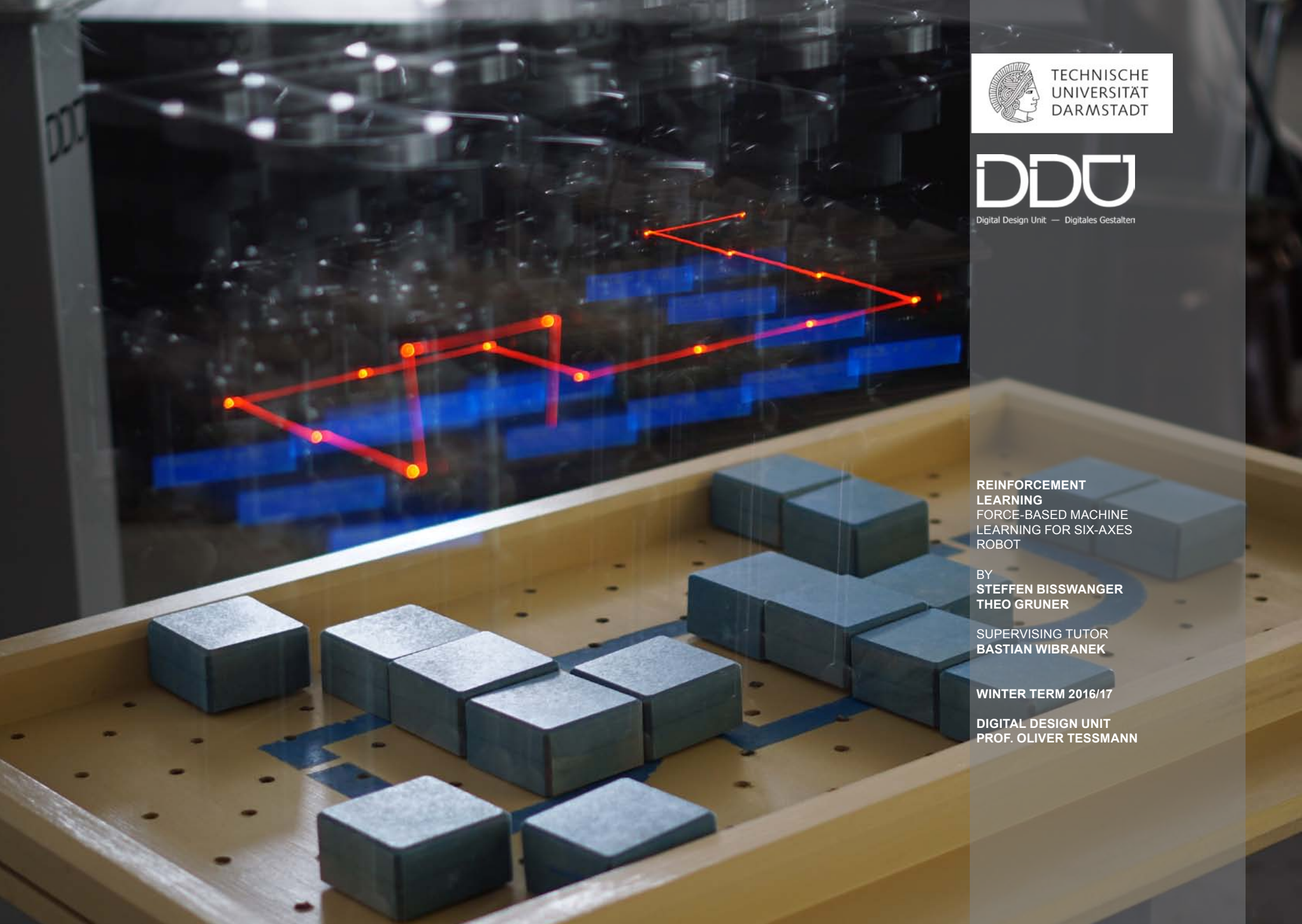
DDU
Digital Design Unit — Digitales Gestalten

