Sensor Path Planning Using Reinforcement Learning

Folker Hoffmann[1], Alexander Charlish[1], Matthew Ritchie[2], Hugh Griffiths[2]
[1]Fraunhofer FKIE, [2]University College London
{folker.hoffmann,alexander.charlish}@fkie.fraunhofer.de
{m.ritchie,h.griffiths}@ucl.ac.uk

***Abstract*—Reinforcement learning is the problem of autonomously learning a policy guided only by a reward function. We evaluate the performance of the Proximal Policy Optimization (PPO) reinforcement learning algorithm on a sensor management task and study the influence of several design choices about the network structure and reward function. The chosen sensor management task is optimizing the sensor path to speed up the localization of an emitter using only bearing measurements. Furthermore, we discuss generic advantages and challenges when using reinforcement learning for sensor management.**

I. Introduction

Reinforcement learning (RL) [1] is the problem of autonomously learning a policy by interacting with the environment. In this setting an agent is given observations of this environment, on which it can act via a defined set of actions. After each interaction the agent receives a reward. The intent of the agent is to learn a policy which maximizes the reward. The agent is especially not given an *a priori* available transfer or observation function of the environment.

Sensor management [2] is the problem of optimizing the configuration and positioning of sensors to achieve a given sensor task, for example localizing a target. A huge number of methods have been proposed for sensor management, often based on explicit modeling of the sensors or based on explicitly derived information theoretic properties of the sensing process.

Reinforcement learning offers the promise to avoid making such explicit sensor-specific optimization, and instead replace them by stating the sensing objective and let the system itself learn a way to optimize the objective. The system then can either learn its behavior online during actual environment interactions or - especially when many interactions are needed - initially in a simulated environment.

In this paper we evaluate the performance of Proximal Policy Optimization (PPO) [3], a state of the art reinforcement learning approach, on a sensor management problem. The sensor management problem consists of optimizing the path of a mobile bearing sensor, for example to localize an emitter via RF-DOA (radio frequency direction of arrival) measurements.

The remainder of the paper is structured as follows: In Section II we survey related work. Section III describes the environment from the point of view of the RL agent. The evaluated algorithm, its input features and network architectures are described in Section IV. We present the results in Section V and conclude the paper in Section VII.

II. Related Work

The published work on reinforcement learning itself is vast and beyond the scope of this paper. We refer the reader to the introductory book by Sutton [1], as well as focused surveys, e.g. [4].

The problem of optimizing the sensor path for emitter localization with bearing only measurements is a classic problem of sensor management. Techniques proposed for this problem are for example lookahead techniques based on the Fisher Information [5], [6], myopic planners [7] or rollout based planners [8]. Similar techniques have been used to optimize the sensor path for tracking moving emitters [9].

Reinforcement learning has also been used for several other sensor management tasks. In [11] temporal difference learning (TD(λ)) has been used to optimize target detection in discrete cells. In [12] a policy was learned to decide on using a short-range and a long-range mode for a radar to track a target. SARSA was used in [13] to optimize waveforms in a MIMO radar system. A Q-learning based algorithm was used in [14] for resource management in a wireless sensor network.

A reinforcement learning approach for bearing only sensor path planning has been proposed in [10]. It uses a linear representation of the policy and differs from the work described in this paper by using a pure policy-gradient based approach, without a critic-function. Additionally, an extension for multi-target localization is presented in the work. This is done by computing the single-target policy for each target. Then a nonlinearity, e.g. the sigmoid function, is applied to each action, and the results are combined by aggregation functions, like the average or the minimum. The resulting vector has one entry for each aggregation function and is used as feature vector for the output layer.

The contribution of this paper is to analyze the performance of a state of the art reinforcement learning algorithm on the emitter localization problem.

III. Environment

The environment of the reinforcement learning agent describes the interaction with the remaining system and consists

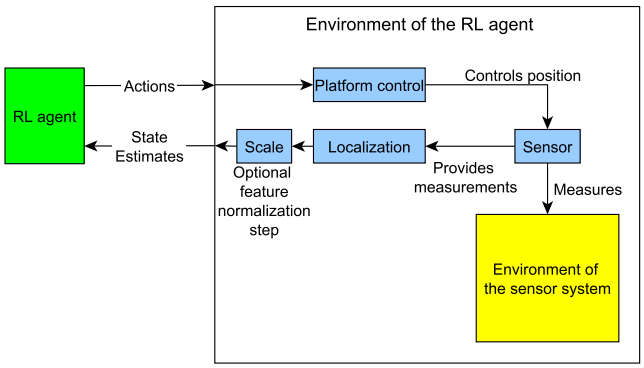

Fig. 1. Environment of the reinforcement learning agent.

of four parts: The movement model of the platform, which describes how the platform moves based on the chosen actions. The sensor model, which formalizes the sensing process and generation of measurements. The localizer integrates these measurements into a state estimate. Finally, the scenario describes the simulated geometry.

