Praktikum: Virtual Neurorobotics in the Human Brain Project

Marie Bommersheim, Rafael Kübler, and Simon Reinkemeier
FZI Forschungszentrum Informatik
Karlsruhe

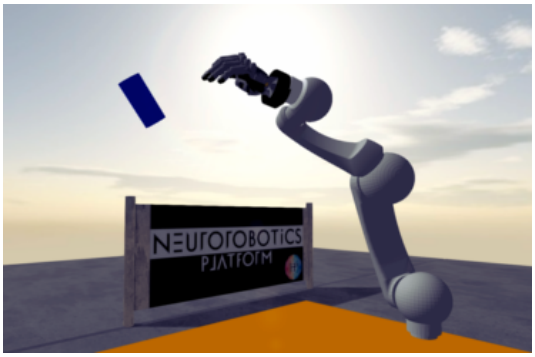

Figure 1. The robot arm throwing the cylinder away from the table.

Abstract

1. Introduction

The aim of our Praktikum is to let the robot arm learn how to throw a cylinder as far away from the table as possible. The robot arm and the cylinder are shown in figure 1.

Sections 2 and 3 describe algorithms for solving optimization problems in general. Sections **??**, **??**, and **??** describe our approaches for solving the Praktikum challenge.

2. Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristics [1]. Metaheuristics are methods for solving optimization problems by iteratively improving candidate solutions in relation to a given quality measure. Population-based metaheuristics use populations of multiple solutions and iteratively change populations to find a good solution. An evolutionary algorithm is based on a population P of possible solutions $e_i \in P$ for the optimization problem or heuristic to be solved. A solution is also referred to as an individual in the context of population-based metaheuristics.

In general, evolutionary algorithms consist of the following steps [1]:

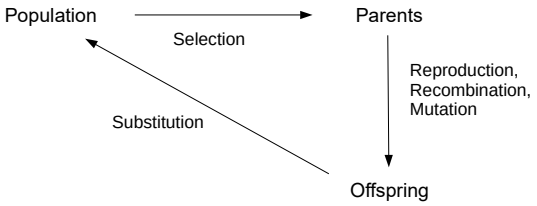

Figure 2. Evolutionary algorithms, adapted from [1]

1. Initialization: The initial population of individuals is generated randomly.
2. Evolutionary process: The individuals are evaluated by calculating their fitness. Therefore a fitness function is needed. According to their fitness, some of the individuals are selected to be the parents of the following generation. Then a process with reproduction, recombination, and mutation starts - similar to biological processes. The parents are being recombinated and the resulting individuals are mutated to form the offspring of the current generation. These steps are being repeated until a new population of individuals is generated. The parents are also added to the new population. This new population substitutes the old population and the process is repeated until an abort criterion is fulfilled (see figure 2).
3. The best individual is selected.

3. Monte Carlo Markov Chain

Monte Carlo Markov Chain algorithms can be used for hard optimization problems. The idea is to construct a Markov Chain that generates samples such that more time is spent (more samples are evaluated) in the most interesting regions of the state space (see figure 3) [2]. The decision to accept or reject a proposal to go from Point 1 to Point 2 is based on the ratio of posterior densities of the two points: uphill steps are always accepted, small downhill steps are usually accepted but huge downhill steps are almost never accepted (see figure 4).

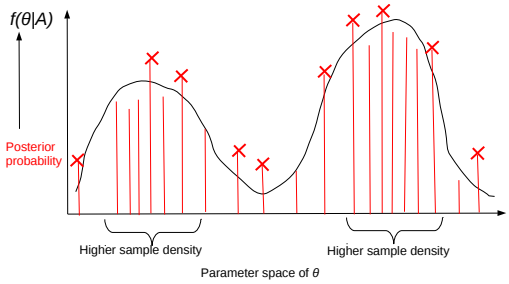

Figure 3. The probability to evaluate or find a sample in an area with high posterior probability is proportional to the posterior distribution, from [2]

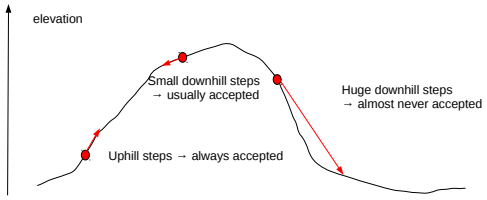

Figure 4. The decision to accept or reject a proposal to go from Point 1 to Point 2 is based on the ratio of posterior densities of the two points/samples, from [2]

4. Prepare and Hit

The first approach is to let the robot arm hit the cylinder away from the table. Therefore, the movement is divided into three phases: reset, prepare, and hit. The phases are implemented as states using the state machine editor.

