ME41125 Introduction to Engineering Research (2020/21 Q4)
R. Varga, Delft, The Netherlands, compiled May 24, 2021

Reinforcement Learning Based Motion Control of Spherical Robot

Roland Varga

***Abstract*—This work presents a SARSA reinforcement learning applied on a spherical robot position tracking controller. The controller was trained and evaluated in a simulation environment, using the OpenAI Gym Python package (custom environment). The model used for the simulation was based on an existing continuous-time state-space model with 4 states.**

I. Introduction

Spherical robots have several advantages compared to wheeled or legged robots in certain applications. They can be deployed on rough terrains thanks to their robust structure. Also they make better robot toys for children because of the lack of outer moving parts and the potentially softer outer shell.

Although the actuation of spherical robots is usually relatively simple, the model and control of such systems can get complex due to the non-linear nature of the design and the modelling constraints (e.g. no slipping). Ohsawa [1] derived the kinematic model for an already commercially available robot, the Sphero (Figure 1), which this project is based on.

Fig. 1. Spherical rolling robot Sphero®; see http://www.sphero.com

One of the main challenges in the control of these robots is the position and trajectory tracking. The desired/optimal trajectory is often precomputed and the tracking is implemented using a feedback controller. However, because of the aforementioned non-linear system model this can be challenging. Model-free reinforcement learning algorithms provide an alternative solution.

For example, for robotic arm force control Perrusquia et al. [2] replaced the model of the system with a learnt environment. The stability and convergence of the control is also discussed in the paper as it is a crucial point when designing reinforcement learning algorithms. The performance of a physical robot system is simulated and compared using LQR

The author is with the institute of TU Delft r.varga@student.tudelft.nl

Kober et al. [3] created a survey which discusses the different types of developed reinforcement learning algorithms. It also lists several advantages, disadvantages and "common pitfalls" for the algorithms. The reduction of required training time and the convergence of the algorithms are also described.

Reinforcement learning algorithms were also developed for mobile robot navigation. Altuntas et al. [4] implemented a SARSA algorithm which enabled a robot with unknown dynamics to navigate in an unstructured environment (potentially with obstacles) between specified points.

The motion planning of a spherical robot using reinforcement learning is discussed in the work of Roozegar et al. [5]. They used a so called eXtended Classifier System (XCS) algorithm to control the robot to achieve trajectory tracking with defined points in a 2D space. The objective of this paper is to develop a SARSA learning algorithm for the same purpose as the previously mentioned paper: position tracking with a spherical rolling robot. The training is executed in a simulation environment using the model from [1]. The environment is implemented in OpenAI Gym as a custom environment.

II. Methods

A. The system model for training

The controller was trained in a simulation environment. For that the system model proposed by Ohsawa [1] was used.Since the objective of this control is to regulate only the position of the center of the spherical robot only part of the state-space model was used. For the derivation of the model and the applied constraints and limitations consult the paper by Ohsawa [1].

$$\dot{\mathrm{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \frac{r\rho}{2h}\left(\dot{\varphi}_1 + \dot{\varphi}_2\right) \begin{bmatrix} -\sin\left(c\left(\varphi_1 - \varphi_2\right)\right) \\ \cos\left(c\left(\varphi_1 - \varphi_2\right)\right) \end{bmatrix} \tag{1}$$

In this continuous-time state-space model x_1 and x_2 are the position coordinates of the center of the spherical robot in the 2D space (Figure 2). The φ_1 and φ_2 are the angle coordinates of the 2 inner wheels as shown on Figure 2. The r, ρ, h and c are system parameters, their description can be found in [1]. The inputs of the system are the angular velocities of the inner wheels.

$$u_1 = \dot{\varphi}_1 \quad u_2 = \dot{\varphi}_2 \tag{2}$$

If we consider discrete-time control of the system with a zero-order hold applied for the input then the discrete-time description of the system can be easily obtained by integration. The detailed derivation can be found in the Appendix.