TECHNISCHE
UNIVERSITÄT
DARMSTADT

REINFORCEMENT LEARNING

FORCE-BASED MACHINE LEARNING
FOR SIX-AXES ROBOT

THEO GRUNER
STEFFEN BISSWANGER



TECHNISCHE UNIVERSITÄT DARMSTADT

DDU
Digital Design Unit — Digitales Gestalten

REINFORCEMENT LEARNING

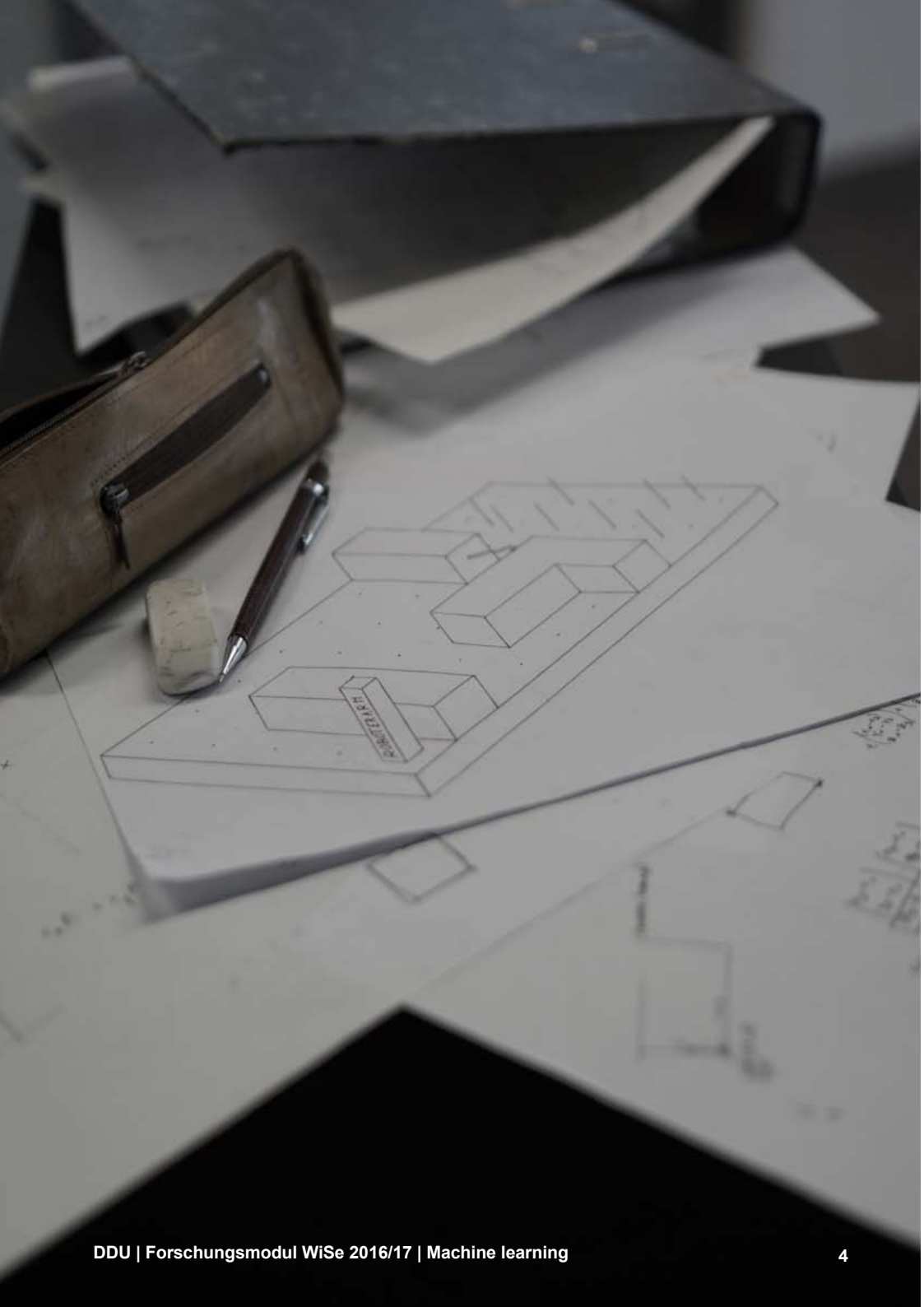
FORCE-BASED MACHINE LEARNING FOR SIX-AXES ROBOT

