POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,
- $\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$
- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$
- $\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$
- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$
- $\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$
- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,
- $\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$
- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$
- $\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$
- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$
- $\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$
- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

MDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t|s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t|s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

- or a function of the belief: $b : S \rightarrow (0, 1)$, $\pi(b)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

- or a function of the **belief**: $b : S \rightarrow \langle 0, 1 \rangle$, $\pi(b)$

POMDP

Definition (Partially Observed Markov Decision Processes POMDP)

Partially Observed Markov Decision Processes is defined by:

- Finite set of states S, $S_i = S$ for any time $t \in \mathbb{N}_0$,

$\rightarrow$ Initial **belief** $b_0(s) = P(S_0)$

- The set of possible actions (decisions) at any time $A = \{a_1, \ldots, a_{|A|}\}$

$\rightarrow$ a set of observations $Z = O = \{z_1, \ldots, z_{|Z|}\}$

- Transition matrix $T(s_{t-1}, a_{t-1}, s_t) = P(s_t | s_{t-1}, a_{t-1})$

$\rightarrow$ observation matrix $O(s_t, a_{t-1}, z_t) = P(z_t | s_t, a_{t-1})$

- Reward(=utility) $R(s, a)$ for each state (and action).
- (discount factor $\gamma \in < 0, 1 >$).

We maximize the expected cumulative reward $max_\pi \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\right]$.

MDP The policy is a function of the state $\pi(s)$

OMDP The policy is a function of the history $\pi(a_{t-1}, z_{t-1}, \ldots, z_1, a_0, b_0)$

- or a function of the **belief**: $b : S \rightarrow \langle 0, 1 \rangle$, $\pi(b)$

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = (0.5, 0.5)$
- The reward R is a function of the state and the action
  - $U(gold, Va) = 10$, $U(tiger, Va) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, Ve) = 10$, $U(tiger, Ve) = -100$, $U(*, listen) = -1$, that is

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S) = (0.5, 0.5)$
- The reward R is a function of the state and the action
  - $U(gold, Ja) = 10$, $U(tiger, Ja) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S) = (0.5, 0.5)$
- The reward R is a function of the state and the action
  - $U(gold, Ø) = 10$, $U(tiger, Ø) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S) = (0.5, 0.5)$
- The reward R is a function of the state and the action
  - $R(gold, A) = 10$, $R(tiger, A) = -100$, $R(*, listen) = -1$, else 0.

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S) = (0.5, 0.5)$
- The reward R is a function of the state and the action
  - $U(gold, A) = 10$, $U(tiger, A) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, A) = 10$, $U(tiger, A) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, Ø) = 10$, $U(tiger, Ø) = -100$, $U(*, listen) = -1$, else 0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, l/r) = 10$, $U(tiger, l/r) = -100$, $U(*, listen) = -1$, that is

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, l/r) = 10$, $U(tiger, l/r) = -100$, $U(*, listen) = -1$, that is

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, l/r) = 10$, $U(tiger, l/r) = -100$, $U(*, listen) = -1$, that is

Tiger / Action	left	right
Listen	-1	-1
left	-100	10
right	10	-100

Z | S=?, A=listen	left	right
TL	0.85	0.15
TR	0.15	0.85
NoInfo	0	0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, l/r) = 10$, $U(tiger, l/r) = -100$, $U(*, listen) = -1$, that is

Tiger / Action	left	right
Listen	-1	-1
left	-100	10
right	10	-100

Z | S=?, A=listen	left	right
TL	0.85	0.15
TR	0.15	0.85
NoInfo	0	0

Tiger Example

The example is a variant of the Monty Hall problem.

- We face two doors.
  - There is a tiger behind one door,
  - there is a gold brick behind the other.
- The Tiger is left or right $S = \{left, right\}$
- We may open any door or listen $A = \{left, right, listen\}$,
- we search optimal policy for given observation and reward tables.
- We observe Z only if we **listen** - we listen the tiger left TL or right TR
- we reset the world at the beginning and after opening any door:
  - the initial belief $P(S_0) = \langle 0.5, 0.5 \rangle$
- The reward R is a function of the state and the action
  - $U(gold, l/r) = 10$, $U(tiger, l/r) = -100$, $U(*, listen) = -1$, that is

$\frac{Tiger}{Action}$	left	right
Listen	-1	-1
left	-100	10
right	10	-100

Z | S=?, A=listen	left	right
TL	0.85	0.15
TR	0.15	0.85
NoInfo	0	0

POMPD

Finite horizon POMPD t, $\gamma = 1$:

- $t = 1$
  $EU_{t=1}(A = left/right) = \frac{-100+10}{2} = -45$
  $EU_{t=1}(A = listen) = -1$
- horizon $t = 2$

$$T(s_{t-1}, a_{t-1}, s_t) = P(s_t|s_{t-1}, a_{t-1})$$
$$O(s_t, a_{t-1}, z_t) = P(z_t|s_t, a_{t-1})$$

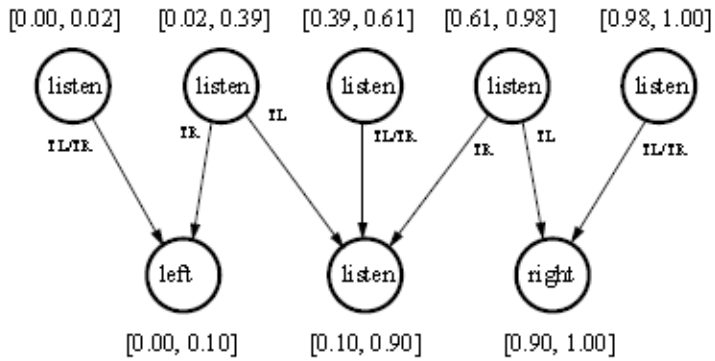
- and $t = 4$

POMPD

Finite horizon POMPD t, $\gamma = 1$:

- $t = 1$
  $EU_{t=1}(A = left/right) = \frac{-100+10}{2} = -45$
  $EU_{t=1}(A = listen) = -1$
- horizon $t = 2$

$$T(s_{t-1}, a_{t-1}, s_t) = P(s_t|s_{t-1}, a_{t-1})$$
$$O(s_t, a_{t-1}, z_t) = P(z_t|s_t, a_{t-1})$$



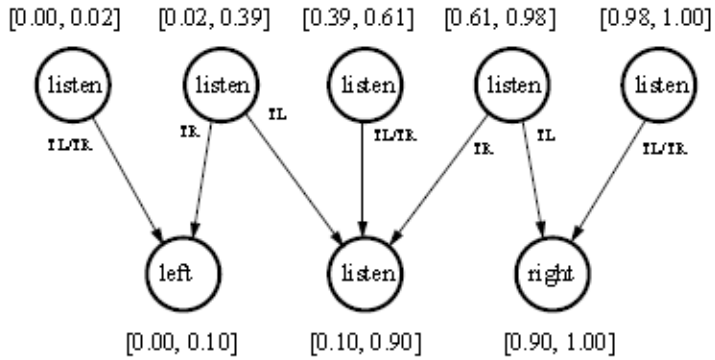
- and $t = 4$

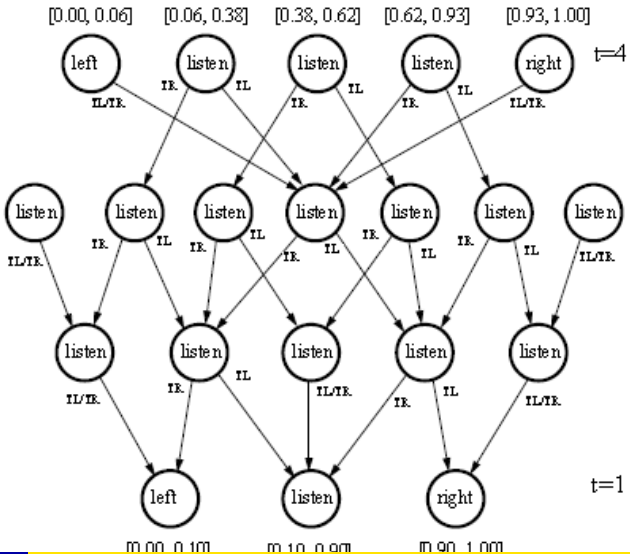
POMPD

Finite horizon POMPD t, $\gamma = 1$:

- $t = 1$
  $EU_{t=1}(A = left/right) = \frac{-100+10}{2} = -45$
  $EU_{t=1}(A = listen) = -1$
- horizon $t = 2$

$$T(s_{t-1}, a_{t-1}, s_t) = P(s_t|s_{t-1}, a_{t-1})$$
$$O(s_t, a_{t-1}, z_t) = P(z_t|s_t, a_{t-1})$$



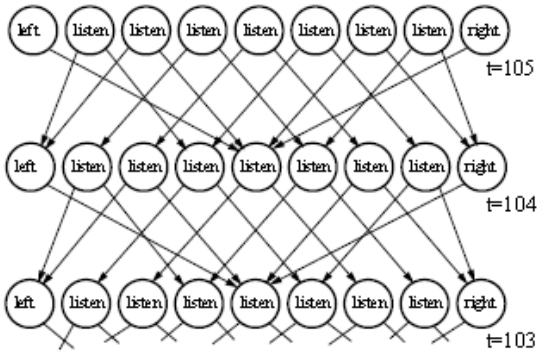


Infinite Horizon

- $\gamma = 0.75$
- we iterate until convergence
- Then, we create a graph by joining two successive time slices together.
- We may omit nodes that are not reachable from the initial belief $b_0(s) = 0.5$.

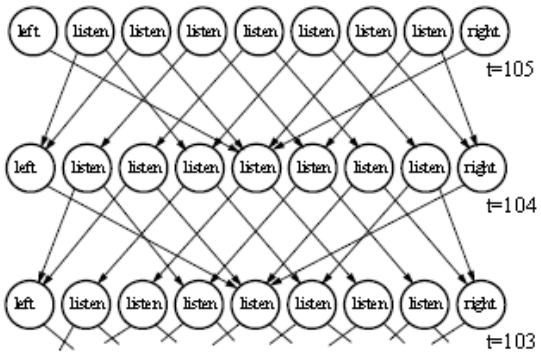
Infinite Horizon

- $\gamma = 0.75$
- we iterate until convergence
- Then, we create a graph by joining two successive time slices together.
- We may omit nodes that are not reachable from the initial belief $b_0(s) = 0.5$.



Infinite Horizon

- $\gamma = 0.75$
- we iterate until convergence
- Then, we create a graph by joining two successive time slices together.
- We may omit nodes that are not reachable from the initial belief $b_0(s) = 0.5$.



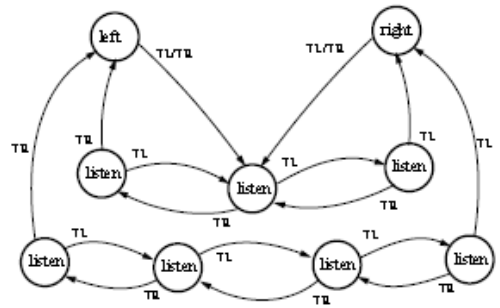
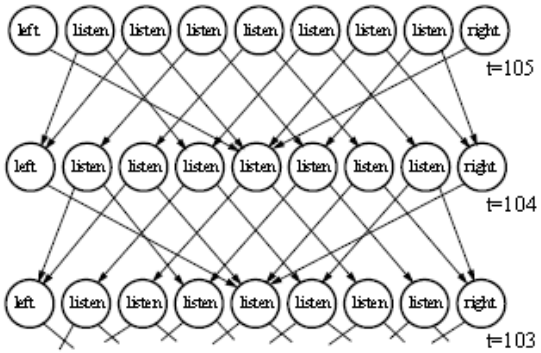


Figure 16: Policy graph for tiger example

Infinite Horizon

- $\gamma = 0.75$
- we iterate until convergence
- Then, we create a graph by joining two successive time slices together.
- We may omit nodes that are not reachable from the initial belief $b_0(s) = 0.5$.



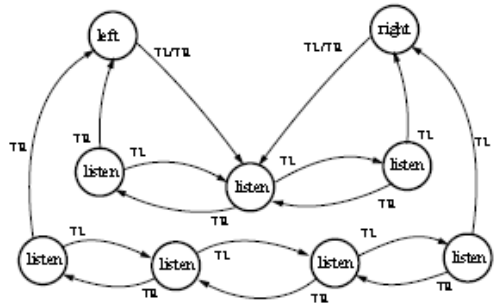


Figure 16: Policy graph for tiger example

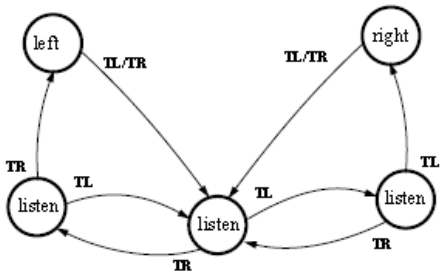


Figure 17: Trimmed policy graph for tiger example

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(left)$, since the other probability is $1 - b(left)$.
- We update belief after any iteration. The update consists of:
  - a transition – we eliminate unobserved s_{t-1}
  - an observation – we condition by z_t
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - belief $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(left)$, since the other probability is $1 - b(left)$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(left)$, since the other probability is $1 - b(left)$.
- We update belief after any iteration. The update consists of:
  - a transition – we eliminate unobserved s_{t-1}
  - an observation – we condition by z_t
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(left)$, since the other probability is $1 - b(left)$.
- We update belief after any iteration. The update consists of:
  - a transition – we eliminate unobserved s_{t-1}
  - an observation – we condition by z_t
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t.
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t.
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t.
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s') \\ &= \frac{\sum_s O(s', a_{t-1}, z_t) T(s, a_{t-1}, s') b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t.
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s^{|}) \\ &= \frac{\Sigma_s O(s^{|}, a_{t-1}, z_t) T(s, a_{t-1}, s^{|}) b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Markov with respect to *belief* over states

- The history is aggregated in the probability distribution over states
  - history $h_t = \{a_0, z_1, a_1, \ldots, z_{t-1}, a_{t-1}, z_t\}$
  - **belief** $b_t(s) = P(S = s | z_t, a_{t-1}, \ldots, a_0, b_0)$,
  - initial belief $b_0(s) = P(S_0)$.
  - In the tiger example a single number $b(\mathit{left})$, since the other probability is $1 - b(\mathit{left})$.
- We update belief after any iteration. The update consists of:
  - a transition - we eliminate unobserved s_{t-1}
  - an observation - we condition by z_t.
- belief update

$$\begin{aligned} \tau(b_{t-1}, a_{t-1}, z_t) &= b_t(s^{|}) \\ &= \frac{\Sigma_s O(s^{|}, a_{t-1}, z_t) T(s, a_{t-1}, s^{|}) b_{t-1}(s)}{P(z_t | b_{t-1}, a_{t-1})} \end{aligned}$$

- Markov with respect to b since τ does not depend on time.

Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \rightarrow a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

- **value function**
  - Initial $V_0(b) = \max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = \max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.

Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \rightarrow a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

- value function
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.

Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \rightarrow a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

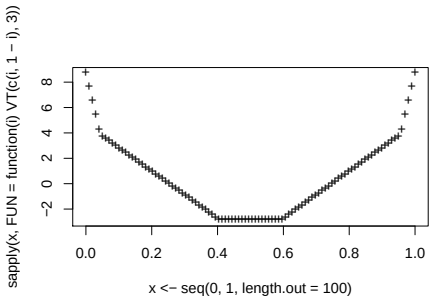
- **value function**
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$,
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.

Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \to a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

- **value function**
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$,
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.

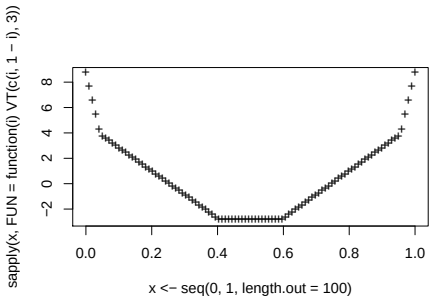


Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \to a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

- **value function**
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$,
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.

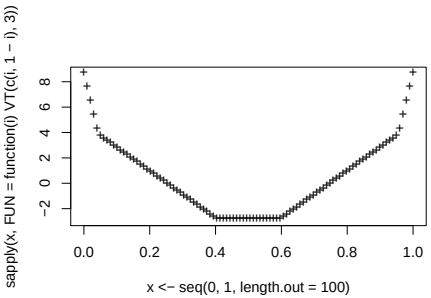


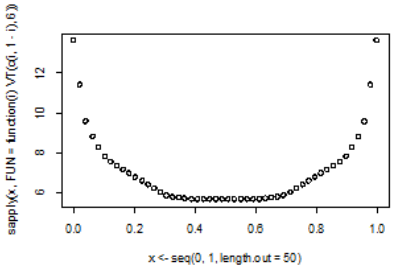
Strategy, Value function

- **Strategy (policy)** is a function $\pi(b) \to a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

- **value function**
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$,
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.



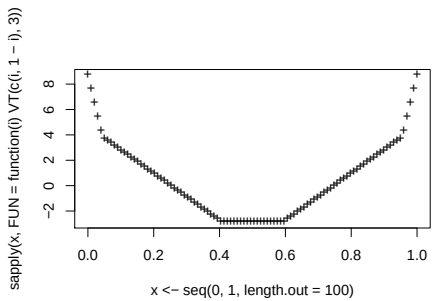


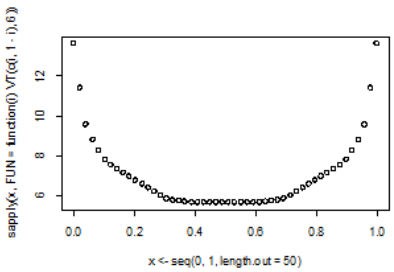
Strategy, Value function

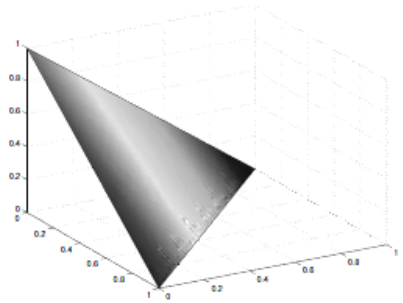
- **Strategy (policy)** is a function $\pi(b) \to a$,
- optimal strategy maximizes the expected discounted cumulative reward

$$\pi^*(b_0) = argmax_\pi \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} (\gamma^t \cdot r_t) | b_0 \right]$$

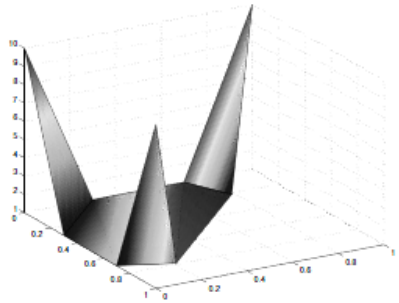
- **value function**
  - initial $V_0(b) = max_a \sum_{s \in S} R(s,a) b(s)$
  - recursively
    $V_t(b) = max_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$,
- optimal strategy for the horizon t:
  $\pi_t^*(b) = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \gamma \sum_{z \in Z} P(z|a,b) V_{t-1}(\tau(b,a,z)) \right]$.





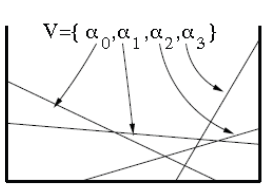


(a) The belief space



(b) The value function

α vectors



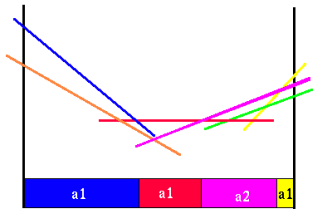


Figure 1: POMDP value function representation

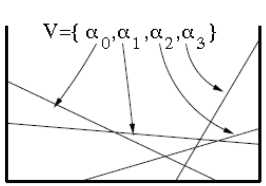
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From

α vectors



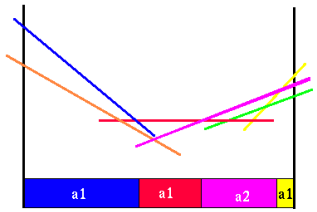


Figure 1: POMDP value function representation

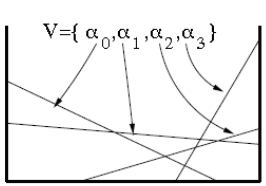
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From

α vectors



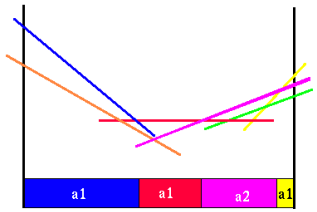


Figure 1: POMDP value function representation

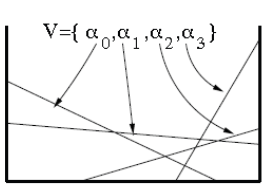
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(b^{a,z})\right]$
  - $P(z|a, b) = \sum_{s' \in S} P(z|a, s') \sum_{s \in S} P(s'|s, a) b(s)$

α vectors



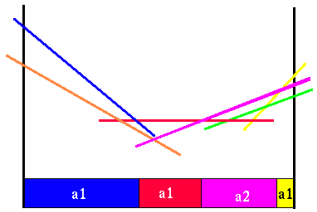


Figure 1: POMDP value function representation

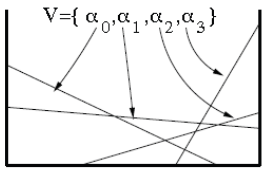
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(\tau(b, a, z))\right]$;
  - $\tau(b, a, z) = \frac{P(z|a, b) b(s')}{P(z|a, b)}$

α vectors



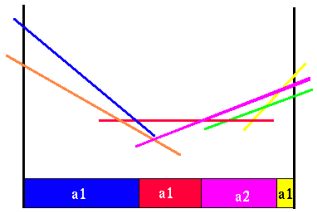


Figure 1: POMDP value function representation

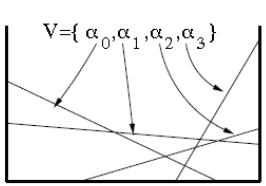
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(\tau(b, a, z))\right]$:
  - $\tau(b_t, a_t, z_{t+1}) = \frac{\Sigma_s O(s^|, a_t, z_{t+1}) T(s, a_t, s^|) b_t(s)}{Pr(z_{t+1}|b_t, a_t)}$

α vectors



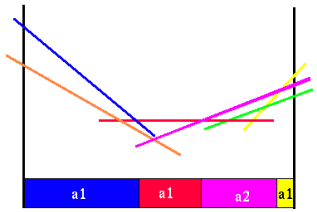


Figure 1: POMDP value function representation

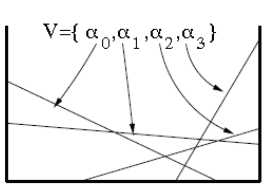
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(\tau(b, a, z))\right]$:
  - $\tau(b_t, a_t, z_{t+1}) = \frac{\Sigma_s O(s^|, a_t, z_{t+1}) T(s, a_t, s^|) b_t(s)}{Pr(z_{t+1}|b_t, a_t)}$

α vectors



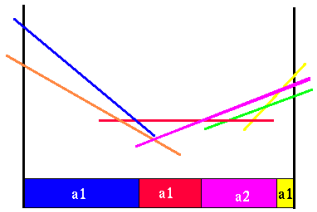


Figure 1: POMDP value function representation

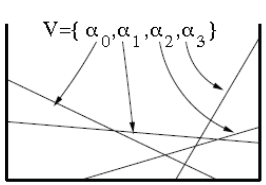
$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(\tau(b, a, z))\right]$:
  - $\tau(b_t, a_t, z_{t+1}) = \frac{\Sigma_s O(s^|, a_t, z_{t+1}) T(s, a_t, s^|) b_t(s)}{Pr(z_{t+1}|b_t, a_t)}$

α vectors



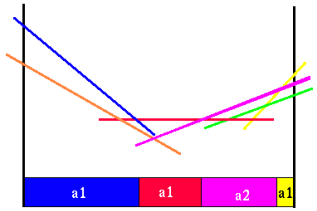


Figure 1: POMDP value function representation

$$|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$$

- *value* function $V_t(b)$ can be represented by a finite number of hyperplanes
  - each hyperplane is represented as a vector α:

$$V_t(b) \Leftrightarrow \Gamma_t = \{a; \alpha_0, \alpha_1, \ldots, \alpha_m\} = \{a; v_0, v_1, \ldots, v_m\}.$$

  - initial: $\Gamma_0(b) = \{\langle a; R(s_1, a), R(s_2, a), \ldots, R(s_{|S|}, a)\rangle\}_{a \in A}$
  - at the time t: $V_t(b) = max_{\alpha \in \Gamma_t} \sum_{s \in S} \alpha(s) b(s)$.
- From
  - $V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} P(z|a, b) V_{t-1}(\tau(b, a, z))\right]$:
  - $\tau(b_t, a_t, z_{t+1}) = \frac{\Sigma_s O(s^|, a_t, z_{t+1}) T(s, a_t, s^|) b_t(s)}{Pr(z_{t+1}|b_t, a_t)}$

$$V_t(b) = max_a \left[\sum_{s \in S} R(s, a) b(s) + \gamma \sum_{z \in Z} max_{\alpha \in \Gamma_{t-1}} \sum_{s' \in S} \sum_{s \in S} T(s, a, s') O(s', a, z) \alpha(s') b(s)\right]$$

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - There are strategies to remove them earlier
  - or to avoid to generate many of them at all [Cassandra et al., 1997]

https://h2r.github.io/pomdp-py/html/index.html

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - there are strategies to remove them earlier
  - or to avoid to generate many of them at all [Cassandra, 1998]

https://h2r.github.io/pomdp-py/html/index.html

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - there are strategies to remove them earlier
  - or to avoid to generate many of them at all [Cassandra, Littman]

https://h2r.github.io/pomdp-py/html/index.html

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - there are strategies to remove them earlier
  - or to avoid to generate many of them at all $|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$.

https://h2r.github.io/pomdp-py/html/index.html

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - there are strategies to remove them earlier
  - or to avoid to generate many of them at all $|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$.

https://h2r.github.io/pomdp-py/html/index.html

One Step of the Time Update

- temporal sets $\forall \alpha_i \in \Gamma_{t-1}$ ($j \in S$ is the α dimension):

$$\begin{aligned} \Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(j) = R(j,a) \\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(j) = \gamma \sum_{s' \in S} T(j,a,s') O(s',a,z) \alpha(s'), \end{aligned}$$

- The utility for the action a summed over possible observation results z_k:

$$\Gamma_t^a = \Gamma_t^{a,+} + \Gamma_t^{a,z_1} \oplus \Gamma_t^{a,z_2} \oplus \ldots \oplus \Gamma_t^{a,z_m}$$

- the new value function for the time t: $\Gamma_t \leftarrow \bigcup_{a \in A} \Gamma_t^a$.
- We remove all α that are dominated by others
  - there are strategies to remove them earlier
  - or to avoid to generate many of them at all $|\Gamma_t| = O(|A| \cdot |\Gamma_{t-1}|^{|Z|})$.

https://h2r.github.io/pomdp-py/html/index.html

POMDP!

procedure POLICY_POMDP(T)
  $\Gamma = \{[None; 0, \ldots, 0]\}$
  for $t = 1$ to T **do**
    $\Gamma' = \emptyset$
    for all $(a', v_1^k, \ldots, v_N^k) \in \Gamma$, all $a \in A$, all $z \in O$ **do**
      for $j = 1$ to N **do** ▷ for all states s_j
        $v_{u,z,j}^k = \gamma \sum_{i=1}^{N} v_i^k p(z|s_i) p(s_i|a, s_j)$
      end for
    end for ▷ next: free choice $k(z) \in \Gamma$ for every z
    for all u, all $k(1), \ldots, k(M) = (1, \ldots, 1)$ to $(|\Gamma|, \ldots, |\Gamma|)$ **do**
      for $j = 1$ to N **do**
        $v_j' = \left[r(s_j, a) + \sum_z v_{a,z,j}^{k(z)} \right]$ ▷ aggregate over $z \in O$
      end for
      add $(a; v_1', \ldots, v_N')$ to Γ' ▷ one Γ_t^a element
    end for
    (optimally prune Γ')
    $\Gamma = \Gamma'$
  end for
  return Γ
end procedure

POMPD policy!

- From the set of α vectors Γ we get the policy:

policy

1: **procedure** POLICY_POMDP($\Gamma, b = [b_1, \ldots, b_N]$)
2: $\quad \hat{a} = \arg\max_{(a; v_1^k, \ldots, v_N^k) \in \Gamma} \sum_{i=1}^{N} v_i^k b_i$
3: $\quad$ return $\hat{a}$
4: **end procedure**

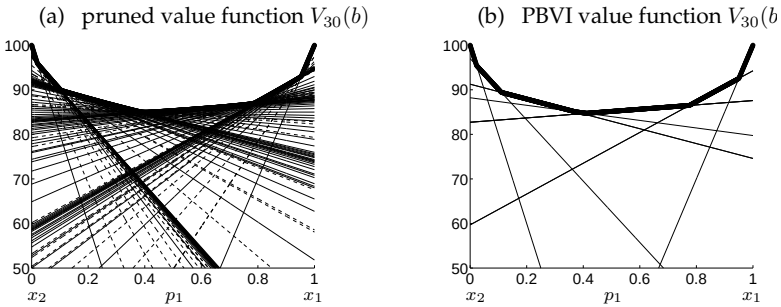


Figure 15.7 The benefit of point-based value iteration over general value iteration: Shown in (a) is the exact value function at horizon $T = 30$ for a different example, which consists of 120 constraints, after pruning. On the right is the result of the PBVI algorithm retaining only 11 linear functions. Both functions yield virtually indistinguishable results when applied to control.

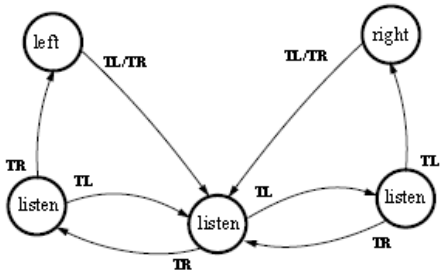
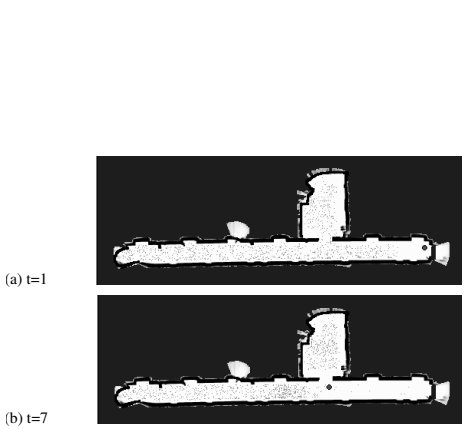
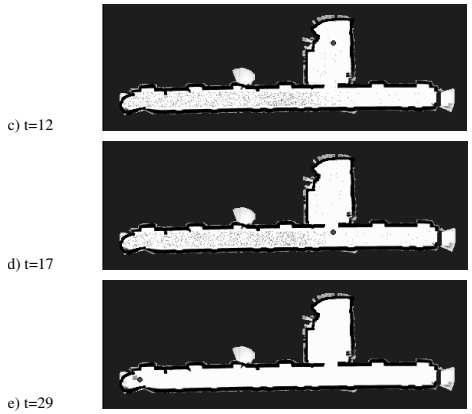


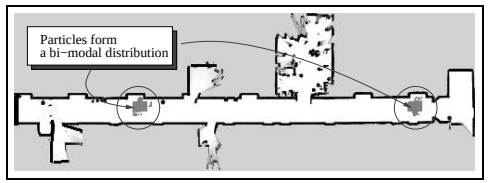
Figure 17: Trimmed policy graph for tiger example

Approximation - We evaluate only some b points

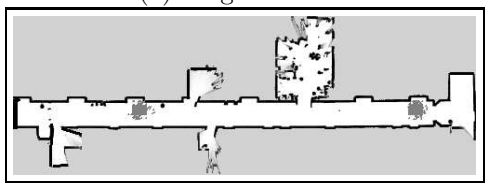
- $\times$ for the rest of today lecture
- Pineau & all.: Anytime Point-Based Approximations for Large POMDPs, JAIR 2006
- Pearl the Nursebot
- Find a person







(a) Original Belief



(b) Reconstruction

Approximation - We evaluate only some b points

- We evaluate the *belief* only in a finite number of points
- only one α vector for each point

$$\begin{aligned}\Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(s) = R(s,a)\\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha(s'),\end{aligned}$$

- max for FINITE number of $b \in B$

$$\begin{aligned}\alpha_b &= argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} argmax_{\alpha \in \Gamma_t^{a,z}} \sum_{s \in S} \alpha(s) b(s) \right]\\ \Gamma_t &= \bigcup_{b \in B} \{\alpha_b\}\end{aligned}$$

- The number of αs does not increase (with respect to the size of B).

Approximation - We evaluate only some b points

- We evaluate the *belief* only in a finite number of points
- only one α vector for each point

$$\begin{aligned}\Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(s) = R(s,a)\\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha(s'),\end{aligned}$$

- max for FINITE number of $b \in B$

$$\alpha_b = argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} argmax_{\alpha \in \Gamma_t^{a,z}} \sum_{s \in S} \alpha(s) b(s) \right]$$

$$\Gamma_t = \bigcup_{b \in B} \{\alpha_b\}$$

- The number of αs does not increase (with respect to the size of B).

