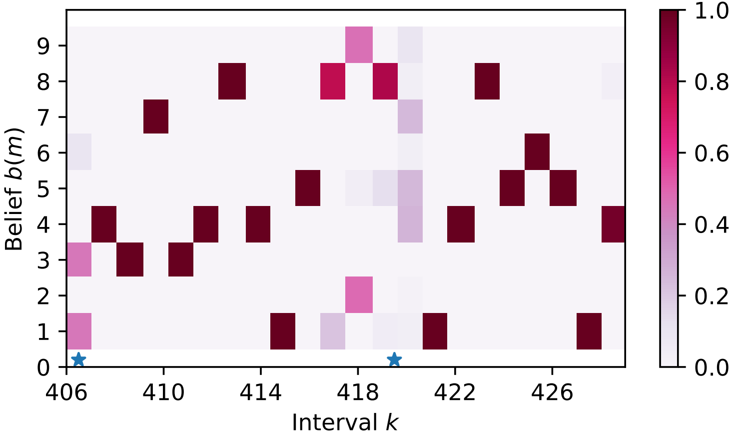
3D Sensing and Asset Movement Model



Belief State Evolution



Belief State Evolution Detailed View



Belief State

- Belief state is a sufficient statistic for deciding the control u_k at stage k
- However: state space has $|\mathcal{M}|$ continuous dimensions
- To apply Q-learning, need to do some form of approximation:

Approximate Q-Learning

Two fundamental approaches

- **Q-function Approximation:**
  - Linear function of beliefs: replicated Q (RQ)-learning
  - Nonlinear: Deep Q-learning

Approximate Q-Learning

Two fundamental approaches

- **Q-function Approximation:**
  - Linear function of beliefs: replicated Q (RQ)-learning
  - Nonlinear: Deep Q-learning
- **Belief State Approximation:**
  - Approximate belief distribution by parameterized distribution (e.g., Gaussian) and quantize parameters: our triple Q (TQ)-learning
  - Belief uncertainty can be represented by universal measures (entropy) or application-specific measures (EMSE)

Belief State Compression

- Compress beliefs and syncing information into triple of **discrete** values $\underline{\mathrm{m}}_k = [\hat{m}_k, T_k^{(s)}, \nu_k]$:

Belief State Compression

- Compress beliefs and syncing information into triple of **discrete** values $\underline{\mathrm{m}}_k = [\hat{m}_k, T_k^{(s)}, \nu_k]$:
  - $\hat{m}_k$: ML estimate for movement state
  - $T_k^{(s)}$: is the number of time since last **sync**
  - ν_k: Quantized EMSE
- Called: *Triple Q-Learning* (TQ-Learning)

TQ-Learning Update

- Use tabular Q-learning with usual update rule:

$$Q(\underline{\mathrm{m}}, u) \mathrel{+}= \alpha \left[c + \gamma \min_{u'} Q(g(\underline{\mathrm{m}}, u), u') - Q(\underline{\mathrm{m}}, u) \right].$$

TQ-Learning Update

- Use tabular Q-learning with usual update rule:

$$Q(\underline{\mathrm{m}}, u) += \alpha \left[c + \gamma \min_{u'} Q(g(\underline{\mathrm{m}}, u), u') - Q(\underline{\mathrm{m}}, u) \right].$$

- Here, c is the cost of performing u from whatever true state the asset actually is in, g is a generic state update function, and u' is the control that minimizes the cost in the next interval.

TQ-Learning Update

- Use tabular Q-learning with usual update rule:

$$Q(\underline{\mathrm{m}}, u) += \alpha \left[c + \gamma \min_{u'} Q(g(\underline{\mathrm{m}}, u), u') - Q(\underline{\mathrm{m}}, u) \right].$$

- Here, c is the cost of performing u from whatever true state the asset actually is in, g is a generic state update function, and u' is the control that minimizes the cost in the next interval.
- The other constants affect how learning progresses:

TQ-Learning Update

- Use tabular Q-learning with usual update rule:

$$Q(\underline{\mathrm{m}}, u) += \alpha \left[c + \gamma \min_{u'} Q(g(\underline{\mathrm{m}}, u), u') - Q(\underline{\mathrm{m}}, u) \right].$$

- Here, c is the cost of performing u from whatever true state the asset actually is in, g is a generic state update function, and u' is the control that minimizes the cost in the next interval.
- The other constants affect how learning progresses:
  - α: learning rate

TQ-Learning Update

- Use tabular Q-learning with usual update rule:

$$Q(\underline{\mathrm{m}}, u) += \alpha \left[c + \gamma \min_{u'} Q(g(\underline{\mathrm{m}}, u), u') - Q(\underline{\mathrm{m}}, u) \right].$$