BY
STEFFEN BISSWANGER
THEO GRUNER

SUPERVISING TUTOR
BASTIAN WIBRANEK

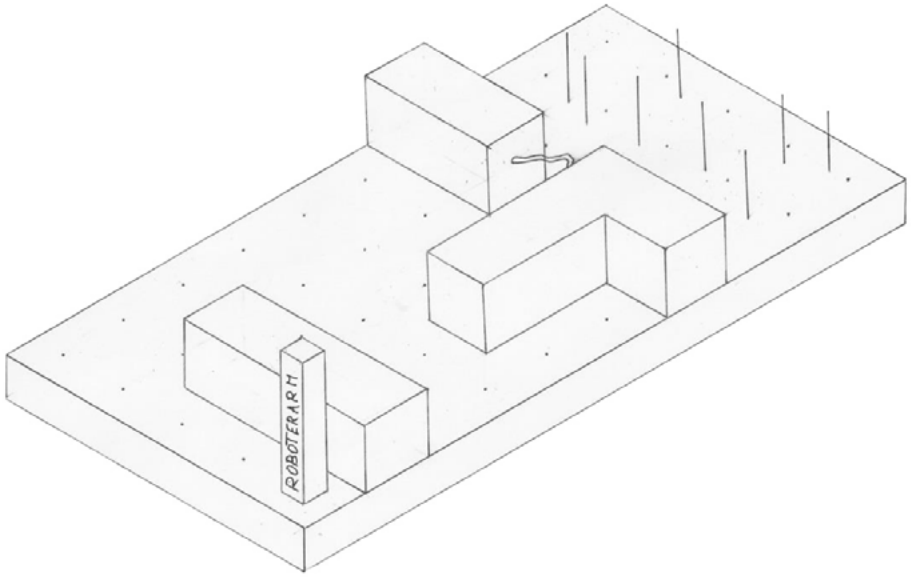
WINTER TERM 2016/17

DIGITAL DESIGN UNIT
PROF. OLIVER TESSMANN



THE TASK

For our research module the objective was to define and solve a problem for a robotic arm using machine learning (ML). We decided to implement a force-based ML task for which collecting feedback should not be too difficult. Also, a two-dimensional environment for the problem would be easier so as not to overcomplicate the task too much, since the topic was completely new to us. Immediately the picture of some kind of labyrinth came to our minds. At first, we thought of some movable components as well as of some elastic rubber-band-type obstacles in addition to the solid obstacles you would expect to find in an ordinary labyrinth.


ROBOTERARM


10kg

“Machine learning
is a core, transformative way
by which we’re re-thinking
everything we’re doing“

Sundar Pichai, Google CEO
Q3 earnings call 2015

The keyword of machine learning is the term „Big Data“. It describes the huge amount of unstructured information given in today’s world, which is mostly unused. Contrary to rule-based systems, it does not rely on complicated algorithms and calculations, but instead uses the stored data to identify generalisations in the data given.
The advantages are very simple algorithms and low operational costs.
So machine learning offers a completely new approach on solving problems and is often used when rule-based systems are too complicated or even impossible to execute.
To analyse the unstructured data, there are many different approaches which suit different kinds of problems.
Following, we listed some commonly used strategies.

