An actor-critic reinforcement learning controller for a 2-DOF ball-balancer

Andreas Stückl

Michael Meyer

Sebastian Schelle

Projektpraktikum: „Computational Neuro Engineering“

Empty side for praktikums paper

Empty side for praktikums paper

Content

Abstract

This report is part of “Projektpraktikum Computational Neuro Engineering” about the Project “An actor-critic reinforcement learning controller for a 2-DOF ball-balancer”. The goal is to balance a ball on a plane, which is moved by two servo motors, the structure of the machine is seen in Figure 1. A DVS (Dynamic Visio Camera) is used as sensor to determine the position and velocity of the ball. As first step a PD-controller is used. At second it is replaced by controller based on reinforcement Q-learning. For both controller types it is important to get good position data from the sensor. The algorithm to determine the position and velocity was iterative improved and designed for speed. The reinforcement learning still lakes of a good reward algorithm, which makes a well adjusted PD-Controller still the better option.

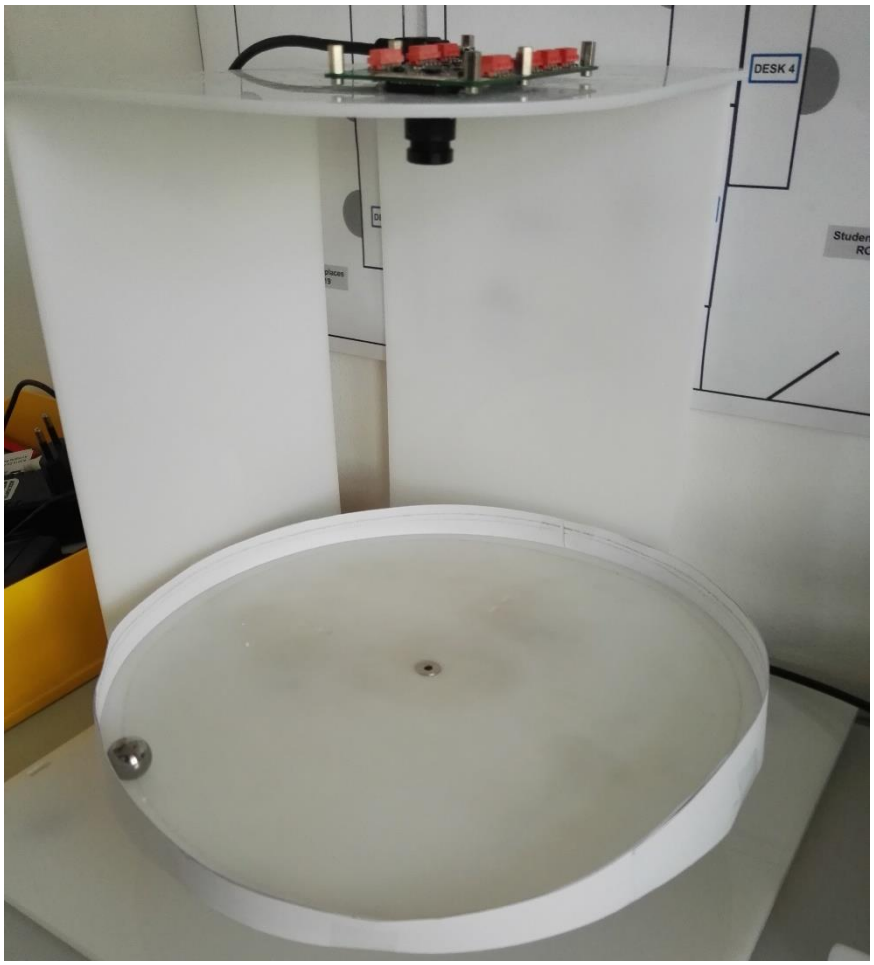

Figure 1: Ball Balancer. DVS Sensor on Top. Ball on the plane with fance for testing.

Task

Two digital servos motors tilt the plane, one for X and one for Y Direction. The midpoint is fixed on a shoulder bearing. The motors are controlled via a USB Connection. A set of commands for moving the motors was provided.

For the DVS, you can find information at the manufacturer website http://inilabs.com/ it is 128x128 pixel. There was a rudimentary code in Matlab available, to use it without time information. Communication was also over USB. [6]

Because of that it is reasonable to use a modern Laptop for computation, as both Devices communicate over USB. Matlab was chosen as IDE and programming language, because of the available code, support for USB communication and good ability to display data. How the code works and its design is explained in the next chapter. In the end there is an evaluation of the controller.

