Learning-based control for a communicating mobile robot under unknown rates

Lucian Buşoniu, Vineeth Varma, Irinel-Constantin Morarescu, Samson Lasaulce

Learning-based control for a communicating mobile robot under unknown rates

Lucian Buşoniu, Vineeth S. Varma, Irinel-Constantin Morărescu, Samson Lasaulce

***Abstract*— In problems such as surveying or monitoring remote regions, a mobile robot must transmit data over a wireless network with unknown, position-dependent transmission rates. We propose an algorithm to achieve this objective that learns approximations of the rate function and of an optimal-control solution that transmits the data in minimum time. The rates are estimated with supervised learning from the samples observed; and the control is found with dynamic programming sweeps around the current state of the robot that exploit the rate function estimate, combined with online reinforcement learning. For both synthetic and realistic rate functions, our experiments show that the learning algorithm empties the data buffer in less than twice the number of steps achieved by a model-based solution that requires to perfectly know the rate function.**

I. Introduction

We consider a problem in which a mobile robot must transmit a data file (or equivalently, empty a data buffer) over a wireless network, with transmission rates given by a position-dependent function that is unknown to the robot. The objective is to move in such a way that the file is transmitted in minimum time. Such a problem appears, e.g., when a robot autonomously collects data in remote areas without network coverage (e.g., photographic surveying or tunnel mapping) and must then return near a human operator to quickly upload this data over an ad-hoc network, before moving on to the next mission.

In the literature, most works treat the problem of trajectory planning for communicating robots with a requirement on the instantaneous communication rate or quality of service, see e.g., [15], [4]. Differently from this, we are interested in a robot that must send data to a fixed (set of) antennas in minimum time, resulting in an optimization problem with an implicit constraint on the integral of the communication rate. In the case of a simple and known model of the wireless communication rate (assuming circular symmetry and only taking path loss into account), some recent works [13], [7] have optimized the trajectory of the robot. In [6], multiple users of a mobile access point lead to a non-circular, but still known and rather specific shape.

In this paper, we do provide a numerical model-based solution for the case when the rates are known, and this solution is more general than [13], [7], [6] since it works for arbitrarily-shaped rate functions. However, our main interest is different: we focus on the case when the robot does not know the properties of the wireless network in advance, i.e. when the rate function is unknown. Such a situation is much more common in practice.

Therefore, as the main contribution of the paper, we propose an algorithm that simultaneously learns approximations of both the rate function, from its values observed so far along the trajectory, and of an optimal-control solution that minimizes the transmission time. This algorithm belongs to the reinforcement learning (RL) class [17], but our problem is quite different from standard RL. Firstly, the robot is only given a single trajectory to transmit its data, and performance is only important along this trajectory, whereas RL usually learns from many trajectories. Secondly, whereas RL approaches often start without any model knowledge, here the motion dynamics of the robot are known, and only the rate function is unknown. Our key idea is to use the learned estimate of this function to perform local model-based updates (dynamic programming sweeps) to achieve fast single-trajectory learning. In addition, we directly exploit transition samples using a version of Q-learning. Such targeted combinations of model-based and model-free learning are rare in RL.

Some recent works like [5], [8] explore control of robots with rates that are uncertain but have a known model. In [19], the trajectory of the robot is fixed, but the velocity is optimized by learning the communication rate while the robot moves. The authors of [19] assume a much more general model for the wireless network that is suitable for practical applications, but as mentioned, consider a fixed trajectory, whereas our algorithm generates the trajectory adaptively. Indeed, to the best of our knowledge there are no methods that learn both the radio map and a near-optimal trajectory, like our algorithm.

Our model-based updates with the learned rate function are related to Dyna [16], which finds a model from samples and then applies DP updates to it. Prioritizing certain areas of the state space is related to prioritized sweeping [11], a way to focus updates on samples that are deemed more important. Our method can also be seen as reusing data in-between RL updates, and so it bears similarities with experience replay [9], which reapplies learning updates to memorized transition and reward samples, and which has recently experienced a resurgence in the field of deep RL [10]. Nevertheless, our method is unique due to the specific structure of the problem that we are considering, which allows us to focus the learning algorithm on the key unknown element: the rate function.