M A C H I N E

L E A R N I N G

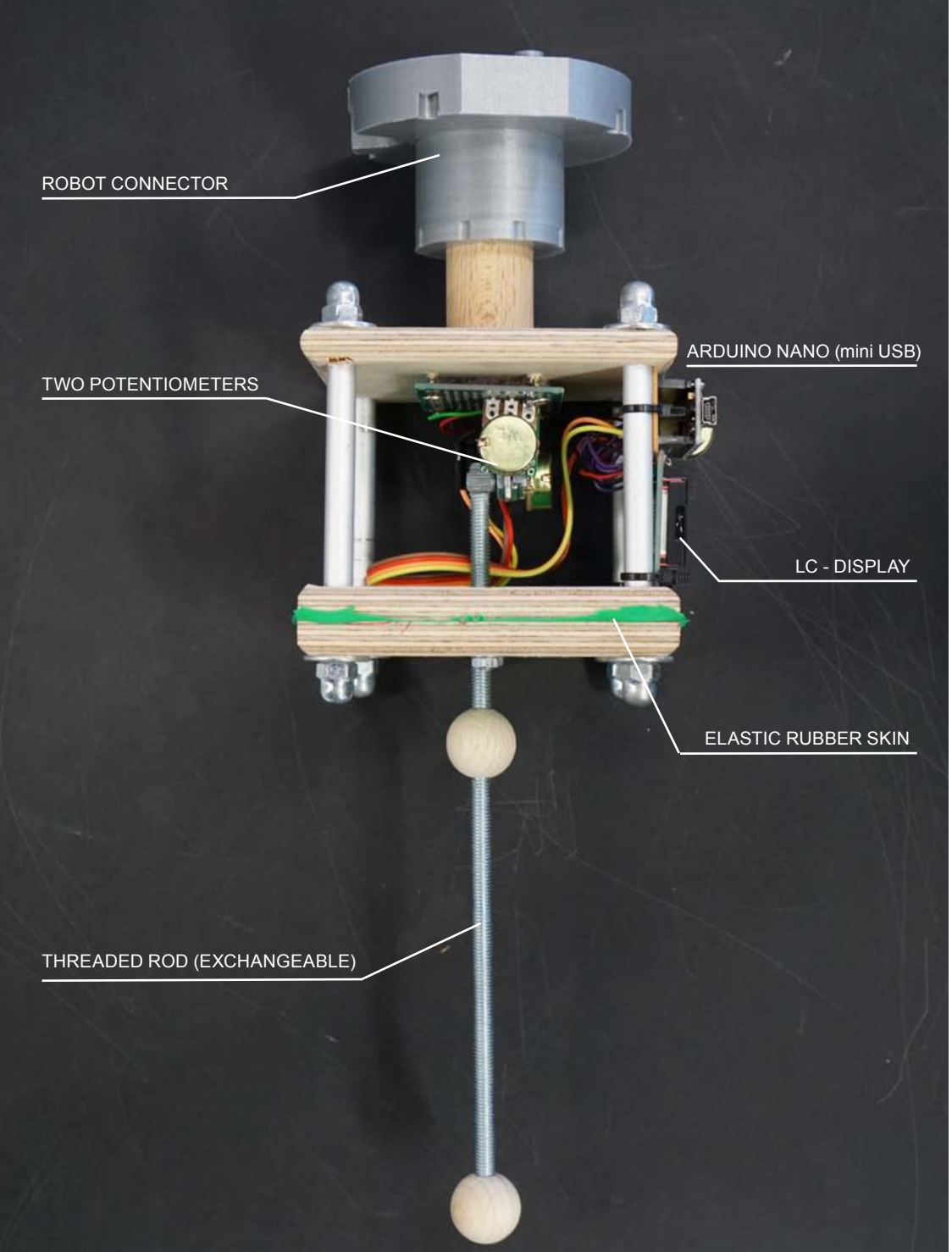
APPROACHES

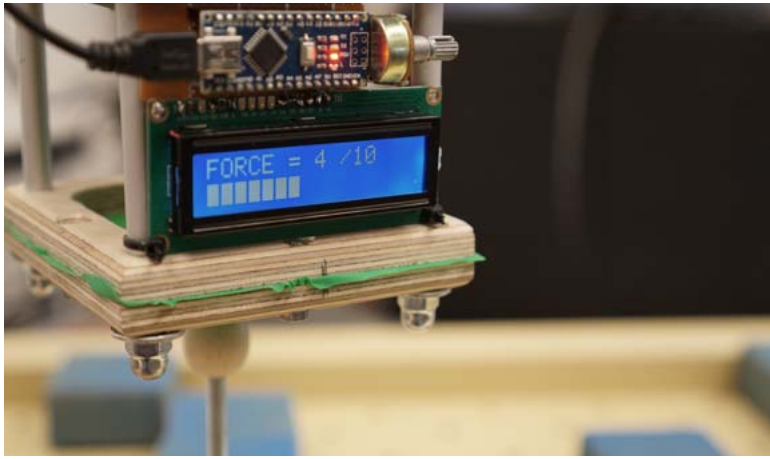
An ARTIFICIAL NEURAL NETWORK is no machine learning approach but helps implementing the other learning tasks. It is modelled on the functioning of the human brain consisting of artificial neurons symbolising functions. They are connected to many other neurons and therefore build a working network. Each neuron receives inputs from the others, analyses them (e.g. sums up all inputs) and sends the result out as an input for another neuron. Artificial networks can consist of up to one million neurons sending information between each other and trying to gather a result out of it.

SUPERVISED LEARNING is well suited for analysing big amounts of already stored and assigned data. It is key to the learning task that the data is labeled in the first place, because the programme's idea is to map the data against the values assigned in before. After evaluating the test data the program should be able to classify the now unlabeled and extern data to the attributes. Companies like "Google" use supervised learning e.g. for image recognition. Simply put, pictures get assigned to certain keywords typed into the playhead of the browser.

REINFORCEMENT LEARNING is mostly used for a procedure in which the programme has to make correct decisions. Based on the principal of human learning, the algorithm uses rewards and punishments to learn from his actions. It requires the introduction of a "state", that describes the condition in which the agent is at a certain moment, and "actions" describing the possible transitions to get to another "state". The learning task is to calculate a value for each state as a function of the rewards given in the current state and of the possible ones in the future. The optimal behavior, the policy, is calculated out of the values.