From the point of view of the RL agent, the environment gets movement actions as input and returns the current state estimate. The actual behavior of the sensor, localizer and movement model is not directly available to the agent. Figure 1 shows a representation of the environment of the RL agent, compared to the true environment of the sensor system. We note that this description of the environment - common in the reinforcement learning literature - differs from the notion in the sensor management literature, where the environment interaction often means only the sensing process, and e.g. the localizer is commonly considered a part of the sensing system.

A. Movement Model

We represent the platform state as a 2-dimensional position with heading:

$$\mathbf{x}_k^p = (x_k^p, y_k^p, \varphi_k^p)^T \ , \tag{1}$$

where x_k^p, y_k^p is the platform position at time k and φ_k^p its heading, measured as counter-clockwise angle from the x-axis.

The control input $u = \Delta\varphi$ denotes the change in heading

$$\varphi_{k+1}^p = \varphi_k^p + u \ . \tag{2}$$

Then the platform moves according to the current heading φ_{k+1}^p and the constant platform speed s^p

$$x_{k+1}^p = x_k^p + \Delta t \cdot s^p \cdot \cos\left(\varphi_{k+1}^p\right) \ , \tag{3}$$

$$y_{k+1}^p = y_k^p + \Delta t \cdot s^p \cdot \sin\left(\varphi_{k+1}^p\right) \ , \tag{4}$$

where Δt denotes the length of a single time step.

The control input is discretized into N^a actions, and action a_i is given as

$$a_i = L^u + \frac{i}{N^a - 1} \cdot (U^u - L^u) \ , \tag{5}$$

where L^u and U^u are lower and upper bound on the control input.

B. Sensor Model

We model a direction-finding sensor, which measures the angle to an emitting, stationary target. At each time step k the sensor generates a measurement, corrupted with Gaussian noise:

$$z_k = h(\mathbf{x}^t, \mathbf{x}_k^p, w_k) = \operatorname{atan2}\left(y^t - y_k^p, x^t - x_k^p\right) + w_k \ , \tag{6}$$

where $\mathbf{x}^t = (x^t, y^t)^T$ is the position of the target and $w_k \sim \mathcal{N}(0, \sigma^2)$ the realization at time step k of the normal distributed noise with standard deviation σ.

C. Localizer

The localizer performs a maximum likelihood estimate of the target position, using the full batch of bearing measurements. As a way to incorporate prior knowledge, an initial Gaussian estimate $\mathbf{x}_0^e, \mathbf{P}_0^e$ is used. After each action, bearing measurements $z_1, ..., z_k$ are received, therefore the log likelihood at step k is proportional to

$$\log l(\mathbf{x}) \propto (\mathbf{x}-\mathbf{x}_0^e)^T \cdot (\mathbf{P}_0^e)^{-1} (\mathbf{x}-\mathbf{x}_0^e) + \sum_{i=1}^{k} \left(\frac{z_i - h(\mathbf{x}, \mathbf{x}_i^p, 0)}{\sigma} \right)^2 . \tag{7}$$

The difference $z_i - h(\cdot)$ is in angle space and takes into account the angular wrap at 0 and 2π. We use the Levenberg–Marquardt algorithm to solve this least squares problem and compute the maximum likelihood estimate

$$\mathbf{x}_k^e = \underset{\mathbf{x} \in \mathcal{R}^2}{\operatorname{argmin}} \log l(\mathbf{x}) \ . \tag{8}$$

The uncertainty of this estimate is approximated by the inverse of the Fisher Information computed at the estimated position

$$\mathbf{P}_k^e = \left((\mathbf{P}_0^e)^{-1} + \sum_{i=1}^{k} \mathbf{I}_i \right)^{-1} \ , \tag{9}$$

where the information of a single measurement is given as

$$\mathbf{I}_i = \frac{1}{\sigma^2 \cdot r^4} \begin{pmatrix} \Delta y^2 & -\Delta x \cdot \Delta y \\ -\Delta x \cdot \Delta y & \Delta x^2 \end{pmatrix} \ . \tag{10}$$

Here $\Delta x = x_k^e - x_i^p$ and $\Delta y = y_k^e - x_i^p$ is the estimated difference in the x- and y-dimension when the measurement was taken, and $r = \sqrt{\Delta x^2 + \Delta y^2}$ the estimated distance to the target.

The estimated root-mean-squared error (RMSE) of the estimate can then be given by

$$\sqrt{\operatorname{tr}\left(\mathbf{P}_k^e\right)} \ , \tag{11}$$

where tr denotes the trace of the covariance matrix.