L. Buşoniu is with the Automation Department, Technical University of Cluj-Napoca, Romania. V.S. Varma and I.C. Morărescu are with Université de Lorraine, CNRS, CRAN, F-54000 Nancy, France. S. Lasaulce is with the Laboratoire des Signaux et Systemes (L2S, CNRS - CentraleSupelec - Univ. Paris Sud), Gif-sur-Yvette, France. Contact: lucian@busoniu.net. This work was supported by a grant of the Romanian Ministry of Research and Innovation, CNCS - UEFISCDI, project number PN-III-P1-1.1-TE-2016-0670, within PNCDI III.

Next, Section II provides a formal statement of the problem addressed, and the algorithm for known rates is given in Section III. Section IV describes the main, learning algorithm for unknown rates. Since this algorithm is empirical, Section V evaluates it in experiments on a synthetic example, while Section VI illustrates the method on a more realistic example. Section VII concludes the paper.

II. Problem definition

Consider a mobile robot with position $p \in P$ where $P \subseteq \mathbb{R}^2$. We will work in discrete-time with k denoting the time step, so the robot has motion dynamics $g : P \times U \to P$:

$$p_{k+1} = g(p_k, u_k) \tag{1}$$

where $u_k \in U$ is the control input, and U is the set of possible inputs. The robot carries a data buffer of size $b \in \mathbb{R}_+$ that it must deliver over a wireless network with a transmission rate that varies with the position, $R(p)$ with $R : P \to \mathbb{R}_+$. Here, $\mathbb{R}_+$ denotes the set of positive real numbers including zero. Therefore, the buffer size evolves like:

$$b_{k+1} = \max\{0, b_k - R(p_k)\}. \tag{2}$$

We denote the overall state by $x := [p^\top, b]^\top \in X$, $X := P \times \mathbb{R}_+$, containing the position and the buffer size. Therefore, the overall dynamics are:

$$x_{k+1} = f(x_k, u_k) := \begin{bmatrix} g(p_k, u_k) \\ \max\{0, b_k - R(p_k)\} \end{bmatrix}. \tag{3}$$

Given an initial position p_0 and buffer size b_0, which together yield the initial state x_0, the objective is to deliver the buffer in minimum time. If the energy taken by the robot to move one step is roughly constant, then this also corresponds to minimizing energy. Define now the stage reward function:

$$\rho(b) = \begin{cases} -1 & \text{if } b > 0 \\ 0 & \text{if } b = 0 \end{cases} \tag{4}$$

and the long-term value function:

$$V^h(x_0) = \sum_{k=0}^{\infty} \rho(b_k) \tag{5}$$

where $x_{k+1} = f(x_k, u_k)$ and $u_k = h(x_k)$ is taken according to the state feedback law $h : X \to U$. Then, we can restate our objective via the optimal control problem:

$$\max_h V^h(x) =: V^*(x), \ \forall x \tag{6}$$

that is, find a control law that minimizes the number of steps until the buffer size becomes zero from any initial state. We choose to (equivalently) use maximization instead of minimization since our learning methods originate in artificial intelligence, where optimal control problems are usually stated in terms of maximizing values.

Remark: The state signal chosen implies that the robot moves according to first-order dynamics. We could include other state variables like velocities, headings, etc. in x, and all the algorithms below can be extended to handle such state signals. Similarly, the reward function could be changed to include additional objectives besides transmitting the buffer, such as energy costs that vary with the motion, or navigating to a goal state. We make simple choices here in the interest of readability, and because they are sufficient to illustrate our ideas. □

III. Solution for known rate functions

If the position-dependent rate R is known, we can apply dynamic programming (DP) to solve the optimal control problem. Construct an initial value function $V_0(x) = 0, \forall x$, and then iterate for $\ell \geq 0$:[1]

$$V_{\ell+1}(x) = \max_{u \in U} \left[\rho(b) + V_\ell(f(x,u))\right], \forall x \tag{7}$$