Approximation - We evaluate only some b points

- We evaluate the *belief* only in a finite number of points
- only one α vector for each point

$$\begin{aligned}\Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(s) = R(s,a)\\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha(s'),\end{aligned}$$

- max for FINITE number of $b \in B$

$$\begin{aligned}\alpha_b &= argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} argmax_{\alpha \in \Gamma_t^{a,z}} \sum_{s \in S} \alpha(s) b(s) \right]\\ \Gamma_t &= \bigcup_{b \in B} \{\alpha_b\}\end{aligned}$$

- The number of αs does not increase (with respect to the size of B).

Approximation - We evaluate only some b points

- We evaluate the *belief* only in a finite number of points
- only one α vector for each point

$$\begin{aligned}\Gamma_t^{a,+} &\leftarrow \alpha^{a,+}(s) = R(s,a)\\ \Gamma_t^{a,z} &\leftarrow \alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s') O(s',a,z) \alpha(s'),\end{aligned}$$

- max for FINITE number of $b \in B$

$$\begin{aligned}\alpha_b &= argmax_a \left[\sum_{s \in S} R(s,a) b(s) + \sum_{z \in Z} argmax_{\alpha \in \Gamma_t^{a,z}} \sum_{s \in S} \alpha(s) b(s) \right]\\ \Gamma_t &= \bigcup_{b \in B} \{\alpha_b\}\end{aligned}$$

- The number of αs does not increase (with respect to the size of B).

POMDP Evaluation for the Fixed Number of B Points

procedure BACKUP(B,Γ_{t-1})
  for each action $a \in A$ **do**
    for each observation $z \in Z$ **do**
      for each solution vector $\alpha_i \in \Gamma_{t-1}$ **do**
        $\alpha^{a,z}(s) = \gamma \sum_{s' \in S} T(s,a,s')O(s',a,z)\alpha(s'), \forall s \in S$
      end for
      $\Gamma_t^{a,z} = \cup_i \alpha^{a,z}(s)$
    end for
  end for
  $\Gamma_t = \emptyset$
  for each belief point $b \in B$ **do**

$$\alpha_b = argmax_a \left[\sum_{s \in S} R(s,a)b(s) + \sum_{z \in Z} argmax_{\alpha \in \Gamma_t^{a,z}} \sum_{s \in S} \alpha(s)b(s) \right]$$

    if $\alpha_b \notin \Gamma_t$ **then**
      $\Gamma_t = \Gamma_t \cup \alpha_b$
    end if
  end for
  return Γ_t
end procedure

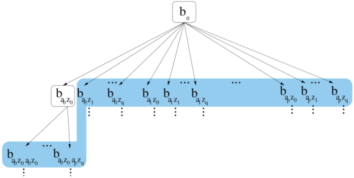
Iterative Number of Points POMDP

1: **procedure** PBVI-MAIN(B_{Init},Γ_0,N,T)
2: $B = B_{Init}$
3: $\Gamma = \Gamma_{Init}$
4: **for** N expansions **do**
5: **for** T iterations **do**
6: $\Gamma = BACKUP(B, \Gamma)$
7: **end for**
8: $B_{new} = EXPAND(B, \Gamma)$
9: **end for**
10: return Γ
11: **end procedure**

T either a horizon or we select a error bound $\gamma^t ||V_0^* - V^*||$.

Expand: New Points Selection

1) at random



2) greedy maximal error improvement

- b' a new candidate
- the upper error bound in b'

$$\epsilon(b') \leq \min_{b \in B} \sum_{s \in S} \begin{cases} \left(\frac{R_{max}}{1-\gamma} - \alpha(s)\right)(b'(s) - b(s)) & b'(s) \geq b(s) \\ \left(\frac{R_{min}}{1-\gamma} - \alpha(s)\right)(b'(s) - b(s)) & b'(s) < b(s) \end{cases}$$

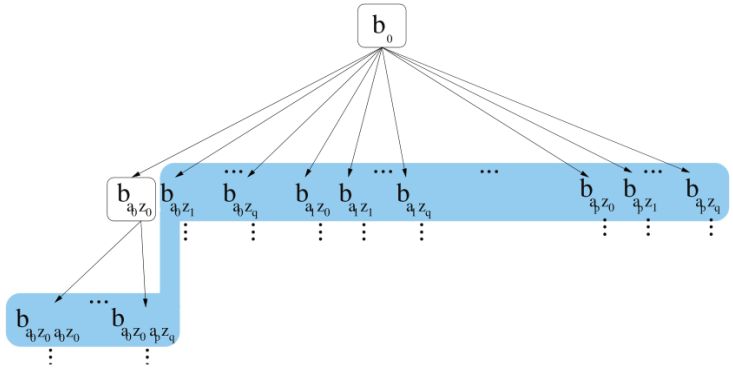
- b on the fringe, the error weighted by the probability of observations

$$\epsilon(b) = \max_{a \in A} \sum_{z \in Z} O(b,a,z)\epsilon(\tau(b,a,z))$$

$$= \max_{a \in A} \sum_{z \in Z} \left[\sum_{s' \in S} \sum_{s \in S} T(s,a,s')O(s',a,z)b(s) \right] \epsilon(\tau(b,a,z))$$

Expand: New Points Selection

1) at random



2) greedy maximal error improvement
   - b' a new candidate
   - the upper error bound in b'

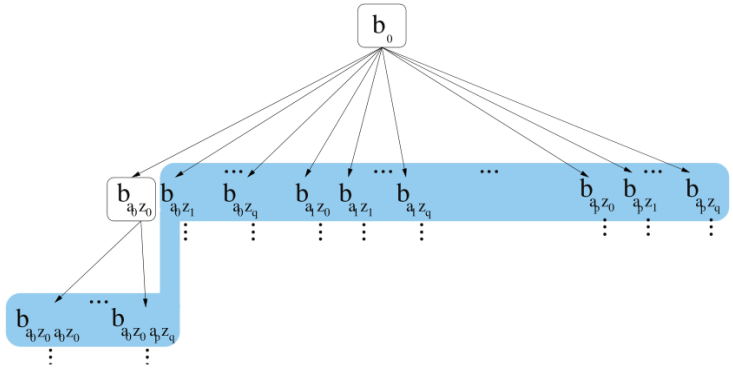
$$\epsilon(b') \leq \min_{b \in B} \sum_{s \in S} \begin{cases} (\frac{R_{max}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) \geq b(s) \\ (\frac{R_{min}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) < b(s) \end{cases}$$

   - b on the fringe, the error weighted by the probability of observations:

$$\begin{aligned} \epsilon(b) &= \max_{a \in A} \sum_{z \in Z} O(b, a, z) \epsilon(\tau(b, a, z)) \\ &= \max_{a \in A} \sum_{z \in Z} \left[\sum_{s' \in S} \sum_{s \in S} T(s, a, s') O(s', a, z) b(s) \right] \epsilon(\tau(b, a, z)). \end{aligned}$$

Expand: New Points Selection

1) at random



2) greedy maximal error improvement
   - b' a new candidate
   - the upper error bound in b'

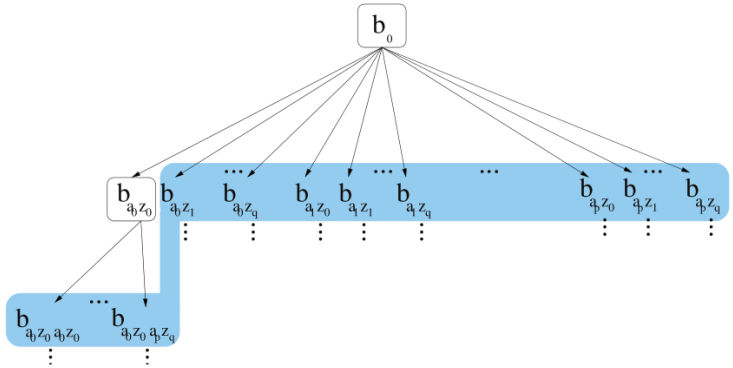
$$\epsilon(b') \leq \min_{b \in B} \sum_{s \in S} \begin{cases} (\frac{R_{max}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) \geq b(s) \\ (\frac{R_{min}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) < b(s) \end{cases}$$

   - b on the fringe, the error weighted by the probability of observations:

$$\begin{aligned} \epsilon(b) &= \max_{a \in A} \sum_{z \in Z} O(b,a,z)\epsilon(\tau(b,a,z)) \\ &= \max_{a \in A} \sum_{z \in Z} \left[\sum_{s' \in S} \sum_{s \in S} T(s,a,s')O(s',a,z)b(s) \right] \epsilon(\tau(b,a,z)). \end{aligned}$$

Expand: New Points Selection

1) at random



2) greedy maximal error improvement
   - b' a new candidate
   - the upper error bound in b'

$$\epsilon(b') \leq \min_{b \in B} \sum_{s \in S} \begin{cases} (\frac{R_{max}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) \geq b(s) \\ (\frac{R_{min}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) < b(s) \end{cases}$$

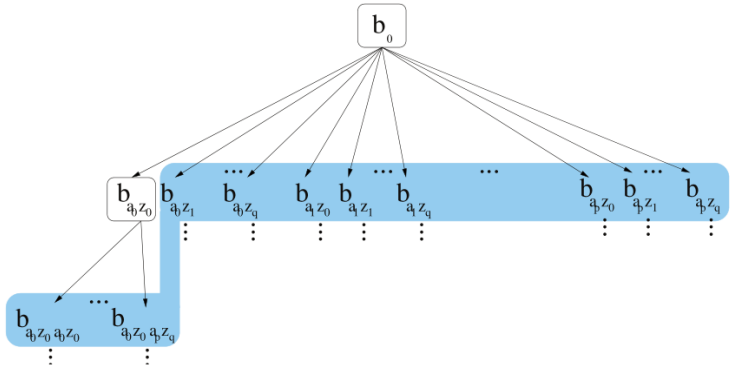
   - b on the fringe, the error weighted by the probability of observations:

$$\epsilon(b) = \max_{a \in A} \sum_{z \in Z} O(b,a,z)\epsilon(\tau(b,a,z))$$

$$= \max_{a \in A} \sum_{z \in Z} \left[\sum_{s' \in S} \sum_{s \in S} T(s,a,s')O(s',a,z)b(s) \right] \epsilon(\tau(b,a,z)).$$

Expand: New Points Selection

1) at random



2) greedy maximal error improvement
   - b' a new candidate
   - the upper error bound in b'

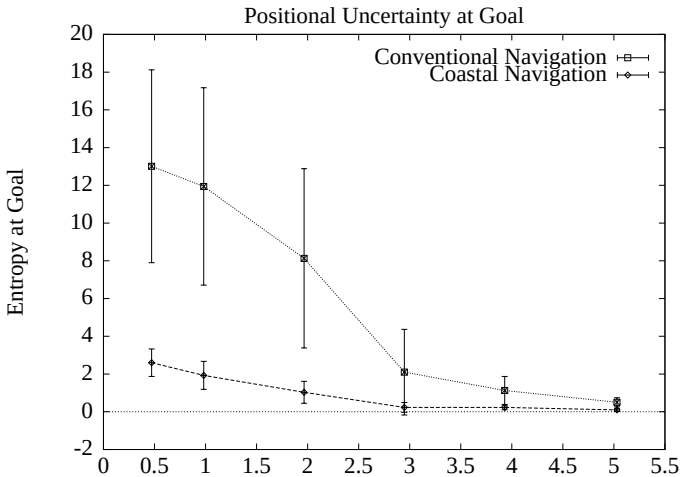
$$\epsilon(b') \leq \min_{b \in B} \sum_{s \in S} \begin{cases} (\frac{R_{max}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) \geq b(s) \\ (\frac{R_{min}}{1-\gamma} - \alpha(s))(b'(s) - b(s)) & b'(s) < b(s) \end{cases}$$

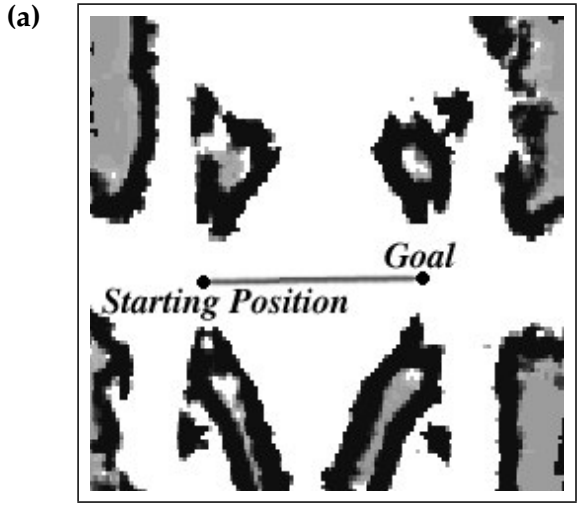
   - b on the fringe, the error weighted by the probability of observations:

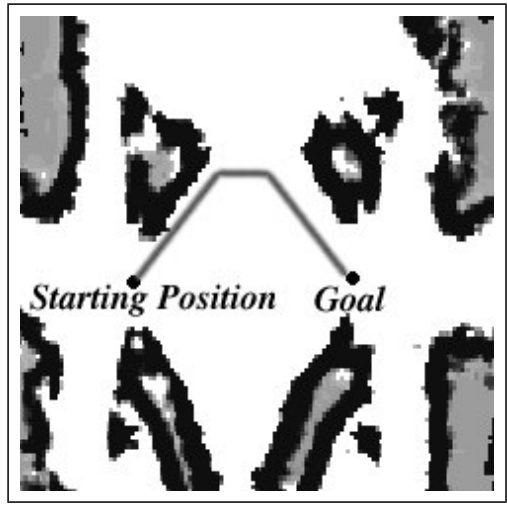
$$\begin{aligned} \epsilon(b) &= \max_{a \in A} \sum_{z \in Z} O(b,a,z)\epsilon(\tau(b,a,z)) \\ &= \max_{a \in A} \sum_{z \in Z} \left[\sum_{s' \in S} \sum_{s \in S} T(s,a,s')O(s',a,z)b(s) \right] \epsilon(\tau(b,a,z)). \end{aligned}$$

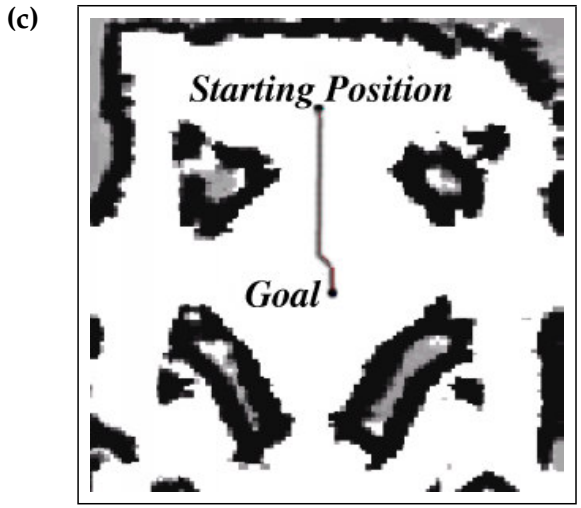
Augmented MDP Application

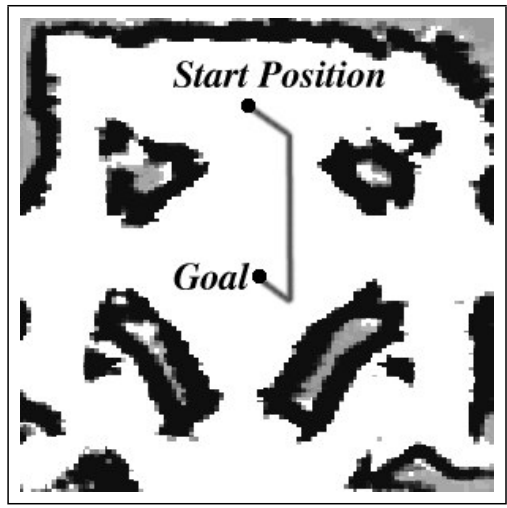
- a) and c) plans that ignore robot's perceptual uncertainty.
- b) and d) was computed by AMDP algorithm. It avoids regions where the robot is more likely to get lost.
- Useful for sensors with maximal range below 4m.





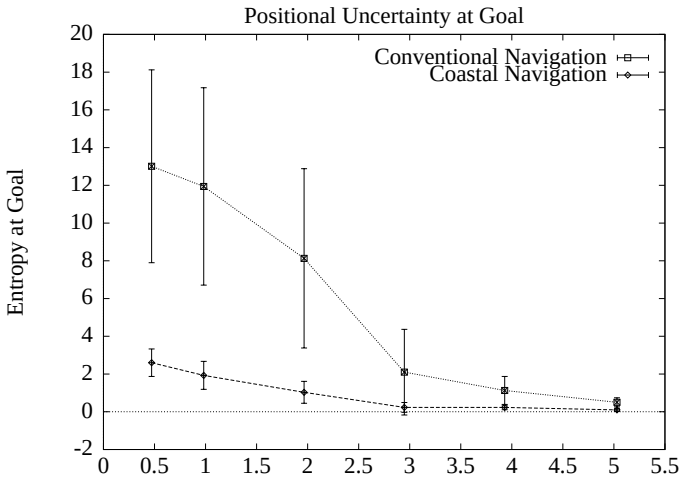


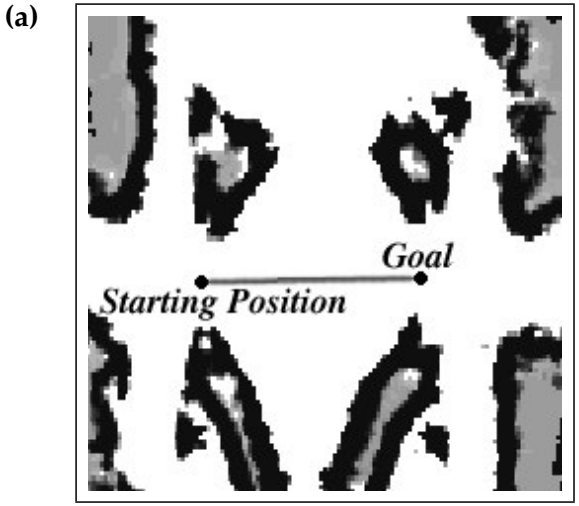


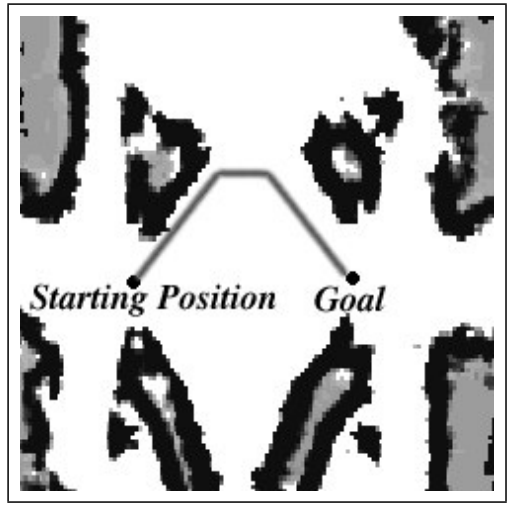


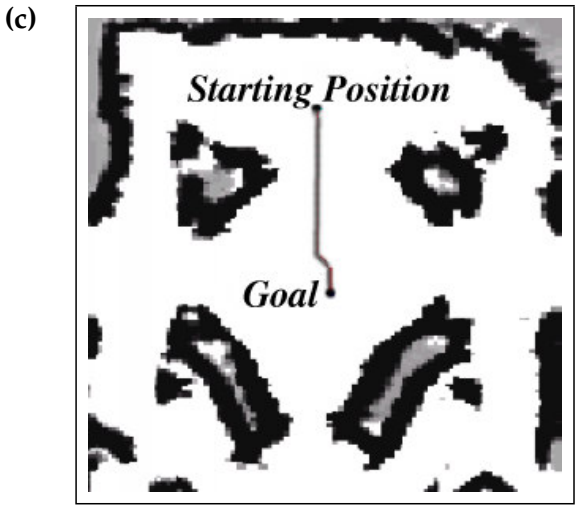
Augmented MDP Application

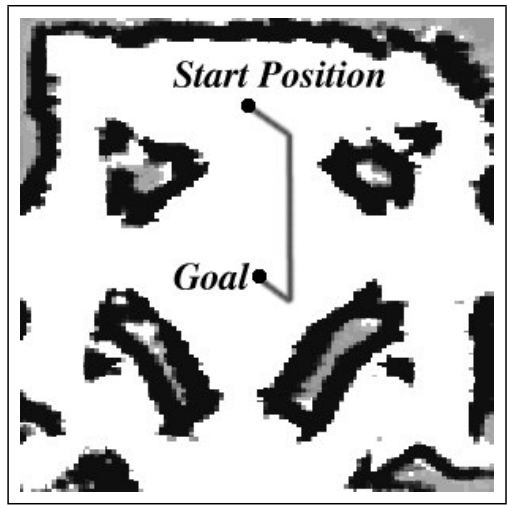
- a) and c) plans that ignore robot's perceptual uncertainty.
- b) and d) was computed by AMDP algorithm. It avoids regions where the robot is more likely to get lost.
- Useful for sensors with maximal range below 4m.





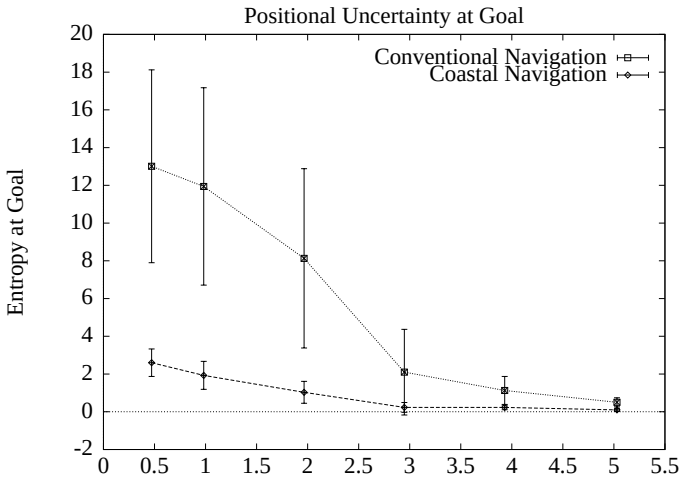


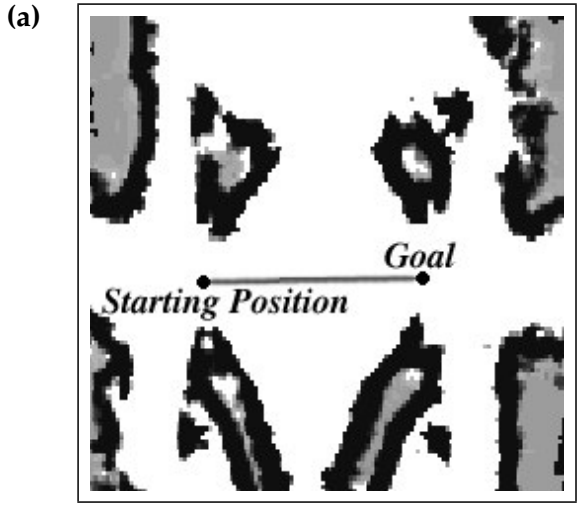


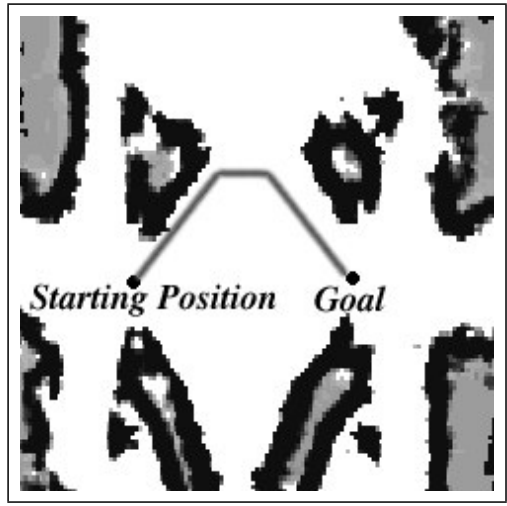


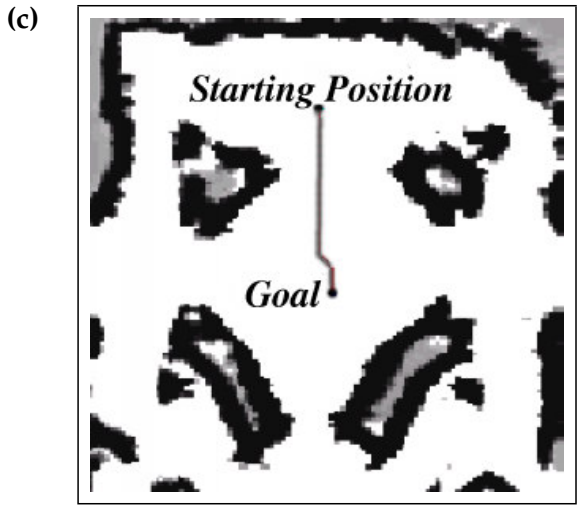
Augmented MDP Application

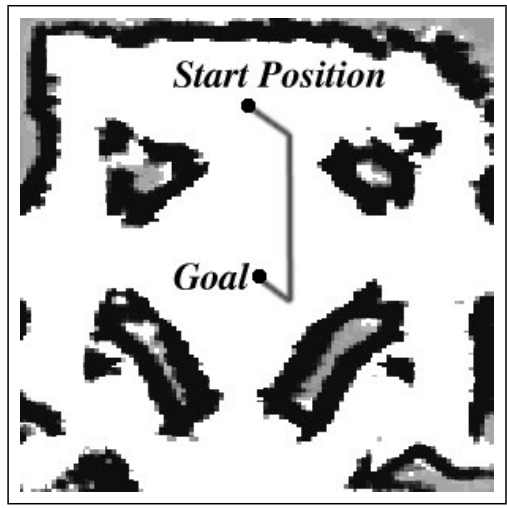
- a) and c) plans that ignore robot's perceptual uncertainty.
- b) and d) was computed by AMDP algorithm. It avoids regions where the robot is more likely to get lost.
- Useful for sensors with maximal range below 4m.











QMPD

- Each belief point adds a single linear 'Bellman' equation

$$\hat{V}(b) = \mathbb{E}_b[\hat{V}(s)] = \sum_{i=1}^{N} b(s_i)\hat{V}(s_i)$$

- Line 4: The value function is calculated in the original space X and controls u
- Line 6: The generalization to belief space.
- $\hat{V}$ can be used as an input to the POMDP evaluation.
- we may continue with T iterations, even small values of T help.

QMDP

1: **procedure** QMDP(b)
2: $\hat{V} = MDP\_discrete\_value\_iteration()$
3: **for** each action $a \in A$ **do**
4: **for** each state $s \in S$ **do**
5: $Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} \hat{V}(s) p(s'|a,s)$
6: **end for**
7: **end for**
8: **return** $\arg\max_a \sum_{s \in S} b(s) Q(s,a)$
9: **end procedure**

Applications

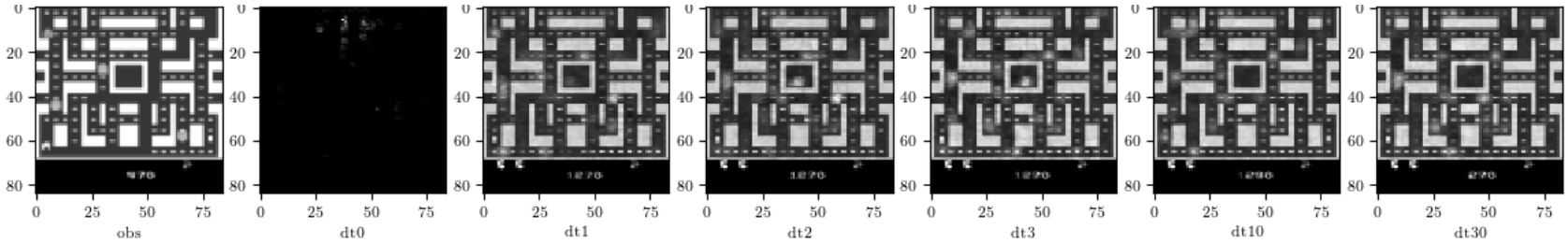
- Pengfei Zhu, Xin Li , Pascal Poupart, Guanghui Miao: **On Improving Deep Reinforcement Learning for POMDPs**, https://arxiv.org/abs/1704.07978v6
- Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, Shimon Whiteson: **Deep Variational Reinforcement Learning for POMDPs**, Proceedings of the 35 th International Conference on Machine

Applications

- Pengfei Zhu, Xin Li , Pascal Poupart, Guanghui Miao: **On Improving Deep Reinforcement Learning for POMDPs**, https://arxiv.org/abs/1704.07978v6
- Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, Shimon Whiteson: **Deep Variational Reinforcement Learning for POMDPs**, Proceedings of the 35 th International Conference on Machine

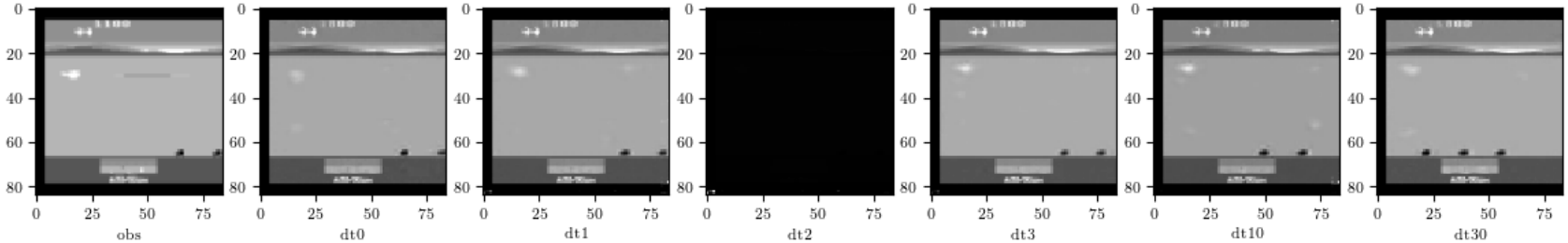
Atari games

- Simple screens, logic
- 18 actions
- used to train an agent by Reinforcement learning with no prior information.
- MDP if fully observable



(e) MsPacman

- flickering for POMPD training (50% of figures are black)

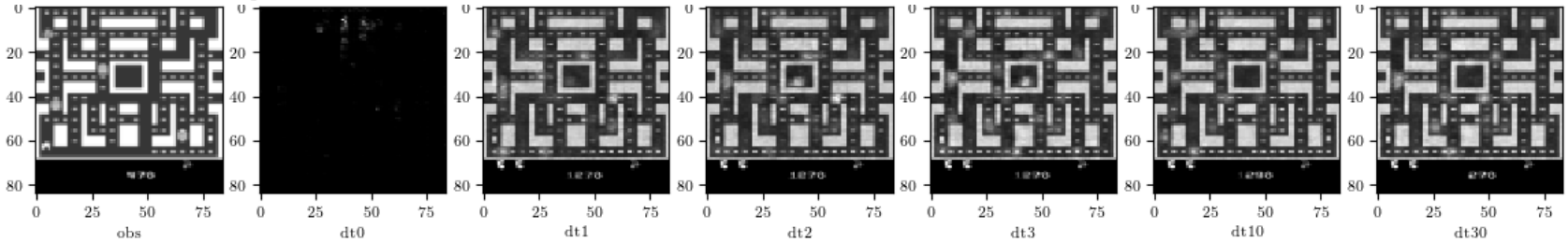


(a) ChopperCommand

Wikipedie

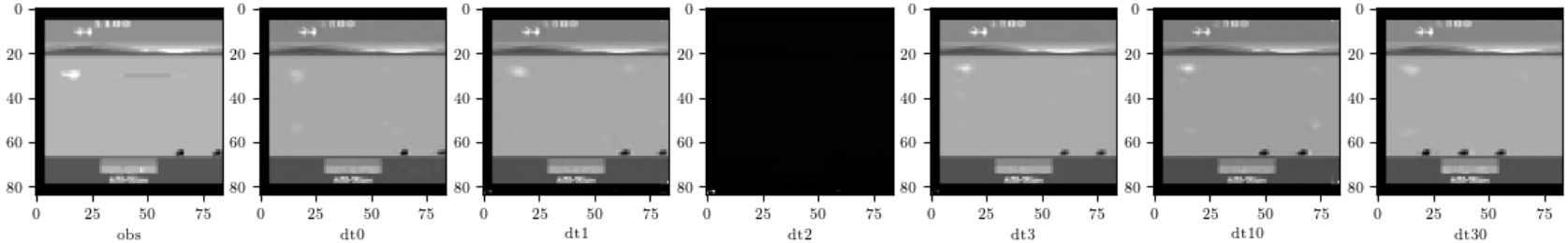
Atari games

- Simple screens, logic
- 18 actions
- used to train an agent by Reinforcement learning with no prior information.
- MDP if fully observable



(e) MsPacman

- flickering for POMPD training (50% of figures are black)