Implementation

To avoid poorly structured code, object oriented programming was used for the software part. The system is divided in several parts which all have their own tasks. The principle separation of concerns is applied. A simplified class diagram is shown in figure 2. The *Controller* acts as a main class, which manages all the other functionalities. The *DVS128* class is used for getting the sensor data. The *Servos* class is used to initialize two servo objects for moving the plane in the two directions. The controller also implements a simple *View*, where the data can be observed, the ball position and velocity are displayed and where the user can interact. The interesting part happens in the *Model* class. It contains a *Filter*, which is used to filter the noisy input data and calculate the ball position. The filtered Data is then stored in a ring buffer. The model class also contains the actual controller, which in this case can be a simple PD controller or a Q learning controller.

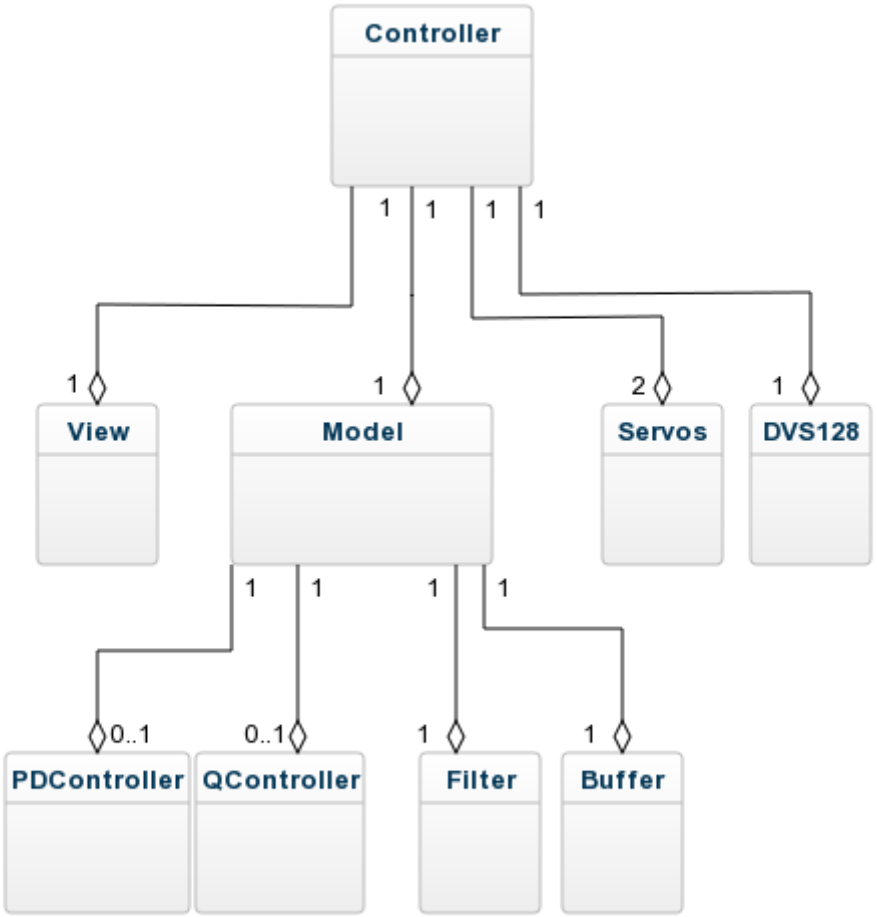

Figure 2: Class Diagram

DVS Data

For the visual input a Dynamic Vision Sensor [6] (DVS) has been used, as it provides immense advantages over conventional frame based solutions. The DVS only sends the local changes in brightness, where a conventional camera would send a whole frame. This allows much higher time resolution and decreases computation requirements, storage and power demands. The output of the DVS is asynchronous, which makes it possible to react much faster to changes or moves in the scene. Another advantage over conventional cameras is the high dynamic range up to 120dB. As the pixels respond to relative changes, a high intra-scene dynamic range does not alter the performance of the DVS too much.