D. Scenario

The geometry of the scenario can be seen in Figure 2. The platform is placed at the origin and the uncertainty ellipse of the prior knowledge is shown. At each instantiation of the scenario, the true target position is randomly sampled from the prior. The scenario is terminated if the expected RMSE falls below a threshold μ_T. To avoid non-termination of the scenario

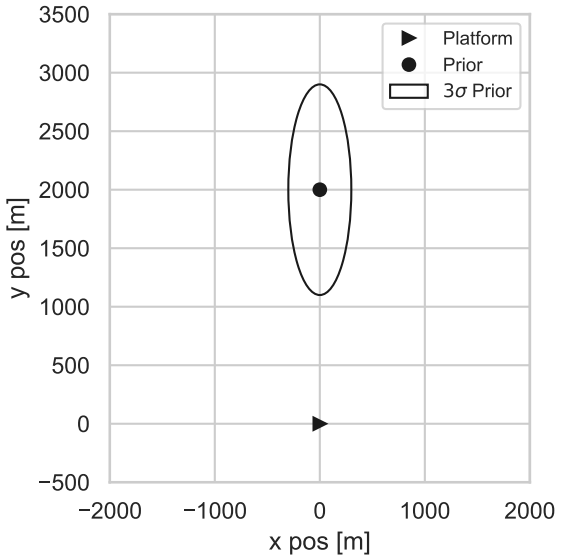

Fig. 2. Geometry of the scenario. Each concrete scenario instantiation samples the true target position from the prior. The prior is available to the localizer.

TABLE I
SCENARIO PARAMETERS

Parameter	Value
x_0^e	0 m
y_0^e	2000 m
$\mathbf{P}_0^e$	diag $\left((100\text{ m})^2, (300\text{ m})^2\right)$
U^u	20°
L^u	−20°
σ	10°
μ_T	10 m
s^p	50 m/step
N^a	11
x_0^p	0 m
y_0^p	0 m
φ_0^p	0°

when the policy does not achieve a localization, the scenario is also terminated if no localization occurs after 500 steps. The full sequence of agent interactions with the environment until termination, i.e. one scenario instantiation, is also called episode. Parameters of the scenario can be seen in Table I.

E. Reward

The stated goal of the agent is to localize the target as fast as possible. We therefore define a constant reward of $r_k = -1$, which serves as a cost and accumulates as long as the environment is active. The only way to avoid this cost is by having the target localized.

As usual in reinforcement learning, the goal of the agent is to maximize the sum of the rewards. Therefore the learning algorithm needs to find a policy parametrization which gives the highest expected reward:

$$\max_{\theta^p} \mathbb{E}\left[\sum_{k=0}^{N^k-1} r_k \mid a_k \sim \pi_{\theta^p}\right] . \tag{12}$$

Here θ_p is the parametrization of a policy π_{θ^p}, for example the weights of a neural network. The actions a_k are chosen according to this policy. N^k is the total number of steps encountered in the scenario, which stops when the target is localized and is capped at 500. The negative of the reward sum is the time until the target is localized.

IV. EVALUATED ALGORITHM

We used the implementation `PPO2` from *Stable Baselines* [15], which is a fork from the *OpenAI Baselines* [16] implementation of reinforcement learning algorithms. Unless otherwise noted, we used the default parameters at version 2.9.0. This includes a discount factor of 0.99 for future rewards.

In this section we describe the choices of our application, including the input features, as well as the policy and value network structures. We additionally perform two variants of reward shaping. First, we use the gain in localization accuracy as a substitute reward. Second, we use a variant of curriculum learning to make it easier for the algorithm to achieve its localization goal.

A. Input Features

While the platform movement and localization take place in absolute coordinates, the localization problem itself is invariant under translation and rotation.

We can therefore represent the state estimate in a coordinate system centered on the platform, with the local x-axis aligned with the platforms heading. Then the estimate in this relative frame is

$$\mathbf{x}_k^r = (x_k^r, y_k^r)^T = \mathbf{R}_k \left(\mathbf{x}_k^e - \mathbf{x}_k^p\right) , \tag{13}$$

with covariance

$$\mathbf{P}_k^r = \mathbf{R}_k \mathbf{P}_k^e \mathbf{R}_k^T . \tag{14}$$

$\mathbf{R}_k$ represents the rotation matrix based on the platform heading at time step k and is given as

$$\mathbf{R}_k = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} , \tag{15}$$

where the rotation angle $\alpha = -\varphi_k^p$ is the negative of the platform heading.

The relative estimate $\mathbf{x}_k^r$ is then chosen as input feature. As input features of the covariance matrix we use the variance of the x- and y-estimate, and their correlation

$$\rho_k = \frac{\operatorname{cov}\left(x_k^r, y_k^r\right)}{\operatorname{std}\left(x_k^r\right) \cdot \operatorname{std}\left(y_k^r\right)} . \tag{16}$$

The full list of input features can be seen in Table II.

TABLE II
INPUT FEATURES FOR THE ALGORITHM

Feature	Description
x_k^r	x-position estimate
y_k^r	y-position estimate
var $\left(x_k^r\right)$	variance of the x-position estimate
var $\left(y_k^r\right)$	variance of the y-position estimate
ρ_k	correlation of the x- and y-position estimates
	All values are of the target position, relative to the platform

B. Proximal Policy Optimization

Giving a full description of the algorithm is beyond the scope of this paper, we refer the reader to the original publication [3], as well the implementations in [16] and [15]. In this section we briefly want to give the basic idea of this algorithm.