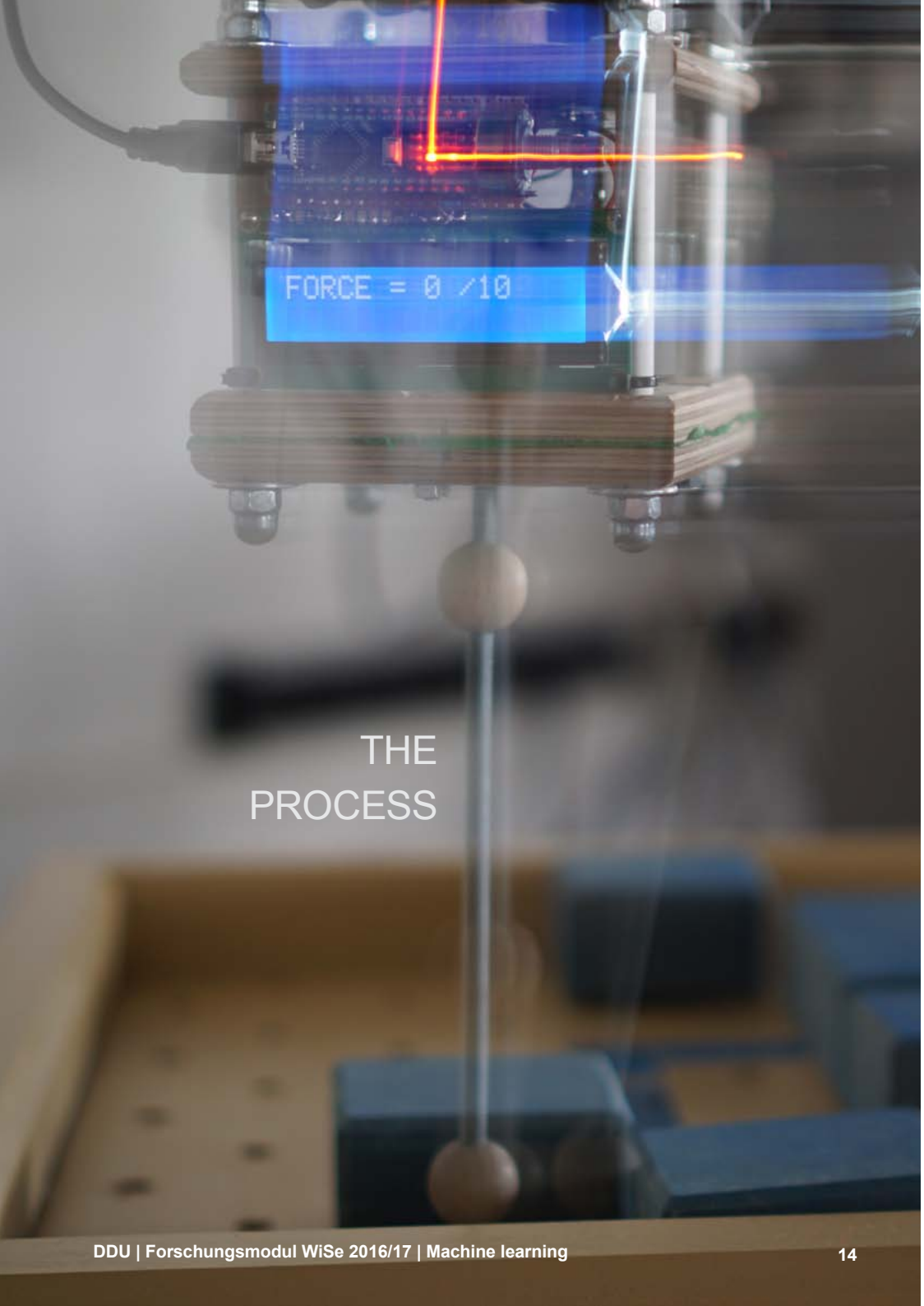
Suiting the best, we decided to implement REINFORCEMENT LEARNING.



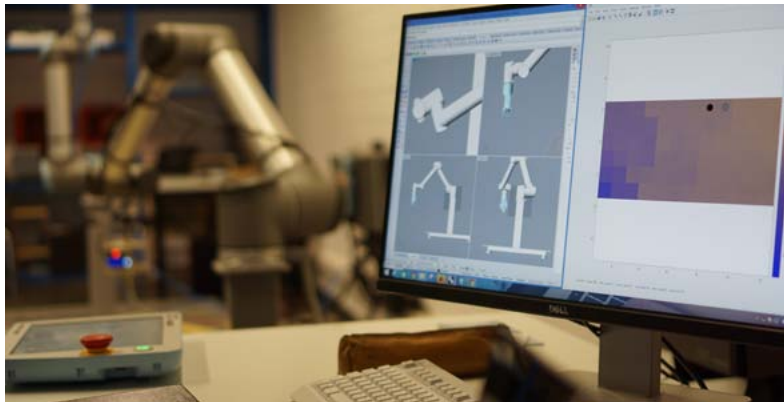


FORCE SENSOR

The sensor is based on an Arduino Nano microcontroller. The deflection of the threaded rod caused by the collision with an obstacle is sensed by the rotation of two analog potentiometers. The relative alignment of the potentiometers is perpendicular to allow for a two-dimensional deflection. The Arduino processes the analog inputs and sends a value between zero and ten via USB to a PC, which can be read by a Python script in Grasshopper. For convenience and debugging purposes the value is displayed and visualised on the LCD as well.