For this project the low computational effort and the fast reaction are the most important advantages. Moreover, the fact, that only changes in brightness are detected, makes it much easier and faster to calculate the current ball position. To track objects with a regular camera, more complex algorithms like Kalman Filters are used. Using the DVS makes tracking objects much easier because the sensor only fires events where actual changes in brightness occur. In this case, the only moving object is the

ball. Therefore, the events are all concentrated around the position of the ball. In practice, there is randomly distributed noise all over the field and some concentrated noise on the edges of the plane, especially when moving the plane. Shadows and changing lighting conditions can also add a large amount of noise to the data. For this reason, the data needs to be filtered beforehand. The flashes (huge amount of events due to changing lighting conditions) and the data which does not lie inside the plane are filtered out. For filtering out the events beyond the border of the plane, the center and radius of the plane need to be identified. Knowing those two coefficients, all events outside this circle, defined by center and radius, can be filtered out. After this step the rest of the events are part of the movement of the ball and some noise. This data is then processed by a cluster algorithm to divide the data into noise and events belonging to the ball movement. To get the position, the mean of all events in the cluster which belongs to the ball is calculated. After calculating the position, the velocity can be obtained. There are two simple methods to calculate an approximation for the velocity. Using only one set of events, the velocity can be approximated through the vector pointing from the ON events, which are represented by the black dots, to the OFF events, which are represented by the red dots (see figure 3). The ON events appear where the ball moves out of a pixel and the OFF events appear where the ball moves in. This is certainly only valid if the ball is darker than the surface of the plane. Basically this method works, but the velocity is very noisy. The length of the vector is dependent on the speed of the ball, but it jitters a lot. This method does not provide reliable data. Furthermore, for each set of events the cluster algorithm has to be called twice, first for the ON and a second time for the OFF events. As the cluster algorithm is the resource hungry part of the input data processing, this should be prevented to increase performance. For the second method two consecutive sets of events are needed. When processing the events, only the position of the ball is calculated. the velocity is calculated by the new position minus the last position divided by the elapsed time. This velocity causes less computational effort and is also less noisy.

Figure 3 shows the raw DVS data. The red dots show events where the pixel got darker and the black dots show pixels where brightness increased. Most of the noise is concentrated around the borders of the plane. Filtering out these regions makes the clustering algorithm much faster and more reliable. Figure 4 shows the calculated ball position and velocity. Here all events beyond the borders are filtered out and almost all events belong to the position of the ball. The arrow corresponds to the ball's velocity, with the origin being the ball's position.