The algorithm follows an actor-critic architecture, which means that it contains an *actor* (also called *policy network*), which decides on the actions based on the current state:

$$\pi_{\theta^p} : \mathbb{R}^{N^f} \rightarrow \mathbb{R}^{N^a} \ . \tag{17}$$

Here θ^p are the weights of the network and N^f the number of input features. The output is a probability for each action, thereby giving the policy a random chance of choosing any action. Due to this randomness, the policy explores the state space. The other element of such an algorithm is the *critic* (also called *value network*), which learns to predict the value (i.e. the expected future rewards) of a state:

$$\pi_{\theta^v} : \mathbb{R}^{N^f} \rightarrow \mathbb{R} \ . \tag{18}$$

During exploration empirical rewards are received by the actor. Whether they are better or worse than in previous interactions can be decided by the value network. Then the policy will be updated using the policy-gradient theorem [1], in a way that actions leading to higher rewards are chosen more frequently in the future. PPO is a variant of this approach, which constrains the policy update to keep the new policy closer to the previous one in regard to the chosen action probabilities.

C. Network Description

We evaluated different networks to represent the policy and value network of the algorithm. We used fully connected layers [17], with a varying number of hidden neurons. The topologies evaluated were a single hidden layer of 10 neurons, as well as 1,2 and 3 hidden layers of 60 neurons each. We evaluated both ReLu and tanh activation methods.

The policy layer used a final fully connected linear layer returning the logits, which represent the probability for each action during training. During evaluation the action with highest probability was chosen. The value layer used a final fully connected linear layer with a single output neuron to represent the value of a state. Otherwise, value and policy network used the same activation function and number of hidden layers for each evaluated configuration. The layers did not use weight sharing between policy and value network.

We also evaluated a scaling step, in which the input features were normalized before feeding them into the network. This was performed by computing the running mean and standard deviation of every feature for all environment interactions during the training, and then normalize the feature by subtracting the mean and dividing by the standard deviation.

D. Reward Shaping

During initial experiments we encountered a problem with the constant reward function $r_k = -1$, together with the localization threshold in Table I. When training starts, it can be that the initial policy parametrization rarely leads to a target localization. In this case it can happen that all episodes end with the worst possible reward of -500. Then the RL algorithm would have no differences in the reward signal, which it could use to improve the policy.

We experimented with two approaches to make the task easier to learn. The first technique was to use the improvement in the localization (*localization gain*) as a substitute reward

$$r_{k+1} = \sqrt{\text{tr}\left(\mathbf{P}_k^e\right)} - \sqrt{\text{tr}\left(\mathbf{P}_{k+1}^e\right)} \ . \tag{19}$$

This reward makes it easy for the reinforcement learning algorithm to pick up on a reward signal, as it provides an immediate feedback on the value of an action.

As a second method to make the task easier to learn, we employed a method based on curriculum learning. Curriculum learning is based on the idea of letting a machine learning algorithm first learn simple tasks, before moving to more complex tasks [18], [19].

Our implementation of curriculum learning was based on making the localization threshold μ_T time dependent. Early in the training it was set to a high value. This makes it easier for the agent to randomly encounter a sensor path which localizes the target. Then we progressively decrease the localization threshold, until it reached the original threshold during half of the training time. This threshold schedule can be seen in Figure 3.

V. RESULTS

As comparison to the trained policies, we used a myopic policy. This policy selected at each step the action that maximizes the Fisher Information, based on the current estimate:

$$a_k = \underset{a}{\text{argmax}} \ \det\left(\left(\mathbf{P}_k^e\right)^{-1} + \mathbf{I}_{k+1}\right) \ . \tag{20}$$

An evaluation with 1000 Monte Carlo runs shows an average time to localization of 60.845 ± 0.655 (95% confidence interval) for the myopic policy.

A. Training

We trained each combination for $400\,000$ environment interactions, which are equal to time steps, with 10 different seeds. After training, the policy was evaluated using 1000 MC runs

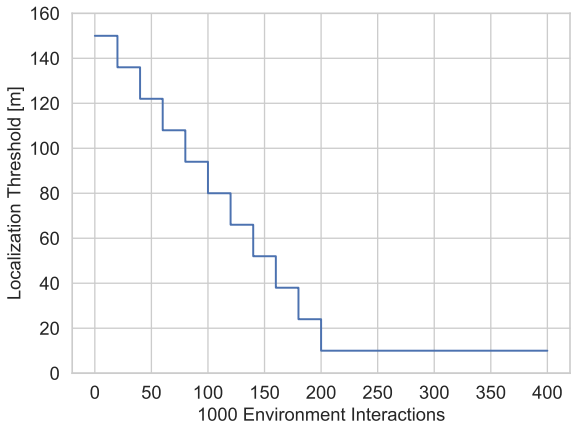

Fig. 3. Localization threshold schedule for curriculum learning.