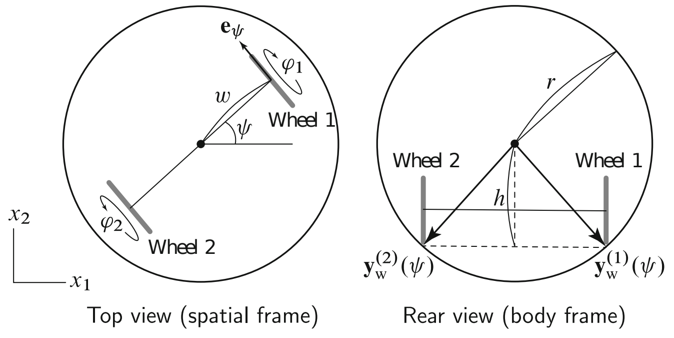

Fig. 2. Coordinates of the model [1]

$$
\begin{aligned}
c_1(k) &= \frac{r\rho}{2h}\left(\dot\varphi_1(k)+\dot\varphi_2(k)\right)\\
c_2(k) &= c\left(\varphi_1(k)-\varphi_2(k)\right)\\
c_3(k) &= c\left(\dot\varphi_1(k)-\dot\varphi_2(k)\right)
\end{aligned}
$$

$$
\begin{bmatrix} x_1(k+1)\\ x_2(k+1)\\ \varphi_1(k+1)\\ \varphi_2(k+1)\end{bmatrix} =
$$

$$
= \begin{bmatrix} x_1(k)+\dfrac{c_1(k)}{c_3(k)}\Big(\cos\left(c_2(k)+c_3(k)T_s\right)-\cos\left(c_2(k)\right)\Big)\\ x_2(k)+\dfrac{c_1(k)}{c_3(k)}\Big(\sin\left(c_2(k)+c_3(k)T_s\right)-\sin\left(c_2(k)\right)\Big)\\ \varphi_1(k)+\dot\varphi_1(k)T_s\\ \varphi_2(k)+\dot\varphi_2(k)T_s \end{bmatrix} \tag{3}
$$

In the equations T_s represents the sampling time. For the control problem the position origin is placed at the location of the desired position and the x_2 axis points in the direction of the front of the robot at the initial time step. This has some advantages compared to placing the origin on the robot starting position. This is described in more detail in the following subsections.

B. Creating the interactive environment

For the creation of the reinforcement learning environment the OpenAI Gym[1] was used. The system was defined as a custom environment (TODO: see the GitHub link). The states of the system are restricted:

$$
\begin{bmatrix} -6\\ -6\\ \pi/c\\ \pi/c\end{bmatrix} \le \begin{bmatrix} x_1(k)\\ x_2(k)\\ \varphi_1(k)\\ \varphi_2(k)\end{bmatrix} \le \begin{bmatrix} 6\\ 6\\ \pi/c\\ \pi/c\end{bmatrix}, \quad \forall k \tag{4}
$$

The restriction on the inner wheel angle was ensured by a wrapping function. The reason why the angle limits are not $-\pi$ and π but they are scaled by $1/c$ is that both angles are multiplied in the state equations by c inside the trigonometric functions. This means that the periodicity of this function with respect to one of the variables is also scaled with c.

[1]https://gym.openai.com/

The reward function was defined to give a small negative reward if the robot gets closer to the origin by taking a step and a bigger negative reward otherwise. A minimum necessary step size was tuned during experiments to ensure convergence. The reason for the negative rewards was to 'encourage' the control to try and bring the robot to the origin as soon as possible. Additionally, a high positive reward was given if the robot reaches a small neighbourhoodof the origin. This case the episode terminates. Finally, to ensure that the robot does not leave the defined position range a high negative reward is given if the robot takes a step which result in a position being out of the range of $[-5,5]$.

At the beginning of an episode the robot is initialized (with uniformly distributed probability) on a position outside the defined small circular neighbourhood of the origin but inside the $[-5,5]$ range. The initial inner wheel angles of the robot arealways 0 for both angles and the robot is facing in the x_2 direction. This does not restrict the applicability of the controller since the wheel angle has no global coordinate and the initial angle can always be considered as reference. Also as already mentioned in the previous subsection the x_2 axis was defined at the initial time of the control to point in the direction of the robot heading axis.

Finally, the environment has discrete action-space. Both wheels can have a predefined, constant positive or negative angular velocity or it can be 0 (for the duration of the sampling time).

$$
(\dot\varphi_1(k),\dot\varphi_2(k)) \in \{-\omega_c,0,+\omega_c\}\times\{-\omega_c,0,+\omega_c\} \tag{5}
$$

C. Design of the SARSA learning components

Since the action-space is continuous the first design choice during implementation is the discretization. The position coordinates were both distributed into 6-6 equal parts as shown in Figure 3.