4.1. Hard-coded movements

At first, the movements are hard-coded:

- Reset: Moves the arm in an upright position and puts the cylinder on the top right corner of the table (as seen from the robot arm).
- Prepare: Moves the arm in an horizontal position next to the cylinder (on the left side of the cylinder as seen from the robot arm).
- Hit: The arm moves towards the cylinder and knocks it off the table.

4.2. Hit movement improved

This approach also has the aim of knocking the cylinder off the table, however, the hit movement is not hard-coded but learned using an evolutionary algorithm. The reset and prepare phases are taken from the hard-coded approach. The evolutionary algorithm which is used to let the robot learn the hit movement is described below:

An individual of each generation consists of an array of six components (genes) that represent the joints of the robot. The start array is defined as $\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$. To generate the initial population of ten individuals the start array is mutated by adding random values between $\{-0.1, 0.1\}$ to each component. Computing the next generation consists of three steps:

1. Selection: The mating pool of four parents is created by selecting the individuals with the highest fitness. The fitness is calculated using the difference between the initial and the end position of the cylinder after knocking it off the table.
2. Mating: In this algorithm, the parents are carried over to the next generation. The recombination process uses one-point crossovers.
3. Mutation: The offspring is mutated by adding a randomly calculated value between $\{-0.5, 0.5\}$ to each component.

These steps are repeated and after ? generations the winner is selected.

4.3. Prepare and Hit movement improved

This approach also uses an evolutionary approach to knock the cylinder off the table. In this approach both the prepare and the hit movement are learned. The reset movement is taken from the hard-coded approach. The evolutionary algorithm is the same as in the previous subsection but with a different start array.

The start array is a combination of the prepare and the hit movement. It is created by appending the hit array, given as $\begin{bmatrix}-0.45 & -0.9 & 0.9 & 0 & 0 & -0.5\end{bmatrix}$ to the prepare array, also given as $\begin{bmatrix}-0.45 & -0.9 & 0.9 & 0 & 0 & -0.5\end{bmatrix}$. In this approach, the start pool consists of 15 individuals which are generated from the start array by adding random values between $\{-0.25, 0.25\}$ to each component. The mating pool consists of three parents. The evolutionary algorithm is executed as in the previous approach (with the described changes).

It is repeated for eight generations.

5. Prepare, Grasp, and Throw

The second approach uses grasp and throw phases to throw the cylinder away from the table. The reset and prepare phases are taken from the first approach and the hit phase is replaced by grasp and throw phases.

5.1. Hard-coded movements

At first, the movements are hard-coded.

- Reset: Moves the arm in an upright position and puts the cylinder on the top right corner of the table (as seen from the robot arm).

- Prepare: Moves the arm next to the cylinder (on the left as seen from the robot arm).
- Grasp: Moves the fingers of the robot hand around the cylinder in order to grab it.
- Throw: With high acceleration, the arm moves backwards beyond the vertical position and the hand opens to throw the cylinder away.

5.2. Throw movement improved

This section describes two different ideas to optimize the throw movement. In both approaches the aim is to let the robot learn which movement is best to throw the cylinder as far away from the table as possible.

5.3. Evolutionary Approach

This approach uses the phases from the second approach. The reset, prepare, and grasp are still hard-coded as in the second approach. The throw phase is trained using an evolutionary algorithm.

5.4. Markov Chain Monte Carlo Approach

This approach also uses the phases from the fourth approach. The reset, prepare, and grasp are still hard-coded as in the second approach. The throw phase is trained using a Markov Chain Monte Carlo algorithm.

6. Prepare, Grasp, Windup, and Throw

References

[1] P. Flick. Evolutionäre Algorithmen. http://parco.iti.kit.edu/henningm/Seminar-AT/seminar-arbeiten/Flick_final.pdf, accessed 03.02.2019, 2012.

[2] A. Stamatakis. Introduction to bioinformatics for computer scientists, lecture 12. https://cme.h-its.org/exelixis/web/teaching/lectures18_19/lecture12.pdf, accessed 10.02.2019, 2018.