- Here, c is the cost of performing u from whatever true state the asset actually is in, g is a generic state update function, and u' is the control that minimizes the cost in the next interval.
- The other constants affect how learning progresses:
  - α: learning rate
  - γ: discount factor

Replicated Q-learning (RQ-learning)

- Replicated Q-learning uses one vector $\mathbf{q}_u$ for each control $u \in \mathcal{U}$ and approximates the value of the Q-function for belief state $\mathbf{b}$ as $Q_{\mathbf{b}}(u) = \mathbf{q}_u \cdot \mathbf{b}$.
- Because our POMDP state contains both motion state M_k and time since last sync, $T_k^{(s)}$,
- The $\mathbf{q}_u$ vector's elements are updated by

$$q_u(x) = q_u(x) + \alpha b(x) \left[c + \gamma \min_{u'} Q_{\mathbf{b}'}(u') - q_u(x) \right] \forall x \in \mathcal{X}.$$

Replicated Q-learning (RQ-learning)

- Replicated Q-learning uses one vector $\mathbf{q}_u$ for each control $u \in \mathcal{U}$ and approximates the value of the Q-function for belief state $\mathbf{b}$ as $Q_{\mathbf{b}}(u) = \mathbf{q}_u \cdot \mathbf{b}$.
- Because our POMDP state contains both motion state M_k and time since last sync, $T_k^{(s)}$,
- The $\mathbf{q}_u$ vector's elements are updated by

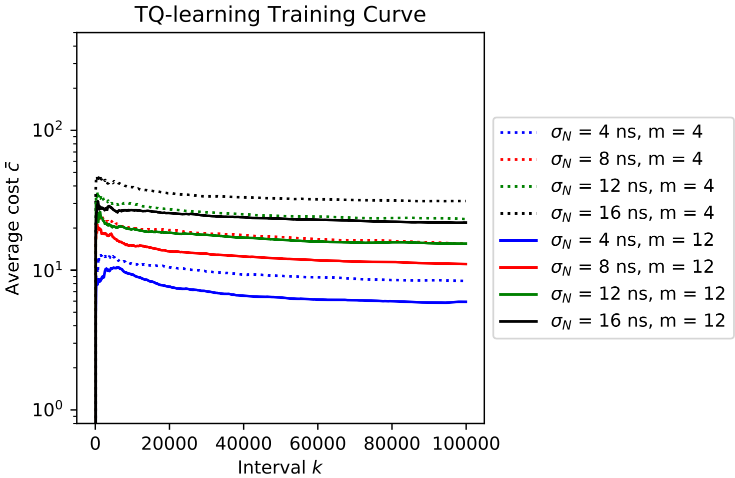
$$q_u(x) = q_u(x) + \alpha b(x) \left[c + \gamma \min_{u'} Q_{\mathbf{b}'}(u') - q_u(x) \right] \forall x \in \mathcal{X}.$$

- Exploit deterministic part of state to represent as $2T_{max}$ vectors of dimensions $|\mathcal{M}|$

Fixed-Rate Deterministic Policies (Model-Free)

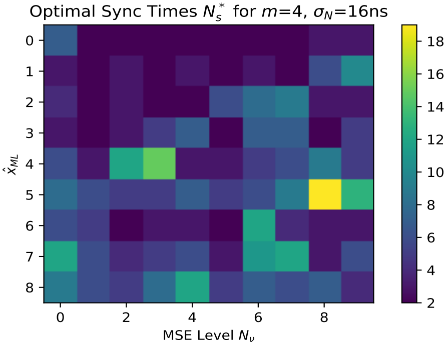
- Fixed-rate deterministic (FRD) is the standard approach used in most of the sensing literature that addresses timing synchronization
- FRD syncs every k intervals, where the value of k is optimzied to minimize the MSE

TQ-learning Training Curve:



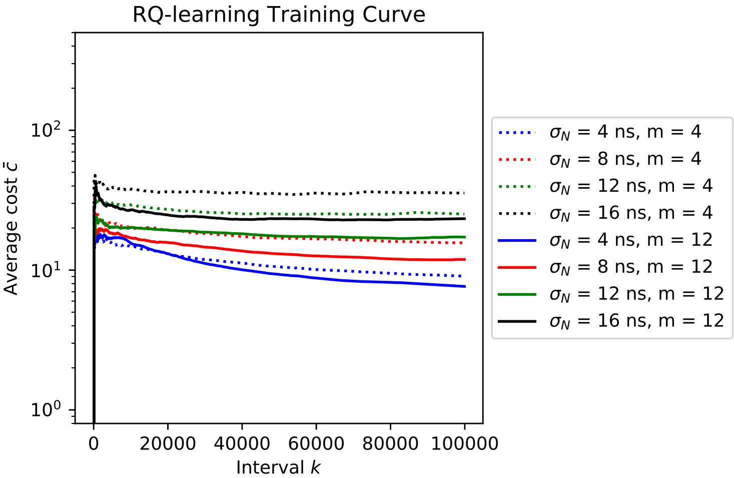
TQ-learning Optimal Sync Times

High Input ToF Variance



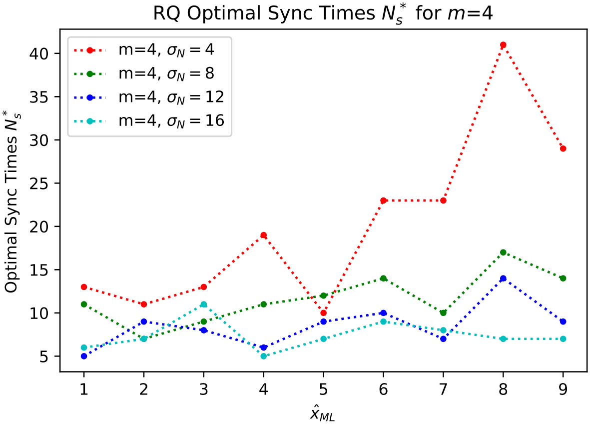
- Note that for the highest input ToF variance, the higher levels of the quantized MSE are reached.
- Also, note that generally the network can optimally wait longer periods of time to resync at the lowest MSE level.

RQ-learning Training Curve:

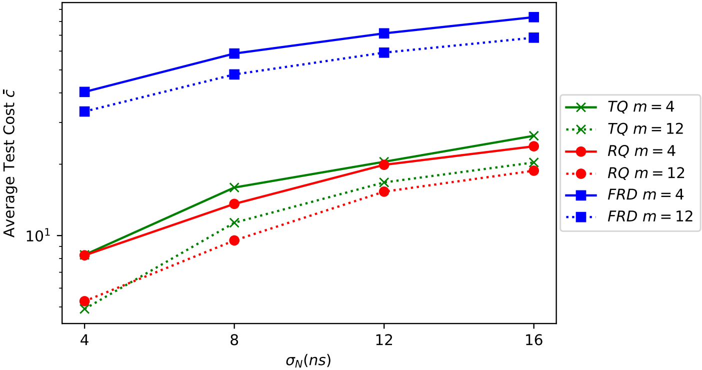


RQ-learning Approximate Sync Times (ML state)

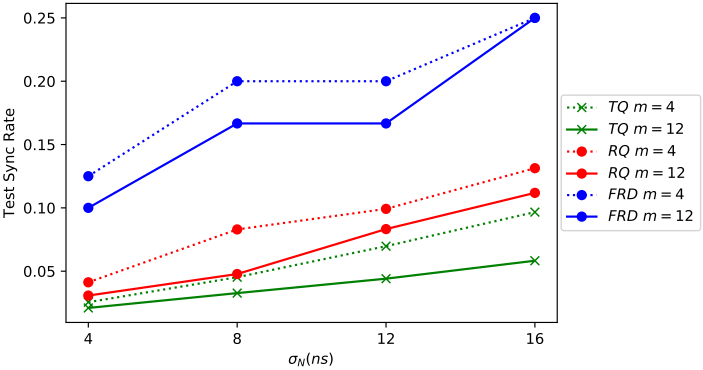
- RQ-learning uses the entire belief vector, so no simple visualization of optimal sync times
- Below we show the sync times if all the belief was concentrated on the ML state:



Performance: Localization MSE



Performance: Average Sync Rates



Conclusion

- Formulated problem of optimizing synchronization times for system of distributed sensors tracking a mobile asset as a POMDP
- Applied dimensionality reduction techniques to perform Q-learning on that POMDP:
  - TQ-learning replaces belief distribution with ML estimate for motion state, time since last sync, and (quantized) expected MSE
  - "old" RQ-learning uses one q vector for each control of size $|\mathcal{M}|$.
  - "new" RQ-learning uses one q vector for each control and stage number of size $|\mathcal{M}|$.
- Q-learning approaches outperform best FRD policies
- Deep Q-learning techniques are a good match for POMDP problems because they can accept the continuous belief state info $\rightarrow$ currently investigating

Conclusion

- Develop approaches for distributed decision making in sensor networks (partially observable MAMDPs)
- Want to deploy and test these ideas using our AFOSR DURIP-funded testbed
- Work towards general framework for joint optimization of stochastic controls and networks