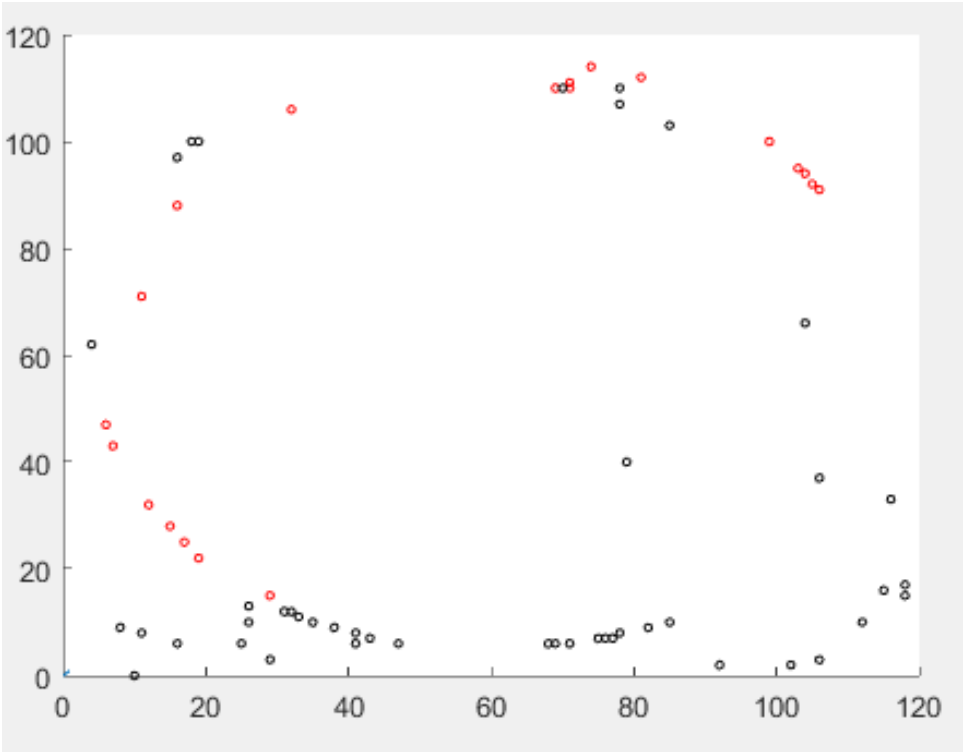

Figure 3: GUI with unfiltered DVS Data

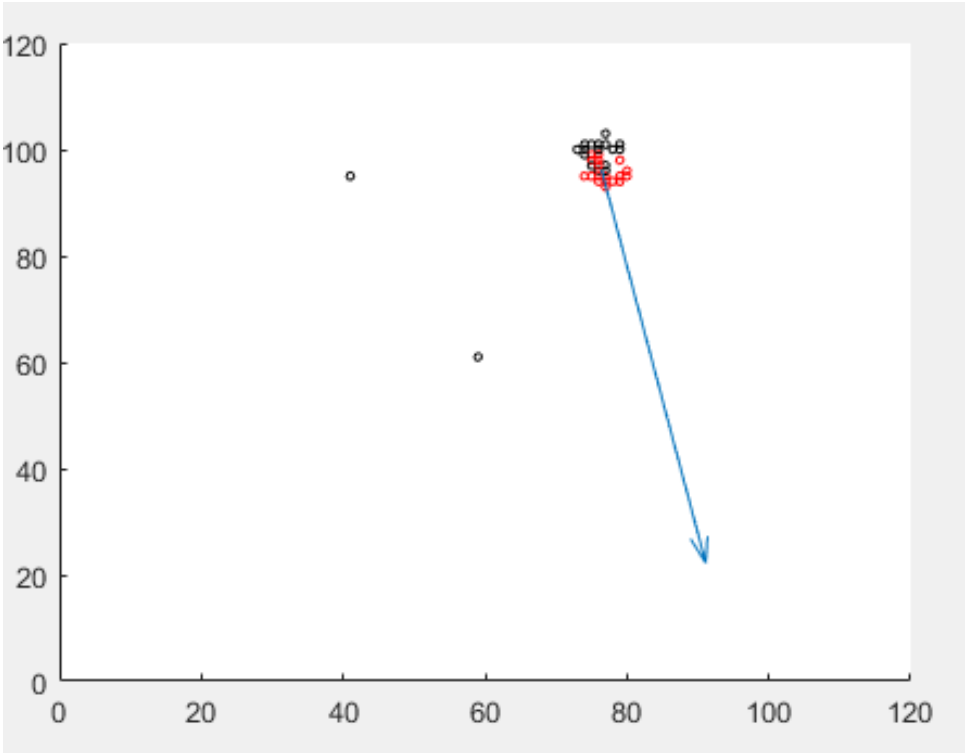

Figure 4: GUI with Filtered Data and Position and Velocity of the Ball

DBSCAN Algorithm

For dividing the events into clusters, the DBSCAN [5] (Density-Based Spatial Clustering of Applications with Noise) algorithm is used. It needs just minimal knowledge about the domain for setting the parameters, it is able to determine different shapes of clusters and it comes with good performance for big data. The data represents a quantity of points in the Euclidean space with a metric *d*. Basically, the algorithm detects how many other points are in the direct environment of every point. The main idea is that inside a cluster the density of points is much higher than at the outside. The points which do not fit into one of the clusters are assigned as noise. With the metric *d* the shape of the cluster can be defined. Using for example the Manhattan distance leads to a square and the Euclidean distance leads to a sphere in the two dimensional space. Dependent on the size of the ball, ε and MinPts can be adjusted. The parameter ε defines the radius where the cluster algorithm searches for neighbors. MinPts defines how many other events need to be in this range to declare an event as part of a cluster.

Fuzzy Learning

Our original goal was to implement a Spiking Neuro-Fuzzy Machine [1]. Because of the limited time available for this project, we together with our supervisor decided to implement a Fuzzy Q-learning algorithm [2] instead.

The Fuzzy logic (see Figure 5) was implemented in such a way, that the number of intervals could easily be adapted. Furthermore, the width of each interval in which there is no overlapping interval could also be changed easily to later be able to adapt the settings to an optimum.

The fuzzy logic was used on the four variables x, dx, y, dy and Q-learning was implemented separately for the states (x, dx) and (y, dy).

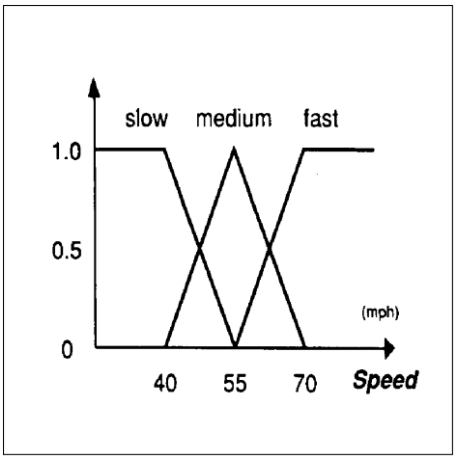

Figure 5: Fuzzy Logic. Instead of defined borders between two intervals, they overlap.Image taken from [3].