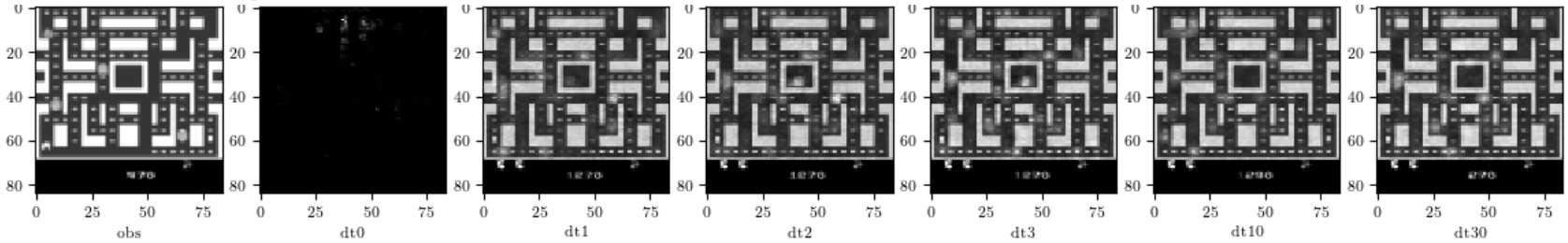


(a) ChopperCommand

Wikipedie

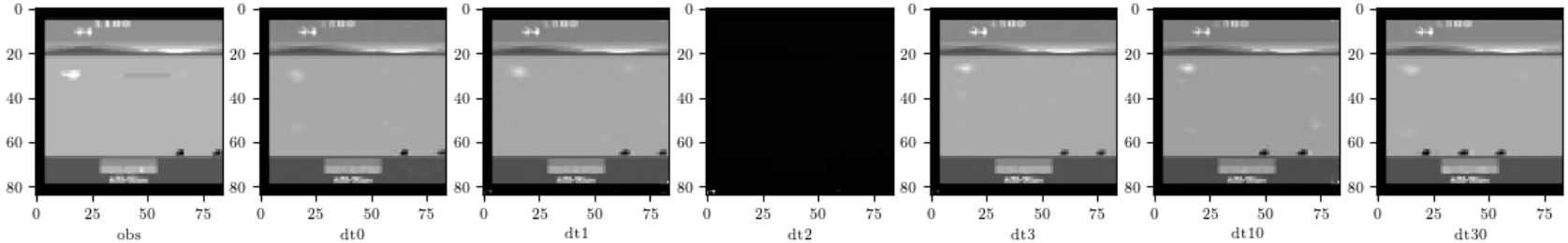
Atari games

- Simple screens, logic
- 18 actions
- used to train an agent by Reinforcement learning with no prior information.
- MDP if fully observable



(e) MsPacman

- flickering for POMPD training (50% of figures are black)

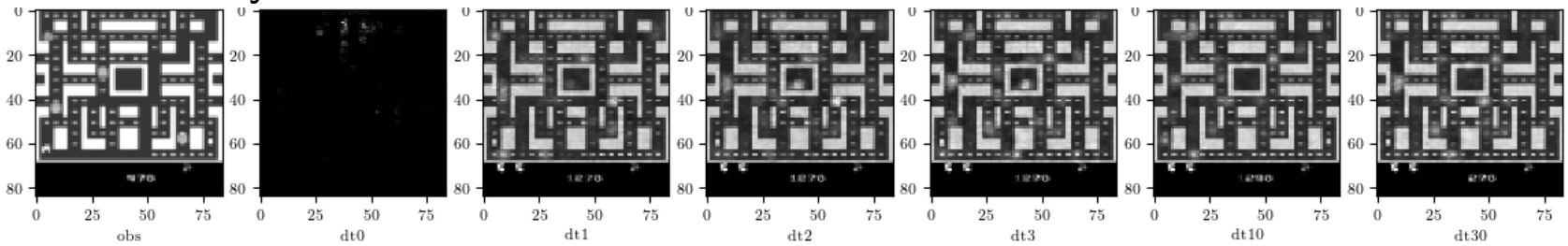


(a) ChopperCommand

Wikipedie

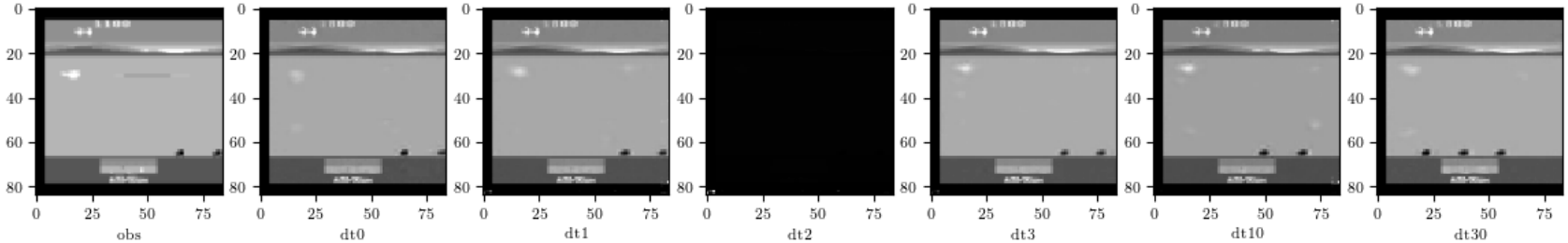
Atari games

- Simple screens, logic
- 18 actions
- used to train an agent by Reinforcement learning with no prior information.
- MDP if fully observable



(e) MsPacman

- flickering for POMPD training (50% of figures are black)

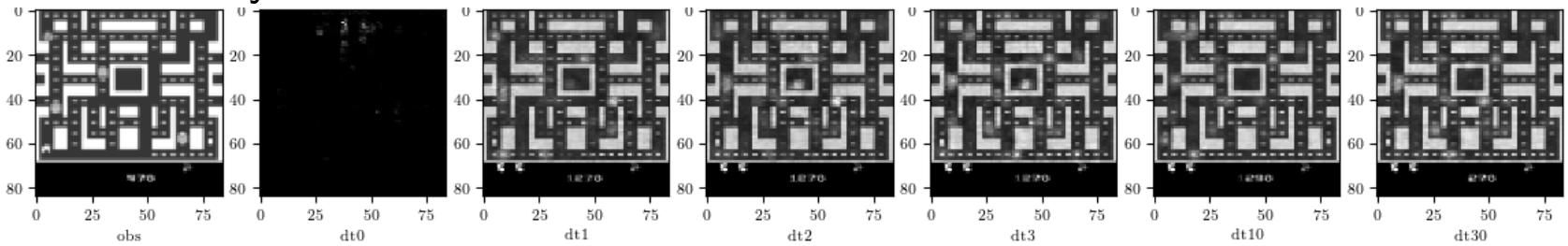


(a) ChopperCommand

Wikipedie

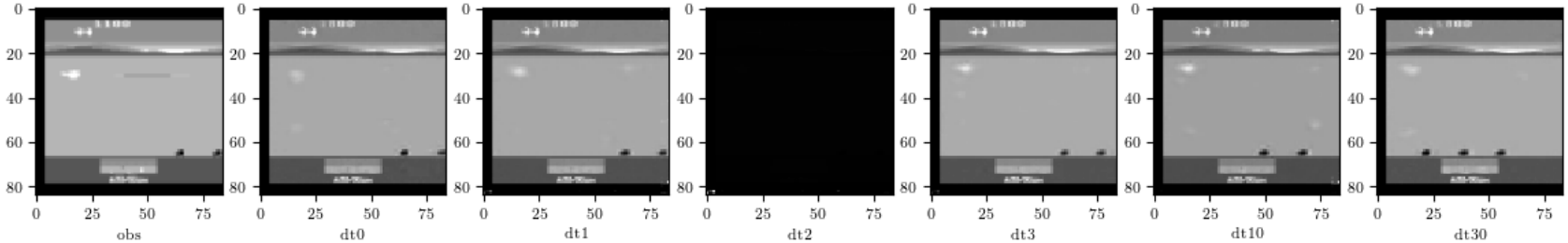
Atari games

- Simple screens, logic
- 18 actions
- used to train an agent by Reinforcement learning with no prior information.
- MDP if fully observable



(e) MsPacman

- flickering for POMPD training (50% of figures are black)



(a) ChopperCommand

Wikipedie

DeepQNetwork - Q function approx. by a neural network

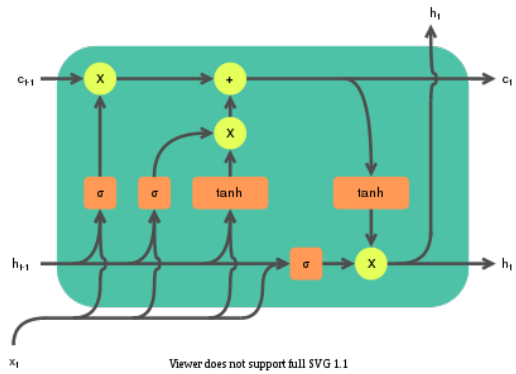
- standard reinforcement learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- NN θ is trained to minimize the loss function

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma max_{a'} Q(s',a';\theta_i^{old}) - Q(s',a';\theta_i))^2].$$

- DRQN uses recurrent LSTM network



- hidden state keeps the track in flickered images
- ADRQN Action-specific Deep Recurent Q-network

DeepQNetwork - Q function approx. by a neural network

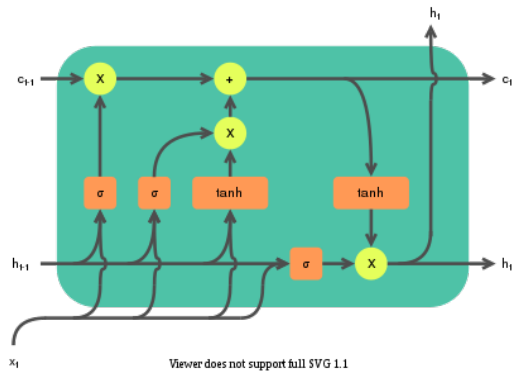
- standard reinforcement learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- NN θ is trained to minimize the loss function

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma max_{a'} Q(s',a';\theta_i^{old}) - Q(s',a';\theta_i))^2].$$

- DRQN uses recurrent LSTM network



  - hidden state keeps the track in flickered images
- ADRQN Action-specific Deep Recurent Q-network

DeepQNetwork - Q function approx. by a neural network

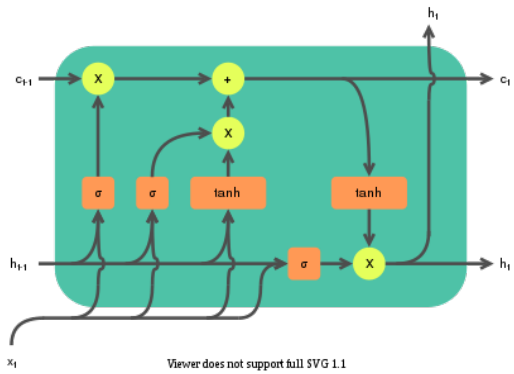
- standard reinforcement learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- NN θ is trained to minimize the loss function

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma max_{a'} Q(s',a';\theta_i^{old}) - Q(s',a';\theta_i))^2].$$

- DRQN uses recurrent LSTM network



  - hidden state keeps the track in flickered images
- ADRQN Action-specific Deep Recurent Q-network

DeepQNetwork - Q function approx. by a neural network

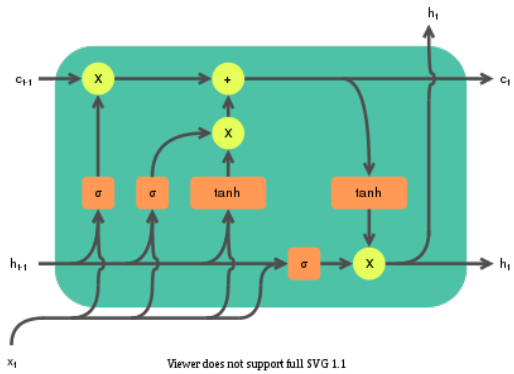
- standard reinforcement learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- NN θ is trained to minimize the loss function

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma max_{a'} Q(s',a';\theta_i^{old}) - Q(s',a';\theta_i))^2].$$

- DRQN uses recurrent LSTM network



  - hidden state keeps the track in flickered images
- ADRQN Action-specific Deep Recurent Q-network

DeepQNetwork - Q function approx. by a neural network

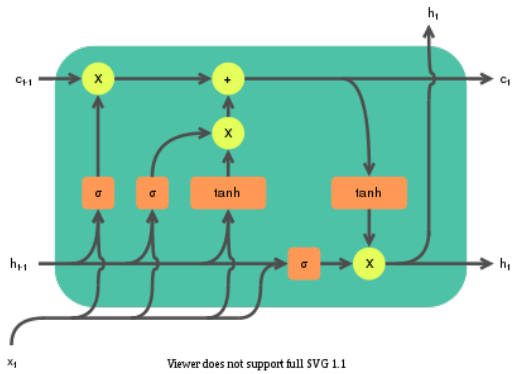
- standard reinforcement learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- NN θ is trained to minimize the loss function

$$L(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma max_{a'} Q(s',a';\theta_i^{old}) - Q(s',a';\theta_i))^2].$$

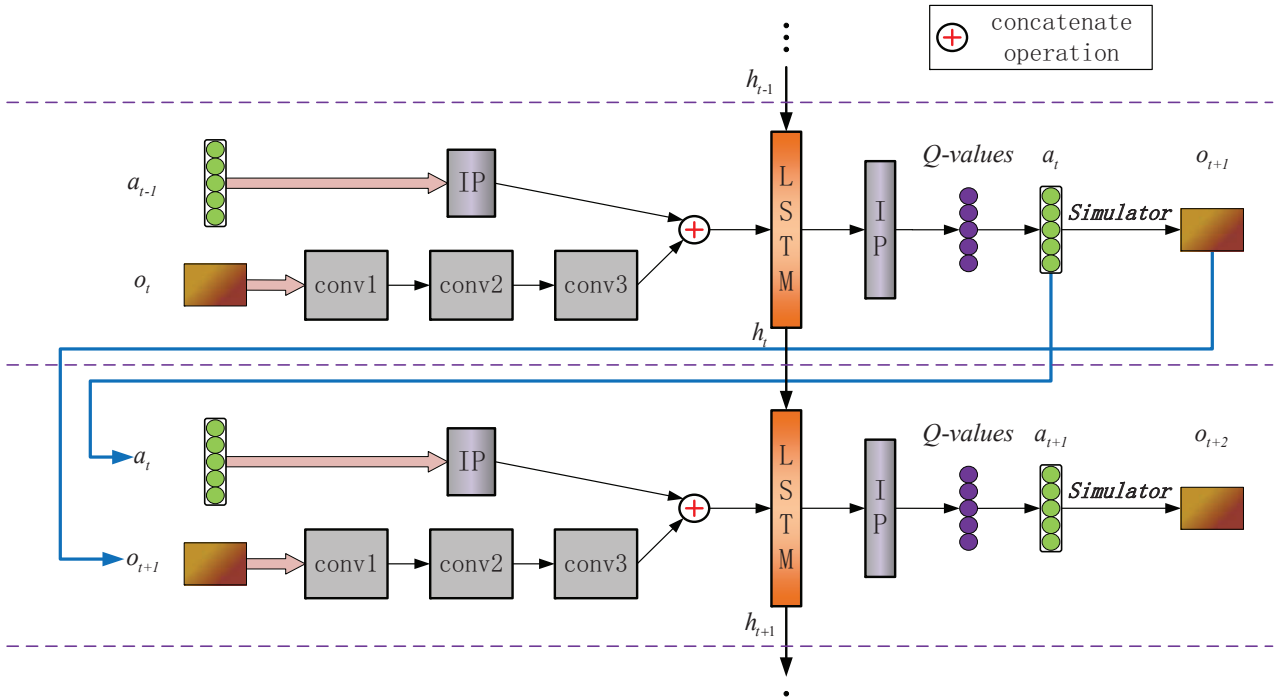
- DRQN uses recurrent LSTM network



  - hidden state keeps the track in flickered images
- ADRQN Action-specific Deep Recurent Q-network

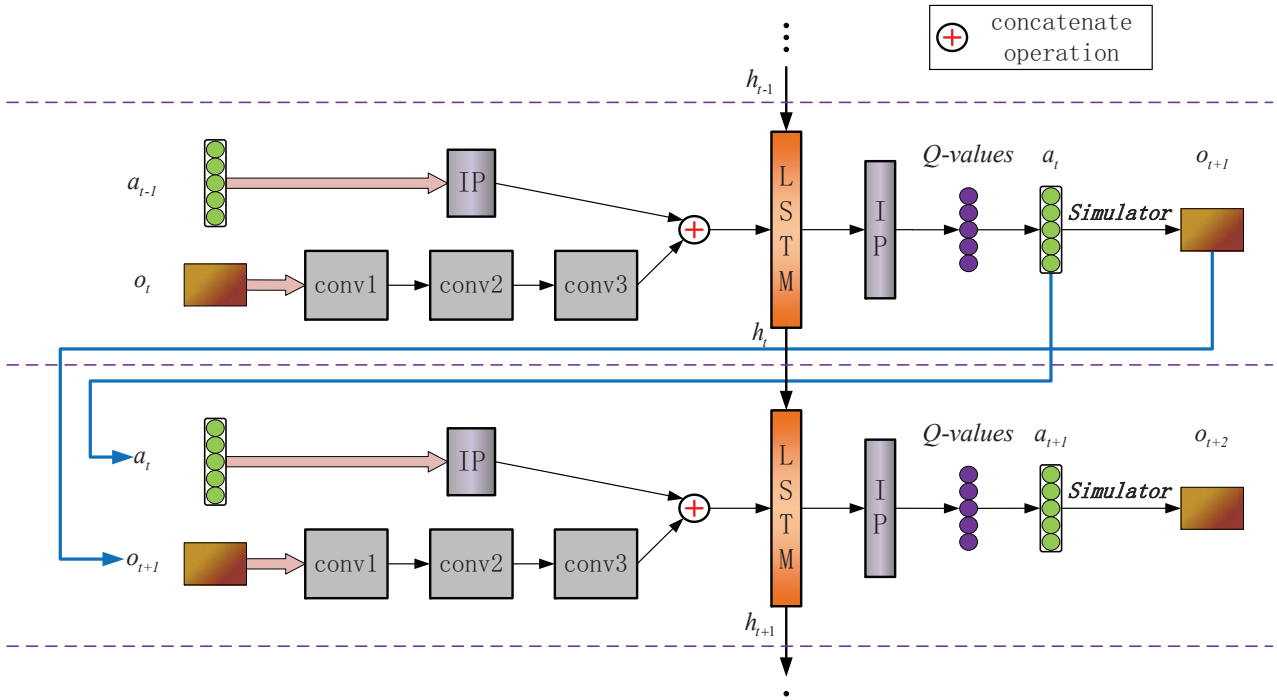
ADRQN Action-specific Deep Recurent Q-network

- The observation (image) is aggregated by three convolutional layers to 3136 dimensions
- the action is extended from 18 dimensional indicator vector to 512 dimensional vector
- these are concatenated and fed to LSTM
- result is reduced to the number of action dimension, an action is selected and passed to simulator
- the simulator provides the observation o and a new pass begins.



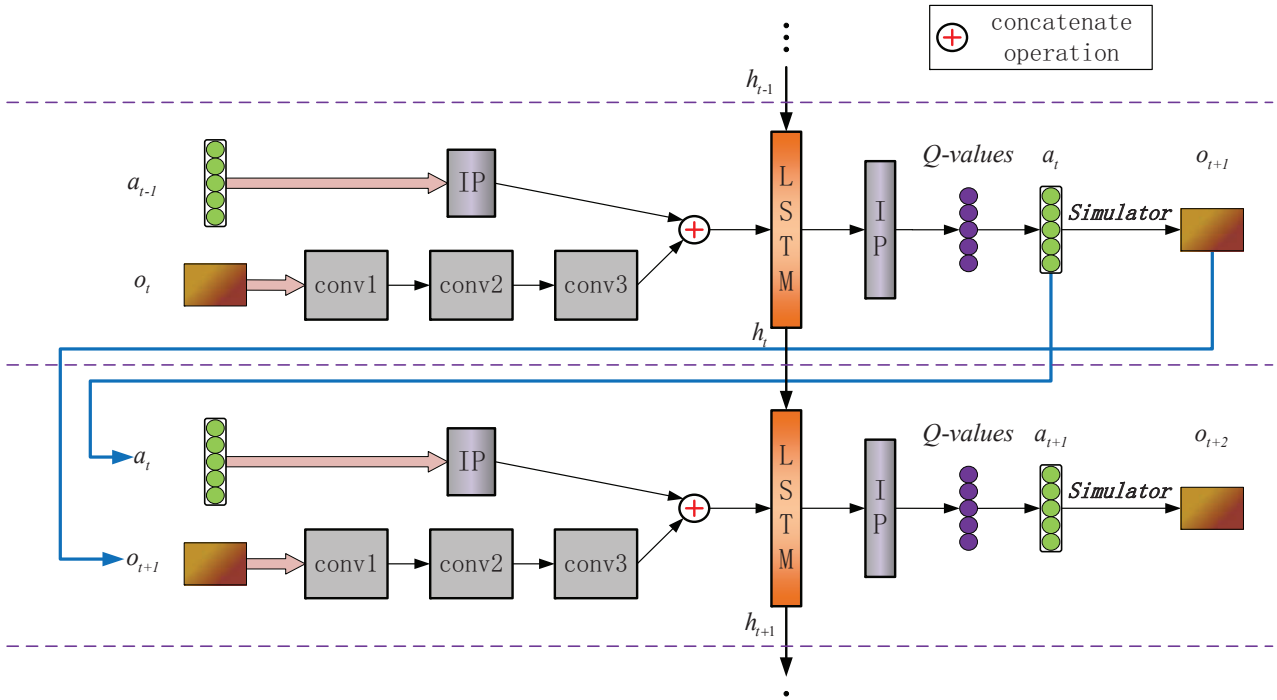
ADRQN Action-specific Deep Recurent Q-network

- The observation (image) is aggregated by three convolutional layers to 3136 dimensions
- the action is extended from 18 dimensional indicator vector to 512 dimensional vector
- these are concatenated and fed to LSTM
- result is reduced to the number of action dimension, an action is selected and passed to simulator
- the simulator provides the observation o and a new pass begins.



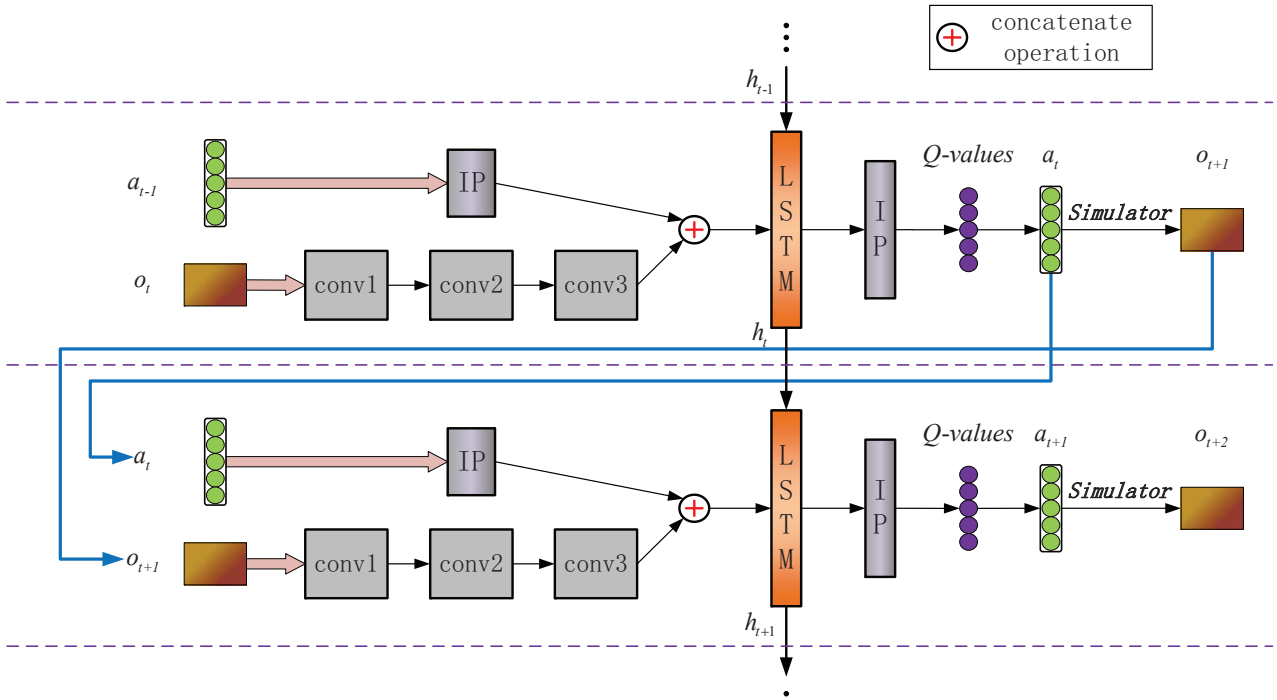
ADRQN Action-specific Deep Recurent Q-network

- The observation (image) is aggregated by three convolutional layers to 3136 dimensions
- the action is extended from 18 dimensional indicator vector to 512 dimensional vector
- these are concatenated and fed to LSTM
- result is reduced to the number of action dimension, an action is selected and passed to simulator
- the simulator provides the observation o and a new pass begins.



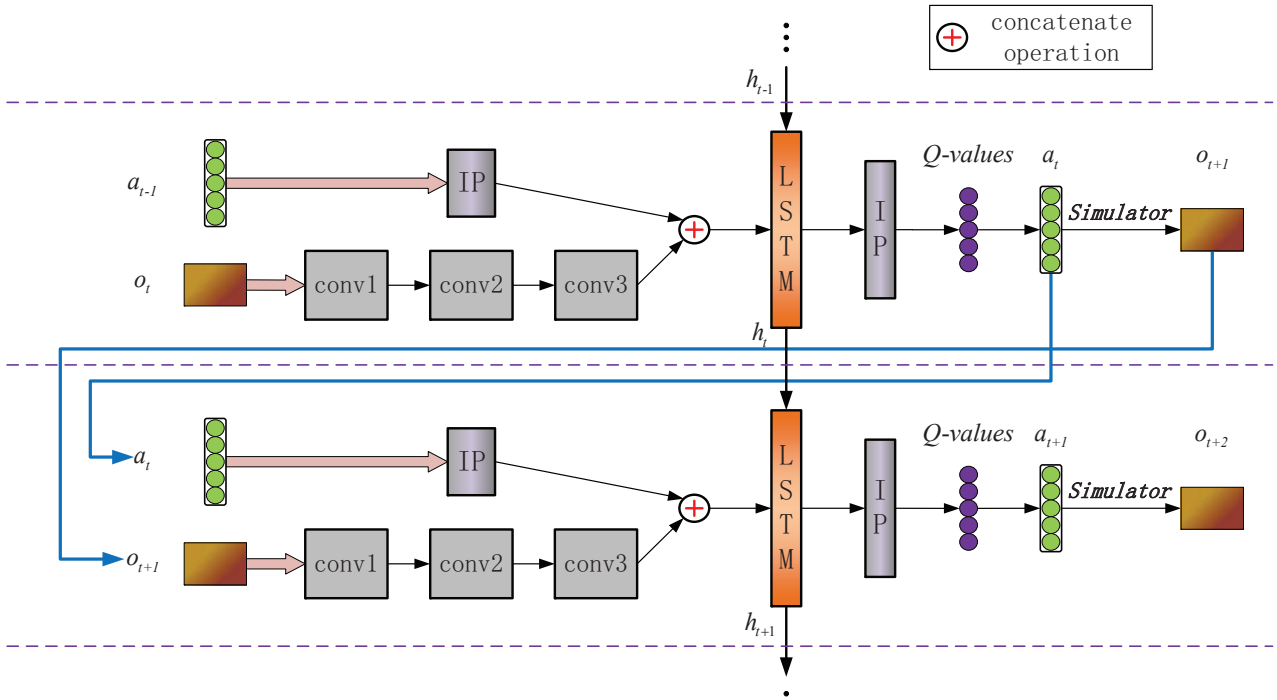
ADRQN Action-specific Deep Recurent Q-network

- The observation (image) is aggregated by three convolutional layers to 3136 dimensions
- the action is extended from 18 dimensional indicator vector to 512 dimensional vector
- these are concatenated and fed to LSTM
- result is reduced to the number of action dimension, an action is selected and passed to simulator
- the simulator provides the observation o and a new pass begins.



ADRQN Action-specific Deep Recurent Q-network

- The observation (image) is aggregated by three convolutional layers to 3136 dimensions
- the action is extended from 18 dimensional indicator vector to 512 dimensional vector
- these are concatenated and fed to LSTM
- result is reduced to the number of action dimension, an action is selected and passed to simulator
- the simulator provides the observation o and a new pass begins.