where b is the buffer size component of state x. Note that knowledge of R is required to simulate $f(x,u)$. The algorithm is stated "forward in iterations", but can also intuitively be seen as running "backwards in time" as would usually be done in finite-horizon applications of DP. Here however, the horizon is not set in advance; instead, since the trajectory must run until the buffer is empty, the horizon until this event occurs depends on the initial buffer size and on the positions along the trajectory. We handle the problem in the infinite-horizon setting, per (5). In general, studying the convergence of such infinite-horizon DP methods is challenging since values may grow unbounded unless specific care is taken to avoid this [2], e.g. by including a discount factor. In our particular problem, with discounting the value function would no longer be the minimal number of steps to zero buffer. However, even without discounting, a simple case where a finite number of iterations is sufficient to find the optimal solution is when the rate is lower bounded by some value $\underline{R}$ (which may represent a minimum quality-of-service requirement) at every p, i.e., $R(p) \geq \underline{R} > 0$, and when $b_0 \leq \bar{b}$. Under this assumption, the buffer will be emptied in at most $\bar{b}/\underline{R}$ step from any initial state. After stopping the algorithm at finite iteration $\bar{\ell}$ larger than this number of steps, we apply the state feedback:

$$h(x) \in \arg\max_{u \in U} \left[\rho(b) + V_{\bar{\ell}}(f(x,u))\right] \tag{8}$$

with ties between maximizing actions resolved arbitrarily.

In general, the algorithm is not implementable as given above, for several reasons: the maximization over u is a possibly nonconcave and nondifferentiable global optimization problem, V cannot be exactly represented in closed form for continuous arguments x, and the rates may be zero at some positions. Below we describe some empirical solutions to these issues, which are rather standard in the field of approximate dynamic programming [1], [17]. First, we assume that U consists of a finite, discrete set of actions, and solve maximization by enumeration. Second, we assume P is bounded and rectangular and that $b \in [0, \bar{b}]$ (ensured by the condition $b_0 \leq \bar{b}$ already discussed above), which

[1] Subscript ℓ in V_ℓ denotes the iteration index, whereas the superscripts used earlier denote either the dependence on the policy h, in V^h, or the particular case of the optimal policy, in V^*.

is reasonable in a practical application. We then represent V approximately, using multilinear interpolation over grids defined along the interval domains of each of the state variables. Denoting the approximate value function by $\widehat{V}$, this representation can be written:

$$\widehat{V}(x;\theta) = \varphi^\top(x)\theta \tag{9}$$

where $\theta \in \mathbb{R}^n$, $\varphi : X \to \mathbb{R}^n$, and n is the total number of points on the grid. Here, θ_i is the parameter associated with point i and $\varphi_i(x)$ is the weight with which point i participates to the approximation, which is easy to obtain from the interpolation procedure. Note that in fact $\varphi(x)$ will be sparse, 0 for most i; indeed the maximal number of points participating to an interpolated value is $2^3 = 8$. However, writing the approximation as (9) highlights that it is a particular type of basis function expansion, and our approach may later be generalized to other such expansions.

Noticing that at point x_i of the grid, $V(x_i) = \theta_i$ since the vector $\varphi(x_i)$ is 1 at position i and 0 everywhere else, an approximate version of (7) can be given:

$$\theta_{\ell+1,i} = \max_{u\in U}\left[\rho(b) + \widehat{V}(f(x_i,u);\theta_\ell)\right], \forall i \tag{10}$$

where vector θ_0 is initialized to zero values.

To circumvent the need to fix the number of iterations in advance, the algorithm is stopped when $\|\theta_{\ell+1} - \theta_\ell\|_\infty \leq \varepsilon$. Finally, a control law is computed with an equation similar to (8) but using $\widehat{V}(\cdot;\theta_{\ell+1})$ on the right hand side.

The approximations used imply that the optimality of the solution is lost. Nevertheless, in practice the accuracy can be increased by making the state interpolation grids and the action discretization finer, and ε smaller. A discounted version of such an interpolated DP algorithm has been analyzed in [3].

IV. Learning algorithm for unknown rate functions

The DP algorithm above requires to know the rate function R, which is usually not possible because R depends on propagation environment effects which are typically unknown (e.g., path loss, shadowing, and fast fading effects). Even if it is possible to model R, the robot may not be provided with the model. Our major goal in this paper is therefore to derive an efficient algorithm for the case when the rate function R is unknown. Rather, we will assume that the robot only has access to realizations of the rate function at particular positions, which may typically be measured via a feedback mechanism (ACK/NACK or more advanced feedbacks such as the signal-to-noise ratio). The robot can therefore accurately sample $R(p_k)$ once it reaches position p_k and can use this information to make decisions at step k.