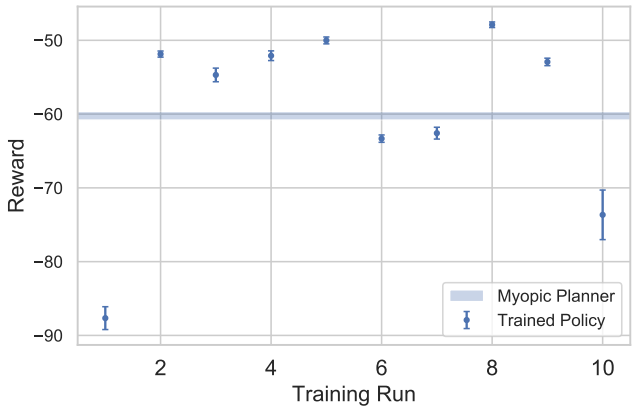

Fig. 4. Performance of the fully trained policy of each training run from (60x60, tanh, Gain, False). The error markers correspond to the 95% confidence interval on the mean after 1000 evaluations of the policy.

on the environment. To save computing resources, we used only 10 MC runs, when the policy did not show any success during training and had an episode reward of -500.

We used each combination of

- Network size: 10, 60, 60x60, 60x60x60
- Activation function: ReLu, tanh
- Reward shaping: None, (Localization) Gain, Curriculum
- Scaling of the input features: True, False

to train 10 times a policy, leading to a total number of $48 \cdot 10$ training runs. We refer to a concrete configuration by the tuple (network, activation, reward shaping, scaling) and to a concrete learned policy by the tuple (network, activation, reward shaping, scaling, run).

Figure 4 shows the result of (60x60, tanh, Gain, False). It can be seen, that even when the same parameters are used, due to the random nature of exploration and the stochastic environment, policies with varying performance are learned.

During training, we evaluated the policy every 2000 environment interactions on 10 randomly chosen instantiations of the scenario, giving an estimate of the current performance

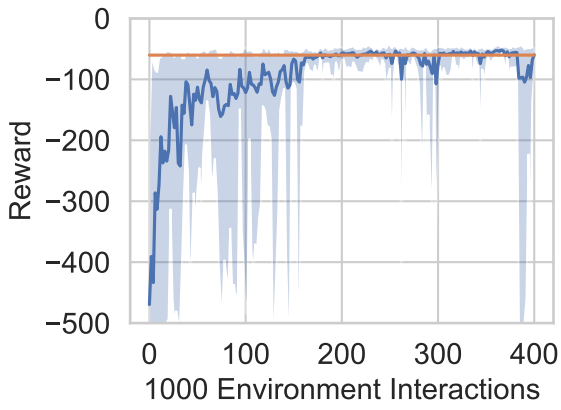

Fig. 5. Policy performance during the training of (60x60, tanh, Gain, False). The blue line is the mean reward for all 10 training runs and the shaded area the worst and best run. The orange line is the myopic policy.

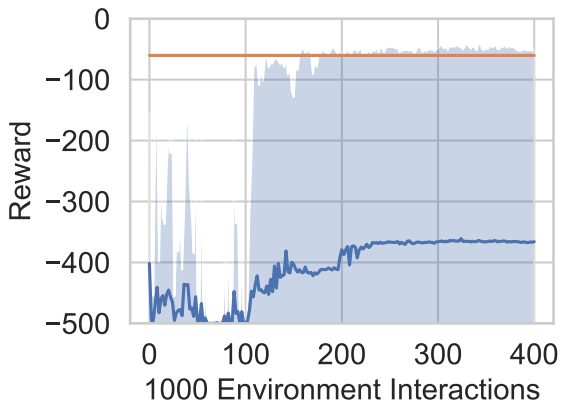

Fig. 6. Policy performance during the training of (60x60, ReLu, —, True). The blue line is the mean reward for all 10 training runs and the shaded area the worst and best run. The orange line is the myopic policy.

of the algorithm. Figure 5 shows the training progress for (60x60, tanh, Gain, False). It can be seen that the performance on average improves, even though for some training runs it degrades temporarily.

Figure 6 shows a less successful training run (60x60, ReLu, —, True). While some policies reach and also surpass the myopic policy, some do not achieve the task at all and the average performance stays quite low.

B. *Influence of the Design Choices*

We analyzed the average influence of those design choices on the performance of the trained policy. We defined a successful training by having a mean reward during evaluation > -499, i.e. the training resulted in a policy which localizes the target at least once during the evaluation. If the mean reward is higher than -60, we say that it beats the myopic policy. From all training runs we performed, 51.4% were successful and 14.3% were better than the myopic planner.