ADRQN Learning

- Simulate M episodes
- actual reward r_j is used
- future reward from the old network θ^{old}
- adjust weights θ by the gradient of $[(r_j + \gamma max_a Q(h_j, a_j, o_{j+1}, a; \theta_i^{old}) - Q(h_{j-1}, a_{j-1}, o_j, a_j; \theta))^2]$

Algorithm 1 Action-specific Deep Recurrent Q-Network

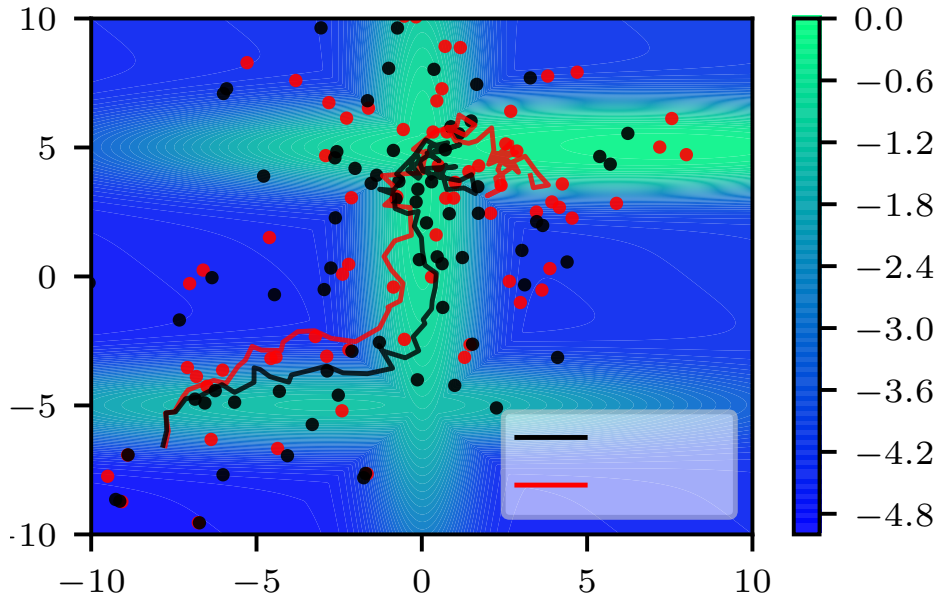
Initialize the replay memory D, # of iterations M
Initialize Q-Network and Target-Network with θ and θ^- respectively
for episode = 1 to M **do**
    Initialize the first action $a_0 = no\,operation$, $h_0 = \mathbf{0}$
    Init.the first obs.o_1 with the preprocessed first screen
    while $o_t \neq terminal$ **do**
        Select a random action a_t with the probability ϵ
        Else select $a_t = \text{argmax}_a\, Q(h_{t-1}, a_{t-1}, o_t, a; \theta)$
        Execute action a_t
        Obtain reward r_t and resulting observation o_{t+1}
        Store transition $\langle \{a_{t-1}, o_t\}, a_t, r_t, o_{t+1} \rangle$ as one record of the current episode in D
        Randomly sample a minibatch of transition sequences $\langle a_{j-1}, o_j, a_j, r_j, o_{j+1} \rangle$ from D
        Compute Q-value of target network

$$y_j = \begin{cases} r_j, & o_{j+1} = terminal \\ r_j + \gamma \max\limits_a Q(h_j, a_j, o_{j+1}, a; \theta^-), & o_{j+1} \neq terminal \end{cases}$$

        Compute the gradient of $(y_j - Q(h_{j-1}, a_{j-1}, o_j, a_j; \theta))^2$ to update θ
    end while
end for

Deep Variational Reinforcement Learning for POMDPs

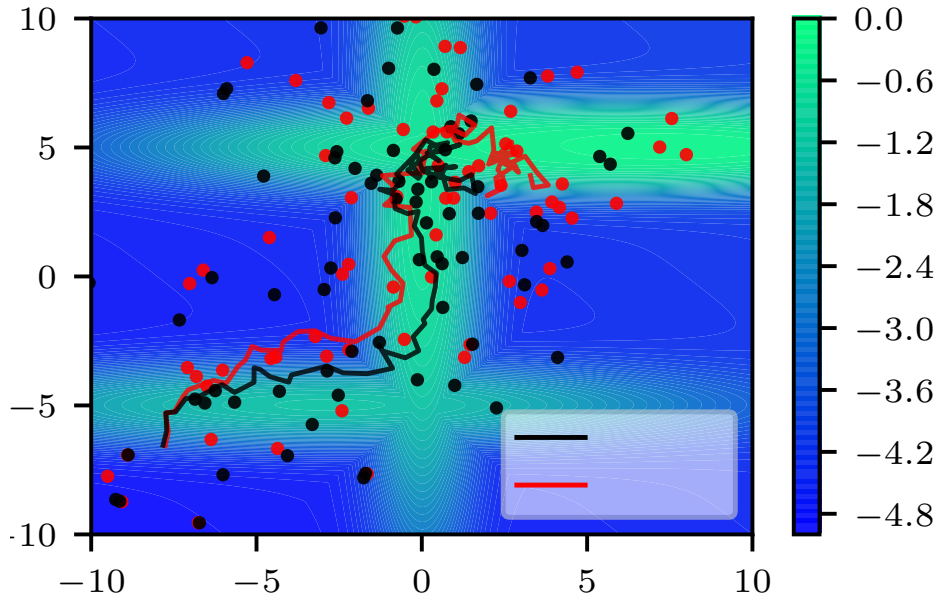
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + a_t + \epsilon_t$,
  - $\epsilon_t \sim \mathcal{N}(0, 0.25 \cdot I)$
  - a_t is a_t with the length capped to $|a_t| \leq 0.5$.
  - $o_t = s_t + \delta_t$,
  - $\delta_t \sim \mathcal{N}(0, \sigma_{obs} \cdot I)$, $\sigma_{obs} \in \{0, 1, 2, 3\}$
  - $R_t = R(s_t, a_t) = 100[|s_t| < 1]$ the color in the figure.
  - $b_0 \sim \mathcal{N}([-8.0, -8.0]^T, I)$.
  - Test 25 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

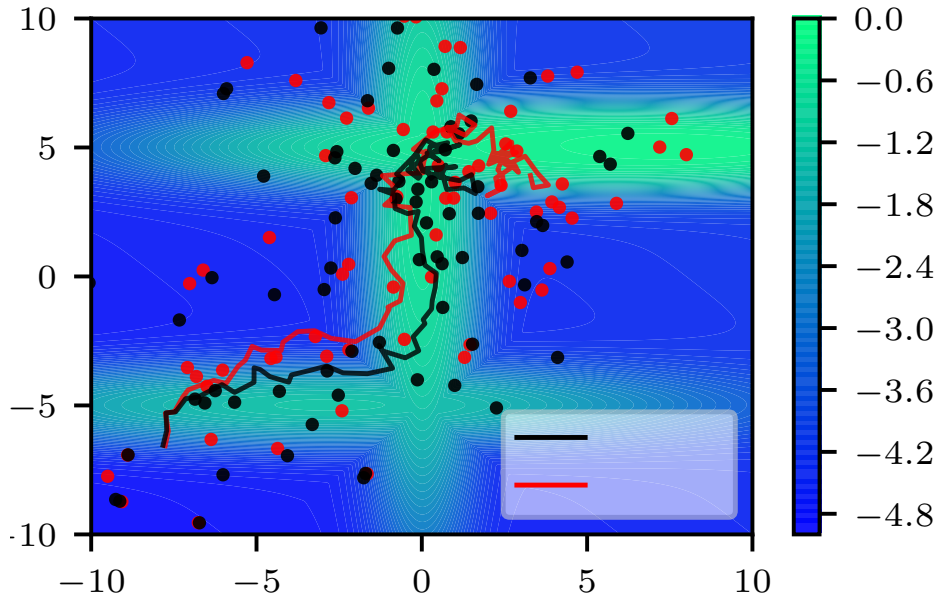
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

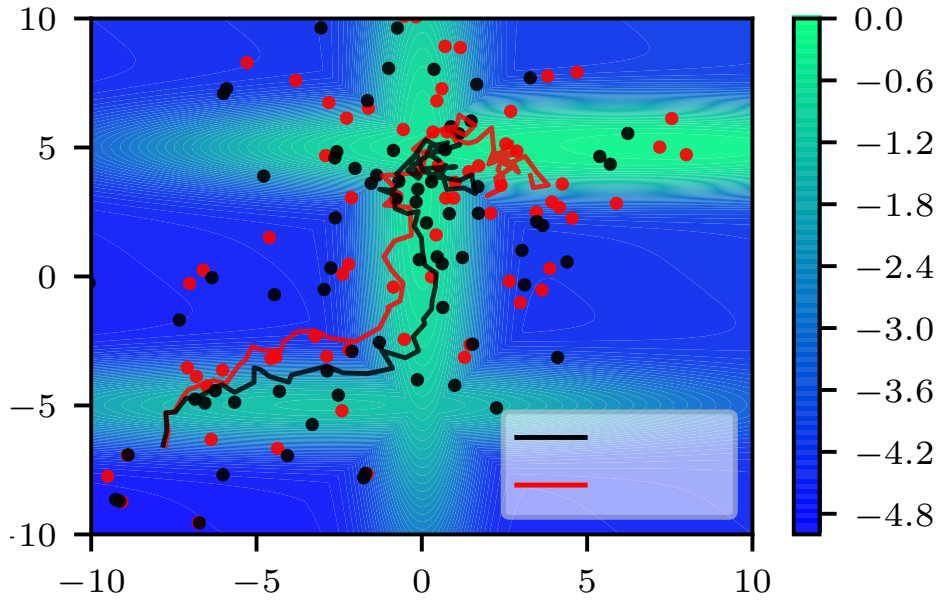
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

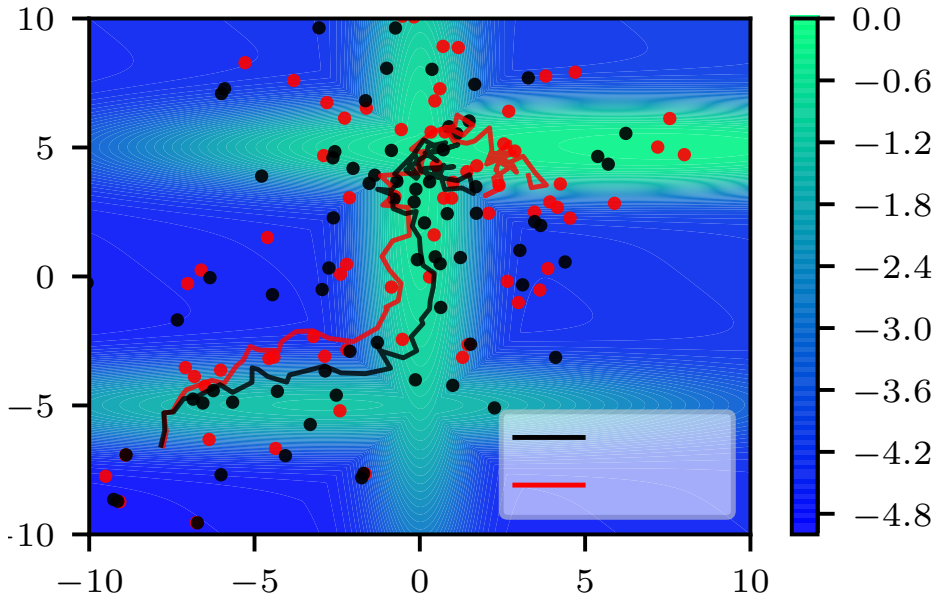
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

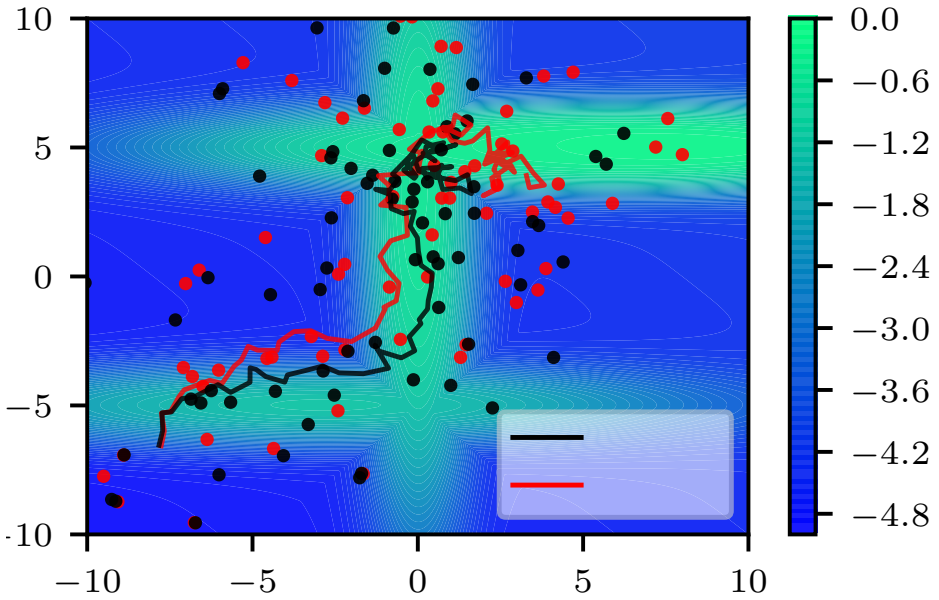
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

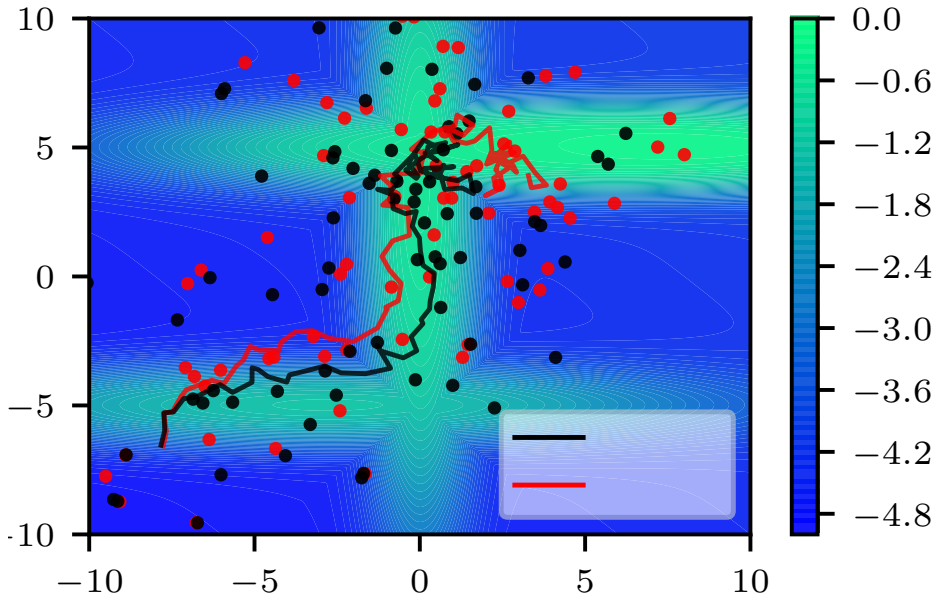
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

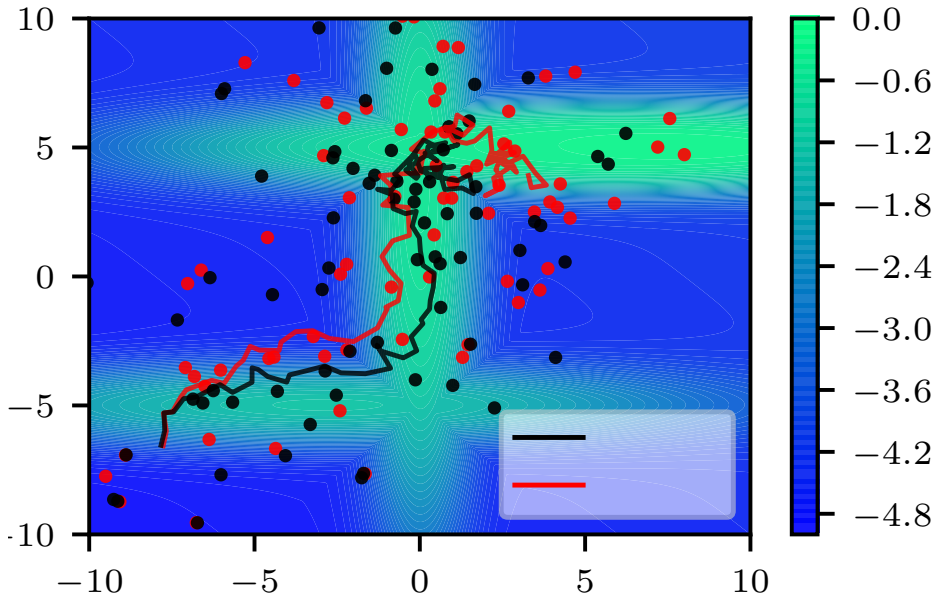
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

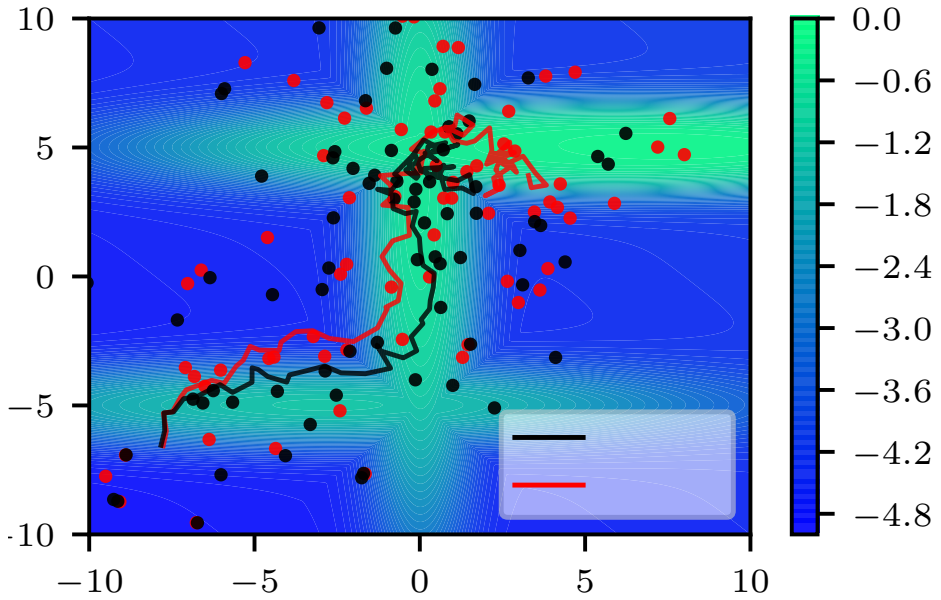
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

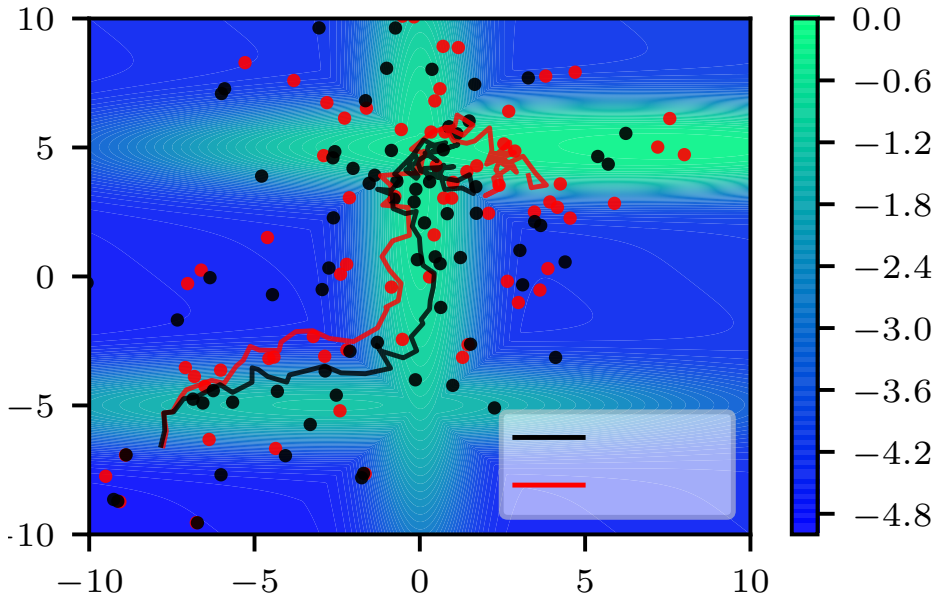
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

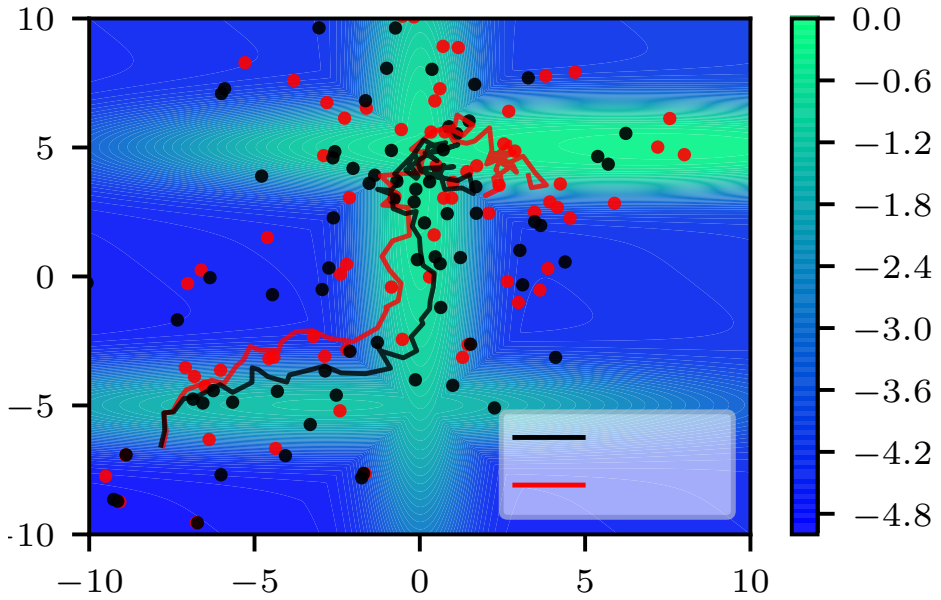
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

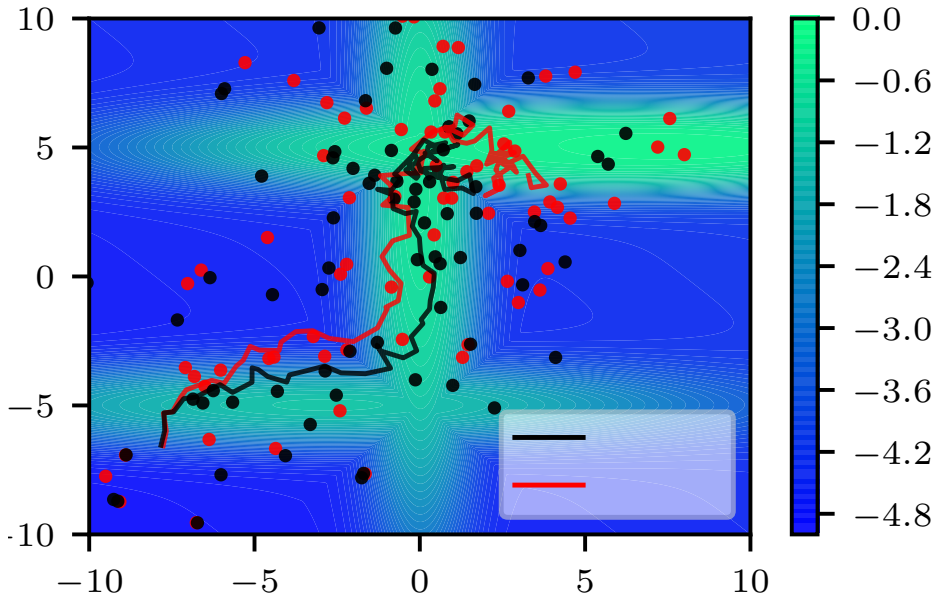
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

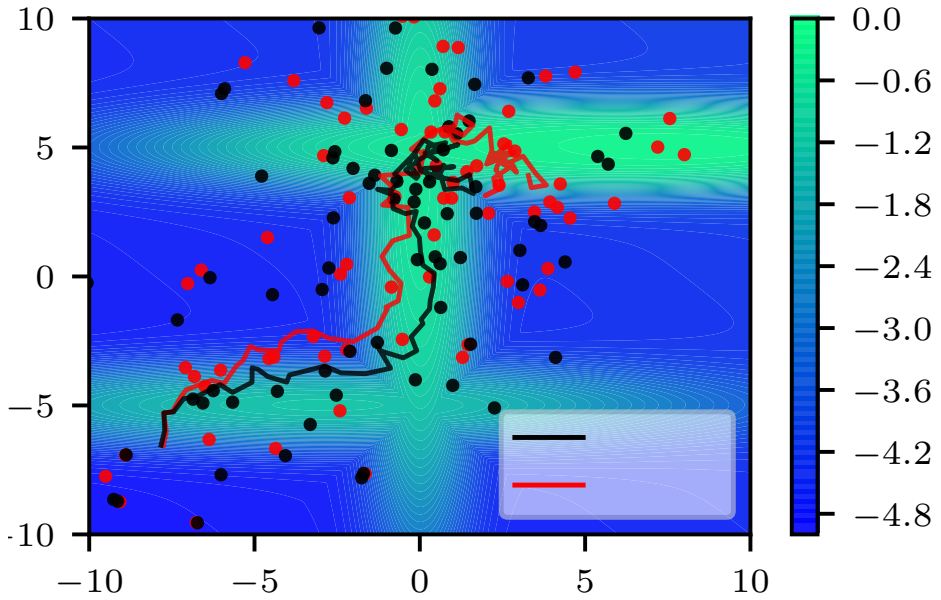
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

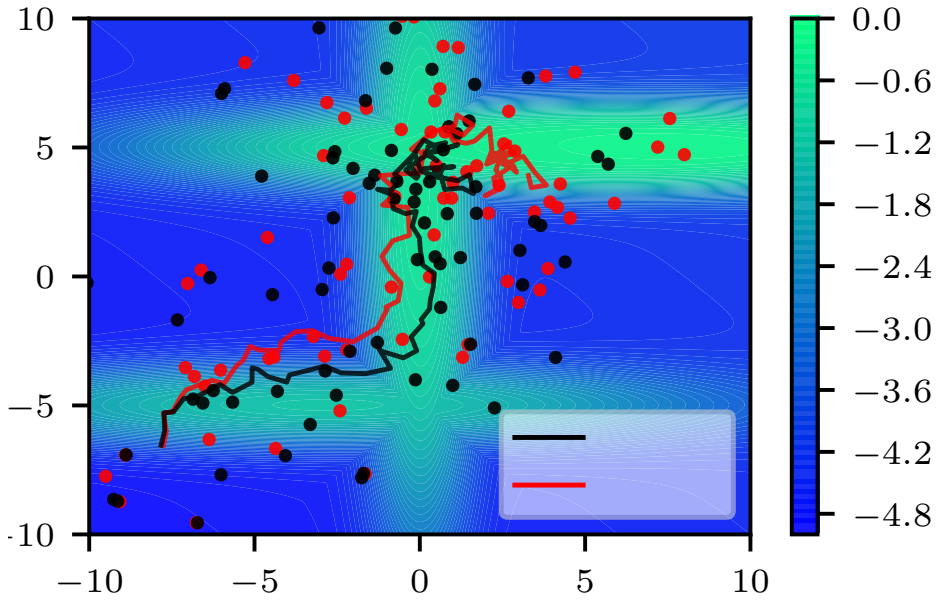
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

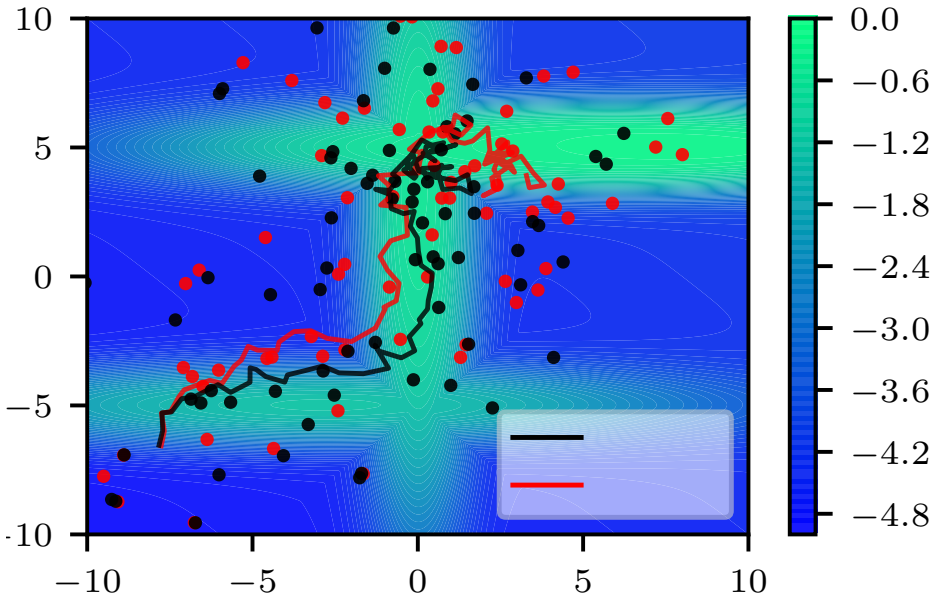
- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

Deep Variational Reinforcement Learning for POMDPs

- Mountain hike
- POMPD
  - $S = O = A = \mathbb{R}^2$
  - state, observed state, the desired step.
  - $s_{t+1} = s_t + \tilde{a}_t + \epsilon_{s,t}$
  - $\epsilon \sim N(0, 0.25 \cdot I)$
  - $\tilde{a}_t$ is a_t with the length capped to $|\tilde{a}_t| \leq 0.5$.
  - $o_t = s_t + \epsilon_{o,t}$
  - $\epsilon \sim N(0, \sigma_0 \cdot I)$, $\sigma_0 \in \{0, 1.5, 3\}$
  - $R_t = r(x_t, y_t) - 0.01|\tilde{a}_t|$, r the color in the figure.
  - $b_0 \sim N([-8.5, -8.5]^T, I)$,
  - T=75 steps.



- Red: Recurrent Neural Network
- Black: Deep Variational Reinforcement Learning
- dots: observations.

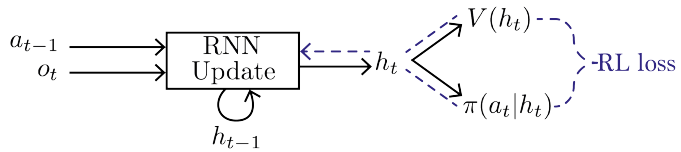
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

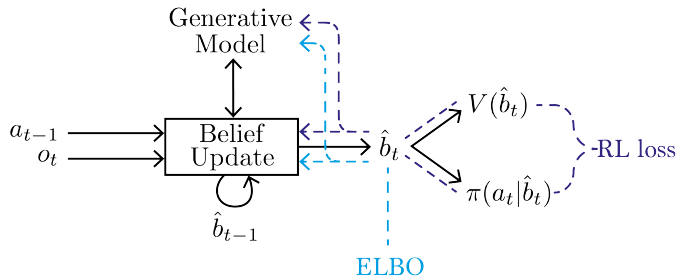
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

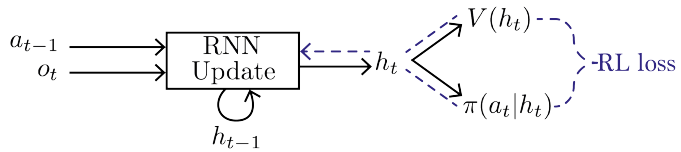
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

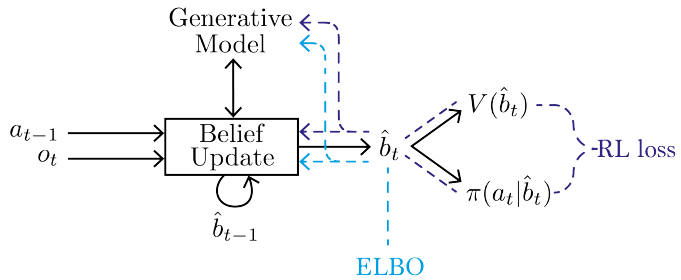
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

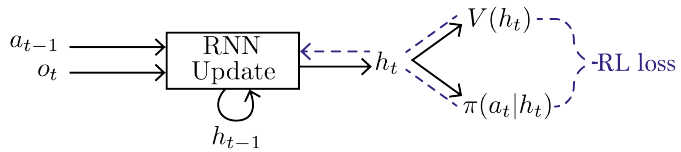
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

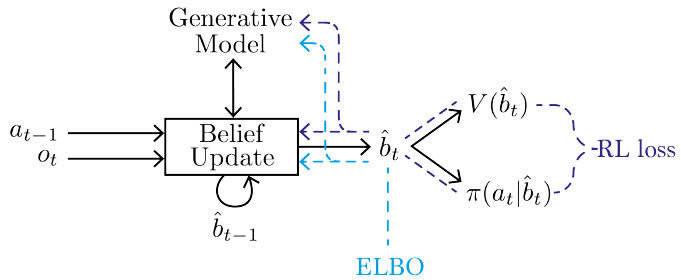
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

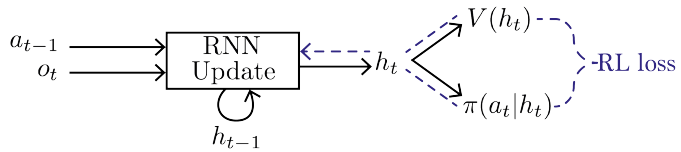
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

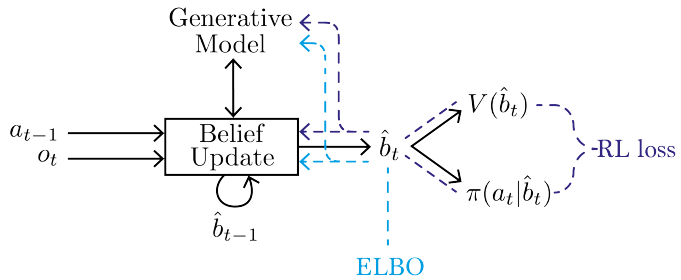
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

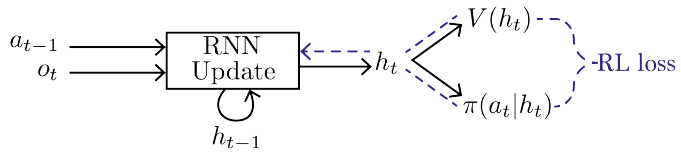
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

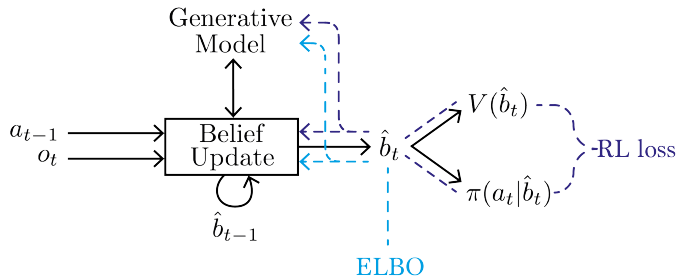
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

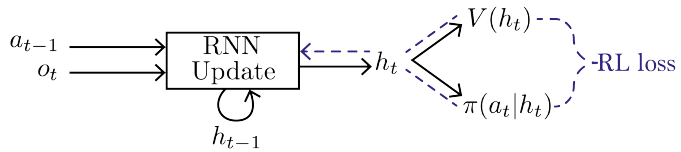
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

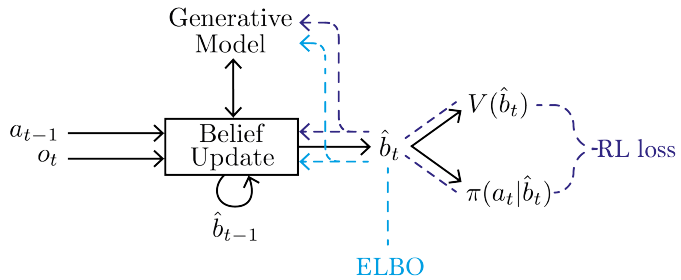
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

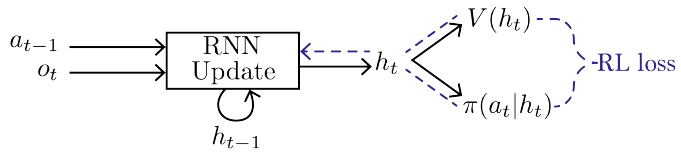
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

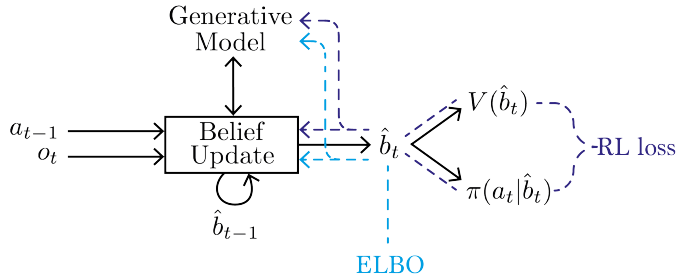
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

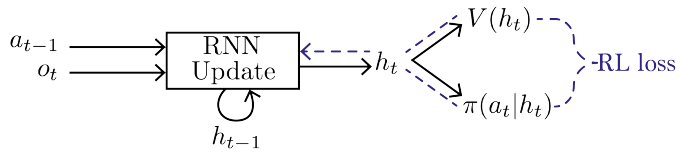
Deep Variational Reinforcement Learning for POMDPs

Recurrent Neural Network

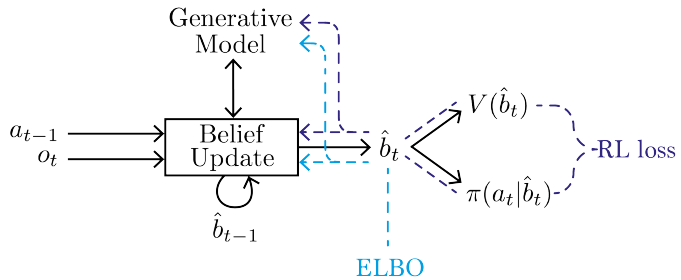
- h_t latent state
- the current update n_s steps
- computational graphs n_g steps
- $n_g > n_s$ greatly improves the performance.