The problem of learning optimal control solutions when the dynamics (and possibly even the rewards) are unknown is the focus of the large field of reinforcement learning (RL) [17]. However, our problem is quite different from standard RL, and the main contribution of our paper stems from the differences. The first difference is that, while the typical paradigm is that RL is applied across many trajectories, seeing the same states over and over again, here we cannot afford to wait several trajectories for good performance: indeed, the robot is only given a single trajectory to transmit its data. Performance is only important during this trajectory, and for only those states encountered along it, most of which will be seen only once. The second difference is that we have significant information about the model: with the exception of R, everything is known in f in (3). Our key idea is to exploit the second difference in order to address the challenge stemming from the first; that is, to learn R directly and use its estimate in f to achieve fast, single-trajectory learning. Such targeted combinations of model-based and model-free learning are rare in RL, since RL approaches nearly always aim to solve the model-free problem in its full generality.

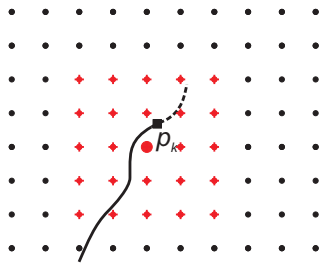

Fig. 1: Illustration of local DP sweeps and related concepts across the two position axes of the state. The grid is denoted by dots, and the current position of the robot by a square. The center of the subgrid (large circle) is the nearest grid point below and to the left of the current position, and the subgrid (red crossed circles) extends $r_{DP} = 2$ points in each direction from this center point. The prior trajectory of the robot is the continuous line, and the dashed line illustrates a possible trajectory across several future steps.

Denote by $\widehat{R}$ the estimate of R, which can be constructed from the samples $(p_j, R(p_j))$, $j \leq k$ seen so far using any function approximation (supervised learning) technique. Before taking a decision at step k, we propose to use $\widehat{R}$ in order to run several DP sweeps of the form (10), but only *locally*, around state x_k. Figure 1 illustrates the idea. A simple reason for these local updates is to reduce computational costs, since a decision must be made online. A deeper motivation however is to avoid extrapolating too much from the samples of R seen so far, which are all probably behind the robot along its trajectory, and not in the direction that it needs to go; and for the same reason, one cannot hope anyway for a decision that is good across more than a few steps – i.e., for smooth dynamics, more than a small distance away in the state space. Indeed, it is likely better to wait until more information is available before attempting to construct such a decision.[2] Constructing the local region around x_k (over which to perform the DP sweeps) in an "optimal" way is difficult. Instead, in this paper, we simply take a subgrid

[2]This is also a key feature of receding-horizon predictive control, so one may wonder why this framework is not applied here. In fact, we have tried a receding-horizon method based on tree search, but it performed poorly.

consisting of r_{DP} grid points to either side of the current state along all 3 dimensions (2 positions and 1 buffer size). The DP range r_{DP}, together with the number of DP sweeps ℓ_{DP}, are tuning parameters of the algorithm.

In addition to the DP sweeps, we will also use a variant of the popular RL algorithm called Q-learning [18] to learn directly from the transition samples seen along the trajectory. Q-learning usually works with Q-functions $Q : X \times U \to \mathbb{R}$, that fix the initial action in addition to x:

$$Q(x, u) = \rho(b) + V(f(x, u)). \tag{11}$$

This is necessary in standard RL because nothing is known about f. However, in our case, once we reach p we observe $R(p)$, and since the motion dynamics g are known, we have all the knowledge required to simulate f. At the cost of some extra calls to f, this allows us to drop the u dimension from the function that must be learned. In particular, for approximate representations of the form (9), we may derive a so-called semi-gradient [17] version of Q-learning adapted to V-functions:

$$\theta_{k+1} = \theta_k + \alpha_k \varphi(x_k) \cdot \cdot \left[\max_u [\rho(b_k) + \widehat{V}(f(x_k, u); \theta_k)] - \widehat{V}(x_k; \theta_k)\right] \tag{12}$$

While in our single-trajectory setting such a learning procedure will not converge, we expect the updates are still useful to extract additional information from the trajectory data.

