*Applied Reinforcement Learning, Sommersemester 2016*
*Ben Pfirrmann, Erik Wannerberg, Zhiwei Han*

# Poppy the Watchdog

- Applied Reinforcement Learning SS 2016: Final Report

## 1. Management Summary

In this project we implemented a learning algorithm that allows the poppy humanoid robot to detect and track coloured objects. This problem was mainly inspired by a watchdog that may track hostile or friendly entities and may react accordingly. As the camera broke in the final week we were only able to record a video of a pre-final software. However, it already demonstrates the learning strategy and the tracking performance.

## 2. General development process[1]

From the very beginning, we realized that the software development process would be difficult and need some structure. Hence, we decided to modularize key functionalities and embed them in individual python classes (c.f. Table 1). These classes were not used to the same extent everywhere, but the concepts were always incorporated.

| **Class** | **Description** |
|---|---|
| StateActionSpace | A model of the states and the actions according to a simple discretized model |
| Actor | Embodies the robot-environment relation and maps the actions in the StateActionSpace to the physical or simulated actions |
| Observer | Transforms the state of the underlying physical, simulated or model (mathematical) world into the states from the StateActionSpace-Class |
| Reward | Allows for designing the reward given for several terminal or non-terminal-states independent of the other modules |
| LearningAlgorithm | Turns state transitions and rewards into actions, thus implementing a Policy, and updating this according its own rules, using Value/Q-functions or similar, thus alsoimplementing the learning strategy |

**Table 1: The base python classes used in the project software**

Overall, we chose to use three domains for our problem. Firstly, a mathematical domain that emulated our model of what would most likely happen in an idealized world, meaning that the action for moving the head upwards could only make the detected point switch one discretized position down in the grid. Secondly, the VREP domain that used the poppy-model implemented there to simulate our problem, including the CV-Task of recognizing object positions and angles. Finally, the third domain was the real poppy robot combined with a coloured object to be

[1] For references and abbreviations not stated here, see the two previous reports.

detected. The first two domains were then used for testing and development, and could finally be used for pre-training the real poppy, to initialize a learning session with a pre-learned policy and value functions.

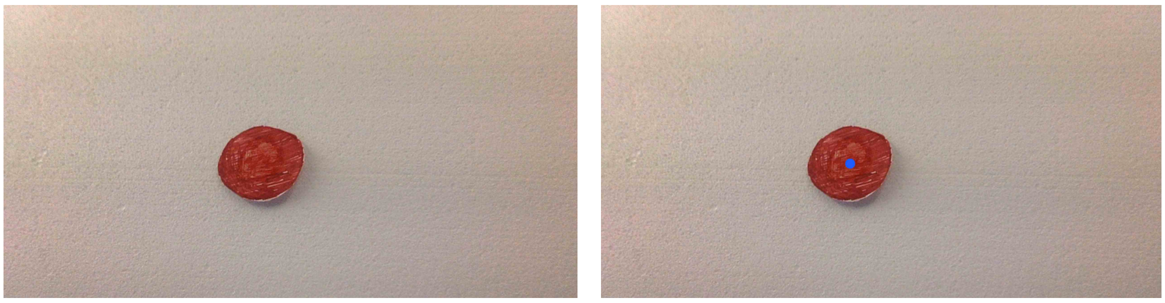
## 3. Task Overview

### 3.1 Computer Vision Task

In the mathematical domain, no CV is explicitly necessary since the state is modelled explicitly in a state variable. In the VREP-Domain we used a relative angle of the poppy-head-direction and a 3D-cube placed on a supporting block, and discretized that into a wanted number of states, keeping a maximum angle beyond which the object would be out-of-sight. This was also relatively simple, thus the main challenge was to handle the real CV-Task.

Here, we compared different solutions. Besides a small research on existing solutions we implemented two prototype solutions to solve the task. One was a coin-segmentation algorithm[2] that did not show a sufficiently robust behavior. Hence, we chose an alternative solution. A mask-based solution[3] was better suited. It tries to detect areas based on their pixel color. Using a white wall of Styrofoam as background and a red paper card target (see Figure 1 for example classification images), the algorithm displayed the desired behavior. A related issue was the light in the lab or more general in poppy's environment. We found out that using the standard lamps in the lab provided a sufficient illumination.

**Figure 1: Target detection using a mask-based algorithm. Left: Original image. Right: Target detected, location illustrated by a blue dot.**



### 3.2 Reinforcement Learning Task

#### 3.2.1 State-Action Space

First, the state action space was critical. Hence, we decided to make it simple. The relative object position is represented as discretized 1D or 2D position in a Cartesian grid. Furthermore, we decided to use two terminal states, one state where the object is centered and one where the object is out of the field of view. All remaining positions in the grid were defined as non-terminal states.