DVRL

- $K = 30$ particles (h_t^k, z_t^k, w_t^k)
- h_t^k the latent state of an RNN
- z_t^k an additional state to learn the transition
- w_t^k an particle importance weight.



(a) RNN-based approach. The RNN acts as an encoder for the action-observation history, on which actor and critic are conditioned. The networks are updated end-to-end with an RL loss.



(b) DVRL. The agent learns a generative model which is used to update a belief distribution. Actor and critic now condition on the belief. The generative model is learned to optimise both the ELBO *and* the RL loss.

Figure 1: Comparison of RNN and DVRL encoders.

Recurrent Latent State Update

- resample particles

$$u_{t-1}^k \sim \mathit{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^k}\right)$$

- model the transition

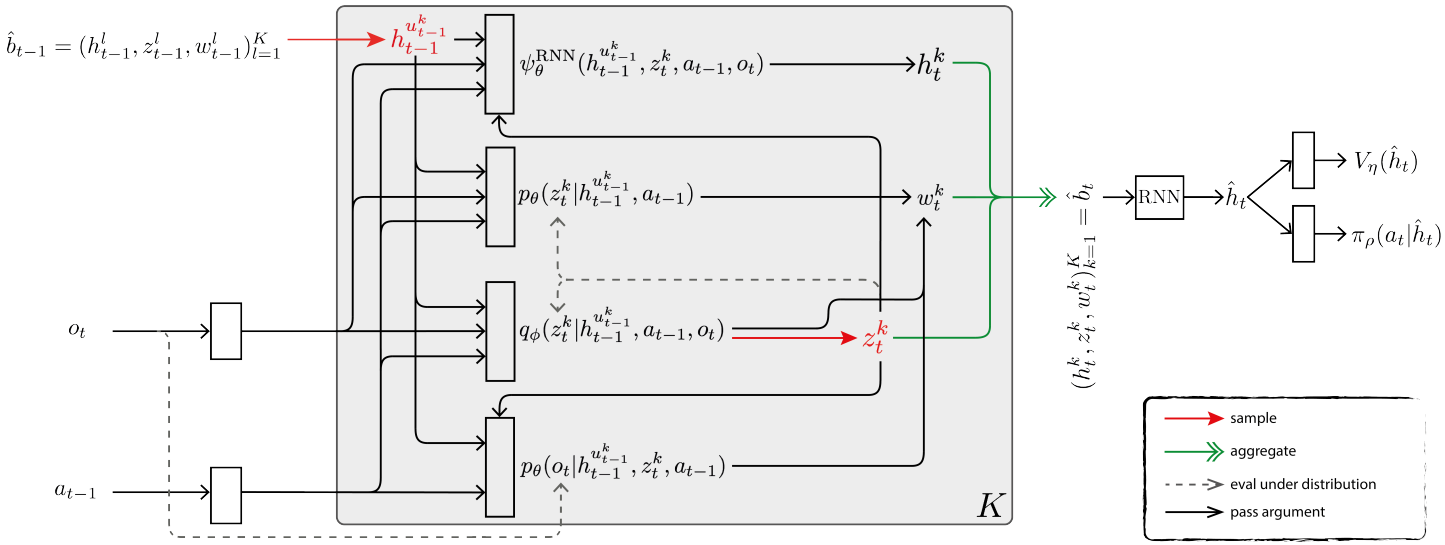
$$z_t^k \sim q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$$

- RNN latent state update

$$h_t^k = \psi_\theta^{RNN}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t)$$

- the weight: probability o_t in our model

$$w_t^k = \frac{p_\theta(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}) p_\theta(o_t | h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)}$$



Recurrent Latent State Update

- resample particles

$$u_{t-1}^k \sim \mathit{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^k}\right)$$

- model the transition

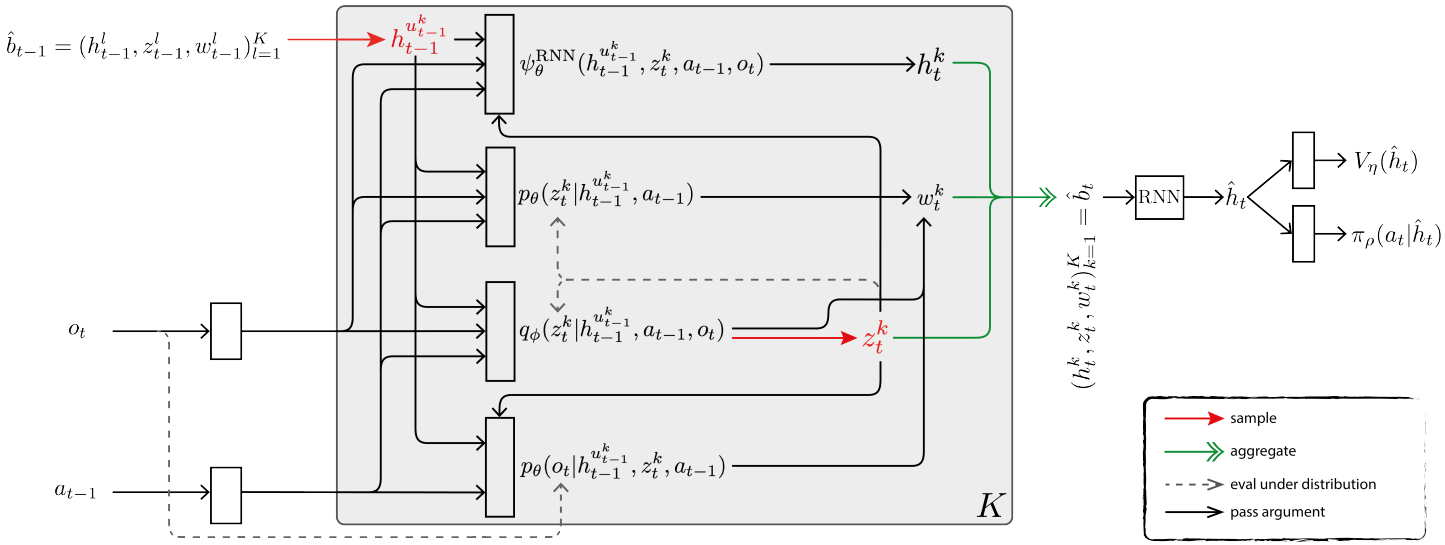
$$z_t^k \sim q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$$

- RNN latent state update

$$h_t^k = \psi_\theta^{RNN}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t)$$

- the weight: probability o_t in our model

$$w_t^k = \frac{p_\theta(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}) p_\theta(o_t | h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)}$$



Recurrent Latent State Update

- resample particles

$$u_{t-1}^k \sim \mathit{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^k}\right)$$

- model the transition

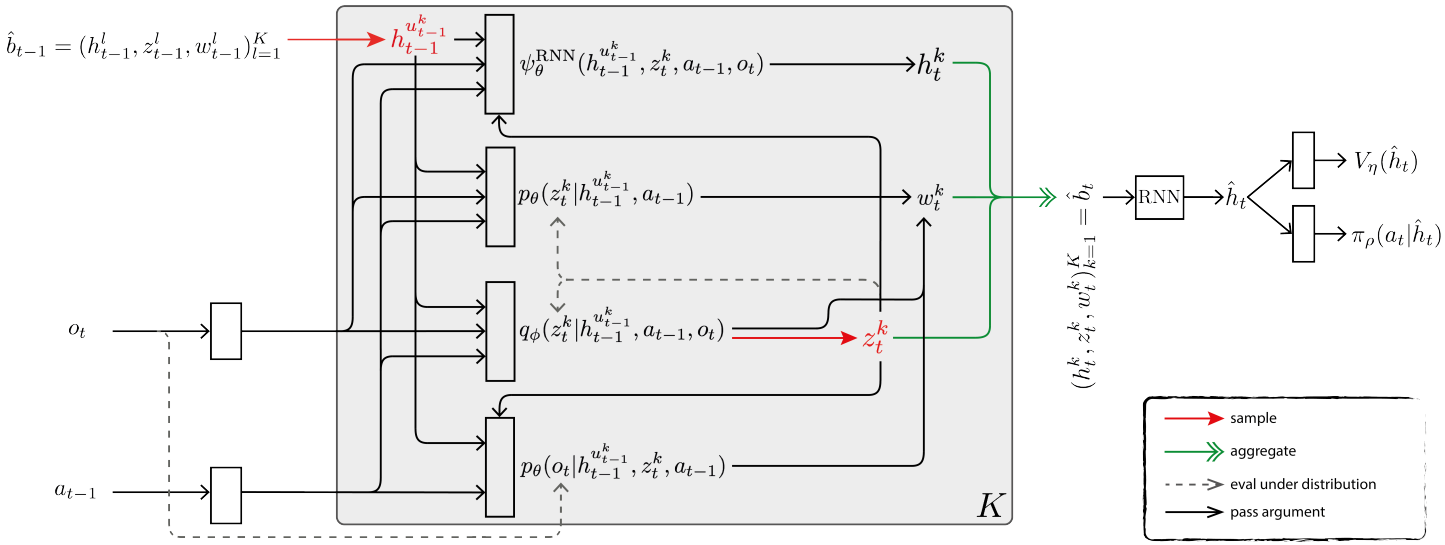
$$z_t^k \sim q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$$

- RNN latent state update

$$h_t^k = \psi_\theta^{RNN}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t)$$

- the weight: probability o_t in our model

$$w_t^k = \frac{p_\theta(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}) p_\theta(o_t | h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)}$$



Recurrent Latent State Update

- resample particles

$$u_{t-1}^k \sim \mathit{Discrete}\left(\frac{w_{t-1}^k}{\sum_{j=1}^K w_{t-1}^k}\right)$$

- model the transition

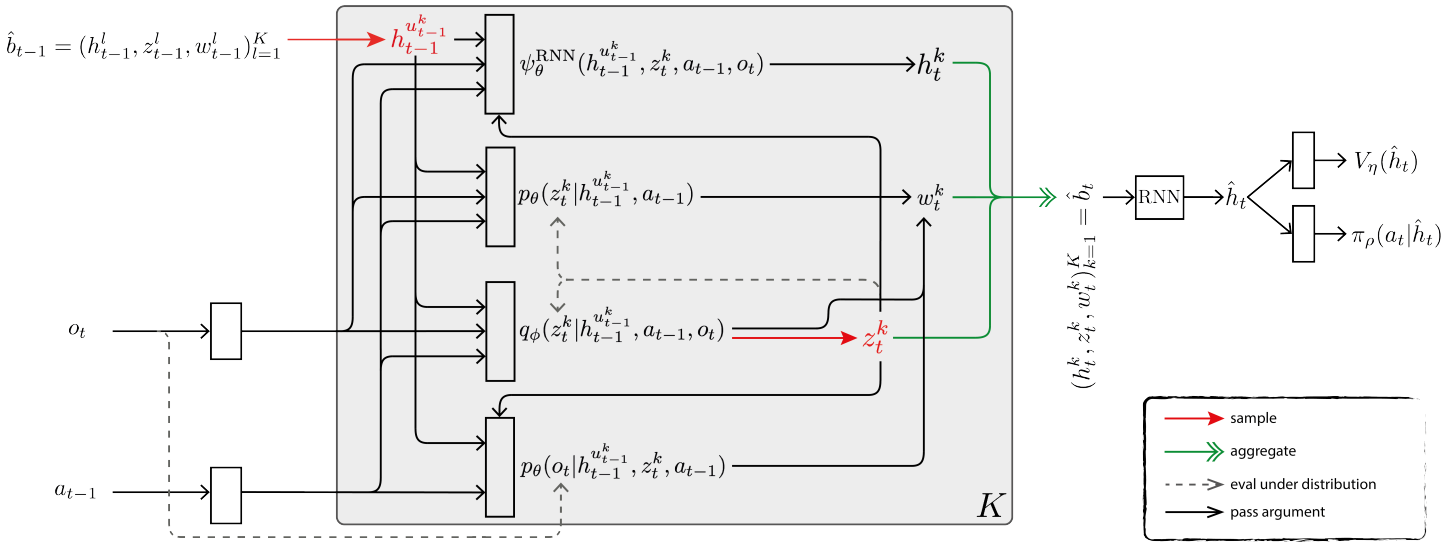
$$z_t^k \sim q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)$$

- RNN latent state update

$$h_t^k = \psi_\theta^{RNN}(h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1}, o_t)$$

- the weight: probability o_t in our model

$$w_t^k = \frac{p_\theta(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}) p_\theta(o_t | h_{t-1}^{u_{t-1}^k}, z_t^k, a_{t-1})}{q_\phi(z_t^k | h_{t-1}^{u_{t-1}^k}, a_{t-1}, o_t)}$$



Learning DVRL

Advantage function

$$A_\eta^{t,i}(a_{t+1}, s_{t+1}) = \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{r+i+j} + \gamma^{n_s-i} V_\eta^{old}(s_{t+n_s}) - V_\eta(s_{t+i}) \right)$$

Loss function

- policy ρ: $L_t^A(\rho) = -\frac{1}{n_e n_s} \sum_{envs} \sum_{i=0}^{n_s-1} log\pi_\rho(a_{t+1}|s_{t+1}) A_\eta^{t,i,old}(a_{t+1}, s_{t+1})$
- value $L_t^V(\eta) = \frac{1}{n_e n_s} \sum_{envs} \sum_{i=0}^{n_s-1} A_\eta^{t,i}(a_{t+1}, s_{t+1})^2$
- exploration $L_t^H(\eta) = \frac{1}{n_e n_s} \sum_{envs} \sum_{i=0}^{n_s-1} H(\pi_\rho(|s_{t+1}))$
- ρ, η: ERO $L_t^{ERO}(\rho, \eta) = \frac{1}{n_e n_s} \sum_{envs} \sum_{i=0}^{n_s-1} log(\pi_\rho(a_{t+1}|s_{t+1}, w_t^A, w_t^C, \eta))$

Learning DVRL

Advantage function

$$A_{\eta}^{t,i}(a_{t+1}, s_{t+1}) = \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{r+i+j} + \gamma^{n_s-i} V_{\eta}^{old}(s_{t+n_s}) - V_{\eta}(s_{t+i}) \right)$$

Loss function

- policy ρ: $\mathcal{L}_t^A(\rho) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \pi_{\rho}(a_{t+1}|s_{t+1}) A_{\eta}^{t,i,old}(a_{t+1}, s_{t+1})$
- value $\mathcal{L}_t^V(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} A_{\eta}^{t,i}(a_{t+1}, s_{t+1})^2$
- exploration $\mathcal{L}_t^H(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} H(\pi_{\rho}(.|s_{t+1}))$
- p, q, fit - ELBO $\mathcal{L}_t^{ELBO}(\theta, \phi) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \left(\frac{1}{K} \sum_{i=1}^{K} w_{t+i}^k \right)$

Learning DVRL

Advantage function

$$A_\eta^{t,i}(a_{t+1}, s_{t+1}) = \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{r+i+j} + \gamma^{n_s-i} V_\eta^{old}(s_{t+n_s}) - V_\eta(s_{t+i}) \right)$$

Loss function

- policy ρ: $\mathcal{L}_t^A(\rho) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log\pi_\rho(a_{t+1}|s_{t+1}) A_\eta^{t,i,old}(a_{t+1}, s_{t+1})$
- value $\mathcal{L}_t^V(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} A_\eta^{t,i}(a_{t+1}, s_{t+1})^2$
- exploration $\mathcal{L}_t^H(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} H(\pi_\rho(.|s_{t+1}))$
- p, q, fit - ELBO $\mathcal{L}_t^{ELBO}(\theta, \phi) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log\left(\frac{1}{K} \sum_{i=1}^{K} w_{t+i}^k\right)$

Learning DVRL

Advantage function

$$A_\eta^{t,i}(a_{t+1}, s_{t+1}) = \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{r+i+j} + \gamma^{n_s-i} V_\eta^{old}(s_{t+n_s}) - V_\eta(s_{t+i}) \right)$$

Loss function

- policy ρ: $\mathcal{L}_t^A(\rho) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \pi_\rho(a_{t+1}|s_{t+1}) A_\eta^{t,i,old}(a_{t+1}, s_{t+1})$
- value $\mathcal{L}_t^V(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} A_\eta^{t,i}(a_{t+1}, s_{t+1})^2$
- exploration $\mathcal{L}_t^H(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} H(\pi_\rho(.|s_{t+1}))$
- p, q, fit - ELBO $\mathcal{L}_t^{ELBO}(\theta, \phi) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \left(\frac{1}{K} \sum_{i=1}^{K} w_{t+i}^k \right)$

Learning DVRL

Advantage function

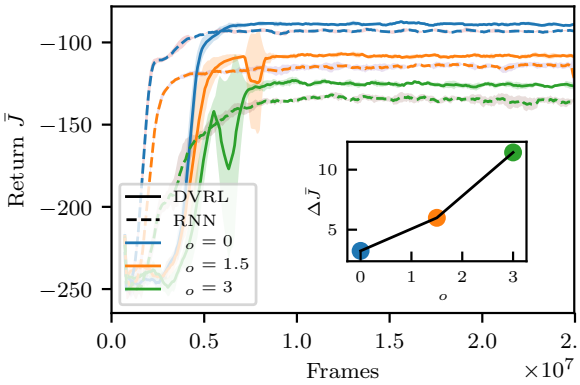
$$A_\eta^{t,i}(a_{t+1}, s_{t+1}) = \left(\sum_{j=0}^{n_s-i-1} \gamma^j r_{r+i+j} + \gamma^{n_s-i} V_\eta^{old}(s_{t+n_s}) - V_\eta(s_{t+i}) \right)$$

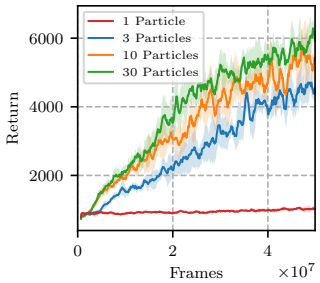
Loss function

- policy ρ: $\mathcal{L}_t^A(\rho) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \pi_\rho(a_{t+1}|s_{t+1}) A_\eta^{t,i,old}(a_{t+1}, s_{t+1})$
- value $\mathcal{L}_t^V(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} A_\eta^{t,i}(a_{t+1}, s_{t+1})^2$
- exploration $\mathcal{L}_t^H(\eta) = \frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} H(\pi_\rho(.|s_{t+1}))$
- p, q, fit - ELBO $\mathcal{L}_t^{ELBO}(\theta, \phi) = -\frac{1}{n_e n_s} \sum_{envs}^{n_e} \sum_{i=0}^{n_s-1} log \left(\frac{1}{K} \sum_{i=1}^{K} w_{t+i}^k \right)$

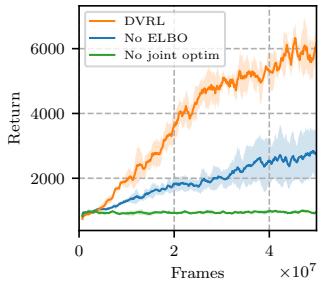
Experiments

- The average return is slightly better for DVRL
- One particle is not enough
- Backpropagation is necessary, ELBO improves the result
- memory length improves the result
- with minimal memory $n_g = 5$ DVRL works, RNN does not.

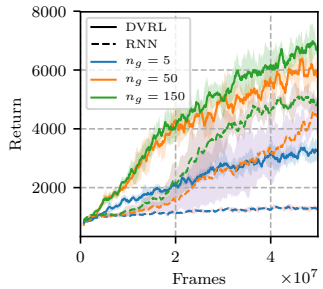




(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.



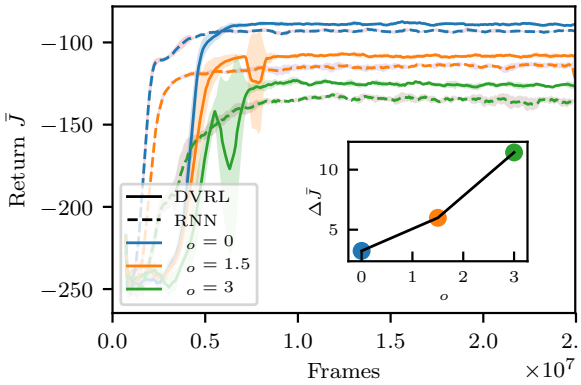
(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").

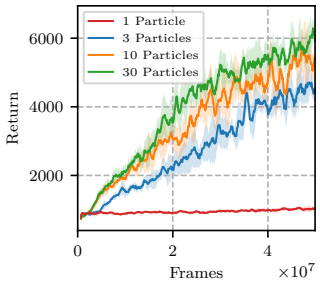


(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

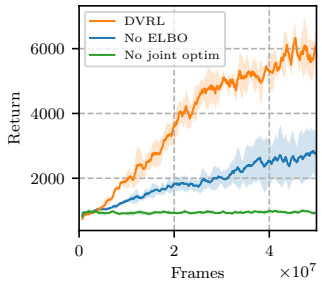
Experiments

- The average return is slightly better for DVRL
- One particle is not enough
- Backpropagation is necessary, ELBO improves the result
- memory length improves the result
- with minimal memory $n_g = 5$ DVRL works, RNN does not.

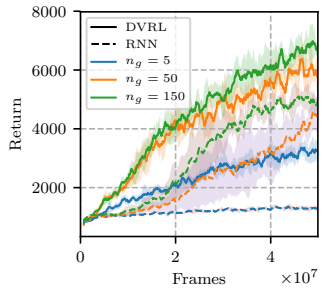




(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.



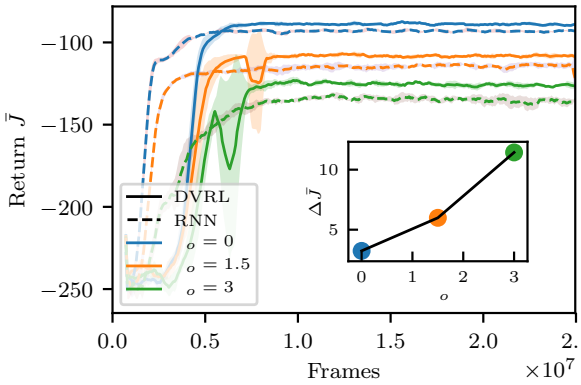
(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").

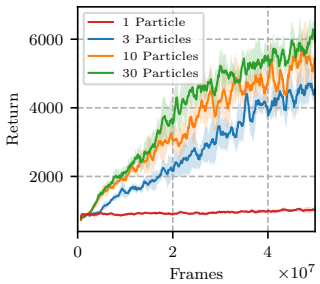


(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

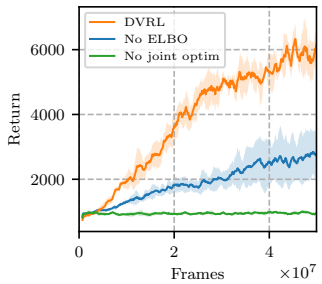
Experiments

- The average return is slightly better for DVRL
- One particle is not enough
- Backpropagation is necessary, ELBO improves the result
- memory length improves the result
- with minimal memory $n_g = 5$ DVRL works, RNN does not.

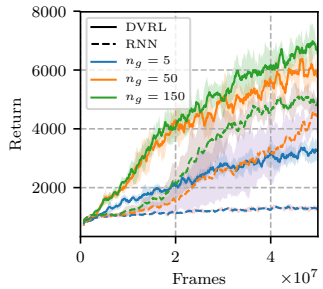




(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.



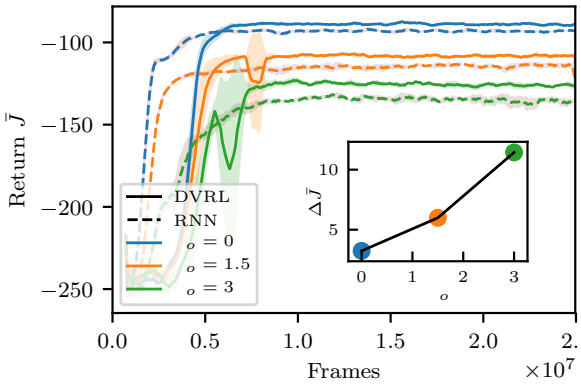
(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").

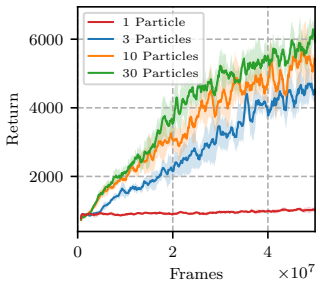


(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

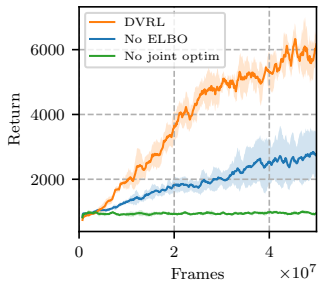
Experiments

- The average return is slightly better for DVRL
- One particle is not enough
- Backpropagation is necessary, ELBO improves the result
- memory length improves the result
- with minimal memory $n_g = 5$ DVRL works, RNN does not.

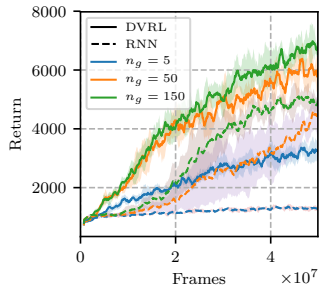




(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.



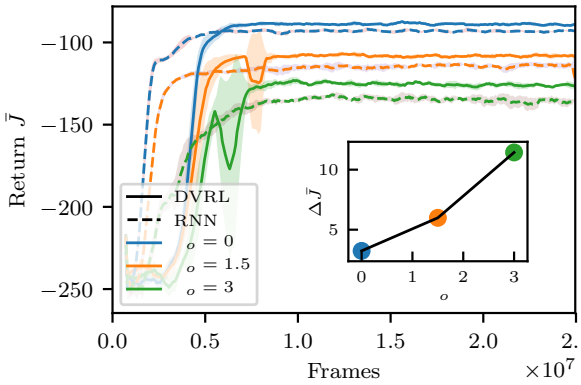
(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").

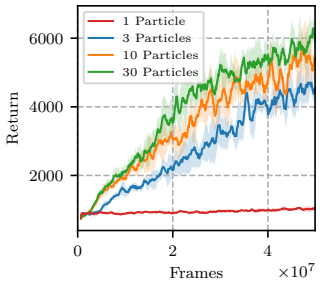


(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

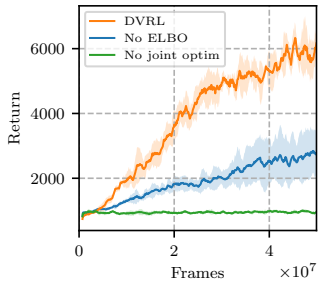
Experiments

- The average return is slightly better for DVRL
- One particle is not enough
- Backpropagation is necessary, ELBO improves the result
- memory length improves the result
- with minimal memory $n_g = 5$ DVRL works, RNN does not.

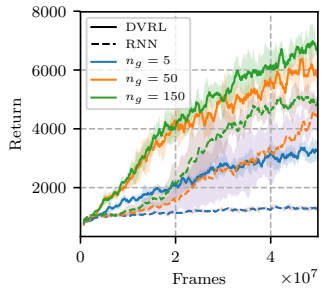




(a) Influence of the particle number on performance for DVRL. Only using one particle is not sufficient to encode enough information in the latent state.



(b) Performance of the full DVRL algorithm compared to setting $\lambda^E = 0$ ("No ELBO") or not backpropagating the policy gradients through the encoder ("No joint optim").



(c) Influence of the maximum backpropagation length n_g on performance. Note that RNN suffers most from very short lengths. This is consistent with our conjecture that RNN relies mostly on memory, not inference.

Next time

- Bring you computer (if possible)
- Check moodle for libraries to install
  - Python 3
  - pgmpy, numpy, matplotlib
  - networkx
  - (graphviz, sklearn)
  - may be others.

Have a nice holiday.

Next time

- Bring you computer (if possible)
- Check moodle for libraries to install
  - Python 3
  - pgmpy, numpy, matplotlib
  - networkx
  - (graphviz, sklearn)
  - may be others.

Have a nice holiday.