A remaining question is how exactly the robot chooses actions at each step. One possibility is to simply apply (8) but with the current $\widehat{V}(\cdot; \theta_k)$ instead of the optimal value-function. This is called the greedy policy. Technically, it does not satisfy the exploration conditions of RL – similarly to persistent excitation in system identification, informative, so-called exploratory actions must be taken in RL to avoid getting stuck in local optima. However, if it is coupled with an optimistic initialization of V (via the parameters θ_0) to some $\bar{V}$ that is larger than the optimal values, then the greedy policy can still work well; intuitively, optimistic initialization will force the algorithm to explore the space of solutions anyway since it believes any unknown solution to be good. Here, we will create this optimistic initial solution by assuming knowledge of the maximal rate $\bar{R}$, and then initializing the parameter $\theta_{0,i}$ for each grid point x_i to $-b_i/\bar{R}$. If $\bar{R}$ is unknown, then θ_0 could be taken 0.

With the same initialization, we will also investigate an explicitly exploratory policy commonly used in RL, called softmax or Boltzmann-Gibbs policy. This policy selects actions randomly, where each action $u \in U$ has a probability π_u related to its Q-values (11) via the formula:

$$\pi_u = \frac{e^{Q(x_k,u)/\tau}}{\sum_{u' \in U} e^{Q(x_k,u')/\tau}}, \quad \forall u \in U \tag{13}$$

where parameter $\tau > 0$ is called the exploration temperature. This temperature controls the tradeoff between exploration and exploitation, with larger values corresponding to more exploration; the fully exploiting, greedy policy is obtained in the limit as $\tau \to 0$.

Algorithm 1 Learning for the communicating robot.

Input: g, $\bar{R}$, state grids, discretized actions U, learning rate α, temperature τ, r_{DP} and ℓ_{DP} for DP sweeps

1: initialize params $\theta_{0,i} = -b_i/\bar{R}$ for all grid centers x_i
2: measure initial state x_0
3: **repeat** at each time step $k = 0, 1, 2, \ldots$
4: sample $R(p_k)$, update approximator $\widehat{R}$
5: $\tilde{\theta}_0 = \theta_k$, and construct DP subgrid around x_k
6: **for** DP sweep $\ell = 0, \ldots, \ell_{DP} - 1$ **do**
7: **for** each point i on the subgrid **do**
8: $\tilde{\theta}_{\ell+1,i} = \max_{u \in U} [\rho(b) + \widehat{V}(\widehat{f}(x_i, u); \tilde{\theta}_\ell)]$
9: **end for**
10: **end for**
11: $\theta_k = \tilde{\theta}_{\ell_{DP}}$
12: $Q(x_k, u) = \rho(b) + \widehat{V}(f(x_k, u)); \theta_k), \ \forall u \in U$
13: **if** using greedy policy **then**
14: $u_k = \arg\max_{u \in U} Q(x_k, u)$
15: **else** (using softmax policy)
16: sample u_k using probabilities $\frac{e^{Q(x_k,u)/\tau}}{\sum_{u'} e^{Q(x_k,u')/\tau}}$
17: **end if**
18: apply action u_k, measure next state x_{k+1}
19: $\theta_{k+1} = \theta_k + \alpha_k \varphi(x_k)[\max_u Q(x_k, u) - \widehat{V}(x_k; \theta_k)]$
20: **until** $b_{k+1} = 0$

Algorithm 1 summarizes the overall procedure. Note that in line 8, the approximate model $\widehat{f}$ uses $\widehat{R}$.

So far, the approximator $\widehat{R}$ has been left unspecified; again, in principle any sample-based function approximator can be used. For our experiments, we will use local linear regression (LLR). To this end, we store each pair $(p_k, R(p_k))$ that was not yet seen in a memory. Then, for each query position p, the K nearest neighbors of p are found using the Euclidean norm, and linear regression on these neighbors is run to find an affine approximator of the form $a^\top p + b$ with $a \in \mathbb{R}^2, b \in \mathbb{R}$. This approximator is then applied to find $\widehat{R}(p)$. The tuning parameter of LLR is K.