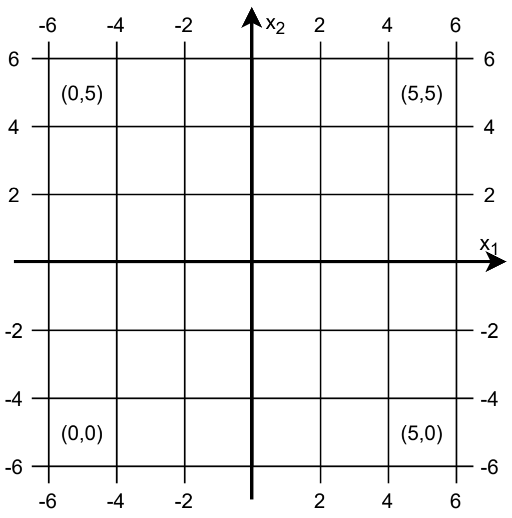

Fig. 3. The discretized state-space

To reduce the number of states in the controller instead of the 2 wheel angle coordinates the subtraction of φ_2 from φ_1 is used. The main idea with this simplification is that the difference in the wheel angles will contain information on the current heading direction of the robot (if the change in both angles is the same then the robot moves forward without turning). This difference will be in the range of $[-2\pi/c, 2\pi/c]$ which is distributed into 20 equal intervals.

The action-space is already discrete. To enable exploration ϵ-greedy method is used for the action selection. The ϵ value was chosen to be a constant ϵ which means that the policy will choose the greedy action with probability $1-\epsilon$ or a random action with probability ϵ.

To further 'encourage' exploration, the Q-table of the algorithm is initialized with ones instead of zeros as following:

$$Q_{init} = J_{6,6,20,3,3} \in \mathbb{R}^{6\times6\times20\times3\times3},$$

where J only contain ones. The update equation of the Q-table is according to the Bellman equation:

$$\begin{aligned} Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha\big(r_{k+1} + \gamma Q(s_{k+1}, a_{k+1}) \\ -Q(s_k, a_k)\big) \end{aligned} \tag{6}$$

, where s_i denotes the state, a_i the action, r_{k+1} the obtained reward by executing an action, α the learning rate and γ the discount factor.

The controller was trained in Python 3.6, using the OpenAI Gym package. The learning parameters were defined as $\alpha = 0.1$, $\gamma = 0.99$ and $\epsilon = 0.3$. The maximum length of an episode was set to 300 steps and the number of episodes for the training was 60000. The robot parameters (same as in [1]): $\rho = 0.3[m]$, $h = 0.75[m]$, $r = 1[m]$, $w = 0.8[m]$, $I_s/J = 0.2[-]$, $T_s = 0.1[s]$. The parameter c can be derived from the other parameters [1].

III. Results

On a laptop with an Intel Core i7-8750H processor the learning took around 68 minutes. Figure 4 shows the reward achieved in the episodes (smoothed) as the learning process progressed.

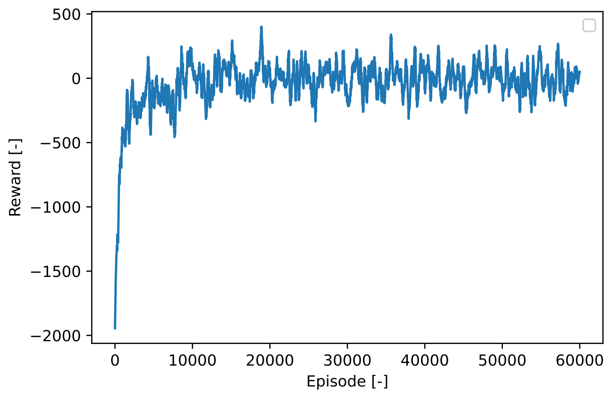

Fig. 4. The improvement in the reward obtained during learning (smoothed)

Figure 5 shows an example how the robot is controlled from the $(-2.87, -0.14)$ initial location to the neighbourhood of the origin in 47 steps. The flat parts in the plot of the position coordinates is when the robot was only rotating but did not make translational movement.