TABLE III
INFLUENCE OF THE REWARD SHAPING

Shaping	Success	Beats Myopic
—	0.25000	0.08750
Curriculum	0.56250	0.18750
Localization Gain	0.73125	0.15625

TABLE IV
INFLUENCE OF SCALING THE INPUT FEATURES

Scaled	Success	Beats Myopic
False	0.295833	0.133333
True	0.733333	0.154167

Table III shows the influence of the reward shaping. It can be seen that without any reward shaping the success rate is quite low, likely because the algorithm does not encounter any trajectory which localizes the target. Curriculum learning improves on this problem, however still requires encountering a successful trajectory - even if finding this is easier. Using the localization gain as substitute reward leads to a comparatively high success rate. This is because even a single action can lead to a training signal.

Scaling the input features clearly improves the probability of success (Table IV). A positive effect can also be seen in using the tanh function instead of ReLu for activation, as can be seen in Table V.

Table VI shows the influence of the network architecture. While the success rate slightly increases with increasing network size, the effect is not that strong. This indicates that a policy to localize the target at all can be represented even with a small number of network parameters. On the other hand, the probability that the trained policy achieves a higher reward than the myopic policy strongly increases with the network capacity.

C. Behavior

Figure 7 shows the behavior of the policy (60x60, tanh, Gain, False, 8) during training. It can be seen that early in the training, the policy moves far away from the target, however, at the end returns towards the target. As this already happens for the first, randomly initialized policy, this training run seems to have started with a good starting policy. During training, it can be seen that the trajectory more and more gets closer to moving towards the target at a sharp angle. However, this progress is also interrupted at some times, for example after 50 000 steps the policy moves further away, then after 30 000 or 80 000.

Figures 8 and 9 show the fully trained policy (60x60, tanh,

TABLE V
INFLUENCE OF THE ACTIVATION FUNCTION

Activation	Success	Beats Myopic
ReLu	0.362500	0.108333
tanh	0.666667	0.179167

TABLE VI
INFLUENCE OF THE NETWORK ARCHITECTURE

Network	Success	Beats Myopic
10	0.483333	0.041667
60	0.500000	0.116667
60x60	0.508333	0.183333
60x60x60	0.566667	0.233333

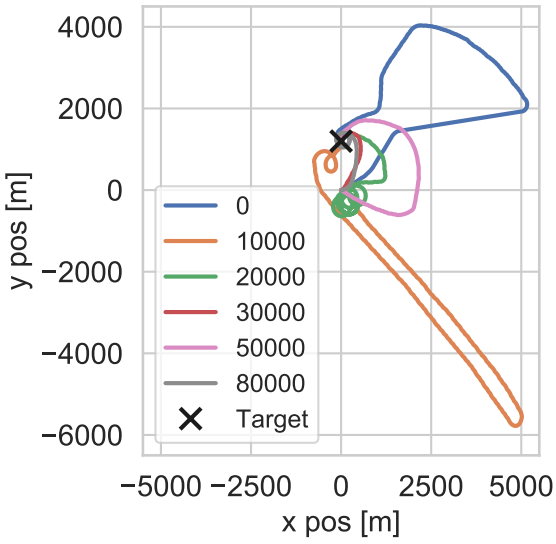

Fig. 7. Training progress of (60x60, tanh, Gain, False, 8), after several numbers of interactions with the environment.

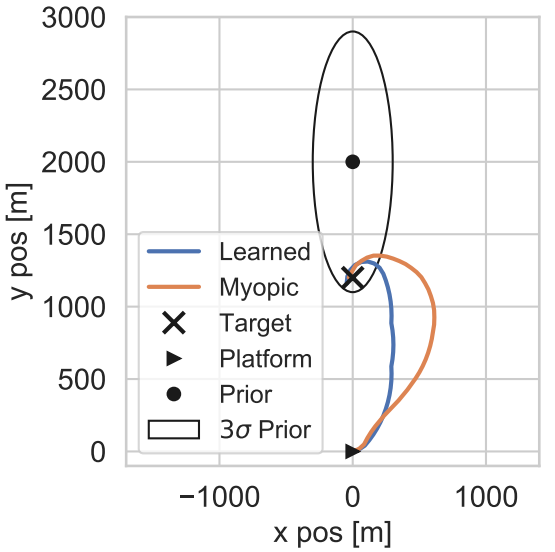

Fig. 8. Comparison of the policy (60x60, tanh, Gain, False, 8) to the myopic policy.

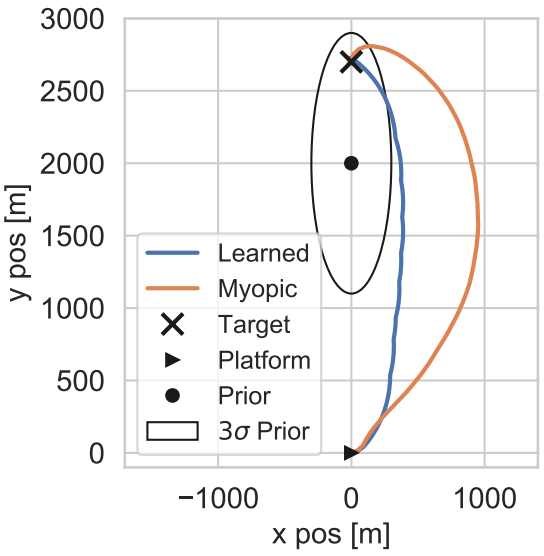

Fig. 9. Comparison of the policy (60×60, tanh, Gain, False, 8) to the myopic policy.