THE PROCESS

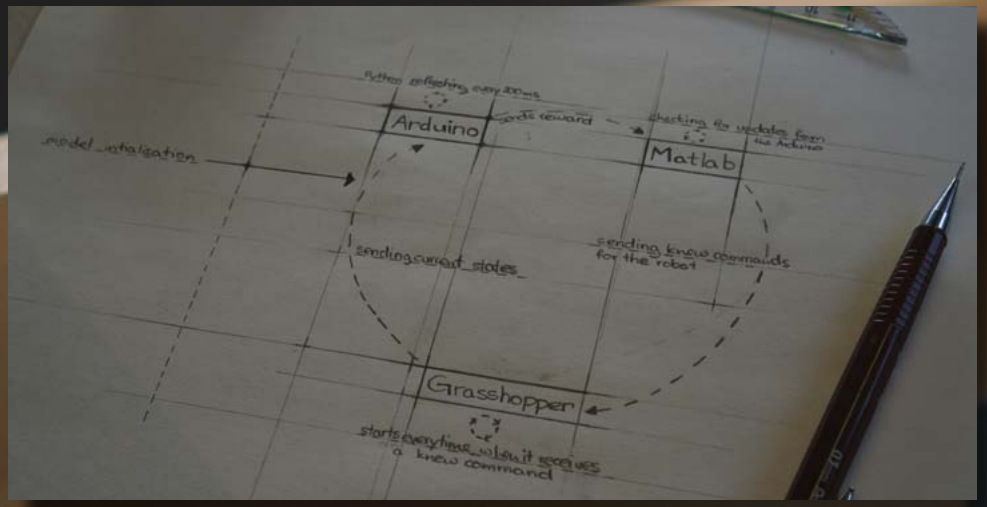


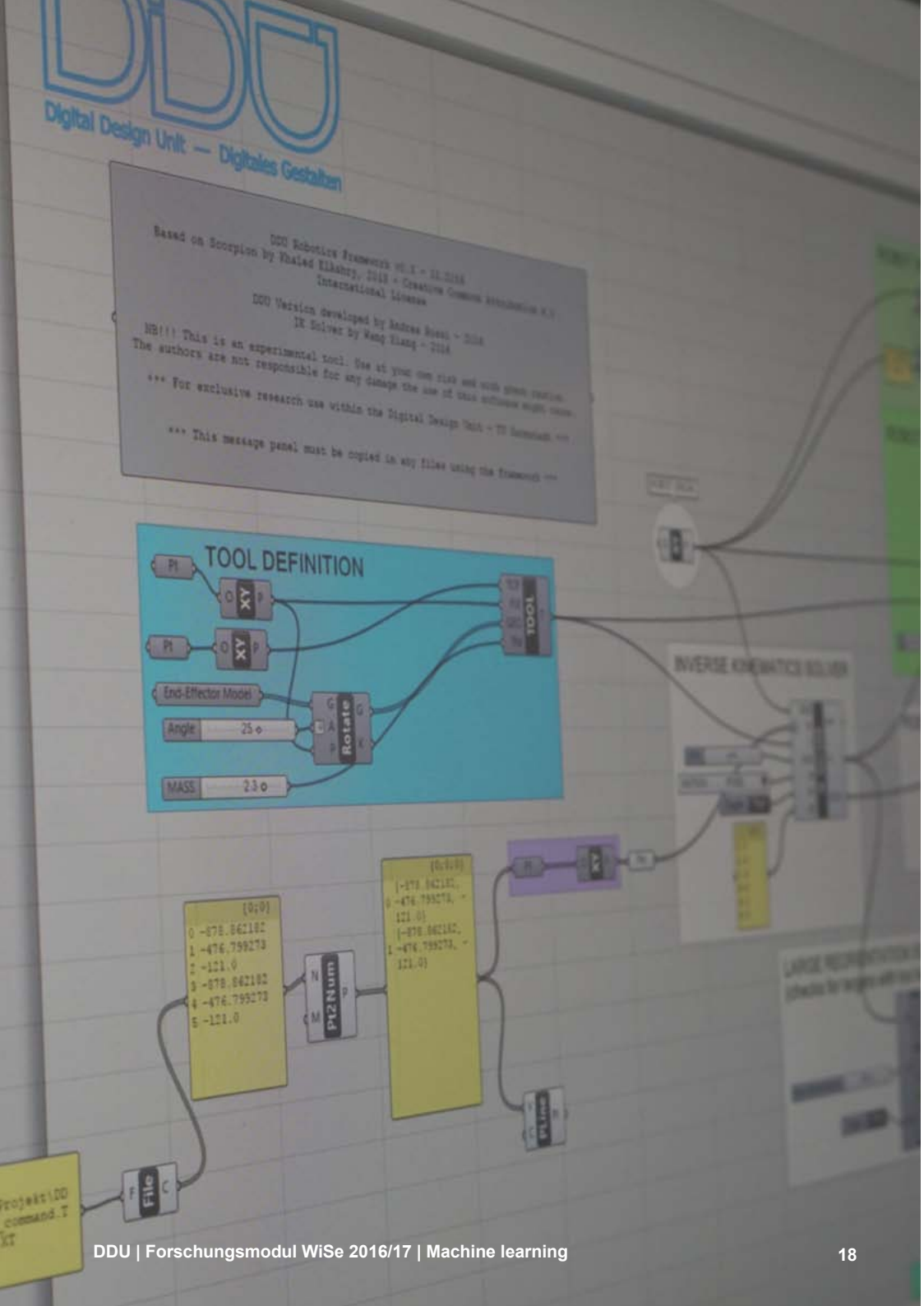
INITIALISATION

To set up the autonomous learning process some known parameters have to be defined in Matlab. This includes the length and width of the labyrinth as a number of steps, the origin, the final destination, three vertices for transforming coordinates, read out from Grasshopper's real-time monitoring and some general parameters for the learning algorithm itself.
The force sensor has to be driven to the starting position by saving its vector into a specific ".txt"-file. The Arduino should be reset being in a vertical position to define the state of zero deflection.