Event buffering

A huge improvement was made by buffering the event from the DVS. The Data from the sensor are coming unregularly. From No Event to hundreds are more, depending on the time from request to request. DB Scan has worst case complexity of $O(n^2)$. With hundreds of input events it's time consuming to calculate the distances between all events. Limiting the number of input values gives the algorithm a fixed worst case time. Having only a few new events or none makes it impossible for DB Scan to find a cluster and position out of that.

A fixed size buffer of 75 events showed good results, fast and precise calculation for the position. With 50 events the position had tolerance about 1cm. A buffer of 100 events had no big improvement, but takes noticeable longer. With the buffer implemented an average calculation cycle needs around 4 us, which uses the potential of the DVS.

Improving plane mount

The ball is rolling on a plane, which should be as flat as possible and reasonable. Each servo moto is connected to the plane with a hook, at this point the plane is a little uneven. When the ball is rolling over this point it shaken a little bit, but do's not change its direction. The plane rotates around the middle point, which is secured at this position with a shoulder bearing. For the connection the bearing got a thread, which is put through a hole in the plan and secured by a nut. The ball can roll against the nut, which makes it immediately change its direction.

By turning the shoulder bearing upside down and using a countersunk screw the ball can no longer roll against it and change its direction, which makes forecasting the ball position more easy.

Results

The result is separated into two parts. The first is about the DVS, Structure of Code and DB Scan. The second is about fuzzy Q-learning controller which was implemented.

Basic Setup

The set-up was tested by reading out and visualizing camera data and moving the servomotors to specific positions. This worked as desired. Afterwards a PD controller was successfully implemented to verify that the position and velocity values which were calculated from the DVS sensor data were correct. The position and velocity determined by DB Scan was improved with several methods. Events from out of bounds are filtered. The number of events in which DB Scan searches for a cluster was fixed to specific size, to limit the maximum calculation time by hundreds of new events and having more accurate results if just a few new events appear. This also improved the execution time.

Controller based on reinforcement learning

The goal was to implement a fuzzy Q-learning controller. To keep the state space as small as possible the learning was performed for x and y direction independently. Unfortunately, the policy did not converge through the Q-learning.
As a first approach to overcome this problem the velocity calculation was improved, because it could be observed, that the calculated values for the velocity were error prone due to noise in the sensor data. The PD controller was used again to test the velocity values. With the new calculations for the velocities the PD controller showed near perfect behaviour. So it can be assumed that position and velocity were calculated quite accurately and therefore, are not the source of error in the Q-learning.
It is observed, that the Q-learning controller tends to rapidly move the ball in a circle far away from the centre. Based on this observation it is suspected, that due to the wall on the outside of the plane, the variables x and y are not decoupled. This makes it impossible for the Q-learning controller to learn a working policy, because during learning the ball will eventually end up on the outside, from where the controller cannot move the ball back closer to the centre due to the coupling.
To overcome this complication, it was started to change the implementation such that x and y are not considered independent. So far the reward function was changed to incorporate both spatial

dimensions. Unfortunately, due to time limitations it was not possible to fully change the implementation to a system where x and y are considered depend on each other.

Conclusion

With the implementation the hardware is able to balance the ball at a desired location on the plane. A Q-learning algorithm was programmed, which unfortunately was not able to learn an optimal policy. From the observations made, it is suspected that the undesired behaviour of the Q-learning is caused by the border of the plane which couples the x and y coordinates and thereby makes the learning impossible.

Future work could include performing the Q-learning on the state (x, dx, y, dy) rather than on the states (x, dx) and (y, dy) individually or to substitute the Q-learning totally by Spike-IDS [1].

Literature

[1] Firouzi, M.; Shouraki, S.B.; Conradt, J.: Sensorimotor Control Learning using a New Adaptive Spiking Neuro-Fuzzy Machine, Spike-IDS and STDP. 24th International Conference on Artificial Neural Networks (ICANN), Hamburg, Germany, Sept 2014, 379-386

[2] Glorennec, P.Y. and Jouffe, L., Fuzzy Q-learning. In: Proc. Sixth Internat. Conf. on Fuzzy Systems (Fuzz-Ieee'97), pp. 659-662.

[3] http://www.logicaldesigns.com/LDFUZ1.htm

[4] Firouzi M, Shouraki S B, Esmaili Paeen Afrakuti I, "*Pattern Analysis by Active Learning Method Classifier*", Journal of Intelligent & Fuzzy Systems, Vol 26, Number 1, January 2014, pp 49-62

[5] http://www.uni-weimar.de/medien/webis/teaching/theses/busch_2005.pdf

[6] http://inilabs.com/products/dynamic-vision-sensors/