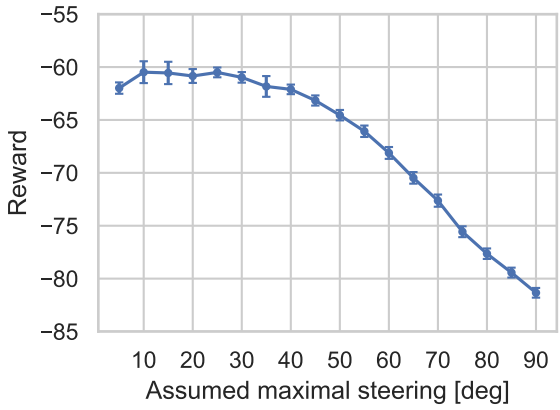

Fig. 10. Effect of false modeling assumptions about the platforms maneuverability by the myopic planner. The maximal steering is equal to $= U^u = -L^u$. The error bars correspond to the 95% confidence interval, computed by 1000 MC runs.

Gain, False, 8) compared to the myopic policy for two exemplary target positions. Both of those methods reduce the range to the target, which leads to improved information content of the measurements (10). Because of the high requirements on the target localization accuracy, the target is first localized when the platform is rather near to the target. It can be seen that the trained policy moves towards the target in a more direct path, leading to a faster reduction in range and eventually faster localization.

VI. Discussion and Challenges

In the paper we have shown that a reinforcement learning algorithm is able to learn a policy for an emitter localization task. While we have seen some fluctuation between different training runs, even with the same parametrization, this does not necessarily prohibit these techniques, as it is sufficient to only learn a single good performing policy. In this section we discuss some general advantages and challenges on using reinforcement learning for sensor management. We use the term *fusion algorithm* as a more general term, encompassing the tracking or localizing process of one or multiple, static or moving targets, via one or multiple sensors.

We see several advantages of using reinforcement learning approaches. An advantage is that a learning algorithm is able to adapt to the sensor and platform properties, e.g. the amount of measurement noise or the field of view without explicitly modeling it. An even stronger advantage is that it also adapts to the fusion algorithm. Information based sensor management maximizes the information-theoretic contribution of the measurement, however it is not necessarily guaranteed that the fusion algorithm makes the best use of it. A reinforcement learning approach as described instead considers the full interaction between sensor and fusion algorithm. Therefore, it could adapt to changes in the sensor data processing, e.g. a different localization threshold or the number of missing measurements causing a track drop.

In our example the myopic policy showed already a good performance. We would expect that a non-myopic policy would be able to improve on this result, possibly even beating the best learned policies. The performance is good, because the myopic policy used prior knowledge about the sensor model, its corresponding Fisher information, the stationarity of the target, and the movement model of the platform. This all is not available to the reinforcement learning algorithm, which has to fully learn the relationship between states, actions and rewards on its own.

However, this explicit modeling is only an advantage as long as the models and assumptions are valid. This is exemplified in Figure 10, where the myopic planner is executed with different assumptions on the minimal and maximal steering L^u and U^u. As they move away from the true value of 20°, the performance of the myopic policy becomes worse. The reinforcement learning algorithm instead just learns based on the actual returns of the actions and is therefore not prone to such modeling errors.

On the other hand, we see also challenges in a reinforcement learning based approach. Especially model-free methods, as the one we used in this work, often require a large number of samples. Reinforcement learning is also known to be very hyperparameter sensible and dependent on the concrete algorithm implementation [20].

A more important challenge is that it is necessary to be able to simulate a sufficient number of representative scenario instantiations. To evaluate a tracking algorithm or an online optimization-based sensor management algorithm, testing the algorithm on a few well-selected benchmark scenarios often gives important knowledge about its behavior. Such scenario-based evaluation is often used for tracking algorithms and sensor management.

To train the reinforcement learning algorithm well, it is required to sample from the whole distribution of possible scenarios. Training on a selected set of scenarios leads to the risk of overfitting and simply memorizing optimal actions for those scenarios without generalizing. With multiple non-stationary and maneuvering targets, and additional scenario constraints it is often possible to describe individually plausible scenarios, however it is more difficult to enumerate each possible realistic scenario. It is even more difficult to assign an appropriate probability distribution over each possible realistic scenario. The general problem of generalization and avoiding overfitting to single environments is still a heavily researched topic for reinforcement learning algorithms [21].

A more technical challenge is, that sensor data fusion often has a variable number of targets, where each target track might possibly be represented by a variable number of hypotheses. In this case the number of input features is variable and cannot be used with a fixed input size policy as described in this paper. One possible solution would be the use of aggregation functions, as done in [10].