V. Empirical study of the learning algorithm

We provide detailed simulations for a synthetic example involving a simple-integrator robot:

$$p_{k+1} = p_k + u_k. \tag{14}$$

with $u \in U$, a set of 5 discrete actions chosen to move the robot on a grid, one step in any cardinal direction, or keep it put. The domain $P = [0, 10] \times [0, 10]$, and $b \in [0, \bar{b}] = [0, \bar{b}]$, with the bounds enforced by saturation. The grid is defined to have 21 points on both position axes and on the buffer size axis of the state space, and we will use this grid in our algorithms as well. Note that if the environment is large (e.g., compared to the turning radius of a wheeled robot), a simple-integrator model is not unrealistic, and in fact such models are often used in e.g., consensus theory [12].

The rate function consists of two Gaussians, see also

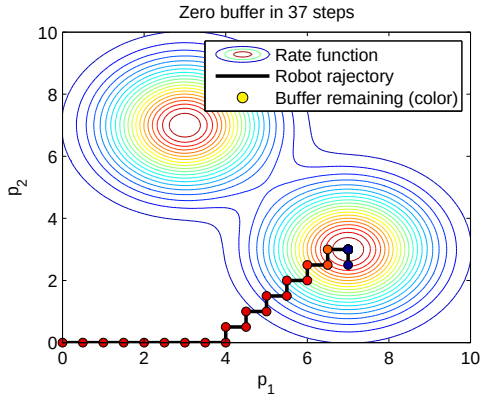

Fig. 2: Model-based control from the bottom-left corner. The contour plot shows the rate function, and the position of the robot at each sampling time is shown by a colored disk. To better follow the trajectory of the robot, these positions are joined by a black line. The color of the disk indicates the remaining buffer size, from dark red (full) to dark blue (empty).

Figure 2 that includes a contour plot:

$$R(p) = 0.1 \sum_{i=1}^{2} \exp\left[-(p - c_i)^\top W (p - c_i)\right] \qquad (15)$$

with $W = \mathrm{diag}[3, 3]$ the radius of the Gaussians and $c_1 = [3, 7]^\top$, $c_2 = [7, 3]^\top$ their centers.

We will study the performance of the algorithm from the bottom-left initial position $p_0 = [0, 0]^\top$, which is furthest away from the rate maxima, and with a full initial buffer, $b_0 = \bar{b} = 2$, as that is the most interesting scenario. It is unclear whether exact optimal solutions can be computed for such a problem, so to obtain a baseline we run instead model-based, interpolative value iteration from Section III. The resulting near-optimal number of steps required to empty the buffer is 37, and Figure 2 illustrates the trajectory.

In preliminary experiments with the learning algorithm of Section IV, which we do not detail here due to space limits, we found the following. For the learning rate α, a value of 1 is best for most settings, which is intuitive since the problem is deterministic and $\alpha < 1$ usually helps for stochastic dynamics. For the exploration temperature τ in softmax, $\tau = 0.1$ works best for most settings (which is close to 0, so action selection is "almost greedy"). For the number of iterations ℓ_{DP} in the DP sweeps, any value above 10 works well; we therefore take it 10. For the number of nearest neighbors in LLR, $K = 4$ works best.

With the above settings, next we vary the range r_{DP} of the DP sweeps, gradually from 0 (which means that the DP sweeps are disabled and pure model-free RL is performed) to 6. Figure 3 reports the results for both the greedy and softmax policies, where for the latter 10 independent experiments were run and mean performances with their 95% confidence intervals are given. The buffer is generally emptied in fewer steps as the DP range grows, showing that the DP sweeps are indeed a useful way to exploit the learned $\widehat{R}$. Indeed, performance is very poor for $r_{DP} = 0$ (no DP sweeps),

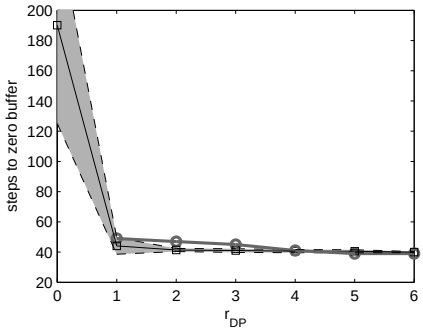

Fig. 3: Learning with the greedy policy (thick gray line with round markers) and with the softmax policy (thin black line with square markers). For the softmax curve, the gray areas are the 95% confidence regions on the mean. Note that for $r_{DP} = 0$, the greedy policy did not succeed in emptying the buffer, so the graph does not include this point.