To launch the process, a specific initialisation function in Matlab is called, which resets some more values and generates the first reward- and transition-matrix as well as the first value function. Based on this function and the starting point, the first action and corresponding vector gets written into the ".txt"-file (see above).

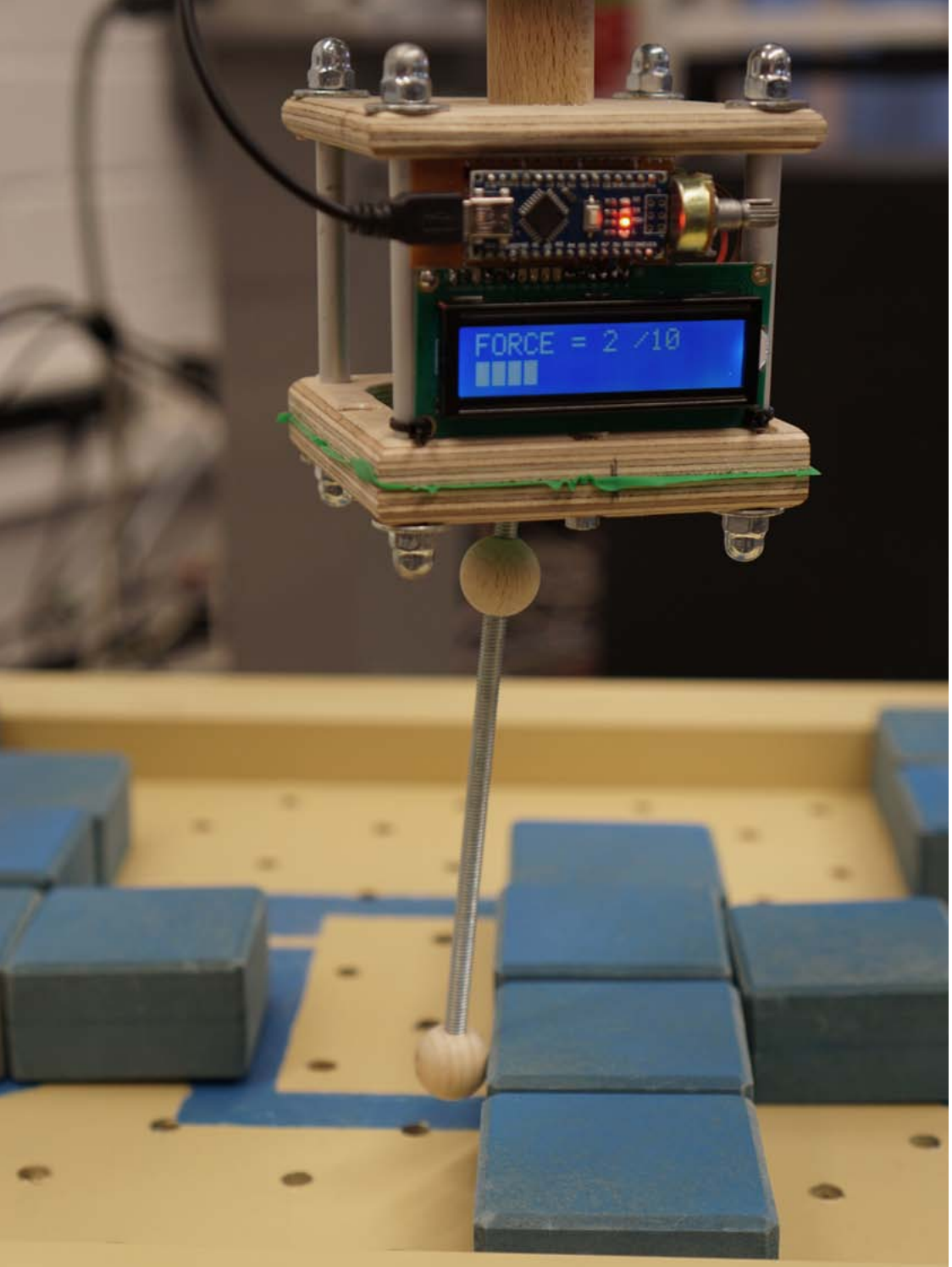
ONE STEP

Once the initialisation has been done, the learning process depends on different programmes executing different tasks. This process repeats itself every time the robot performs a step. Due to computation time and the lag in some feedbacks, the robot takes around three to four seconds to complete one cycle.