A possibility to avoid those challenges would be to consider the RL agent as a single building block of a larger sensor manager. For example in a multi-emitter localization problem, a higher-level planner could create a sequence in which order the emitters are localized. Following this sequence, each emitter is considered a single-emitter localization problem, for which a learned policy as described above could be used.

An alternative to the model-free algorithm we used in this work would be to use model-based reinforcement learning approaches. In this case the algorithm would learn the mapping from actions and the current state estimate to the next state estimate in Figure 1, i.e. a model including the platform control, sensor and localizer. The actions would then be selected using some online control method, e.g. model predictive control. Model based reinforcement algorithms are often more efficient regarding the number of required environment interactions. This approach would still keep the advantage of taking into account the actual behavior of the fusion algorithm.

VII. Conclusion

In this paper we have presented a sensor path planning problem and shown that a reinforcement algorithm can learn a good policy for this task. We have presented input features for the problem, and evaluated different possible choices of the policy representation and reward function.

We found that the algorithm had a notable fluctuation in its resulting policy performance, but that a good trained policy can show a performance better than a myopic planner. Several choices were analyzed on their effect on the final training performance. We found that scaling the input features and a larger network size led to better results. We also found a better performance of tanh compared to ReLu as activation function on this problem. Finally, we found that modifying the reward function led to better performance. Not surprisingly learning a policy was substantially easier, when the reward was replaced by the immediate localization gain and therefore not delayed.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, 2nd ed. Cambridge, Massachusetts, USA: MIT Press, 2018.

[2] A. O. Hero III and D. Cochran, "Sensor Management: Past, Present, and Future," *IEEE Sensors Journal*, vol. 11, no. 12, pp. 3064–3075, 2011.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 5, pp. 26–38, 2017.

[5] S. E. Hammel, P.-T. Liu, E. Hilliard, and K. F. Gong, "Optimal Observer Motion for Localization with Bearing Measurements," *Computers & Mathematics with Applications*, vol. 18, no. 1-3, pp. 171–180, 1989.

[6] Y. Oshman and P. Davidson, "Optimization of Observer Trajectories for Bearings-Only Target Localization," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 3, pp. 892–902, 1999.

[7] O. Cliff, R. Fitch, S. Sukkarieh, D. Saunders, and R. Heinsohn, "Online Localization of Radio-Tagged Wildlife with an Autonomous Aerial Robot System," in *Robotics: Science and Systems XI*. Rome, Italy: Robotics: Science and Systems Foundation, 2015.

[8] F. Hoffmann, H. Schily, A. Charlish, M. Ritchie, and H. Griffiths, "A Rollout Based Path Planner for Emitter Localization," in *Proceedings of the 22nd International Conference on Information Fusion (FUSION)*, Ottawa, Canada, 2019.

[9] M. L. Hernandez, "Optimal Sensor Trajectories in Bearings-Only Tracking," in *Proceedings of the 7th International Conference on Information Fusion (FUSION)*, Stockholm, Sweden, 2004.

[10] A. Gorji and R. Adve, "Policy Gradient for Observer Trajectory Planning with Application in Multi-target Tracking Problems," in *52nd Asilomar Conference on Signals, Systems, and Computers*. Pacific Grove, CA, USA: IEEE, 2018, pp. 2029–2033.

[11] R. Malhotra, E. P. Blasch, and J. D. Johnson, "Learning sensor-detection policies," in *Proceedings of the IEEE 1997 National Aerospace and Electronics Conference. NAECON 1997*, vol. 2, no. July. Dayton, OH, USA: IEEE, 1997, pp. 769–776.

[12] F. Smits, A. Huizing, W. Van Rossum, and P. Hiemstra, "A cognitive radar network: Architecture and application to multiplatform radar management," in *European Radar Conference (EURAD)*, Amsterdam, Netherlands, 2008, pp. 312–315.

[13] L. Wang, S. Fortunati, M. S. Greco, and F. Gini, "Reinforcement learning-based waveform optimization for MIMO multi-target detection," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. Pacific Grove, CA, USA: IEEE, 2018, pp. 1329–1333.

[14] K. Shah and M. Kumar, "Distributed Independent Reinforcement Learning (DIRL) Approach to Resource Management in Wireless Sensor Networks," in *2007 IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems*. Pisa, Italy: IEEE, 2007, pp. 1–9.

[15] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[16] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[18] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative Multi-agent Control Using Deep Reinforcement Learning," in *AAMAS 2017: Autonomous Agents and Multiagent Systems*, São Paulo, Brazil, 2017, pp. 66–83.

[19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, Montreal, Quebec, Canada, 2009, pp. 41–48.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," 2017. [Online]. Available: http://arxiv.org/abs/1709.06560

[21] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging Procedural Generation to Benchmark Reinforcement Learning," pp. 1–27, 2019. [Online]. Available: http://arxiv.org/abs/1912.01588