#### 3.2.2 Reward Scheme:

The reward scheme can basically be built as one of infinitely many solutions, as it is always a heuristic engineering solution. However, it should be simple and best allow for illustrating the

[2] http://blog.christianperone.com/2014/06/simple-and-effective-coin-segmentation-using-python-and-opencv/
[3] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html

theoretical results from reinforcement learning. Hence, we decided to only reward the centered (desired) terminal state and (if implemented) penalize the out-of-range terminal state. All the non-terminal states were given a small negative reward, to give a faster path to the goal a higher reward. This turned out to be sufficient for our purposes.

### 3.2.3 Learning Algorithm

For the learning algorithm, we used several approaches. The first one was a TD Policy-Iteration. It updates the value function according to the standard formula for a TD-update. The policy update was handled by recording the numbers of each state-action-state transitions, and hence the frequency with which a certain action in a state lead to other states, and using this to compute expectations. These were then used to choose the optimal action. Computing expectations was required as the implication of using a specific action was not deterministic, the main reason for this immanent stochastic being behavior that we used a fixed rotational step size, and that the image was depicting angles nonlinearly. Hence, the actual state-transition depends on the image acquisition nonlinearities as well as the different possible state-space-grids. Besides the TD Policy Iteration, we used SARSA(λ) as well as SARSA(0). However, both of them were never used on the real poppy. Naturally, instead of a Value-Function a Q-function was used there. In SARSA(λ) and SARSA(0), the first step is always to learn an action-value function rather than a state-value function as in TD Policy-Iteration. In particular, for an on-policy method, the Q function must be estimated for the current policy and for all states and actions with $\varepsilon$-greedy method. This can be done by using the same TD update rule described above (learning state-value function). As we were never able to run all of them in practice, we could not do a sophisticated analysis based on performance differences. However, the TD Policy-Iteration based on some simplifications[4].

In our development we followed different approaches and so we also used different visualization methods. For the TD Policy Iteration a gray-scale image of each states value function can be found in Fig. 2. For SARSA(λ) as well as SARSA(0) the plots can be found in Fig. 3. However, different parameters were used here. Overall, only the TD Policy Iteration was tested in all three domains. The SARSA-Implementations were only used on the mathematical and the VREP-Domain.

[4] In our video, we used a 1D-State-Representation with the TD Policy iteration with only one terminal state in the center. The case of an object out of the camera range was not yet handled at this time. However, as the algorithms used a pre-trained policy for further learning and a very limited geometric setup, the robot was able to handle the cases without this exception.

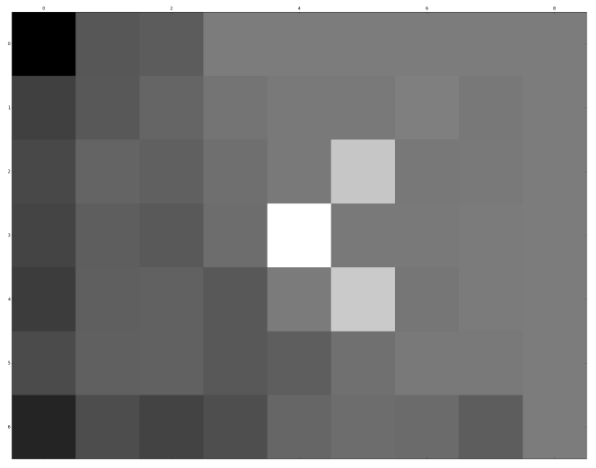


**Figure 2: Value function of the TD Policy Iteration[5]. The colour scale goes from white (maximum value) to black (minimum value).**

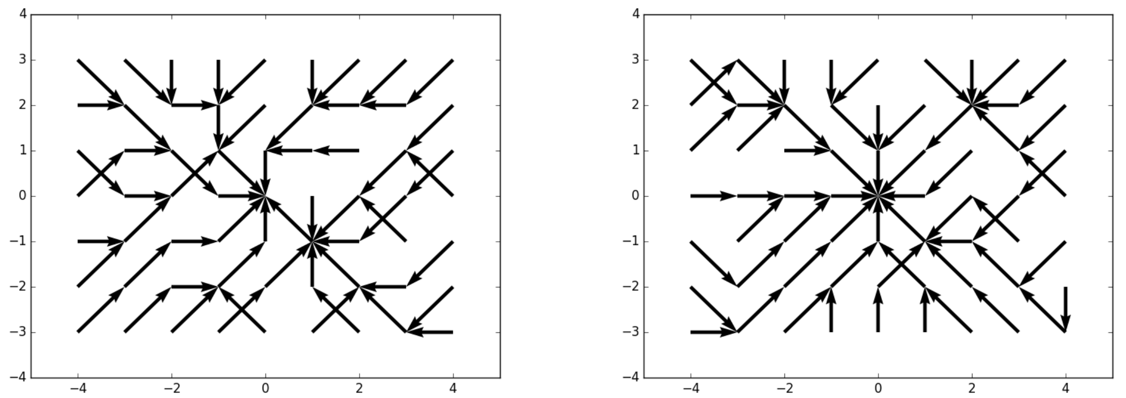


**Figure 3: Policies illustrated by arrows of which grid states lead to which. Left: Policy for SARSA(0.5). Right: Policy for SARSA(0)[6]**