POMDP Applications

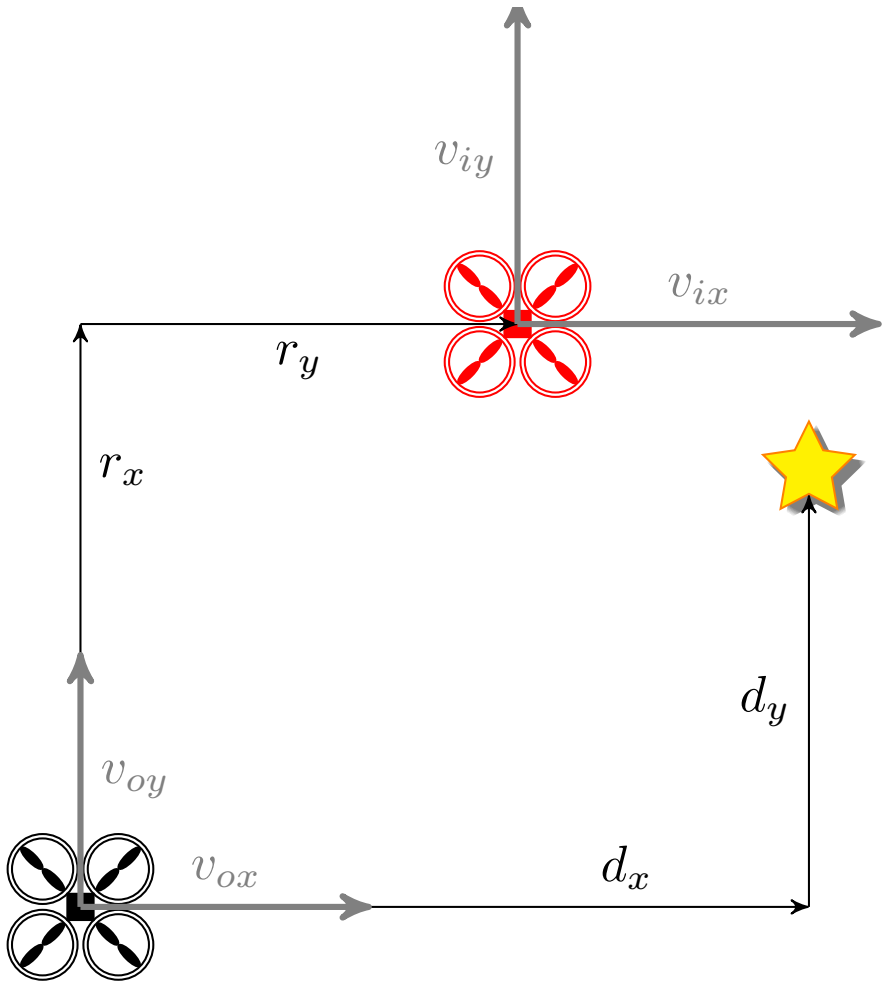
- Karkus, Hsu, Lee: **QMDP-Net: Deep Learning for Planning under Partial Observability**
  https://proceedings.neurips.cc/paper/2017/file/e9412ee564384b987d086df32d4Paper.pdf
- Eric Mueller and Mykel J. Kochenderfer :**Multi-Rotor Aircraft Collision Avoidance using Partially Observable Markov Decision Processes**, American Institute of Aeronautics and Astronautics

POMDP Applications

- Karkus, Hsu, Lee: **QMDP-Net: Deep Learning for Planning under Partial Observability**
  https://proceedings.neurips.cc/paper/2017/file/e9412ee564384b987d086df32d4Paper.pdf
- Eric Mueller and Mykel J. Kochenderfer :**Multi-Rotor Aircraft Collision Avoidance using Partially Observable Markov Decision Processes**, American Institute of Aeronautics and Astronautics

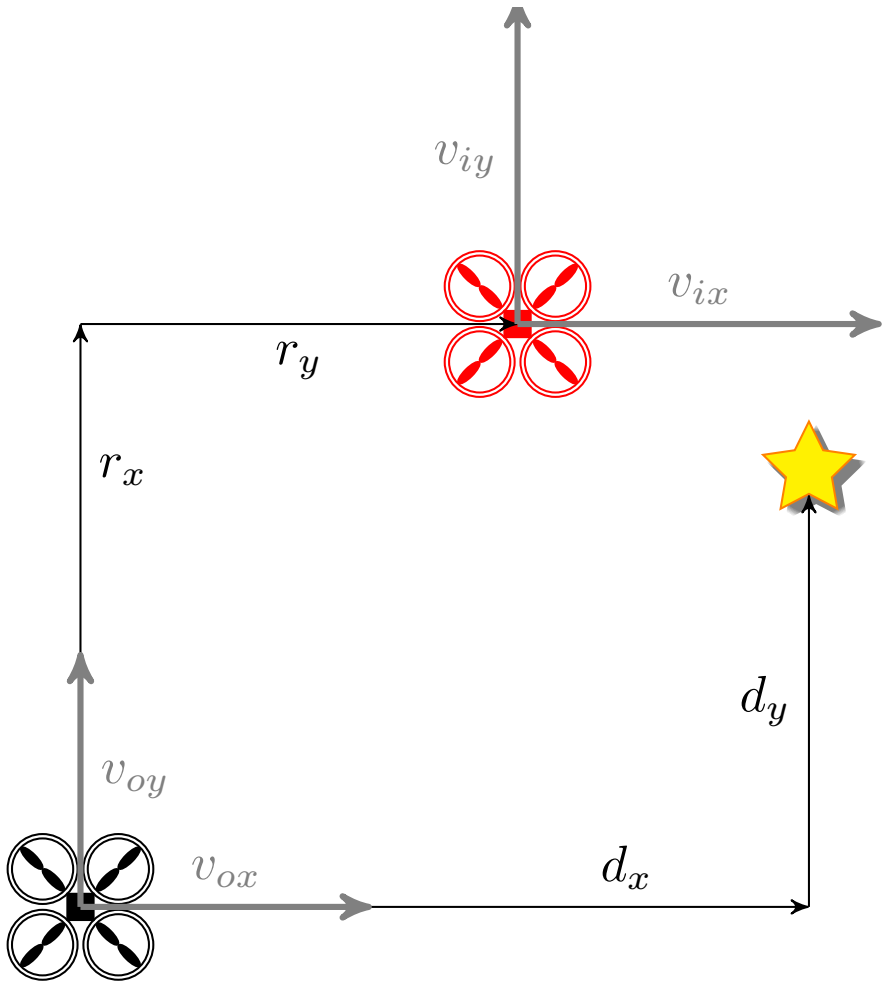
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the x axis and zero velocity in the y axis



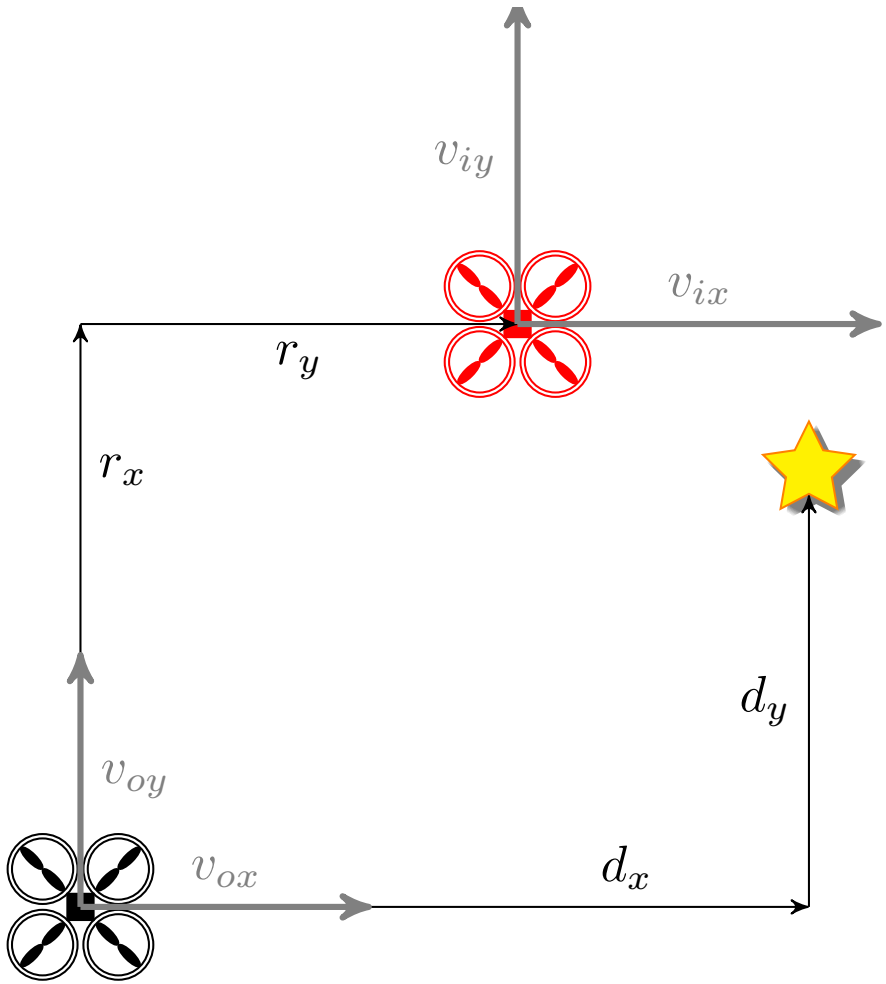
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the x axis and zero velocity in the y axis



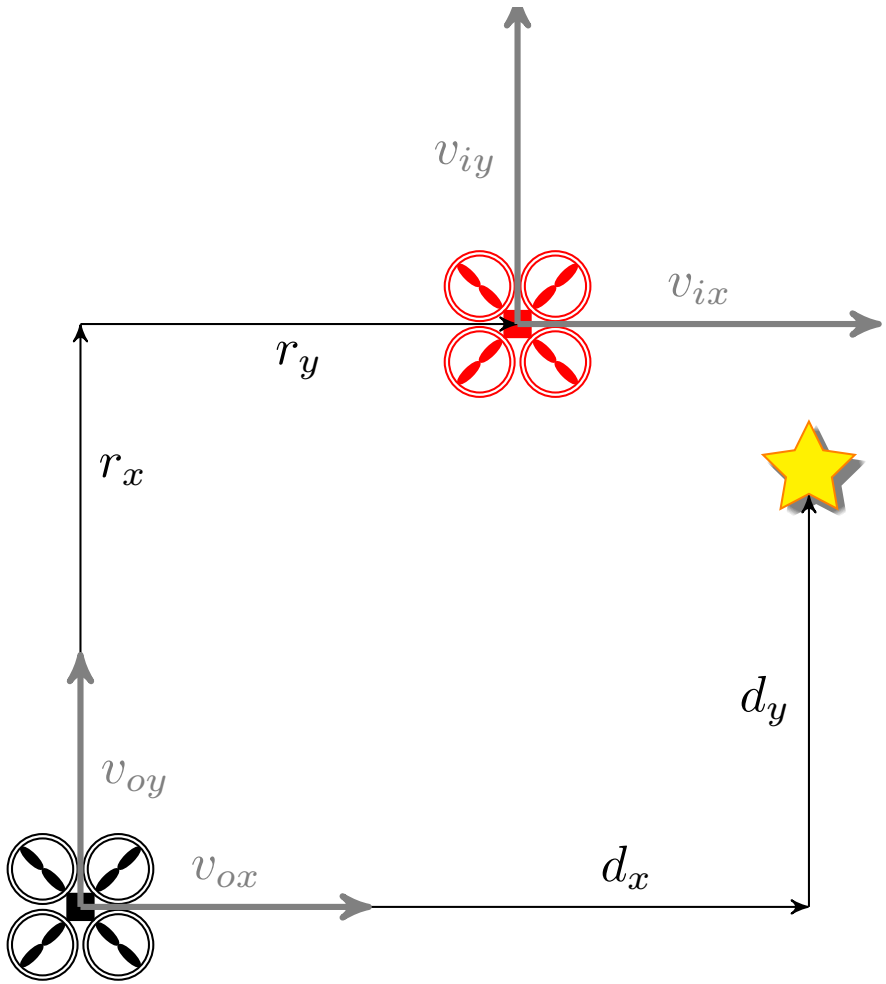
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the x-axis and zero velocity in the y-axis



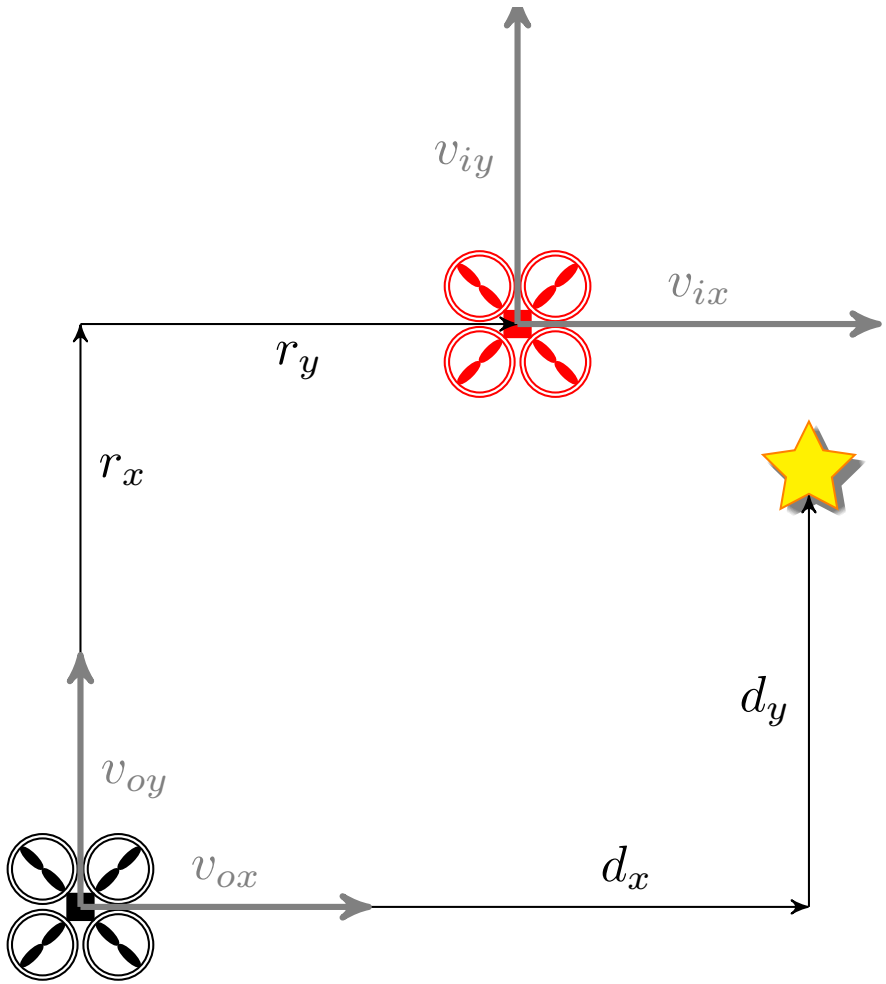
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ix}, v_{iy}, (v_{iz})
  - velocities for the intruder v_{ox}, v_{oy}, (v_{oz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the x-axis and zero velocity in the y-axis



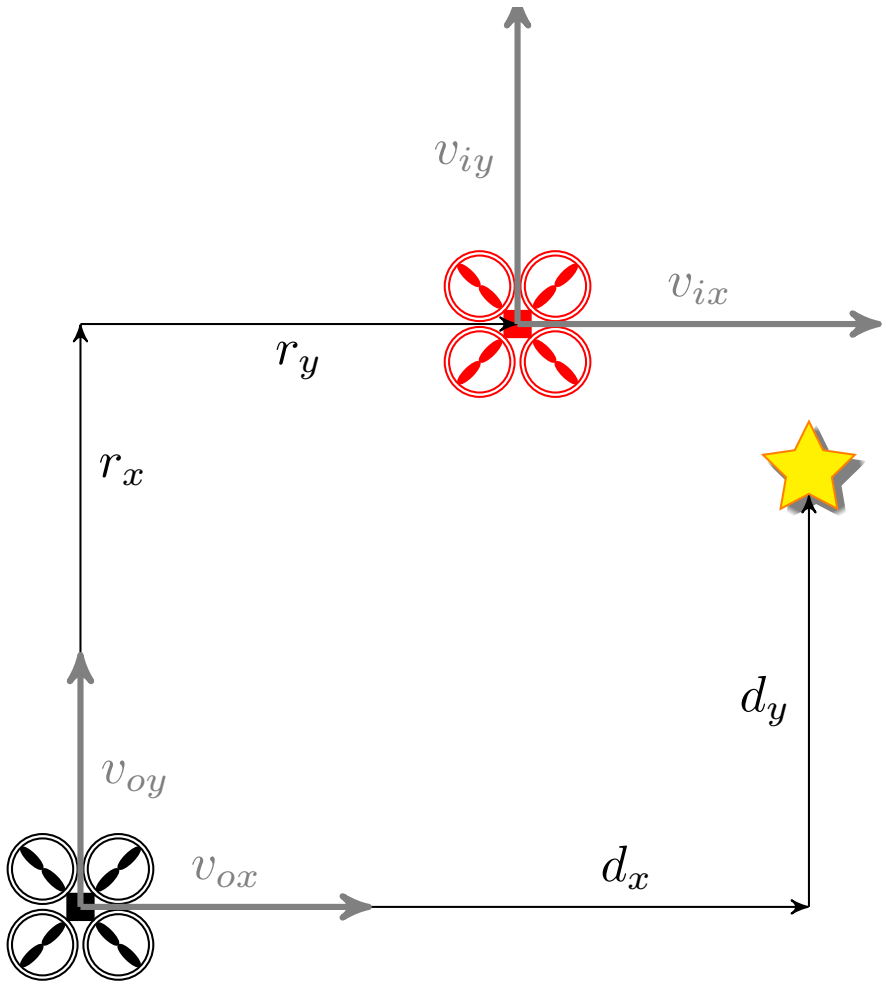
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the x axis and zero velocity in the y axis



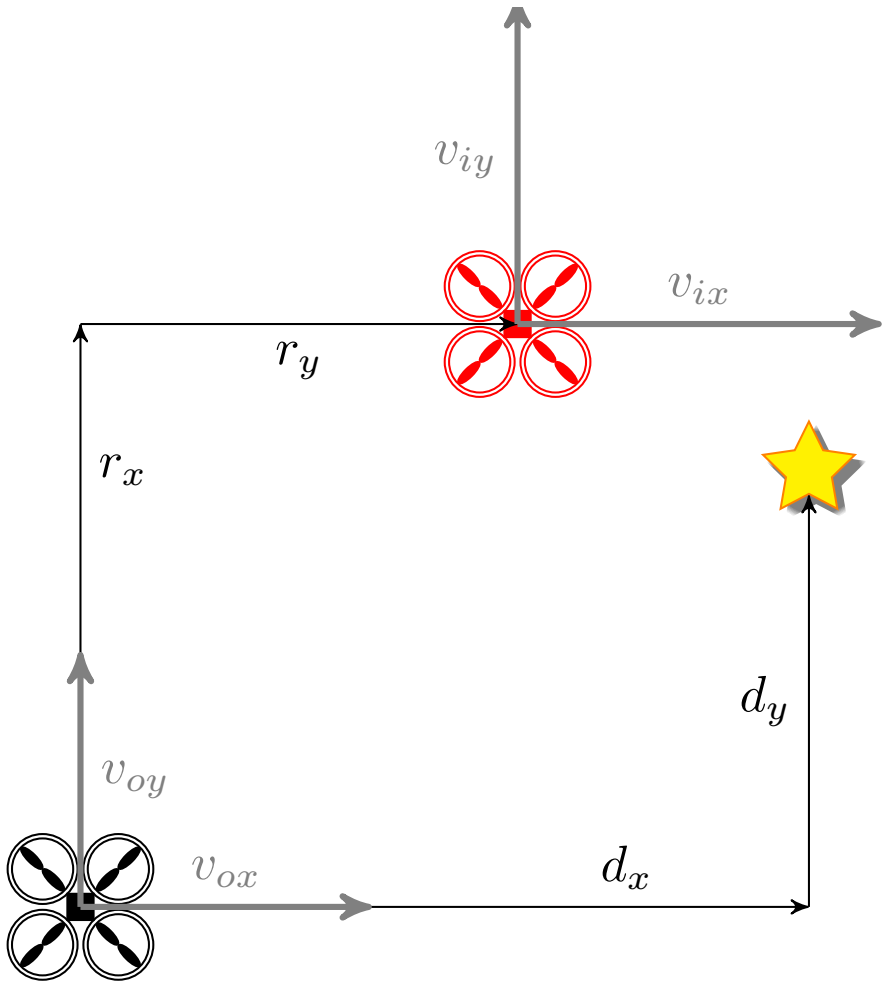
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



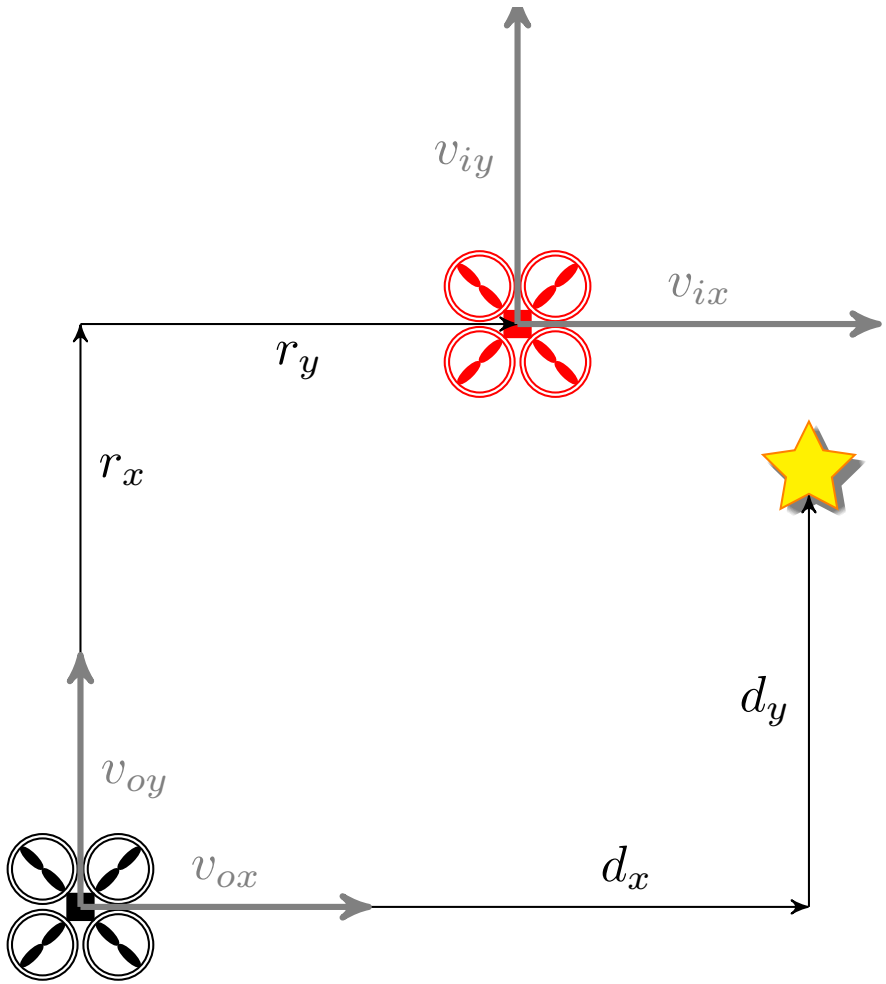
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



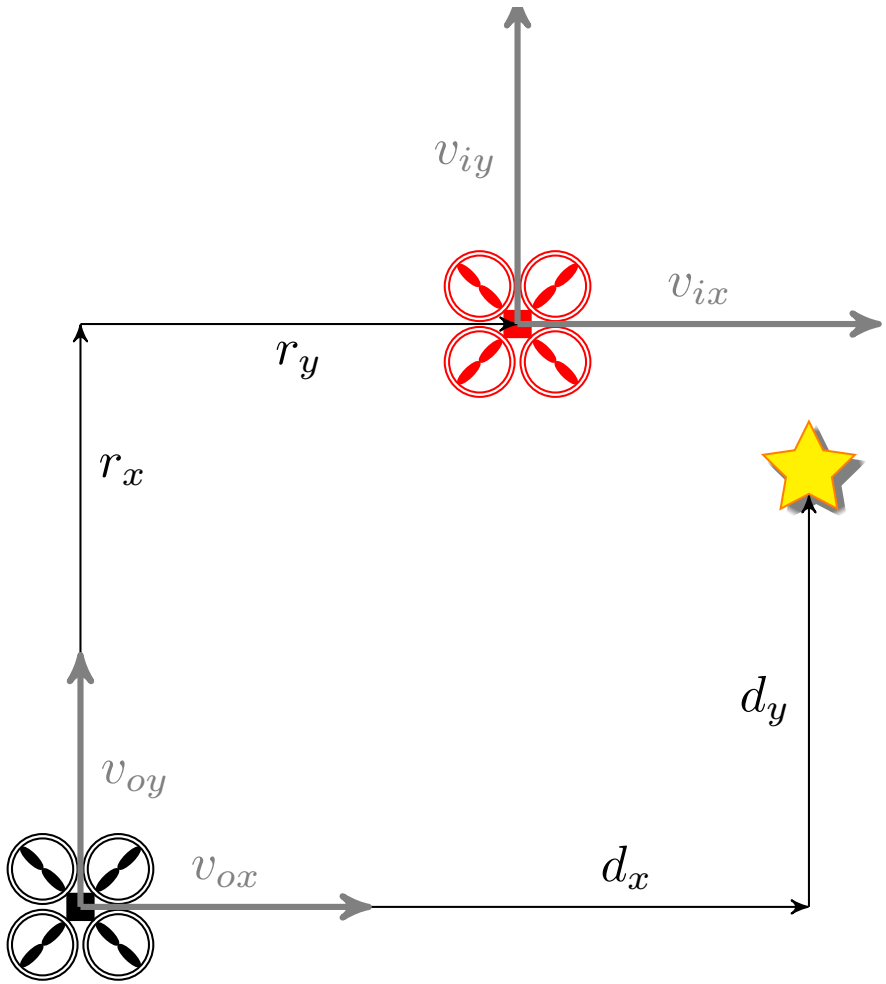
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



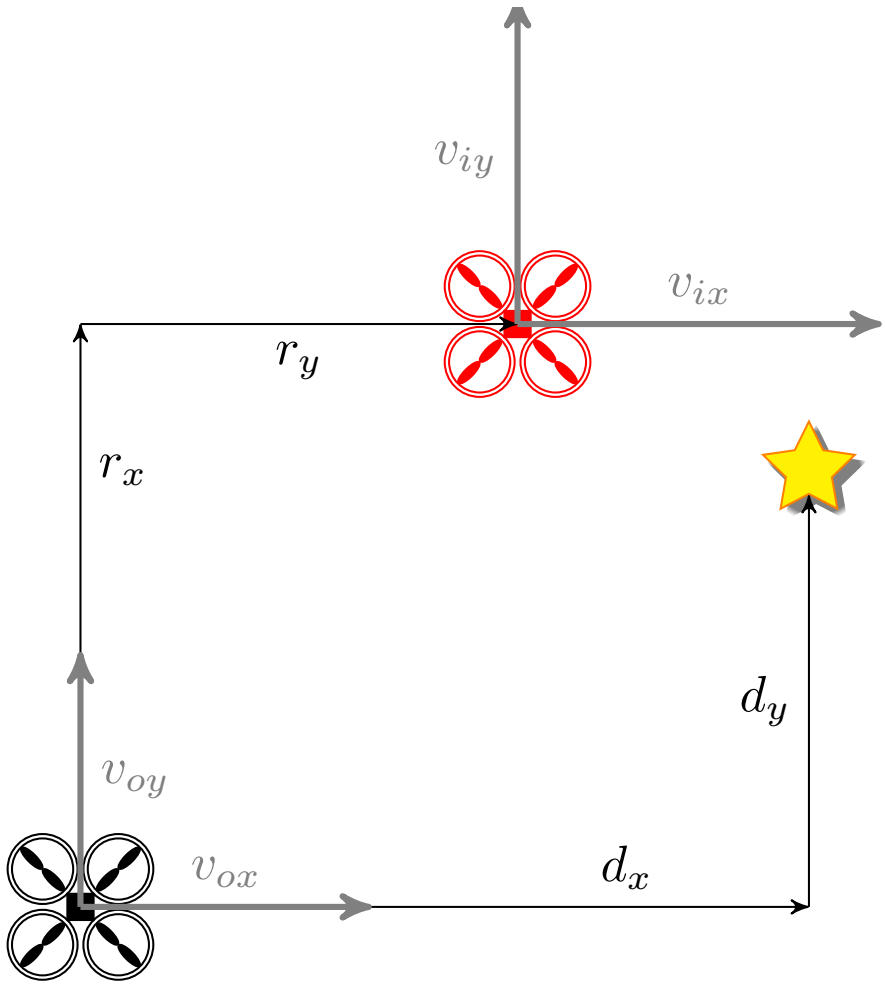
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



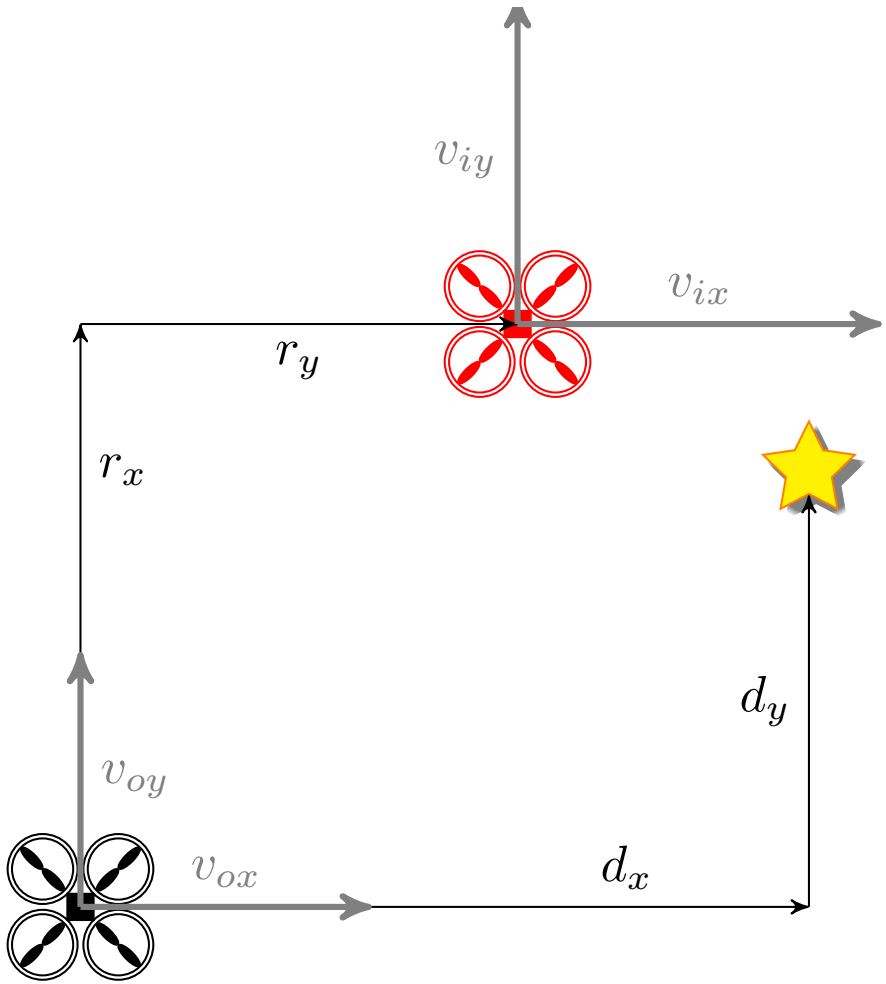
POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



POMDP Aircraft Collision Avoidance

- the algorithms designed for fixed-wing aircraft analyze
  - turns
  - vertical meneuvers
- multirotor aircraft (drones) and helicopters can also
  - horizontal plane accelerations
- state 2D, (3D)
  - relative range states r_x, r_y, (r_z)
  - velocities for the ownship v_{ox}, v_{oy}, (v_{oz})
  - velocities for the intruder v_{ix}, v_{iy}, (v_{iz})
  - absolute displacement from the desired trajectory d_x, d_y, (d_z)
  - the desired trajectory is normalized to unit velocity in the $x$axis and zero velocity in the y axis.



MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  $N_* \sim \mathcal{N}(\mu = 0, \sigma = 3)$, $N_* \in [-9, 9]$
- Bellman update
  transition from s with acceleration a to s'

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s'} T(s'|s,a) max_{a'} Q[s',a'].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  $N_* \sim \mathcal{N}(\mu = 0, \sigma^2 = 3), N[\mu = 0, \sigma^2 = 3]$
- Bellman update
  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  $N_* \sim \mathcal{N}(\mu = 0, \sigma^2 = 0.025^2)$, $N_*[\mu = 0, \sigma^2]$
- Bellman update
  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  $N_* \sim \mathcal{N}(\mu = 0, \sigma^2)$, $N[\mu = 0, \sigma^2]$
- Bellman update
  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  - $N_o(\mu = 0, 0.30s^{-2})$, $N_i(\mu = 0, 0.45s^{-2})$,
- Bellman update

  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  - $N_o(\mu = 0, 0.30s^{-2})$, $N_i(\mu = 0, 0.45s^{-2})$,
- Bellman update

  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  - $N_o(\mu = 0, 0.30s^{-2})$, $N_i(\mu = 0, 0.45s^{-2})$,
- Bellman update

  transition from s with acceleration a to $s^{|}$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^{|}} T(s^{|}|s,a) max_{a^{|}} Q[s^{|}, a^{|}].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  - $N_o(\mu = 0, 0.30s^{-2})$, $N_i(\mu = 0, 0.45s^{-2})$,
- Bellman update

transition from s with acceleration a to $s^|$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^|} T(s^||s,a) max_{a^|} Q[s^|, a^|].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

MDP Transitions

- the prediction horizon is very short
- updates done every 0.1 to 1 seconds
- simple update equation are sufficient
- not a benefit to using more complex dynamic equations.
- a_x, a_y acceleration by the ownship
- N_* noise to the ownship, intruder, x and y axis
  - $N_o(\mu = 0, 0.30s^{-2})$, $N_i(\mu = 0, 0.45s^{-2})$,
- Bellman update

  transition from s with acceleration a to $s^|$

$$Q[s,a] \leftarrow R(s,a) + \gamma \sum_{s^|} T(s^||s,a) max_{a^|} Q[s^|, a^|].$$

$$\begin{aligned}
\dot{r}_x &= v_{ix} - v_{ox} \\
\dot{r}_y &= v_{iy} - v_{oy} \\
\dot{v}_{ox} &= a_x + N_{ox} \\
\dot{v}_{oy} &= a_y + N_{oy} \\
\dot{v}_{ix} &= N_{ix} \\
\dot{v}_{iy} &= N_{iy} \\
\dot{d}_x &= v_{tx} - v_{ox} \\
\dot{d}_y &= v_{ty} - v_{oy}
\end{aligned}$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

Reward

- Minimum reward R_{min}
  - collision
  - physically impossible states
  - keeps the sum finite
- we prefer no acceleration
- we prefer long distance to the intruder
- we prefer short distance to the desired trajectory
- K_s, K_T, R_{min} weights was learned, k weights was $= 1$.

$$R(s,a) = max\left[R_{min}, -(k_{ax}|a_x| + k_{ay}|a_y|) - K_s\frac{1}{k_{rx}r_x^2 + k_{ry}r_y^2} - K_T(k_{dx}d_x^2 + k_{dy}d_y^2)\right]$$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 7056 discrete states
  - the finely discretized version contained 9 520 800 states

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 7056 discrete states
  - the finely discretized version contained 9 520 800 states

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 7,056,625 discrete states
  - the finely discretized version contained 9,529,569 states

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contains a total of 7 65 625 discrete states
  - the finely discretized version contained 9 520 760 states

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 765,625 discrete states
  - the finely discretized version contained 9,570,160 states

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 765,625 discrete states
  - the finely discretized version contained 9,529,569 states.

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 765,625 discrete states
  - the finely discretized version contained 9,529,569 states.

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

QMDP Approximation

- offline optimization
  - a few hours for coarse discretization, 1 PC
  - initially stationary intruders
  - intruders moving at uniform velocity with a variety of relative headings angles
  - intruders state and dynamic uncertainty were added to the encounters.
- all values normalized
  - the coarse set contained a total of 765,625 discrete states
  - the finely discretized version contained 9,529,569 states.

State variable	State Description	Discretization
r_x, r_y	Intruder range components	$-15, [-7, -3], -1, 0, 1, [3, 7], 15$
v_{ox}, v_{oy}	ownship velocity components	$-5, -3, -1, 0, 1, 3, 5\ s^{-1}$
v_{ix}, v_{iy}	intruder velocity components	$-5, [-3], -1, 0, 1, [3], 5\ s^{-1}$
d_x, d_y	desired trajectory distance	$-10, [-3], -1, 0, 1, [3], 10$

Evaluation Function

- The primary goal is to remain safely separated from the intruder aircraft.
  - $r_{5\%CPA}$ 'the closest point of approach', we allow 5% trajectories a little bit closer.
  - Figure: required 1.5 units, never closer than 1.1 units.
  - Mean deviation distance from the desired trajectory μ_{dev}.