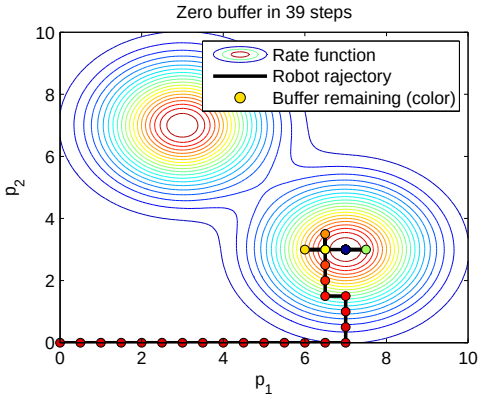

Fig. 4: Model-free control from the bottom-left corner.

showing that a pure model-free RL algorithm does not work well in our single-trajectory setting. The performance of the two exploration strategies is quite similar for $r_{DP} \geq 1$, and for large r_{DP}, the algorithm empties the buffer in about 40 steps, close to the model-based number. This is a very good result, keeping in mind that the rate function must be learned *at the same time as* using it to transmit.

On the flip side, increasing the DP range requires of course larger computational costs, here roughly cubical in r_{DP} due to the three-dimensional state space.

Figure 4 illustrates the trajectory with the greedy policy for $r_{DP} = 6$. Note that, in contrast to the model-based solution from Figure 2, which goes along the shortest path towards the maximum rate (since it knows where it is), the learning algorithm first looks around to observe samples from R and build its estimate, and as this estimate becomes better and the buffer gets smaller, it goes near a maximum to finish transmitting.

Stochastic rates are typical in many problems, so we briefly check how our algorithm handles them. The problem is changed so that the deterministic rate function from before is affected by additive zero-mean Gaussian noise with a standard deviation of 0.01 (i.e., about 10% of the rate function magnitude). The greedy algorithm is run with the settings above and $r_{DP} = 4$, chosen because it provides a reasonable

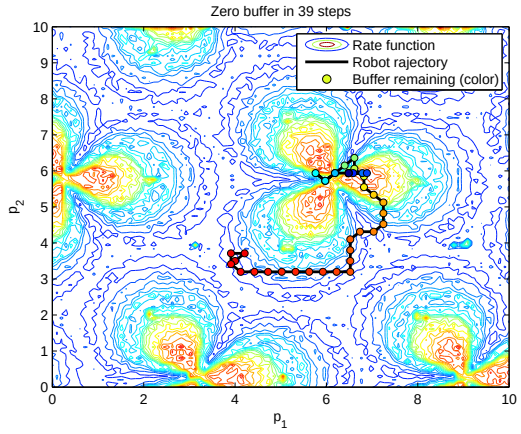

Fig. 5: Model-free control in the realistic-rate problem.

compromise between performance and computational cost. The resulting number of steps to empty the buffer is 42.5 on average, with a 95% confidence interval of 42.5 ± 2.8063. This is not far from the deterministic result, illustrating that the algorithm is in this case resilient to noise.

VI. Illustration for a realistic rate function

The second problem is chosen to have a more realistic rate function, inferred from the setup used in Fig. 5 of [14] and rescaled to a position domain $P = [0, 10] \times [0, 10]$ m. The motion dynamics are also changed to be nonlinear, unicycle-like:

$$\begin{aligned} p_{k+1,1} &= p_{k,1} + u_{k,1} \cos(u_{k,2}) \\ p_{k+1,2} &= p_{k,2} + u_{k,1} \sin(u_{k,2}) \end{aligned} \tag{16}$$

i.e., the first input is the velocity and the second the heading of the robot. The discretized actions consist of all possible combinations between velocities 0.1, 0.3 m/s and headings $0, \pi/4, \ldots, 7\pi/4$ rad; together with a 0-velocity action. To illustrate the robustness of the algorithm, we do not retune any parameter but just use the values from the integrator problem: a $21 \times 21 \times 21$ interpolation grid, $\alpha = 1$, $\tau = 0.1$, $\ell_{DP} = 10$, and $K = 4$.