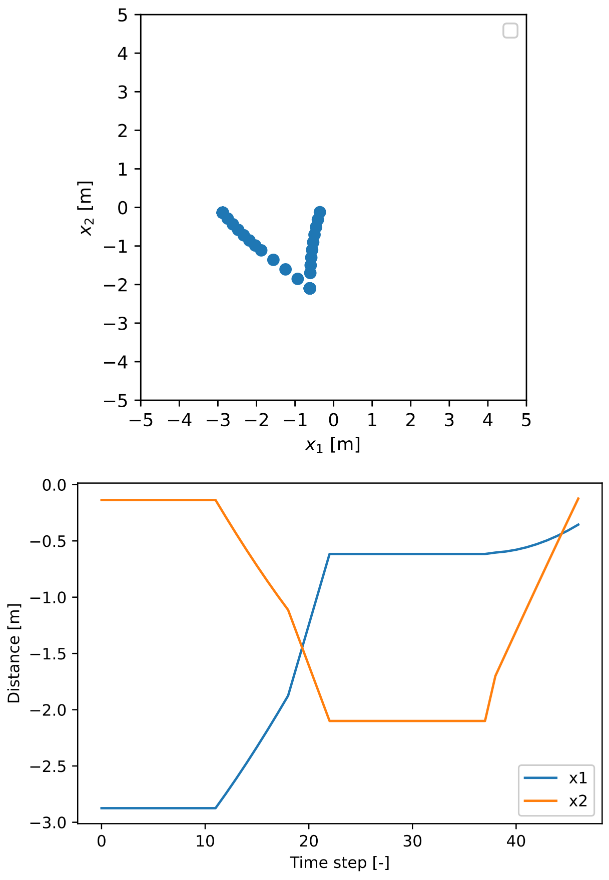

Fig. 5. A successful position control

A test of 1000 episodes after learning showed that the robot only reaches the origin around 50% of the runs. The state-value function is shown on Figure 6 (only the position states), which provides a ground for further discussion about the effectiveness of the control in the next section. The plot shows the maximum expected reward at a given position cell among all possible orientations when choosing the greedy action.

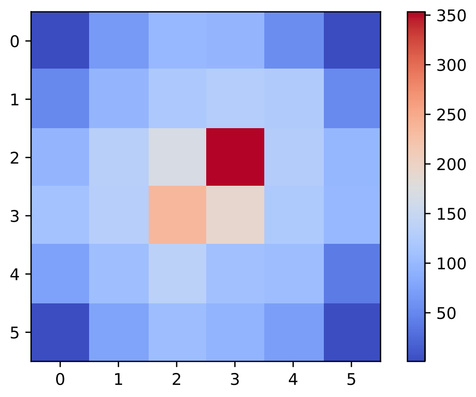

Fig. 6. The state-value function (only the position states)

Also manual runs and observations showed that when the robot does not reach the origin it still gets closer to it, except if it was initialized close to the borders.

IV. Discussion

The state-value function (Figure 6) showed that the highest rewards can be achieved when the robot is close to the origin (as expected), although there is difference between the neighbouring fields. If the robot is initialized close to the border of the position map the control is less effective. This can be explained by the lack of "experience" in those regions and also by the fact that high negative reward is given if the robot leaves the $[-5,5]\times[-5,5]$ region. This is also supported by the manual checks: the robot is less certain when it is initialized close to the borders.

Possible future work includes the investigation of the influence of different state discretizations, e.g. the modification of the cell ranges to have separate states for the positions outside the $[-5,5]\times[-5,5]$ region. Also the integration of the Hindsight Experience Replay (HER) method described in the systematic review work of Kober et al. [3] could help reduce the learning time as it handles the data more efficiently.

V. Conclusion

The SARSA reinforcement learning algorithm has the most benefit in applications where the state- and action-spaces has only a few elements. In this seemingly simple example project the learning time quickly increased by the refinement of these spaces. Also, although there are some general guidelines for the tuning of the learning parameters, it is still heuristic and requires experience.

VI. Data Availability Section

The data analysis scripts/code that are used in this study are available in the corresponding GitHub repository at the following link.

Acknowledgments

Many thanks to the organizers of the Introduction to Engineering Research subject at TU Delft for the provided paper template, scientific writing and research recommendations and technical support.

References

[1] T. Ohsawa, "Geometric Kinematic Control of a Spherical Rolling Robot," *Journal of Nonlinear Science*, vol. 30, no. 1, pp. 67–91, 2 2020.