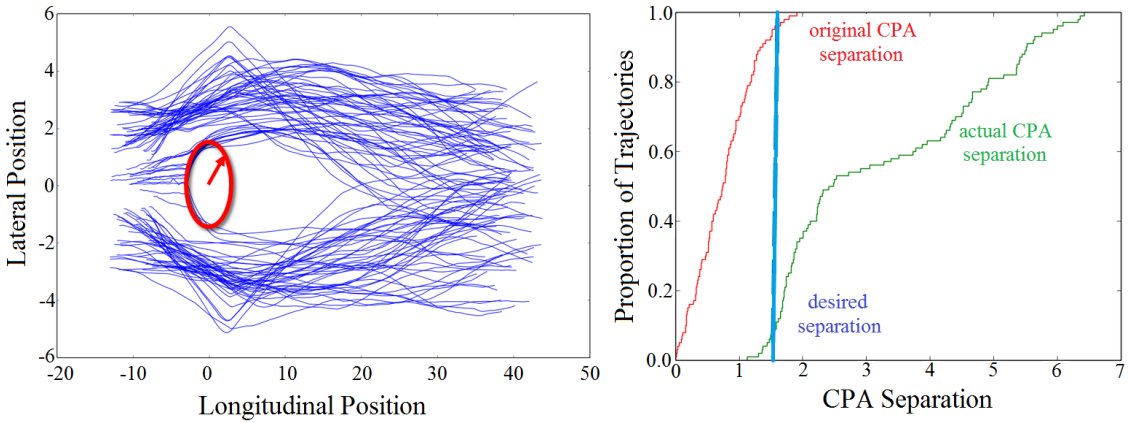
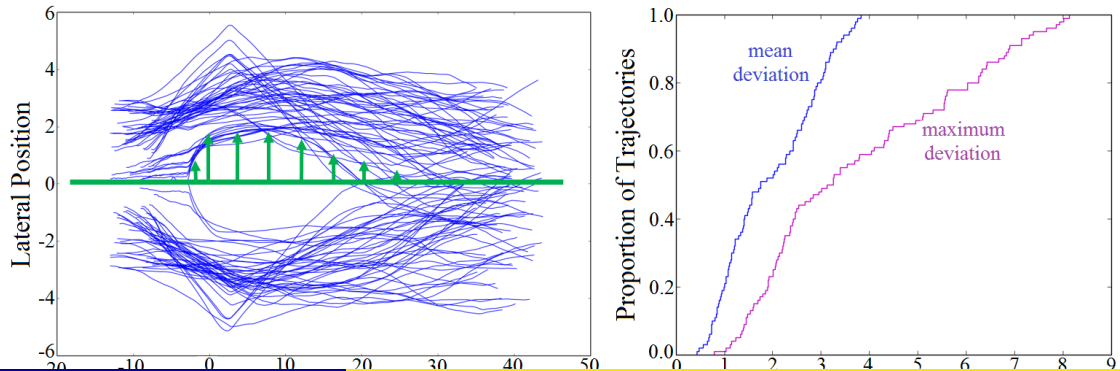


Figure 3: Separation metric used to evaluate the collision avoidance algorithm



Evaluation Function

- The primary goal is to remain safely separated from the intruder aircraft.
  - $r_{5\%CPA}$ 'the closest point of approach', we allow 5% trajectories a little bit closer.
- Figure: required 1.5 units, never closer than 1.1 units.
- Mean deviation distance from the desired trajectory μ_{dev}.

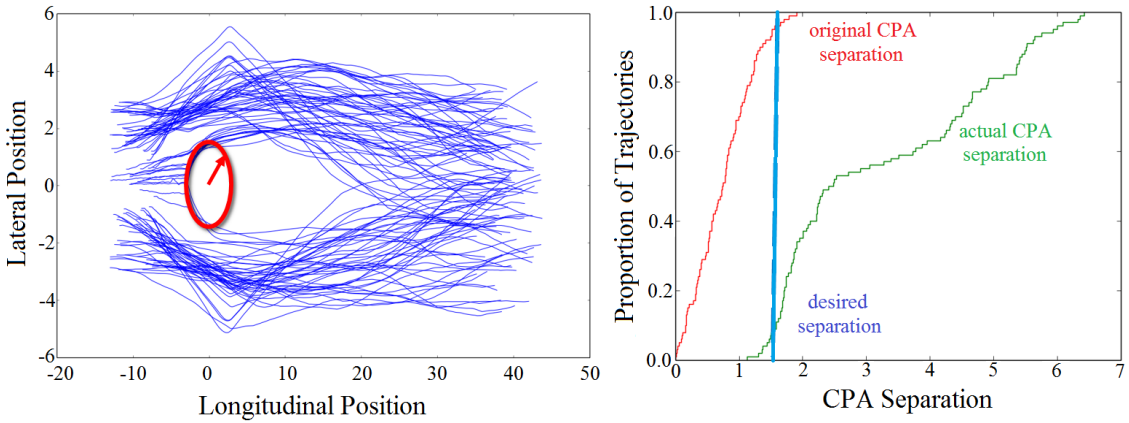
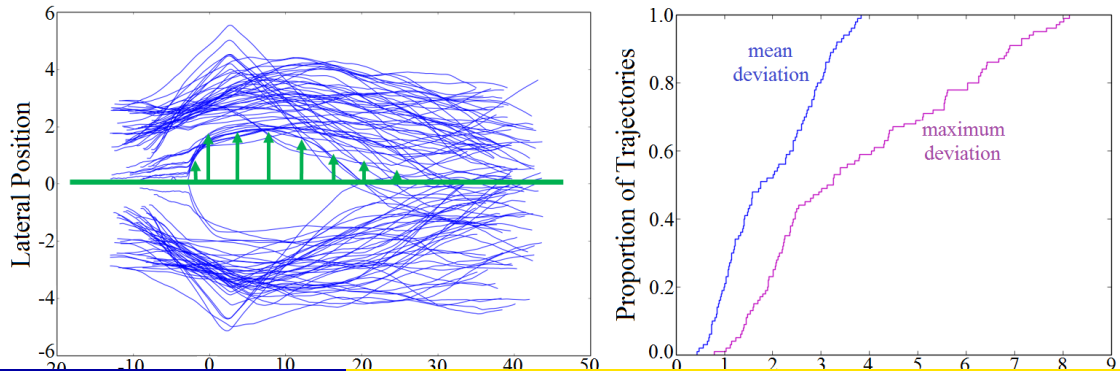


Figure 3: Separation metric used to evaluate the collision avoidance algorithm



Evaluation Function

- The primary goal is to remain safely separated from the intruder aircraft.
  - $r_{5\%CPA}$ 'the closest point of approach', we allow 5% trajectories a little bit closer.
- Figure: required 1.5 units, never closer than 1.1 units.
- Mean deviation distance from the desired trajectory μ_{dev}.

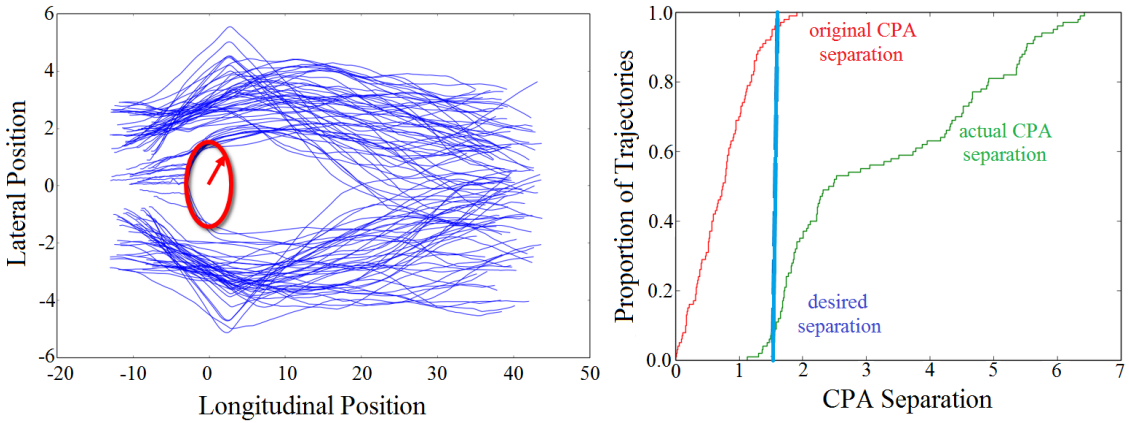
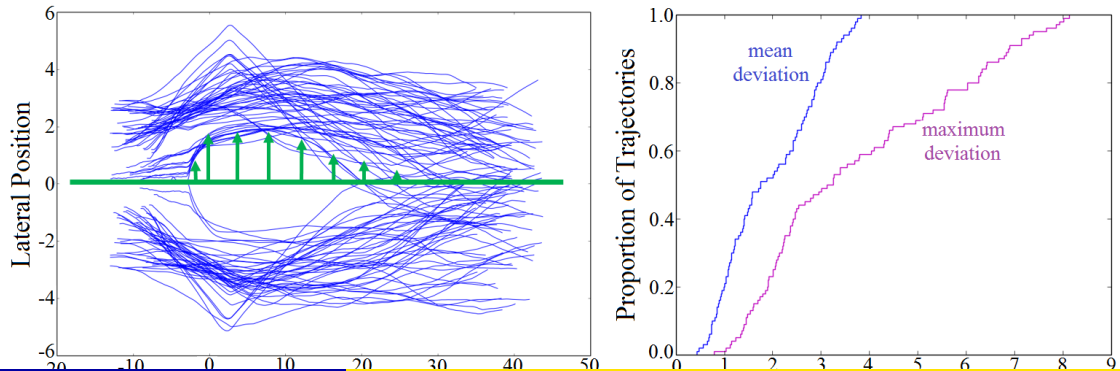


Figure 3: Separation metric used to evaluate the collision avoidance algorithm



Evaluation Function

- The primary goal is to remain safely separated from the intruder aircraft.
  - $r_{5\%CPA}$ 'the closest point of approach', we allow 5% trajectories a little bit closer.
- Figure: required 1.5 units, never closer than 1.1 units.
- Mean deviation distance from the desired trajectory μ_{dev}.

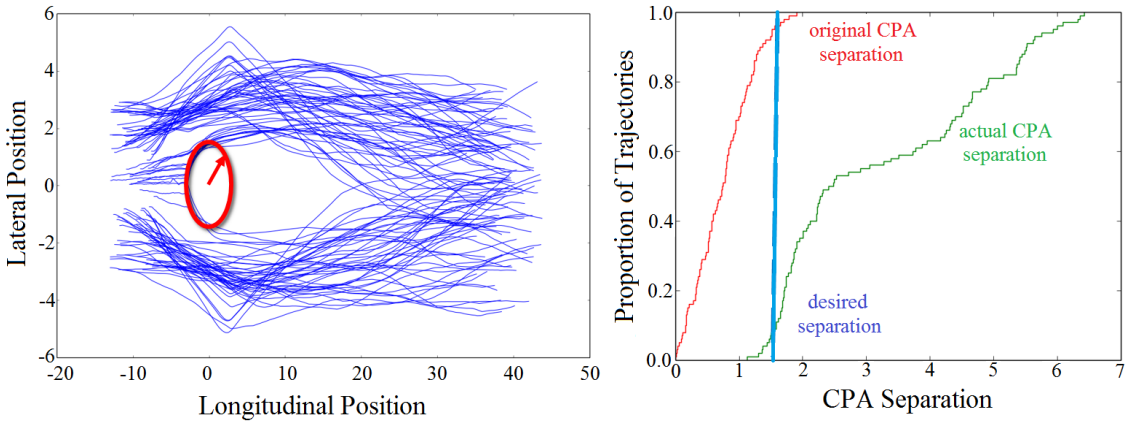
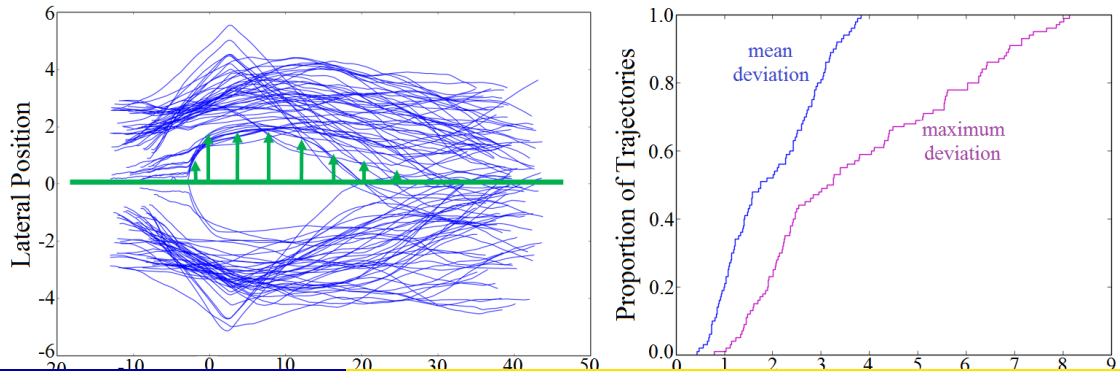


Figure 3: Separation metric used to evaluate the collision avoidance algorithm



Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

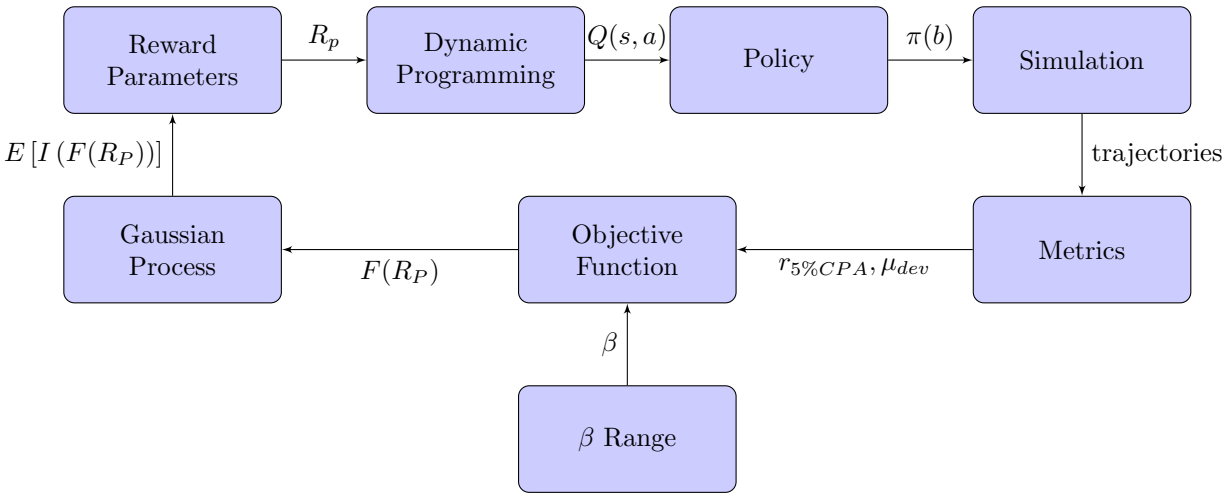


Figure 5: Process for tuning POMDP reward parameters

Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

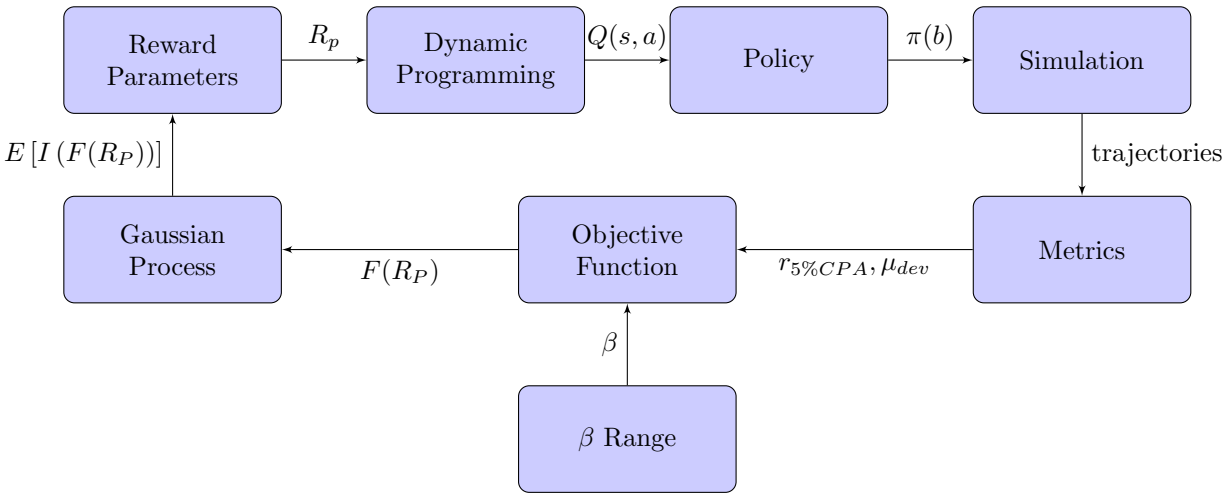


Figure 5: Process for tuning POMDP reward parameters

Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

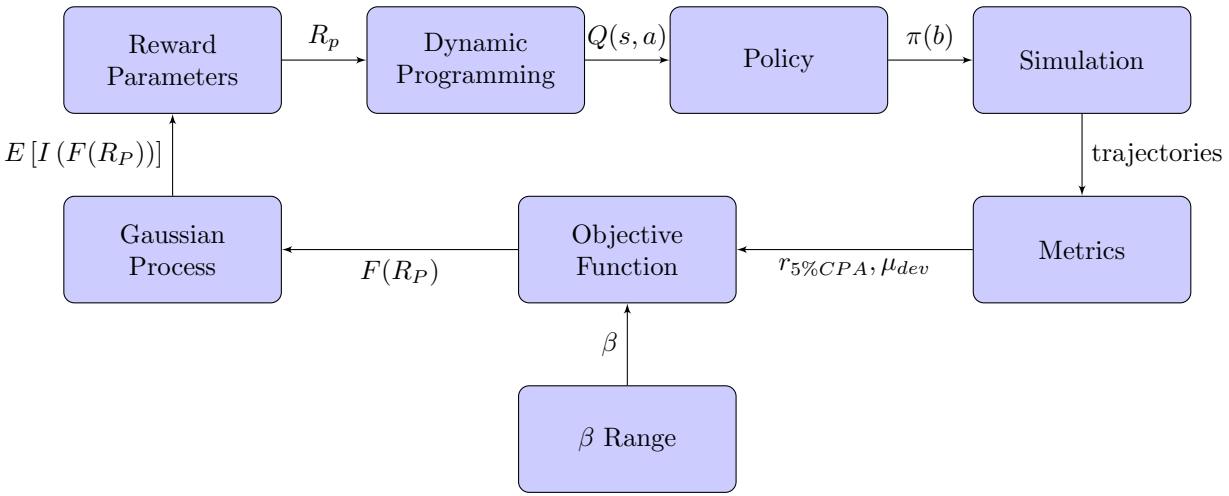


Figure 5: Process for tuning POMDP reward parameters

Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

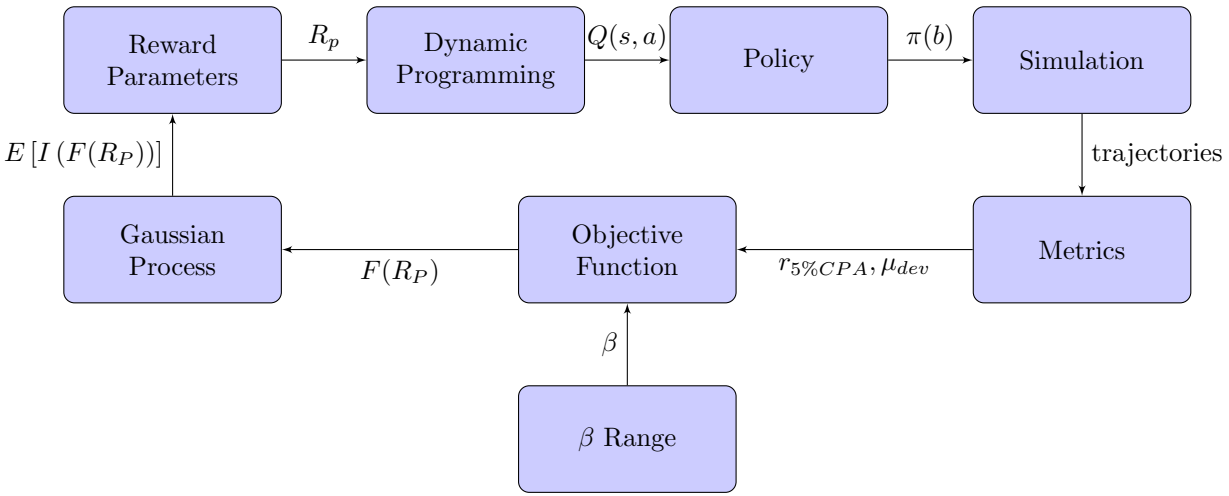


Figure 5: Process for tuning POMDP reward parameters

Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

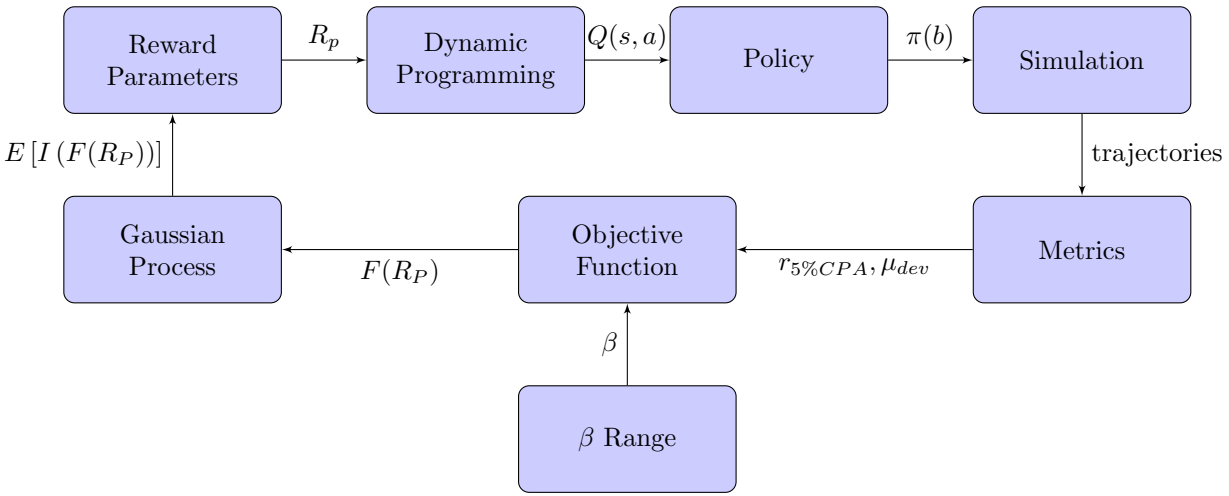


Figure 5: Process for tuning POMDP reward parameters

Reward Tuning – Bayesian Optimization

- We tune $R_P = (K_T, K_s, R_{min})$
- β weights the two objective functions

$$F(R_P) = (\beta \times (r_{5\%CPA})^{-1} + (1-\beta) \times \mu_{dev}).$$

- Gaussian process models $F(R_P)$.
- We determine the point at which the objective function is expected to have the largest improvement, $E[I(F(R_P))]$ over that of the current minimum.
- This set of R_P is passed to QMDP to evaluate.
- until convergence.

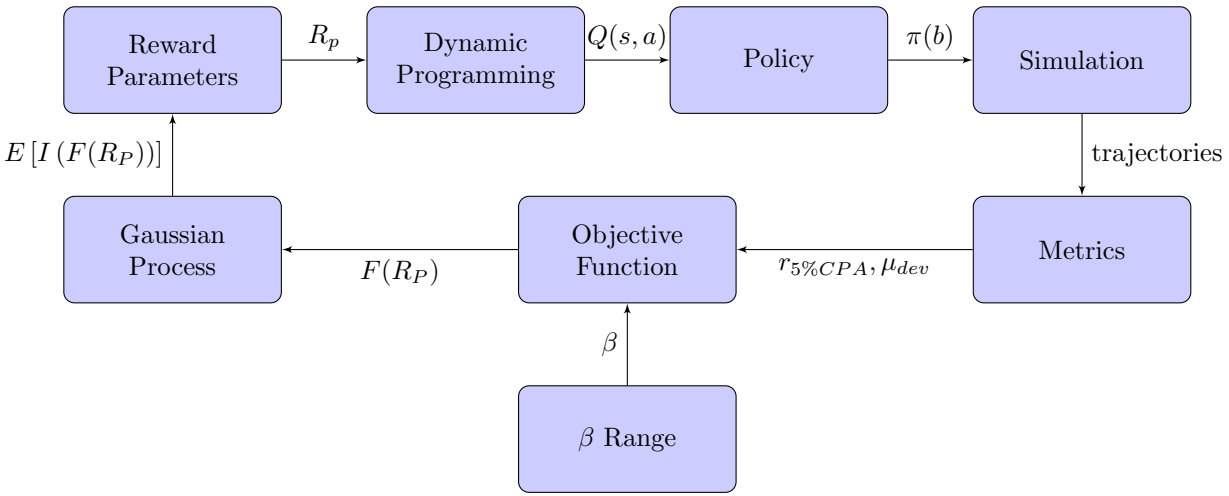
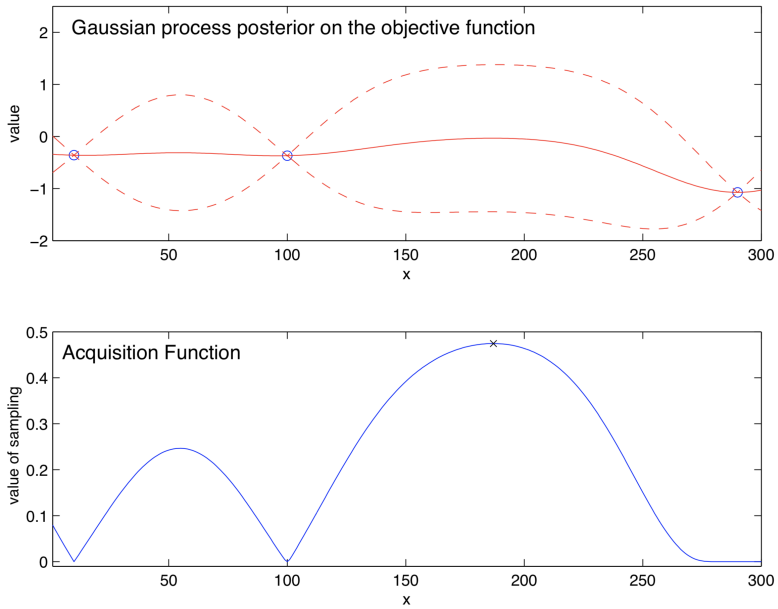


Figure 5: Process for tuning POMDP reward parameters

Bayesian Optimization

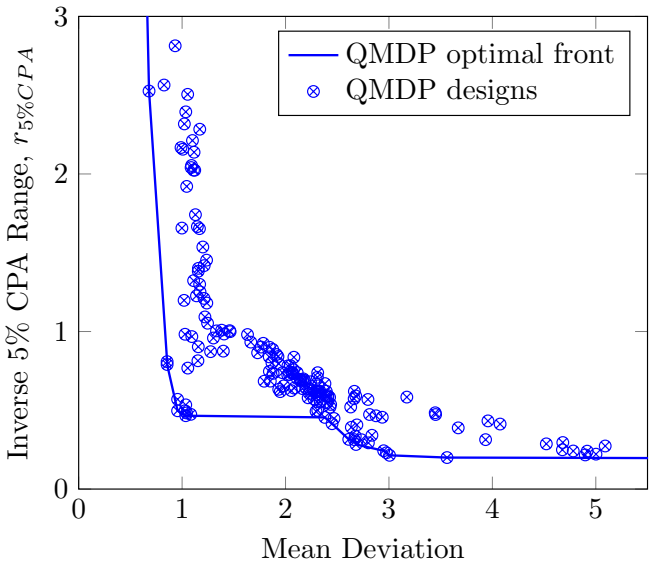
- we know QMDP and F values for one or more $\mathbf{x} = R_P$ points
- we search the point $x = R_P^*$ to observe
- we minimize $y = F(R_P^*)$ and search the maximal probability of improvement
- 'the chance to improve' is expressed by the **Expected improvement** (EI)



Peter I. Frazier: A Tutorial on Bayesian Optimization, rXiv:1807.02811v1 [stat.ML] 8 Jul 2018

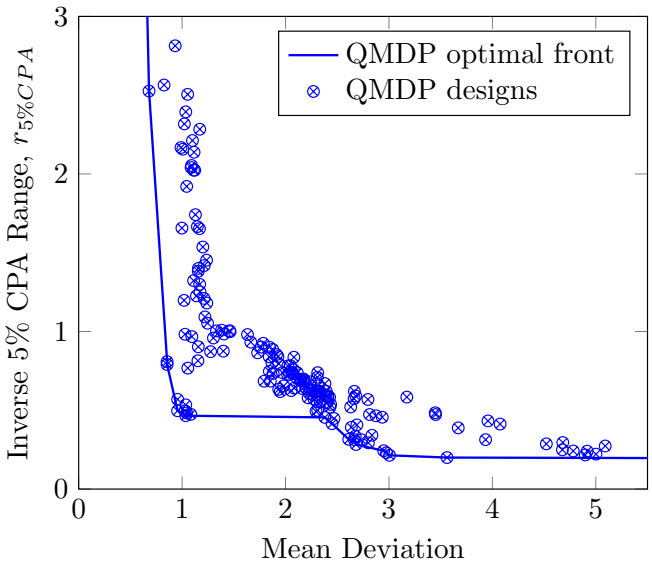
Pareto Optimal frontier

- 194 parameter sets evaluated
- β between 0.01 and 0.99 .
- resulting in nine non-dominated, Pareto–optimal designs.



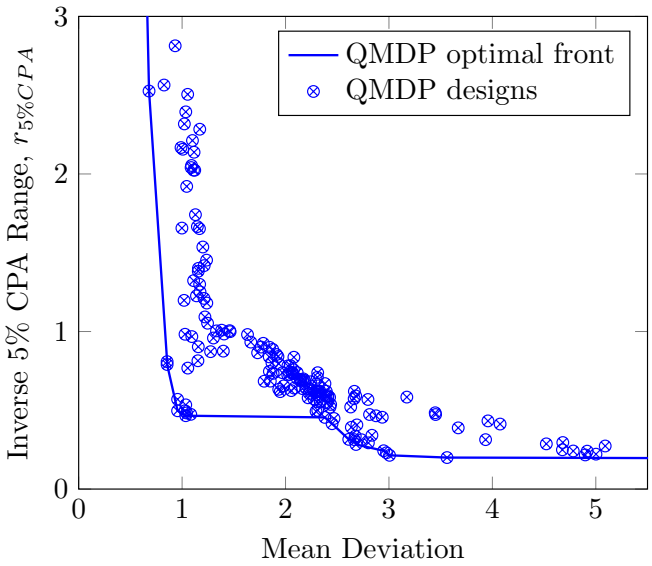
Pareto Optimal frontier

- 194 parameter sets evaluated
- β between 0.01 and 0.99 .
- resulting in nine non-dominated, Pareto–optimal designs.



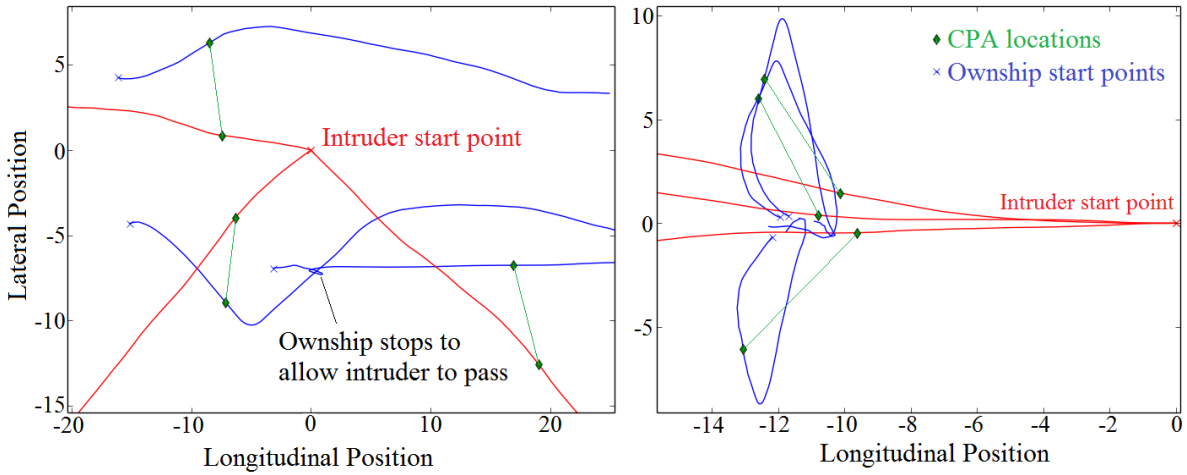
Pareto Optimal frontier

- 194 parameter sets evaluated
- β between 0.01 and 0.99 .
- resulting in nine non-dominated, Pareto–optimal designs.