Figure 5 illustrates the trajectory with the greedy policy from initial position $[5, 3]^\top$ and an initial buffer size of 80Mb (megabits, i.e. 8 megabytes). Note that the rate function varies roughly between 0.5 and 5 Mb/s. The buffer is emptied in 39 steps, some of which are spent "skirting" the lobes of the rate function; we hypothesize this helps to learn it, but further study is needed to confirm that. Note that the model-based solution with known rates empties the buffer in 27 steps, so the learning algorithm manages to work in less than double this number of steps.

VII. Conclusions

We have presented a learning-based algorithm that a mobile robot can use to transmit data over a wireless network with an unknown rate map. The algorithm was evaluated in experiments with simple-integrator robot motion dynamics and a synthetic rate function with two maxima (antennas); and was also illustrated to work well in a problem with realistic rates and unicycle-like, nonlinear motion dynamics.

In future work it will be important to derive analytical guarantees that take into account the approximation errors for V and R. Handling stochastic rates algorithmically is also needed. Finally, it will help to validate the method in a practical experiment.

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Athena Scientific, 2012, vol. 2.

[2] D. P. Bertsekas and S. E. Shreve, *Stochastic Optimal Control: The Discrete Time Case*. Academic Press, 1978.

[3] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Approximate dynamic programming with a fuzzy parameterization," *Automatica*, vol. 46, no. 5, pp. 804–814, 2010.

[4] N. Chatzipanagiotis, Y. Liu, A. Petropulu, and M. M. Zavlanos, "Controlling groups of mobile beamformers," in *Proceedings 51st IEEE Conference on Decision and Control (CDC)*, Maui, Hawaii, 10–13 December 2012, pp. 1984–1989.

[5] J. Fink, A. Ribeiro, and V. Kumar, "Robust control for mobility and wireless communication in cyber–physical systems with application to robot teams," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 164–178, 2012.

[6] R. Gangula, P. de Kerret, O. Esrafilian, and D. Gesbert, "Trajectory optimization for mobile access point," in *51st Asilomar Conference on Signals, Systems, and Computers*, Oct 2017, pp. 1412–1416.

[7] D. B. Licea, V. S. Varma, S. Lasaulce, J. Daafouz, and M. Ghogho, "Trajectory planning for energy-efficient vehicles with communications constraints," in *Proceedings 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM16)*, Fez, Morocco, 26–29 October 2016, pp. 264–270.

[8] D. B. Licea, V. S. Varma, S. Lasaulce, J. Daafouz, M. Ghogho, and D. McLernon, "Robust trajectory planning for robotic communications under fading channels," in *Ubiquitous Networking: Third International Symposium, UNet 2017, Casablanca, Morocco, May 9-12, 2017, Revised Selected Papers*, vol. 10542. Springer, 2017, p. 450.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3–4, pp. 293–321, Aug. 1992, special issue on reinforcement learning.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[11] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[12] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[13] C. C. Ooi and C. Schindelhauer, "Minimal energy path planning for wireless robots," *Mobile Networks and Applications*, vol. 14, no. 3, pp. 309–321, 2009.

[14] P. Pietraski, G. Charlton, R. Yang, and C. Wang, "Enhanced cell-edge performance with transmit power-shaping and multipoint, multiflow techniques," *ZTE Communications*, no. 4, 2011.

[15] M. N. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Engineering Practice*, vol. 15, no. 4, pp. 435–445, 2007.

[16] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings 7th International Conference on Machine Learning (ICML-90)*, Austin, US, 21–23 June 1990, pp. 216–224.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., ser. Adaptive Computation and Machine Learning. A Bradford Book, 2018.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[19] Y. Yan and Y. Mostofi, "Co-optimization of communication and motion planning of a robotic operation under resource constraints and in fading environments," *IEEE Transactions on Wireless Communications 12.4 (2013):*, vol. 12, no. 4, pp. 1562–1572, 2013.