[2] A. Perrusquía, W. Yu, and A. Soria, "Position/force control of robot manipulators using reinforcement learning," *Industrial Robot*, vol. 46, no. 2, pp. 267–280, 3 2019.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 9 2013.

[4] N. Altuntas, E. Imal, N. Emanet, and C. N. Öztürk, "Reinforcement learning-based mobile robot navigation," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 24, no. 3, pp. 1747–1767, 2016.

[5] M. Roozegar, M. J. Mahjoob, M. J. Esfandyari, and M. S. Panahi, "XCS-based reinforcement learning algorithm for motion planning of a spherical mobile robot," *Applied Intelligence*, vol. 45, no. 3, pp. 736–746, 10 2016.

Appendix

A. Derivation of the discrete-time model

Since the position equations are not coupled

$$x_1(t_0+T_s) = x_1(t_0) + \int_{t_0}^{t_0+T_s} -\frac{\rho}{2h}(\dot\varphi_1+\dot\varphi_2)\cdot \sin(c(\varphi_1-\varphi_2))\,\mathrm{d}t \tag{7}$$

Now let's assume that the t_0 is the starting time of a sampling period. This means that in the integral the $(\dot\varphi_1+\dot\varphi_2)$ term is constant (discrete-time control with zero-order hold).

$$c_1 = \frac{\rho}{2h}(\dot\varphi_1(t)+\dot\varphi_2(t)), \quad \forall t\in[t_0,t_0+T_s) \tag{8}$$

$$x_1(t_0+T_s) = x_1(t_0) - c_1\int_{t_0}^{t_0+T_s}\sin(c(\varphi_1(t)-\varphi_2(t)))\,\mathrm{d}t \tag{9}$$

The trajectory of the wheel angles is known for a given time step, since the input, the derivative of this angle is determined at the beginning of the sample period.

$$\varphi_1(t) = \varphi_1(t_0) + \dot\varphi_1\cdot(t-t_0), \quad \forall t\in[t_0,t_0+T_s) \tag{10}$$

$$\varphi_2(t) = \varphi_2(t_0) + \dot\varphi_2\cdot(t-t_0), \quad \forall t\in[t_0,t_0+T_s) \tag{11}$$

Substituting these to the integral in Equation 9 and by applying change of variables ($\tau = t-t_0$) we obtain:

$$c_1 = \frac{\rho}{2h}(\dot\varphi_1(t)+\dot\varphi_2(t)), \quad \forall t\in[t_0,t_0+T_s) \tag{12}$$

$$c_2 = c(\varphi_1(t_0)-\varphi_2(t_0)) \tag{13}$$

$$x_1(t_0+T_s) = x_1(t_0) + c_1\int_0^{T_s} -\sin(c_2 + \underbrace{c(\dot\varphi_1-\dot\varphi_2)}_{c_3}\tau)\,\mathrm{d}\tau \tag{14}$$

Computing the integral result in:

$$x_1(t_0+T_s) = x_1(t_0) + c_1\left[\frac{\cos(c_2+c_3\tau)}{c_3}\right]_0^{T_s} \tag{15}$$

$$= x_1(t_0) + \frac{c_1}{c_3}(\cos(c_2+c_3T_s)-\cos(c_2)) \tag{16}$$

With discrete-time notation:

$$x_1(k+1) = x_1(k) + \frac{c_1(k)}{c_3(k)}\Big(\cos\big(c_2(k)+c_3(k)T_s\big) - \cos\big(c_2(k)\big)\Big) \tag{17}$$

The reformulation of Equation 10 and 11 to discrete-time:

$$\varphi_1(k+1) = \varphi_1(k) + \dot\varphi_1(k)\cdot T_s \tag{18}$$

$$\varphi_2(k+1) = \varphi_2(k) + \dot\varphi_2(k)\cdot T_s \tag{19}$$

The discrete-time equation for x_2 can be derived similarly to x_1 resulting in the following:

$$x_2(k+1) = x_2(k) + \frac{c_1(k)}{c_3(k)}\Big(\sin\big(c_2(k)+c_3(k)T_s\big) - \sin\big(c_2(k)\big)\Big) \tag{20}$$

Equation 17-20 are the final discrete-time, state-space equations.