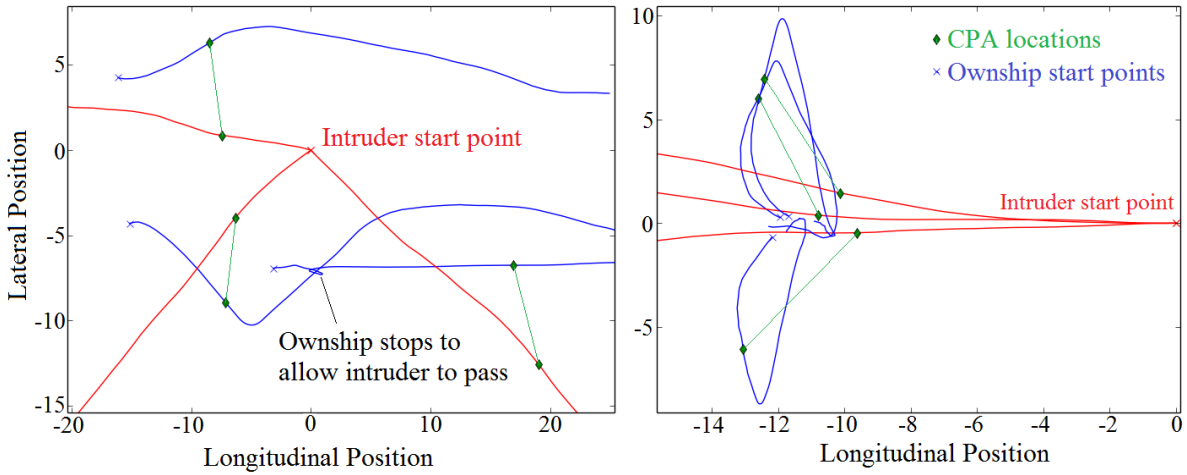
Human Expert Check

- Left: intruder starts at $(0, 0)$,
- random heading, fixed velocity of the intruder
- the ownship starts at the blue cross
- Right: The goal is hovering
- the intruder comes from the right with the unknown behaviour.



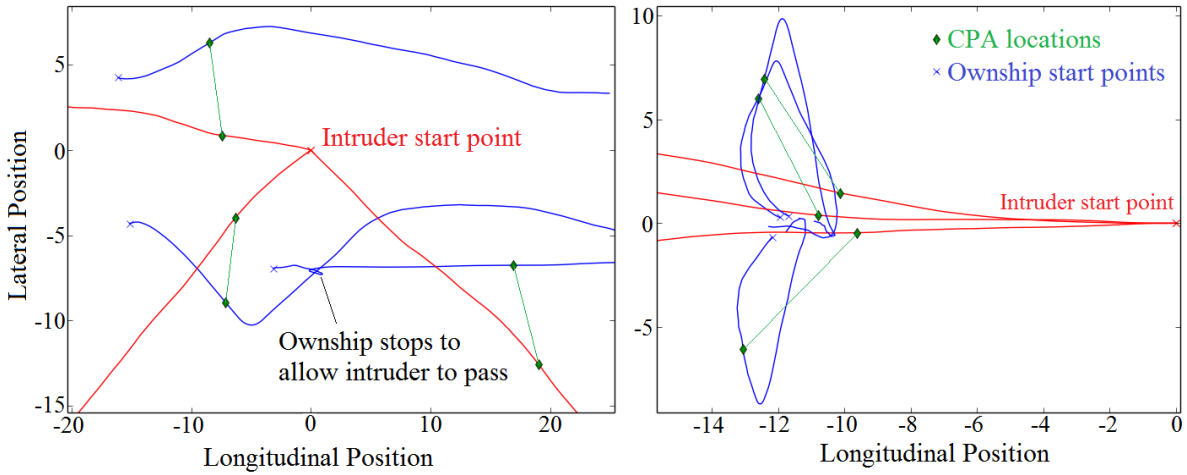
Human Expert Check

- Left: intruder starts at $(0, 0)$,
- random heading, fixed velocity of the intruder
- the ownship starts at the blue cross
- Right: The goal is hovering
- the intruder comes from the right with the unknown behaviour.



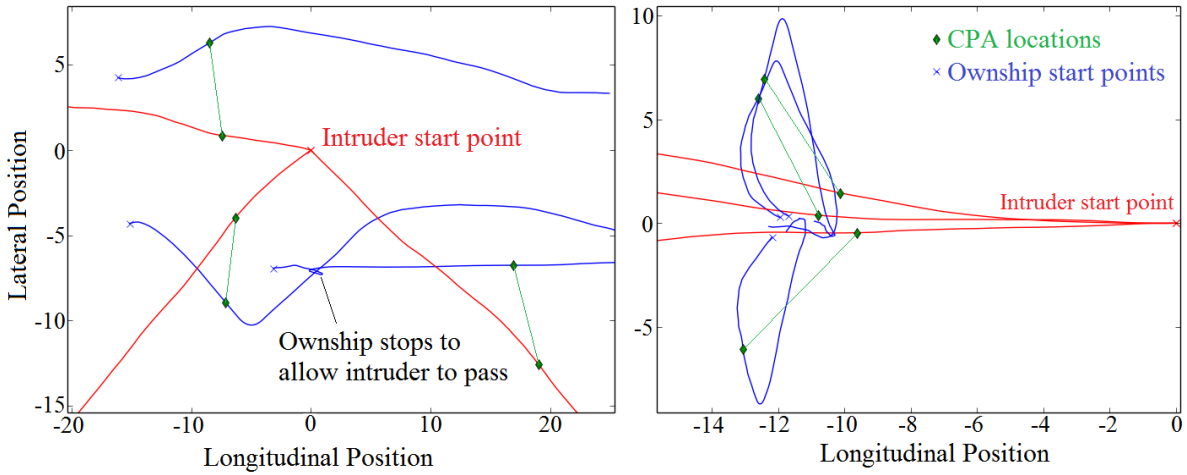
Human Expert Check

- Left: intruder starts at $(0, 0)$,
- random heading, fixed velocity of the intruder
- the ownship starts at the blue cross
- Right: The goal is hovering
- the intruder comes from the right with the unknown behaviour.



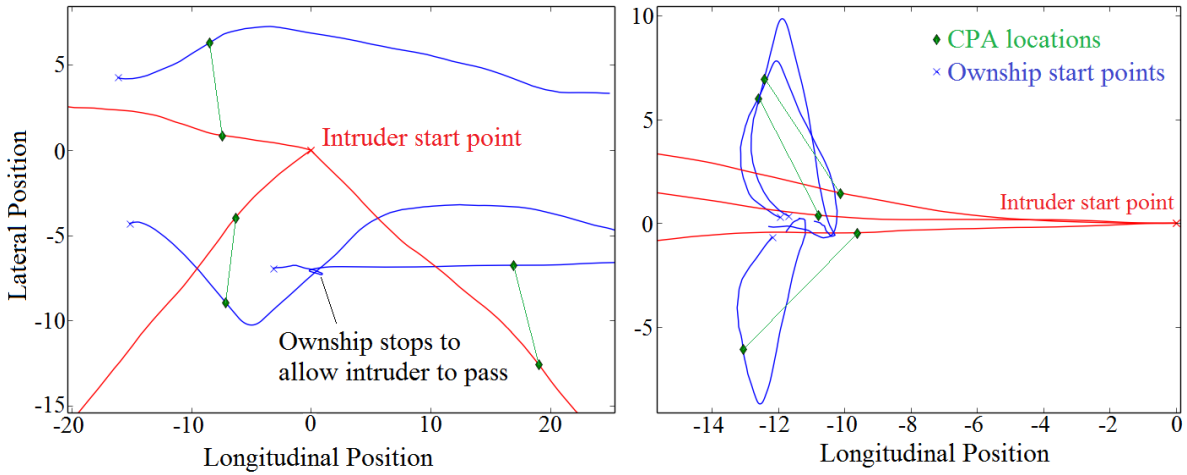
Human Expert Check

- Left: intruder starts at $(0, 0)$,
- random heading, fixed velocity of the intruder
- the ownship starts at the blue cross
- Right: The goal is hovering
- the intruder comes from the right with the unknown behaviour.



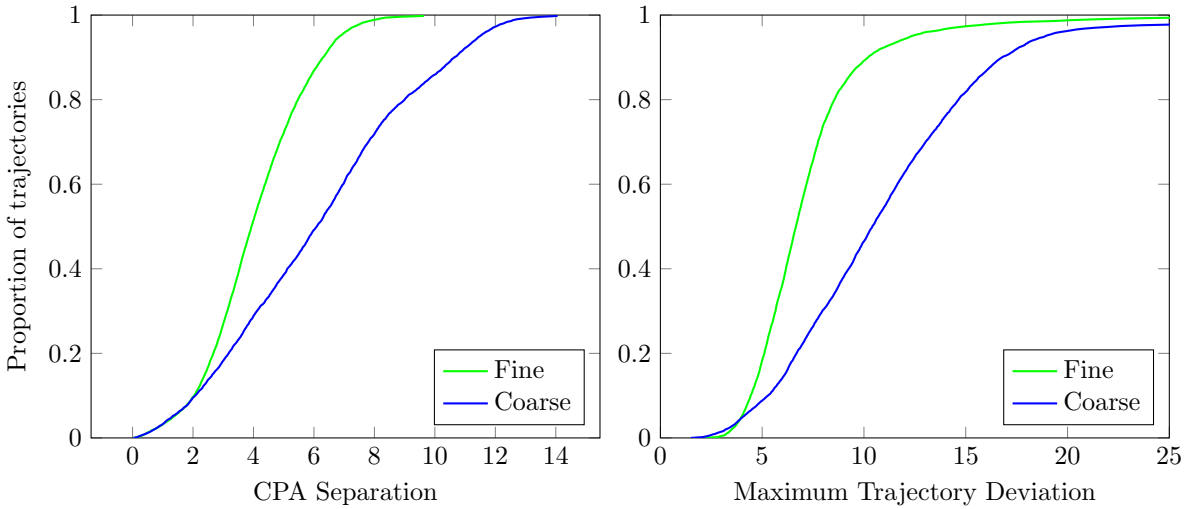
Human Expert Check

- Left: intruder starts at $(0, 0)$,
- random heading, fixed velocity of the intruder
- the ownship starts at the blue cross
- Right: The goal is hovering
- the intruder comes from the right with the unknown behaviour.



State Discretization

The fine discretization improves the results.



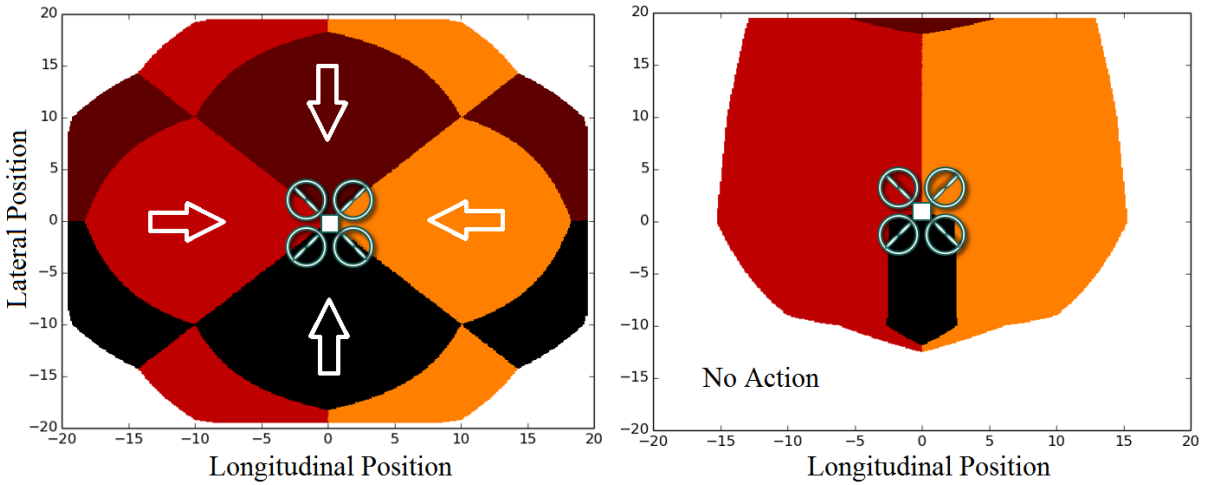
(a) CPA separations

(b) Maximum trajectory deviations

Figure 10: Cumulative distributions of encounter model metrics as a function of state discretization

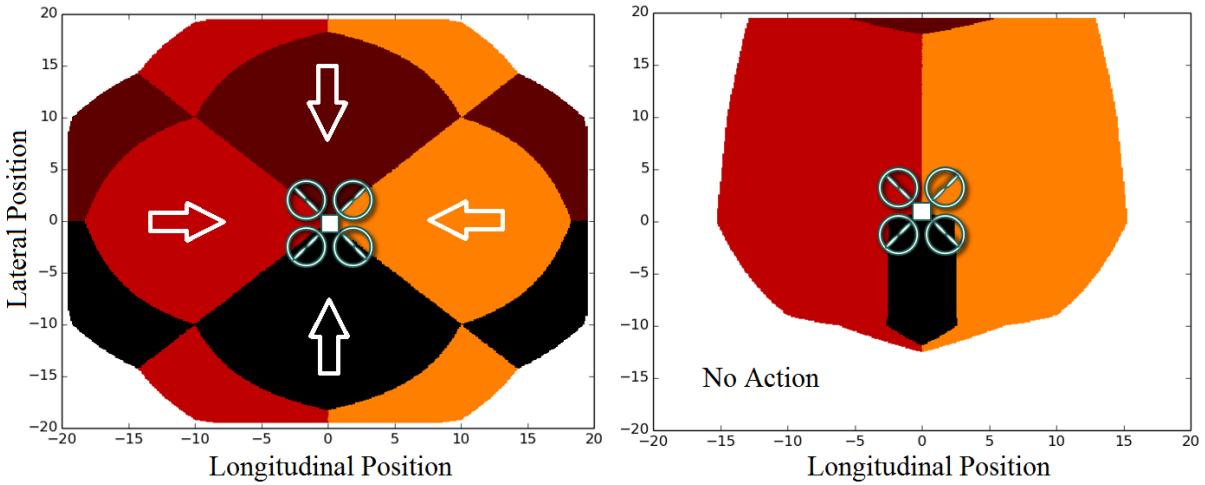
Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



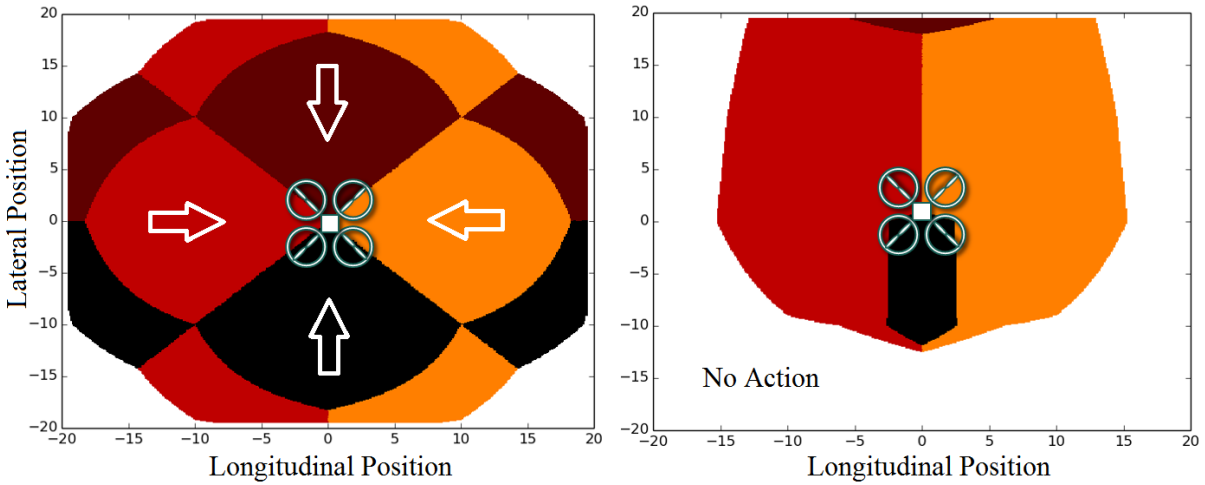
Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

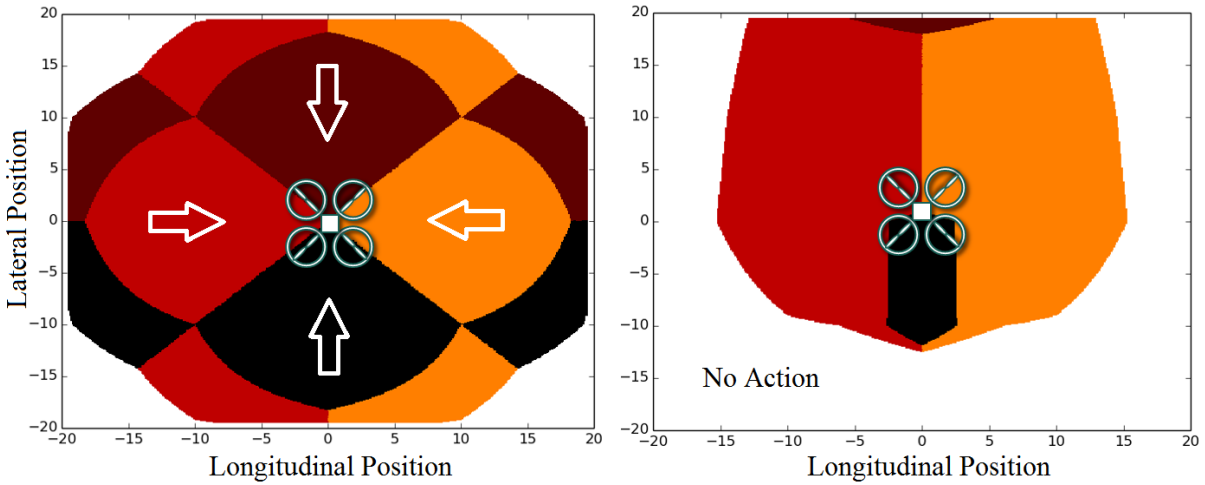
- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

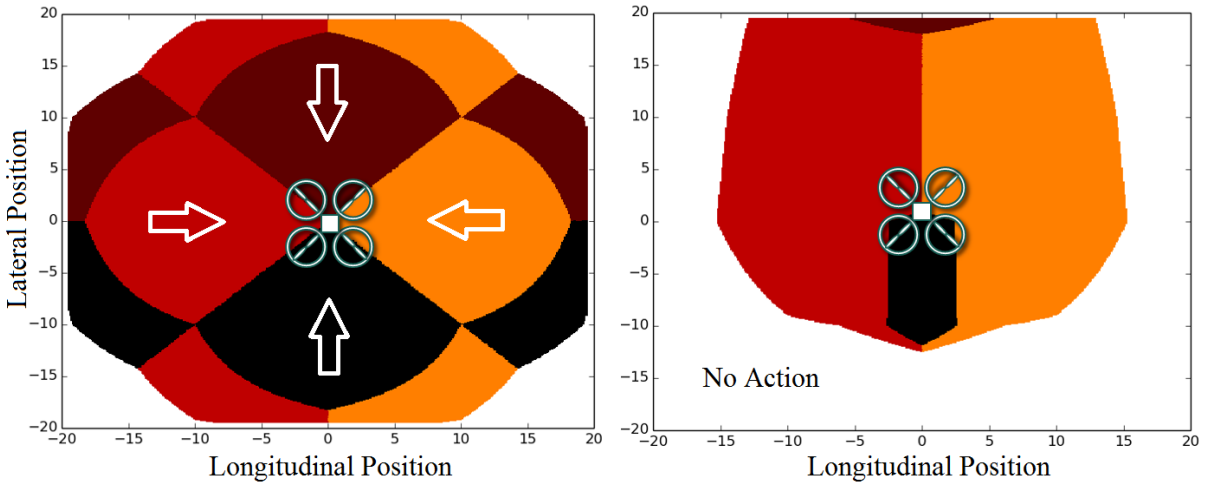
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

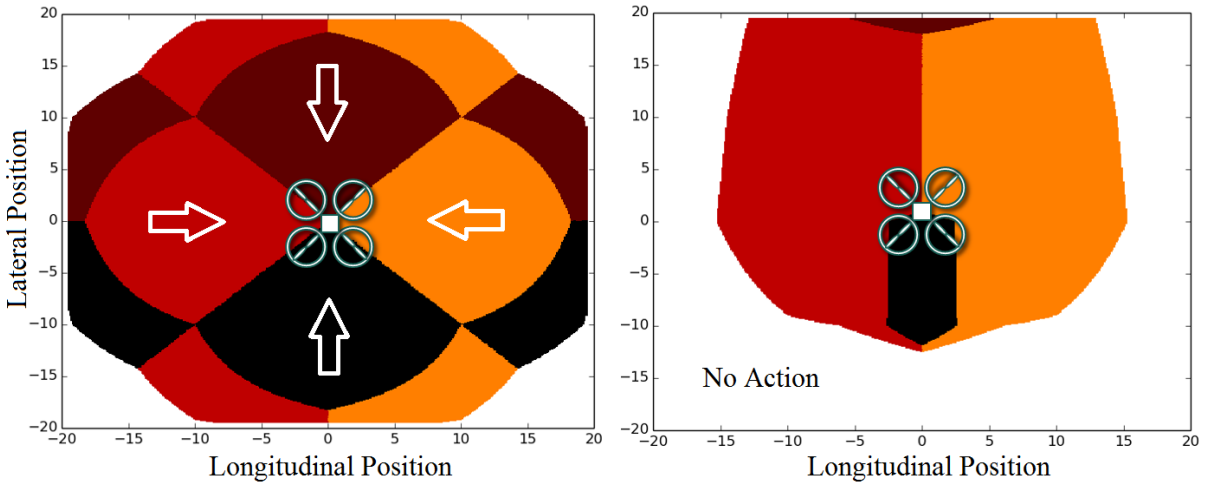
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

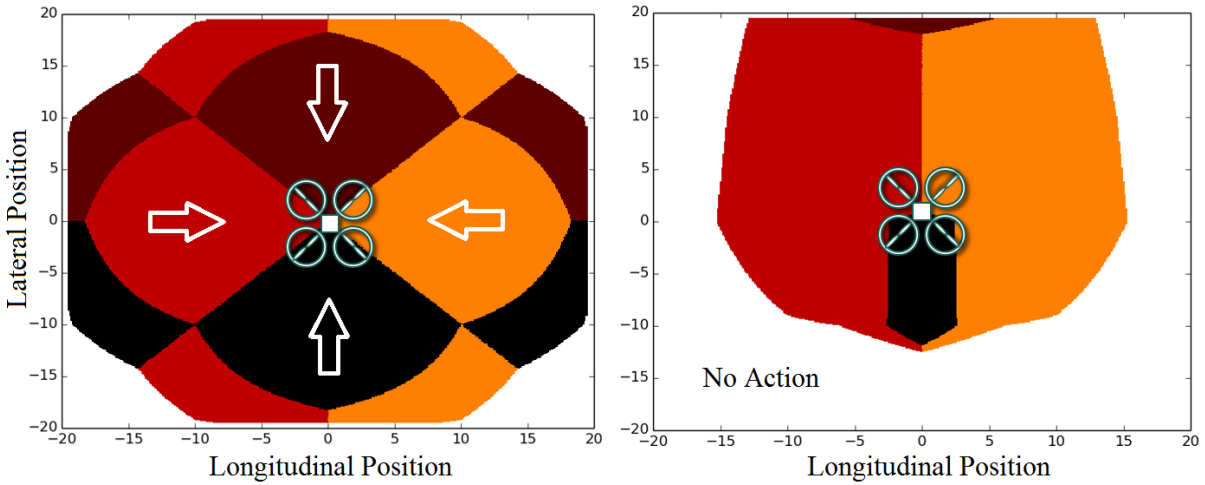
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

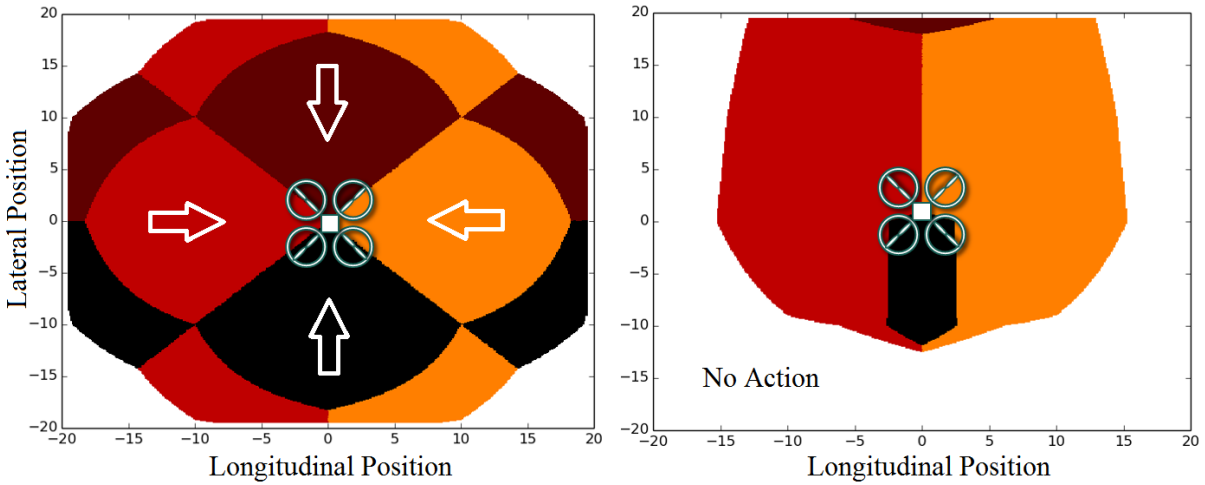
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

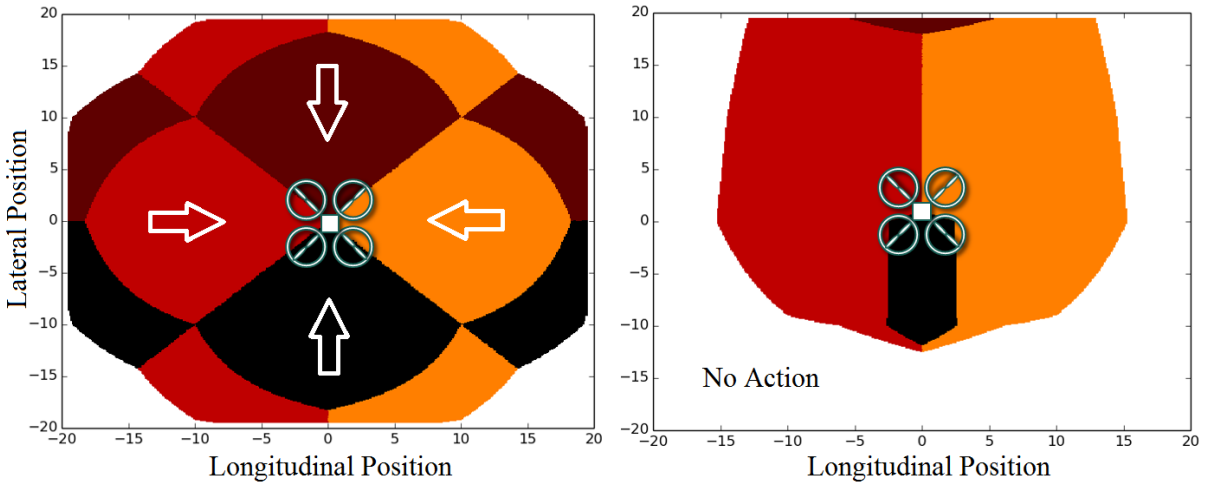
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

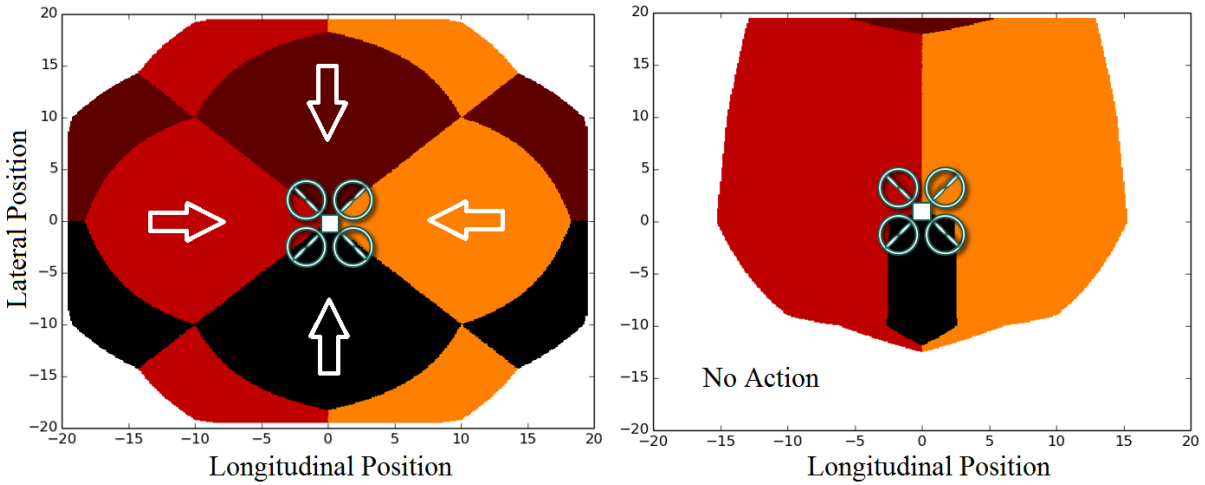
- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Value Iteration, QMDP Policies

- considerable improvement in convergence speed by initializing by the value of previously evaluated policy
- the value of maximum negative reward influenced the convergence speed
- $\gamma \leftarrow 0.99$ taking hundred iterations to converge.
- smaller γ did not ensure the return to the desired path.

- Figures: Owhship at the origin
- different intruder positions
- policies indicated by color: black=up, red=right
- left: both own and intruder velocities are zero, $d = 0$
- right: owhship is moving in the positive y–axis direction at 1 s^{-1} with zero trajectory error and nominal trajectory matches the velocity.
- The intruder is stationary.



Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbours
  - simplex interpolation between $n + 1$ nearest neighbours
  - prior work has found little benefit to using more sophisticated approaches

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbours
  - simplex interpolation between $n+1$ nearest neighbours
  - prior work has found little benefit to using more sophisticated approaches

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbours
  - simplex interpolation between $n+1$ nearest neighbours
  - prior work has found little benefit to using more sophisticated approximations

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbor
  - simplex interpolation between $n+1$ nearest neighbors
  - prior work has found little benefit to using more sophisticated approaches

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbor
  - simplex interpolation between $n + 1$ nearest neighbor
  - prior work has found little benefit to using more sophisticated approaches.

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbor
  - simplex interpolation between $n + 1$ nearest neighbor
  - prior work has found little benefit to using more sophisticated approaches.

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbor
  - simplex interpolation between $n + 1$ nearest neighbor
  - prior work has found little benefit to using more sophisticated approaches.

Beliefs

- Uncertainty does not increase with time, QMPD is justifiable.
- State uncertainty is incorporated only when actions are selected
- a set of potential states is calculated from the observations received at each step.
- the probability distribution on potential states is the belief used to select an action.

$$\pi(b) = max_a \left[\sum_{s \in S_p} Q(s, a) b(s) \right]$$

- The value $Q(s^{(k)}, a) b^{(k)}$ approximated from QMDP solutions
  - rectangular interpolation between 2^n nearest neighbor
  - simplex interpolation between $n + 1$ nearest neighbor
  - prior work has found little benefit to using more sophisticated approaches.

Table of Content

Summary Links

- BN basics
  - Bayesian Network, Conditional Independence, Separation, d-separation, Markov Blanket, ...
  - Naive Bayes Classifier, Functions MI, KL, CMI, loglik, BIC, AIC
- BN Evaluation
  - Variable Elimination Algorithm, Junction Tree Algorithm
  - Likelihood weighting, Gibbs Sampling, (Metropolis Hastings Sampling)
- Parameter Learning
  - Frequency Ratio, Dirichlet, BDeu priors, Bayesian Learning BO, MAP, ML, Missing Data, EM algorithm
- Structure Learning
  - Chow–Liu Tree, Learning TAN Classifier
  - Myopic Structure Search, PC–Algorithm, (Structural EM)
- Gaussian Variables
  - Gaussian Graphical Models, Graphical Regression, GGM Model Selection (deviance, idev, lrt)
  - Gaussian Process, (Bayesian Optimization)
- Decisions
  - Decision Tree, DT Evaluation
  - Decision Graphs =IDs, Variable Elimination for DG
  - (Markov Decision Processes, Value Iteration Algorithm,) Partially Observed Markov Decision Processes, Policy Graph
- Variational Approximation
  - Variational Approximation (, Latent Dirichlet Allocation).