## 3.3 Practical Implementation

### 3.3.1 Setting up the Problem World

As often in engineering problems, theoretical models deviate from the real environment. One major challenge arose from naïve assumptions about poppy's sight. As the head position is not always a well-suited indicator for the camera's field of view, we needed to trace back the image and to adjust the relative poppy-wall-object-constellation. This required placing the poppy-robot on a box as well as on a tape roll to guarantee both, the correct height and a suitable inclination. Moreover, we found that placing the entire setup on a cart made it adaptive to external factors and allowed to place the setup in a well-illuminated environment. Finally, using a white

[5] positionMatrix = (9, 7), num_episodes = 50, epsilon = 0.1, gamma = 0.7, learning_rate = 0.1

[6] dimension = (9, 7), epsilonGreedy = 0.6, learningRate = 0.1, gamma = 0.7, numEpisodes = 4500, lambdaDiscount = 0.5, iterNumLimit = 500

Styrofoam board as well as a red paper card provided sufficient contrast in the real world and allowed for quick changes of the relative object position. Hence, we were easily able to demonstrate poppy's tracking performance by simply moving the wall together with the object.

**3.3.2 Computer Platform**

Another issue resulted from the shipped ODROID-board. The limited number of USB-Ports did not allow for simple access and control. Hence, we decided to use regular computers (i.e. notebooks). However, as we used three different operating systems, this required to sometimes reconsideration of implementation details.

**3.3.3 Broken Camera**

Unfortunately, the camera broke in the final week. A single cable from the USB-Connection slipped out of the connector. By trying, to fix this issue, the connector of the camera board broke loose and we were unable to fix this issue sufficiently. We were, grateful to be offered a substitute camera. Unfortunately, we did not manage to correctly control the camera from python because we were unable to control the drivers we found online correctly or may have chosen the wrong drivers.

# 4. Summary

## 4.1 Major Learnings

### 4.1.1 General Project Management

Three people working on one software project is a situation that will easily create a large and unstructured code environment. Therefore, it was critical to provide some structure for the development process. Besides the aforementioned code framework, we decided to use GitHub (https://github.com) for version control and this turned out to be a very useful tool.

People have different work and communication habits. That is something that also needs to be considered in engineering projects, as individual team member will have different assumptions on what they are required to do. Criticism is not always expressed directly and hence a communication plan could have been helpful. Such a plan would specify each team member's communication preferences like "how to criticize" or "how to be criticized". Writing down tasks was very helpful. However, some minor tasks were not written down and the respective distribution of workload then leads to small fluctuations from time to time.

### 4.1.2 Implementation Learnings

In the end we were glad to have captured a video of a pre-final version, as the hardware failure did not put us to the very beginning but just a small step back. This is a valuable experience that will help each of us to succeed on future projects.

A good and simple state-space model is key and allowed us not to waste much time on the parameter tuning but rather on the interfaces and the big algorithm ideas as the learning algorithm worked well with best-guess parameters. Moreover, a raw comparison of the different learning strategies did make the Q function approaches superior to the value function based approach. Hence, the state-space may have even allowed us to use a simpler learning strategy as it was very easy to control and to observer.

## 4.2 Outlook

In our latest implementation we used a learning algorithm that was pre-trained and continued learning on the real poppy already displaying a good tracking performance. In the future, we may have stopped the learning procedure and only let the robot do its job.

The CV-Approach would have allowed for detecting different colours we did not use. Hence, we could have extended and possibly duplicated the state space for two different colours. Then, we could have further associated different actions to the terminal state. Poppy may be able to act like a real watchdog and greet friendly objects as well as threaten hostile intruders. Another possible extension could be to classify pre-defined human faces as friendly or hostile and hence making it more real-world oriented. This could be done using standard pattern recognition faces for face recognition. Regarding the CV-Algorithm we could also further expand on the out-of-range case and develop more generic strategies that make poppy search for the object without possibly ruining its value function values by permanently propagating the penalty to all states. This would require a sophisticated parameter tuning as well as sufficiently thorough pre-training.

Finally, the learning algorithm design can be pushed forward. One may use a reward scheme that also penalizes non-terminal states with a small negative reward. This approach would allow for developing fast tracking solutions.

Incorporating more sophisticated state-spaces may also have led to a state-space explosion and one could possibly use learning algorithms that perform dimensionality reduction.