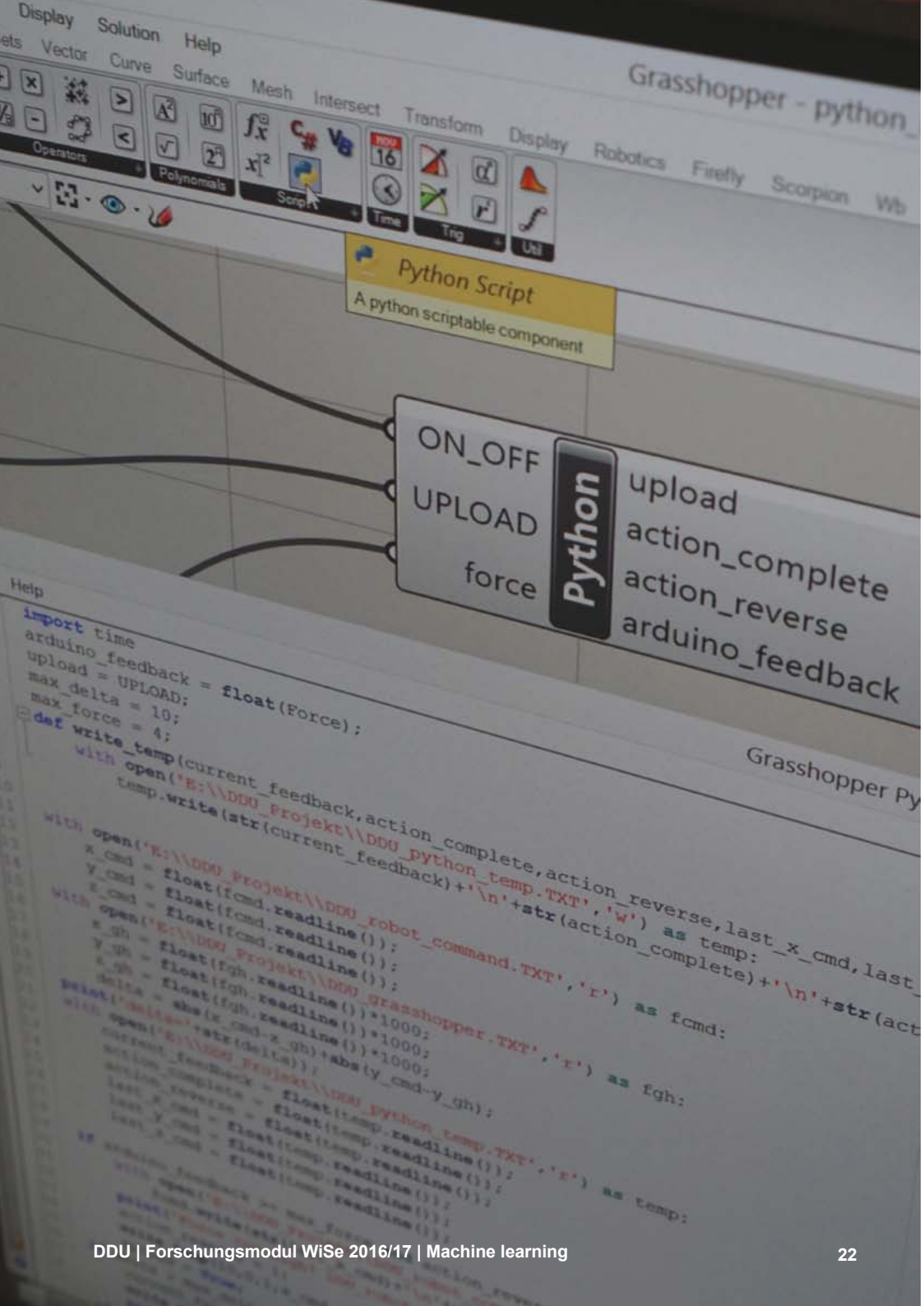




GRASSHOPPER serves as an interface between the robot's own control software and Matlab. It simply passes commands from a ".txt"-file to the robot constraining the robot's arm and therefore the sensor's rod to a vertical orientation. Every 500 milliseconds Grasshopper processes a real-time monitoring from the robot and streams it into a file to make it accessable for a Python script.

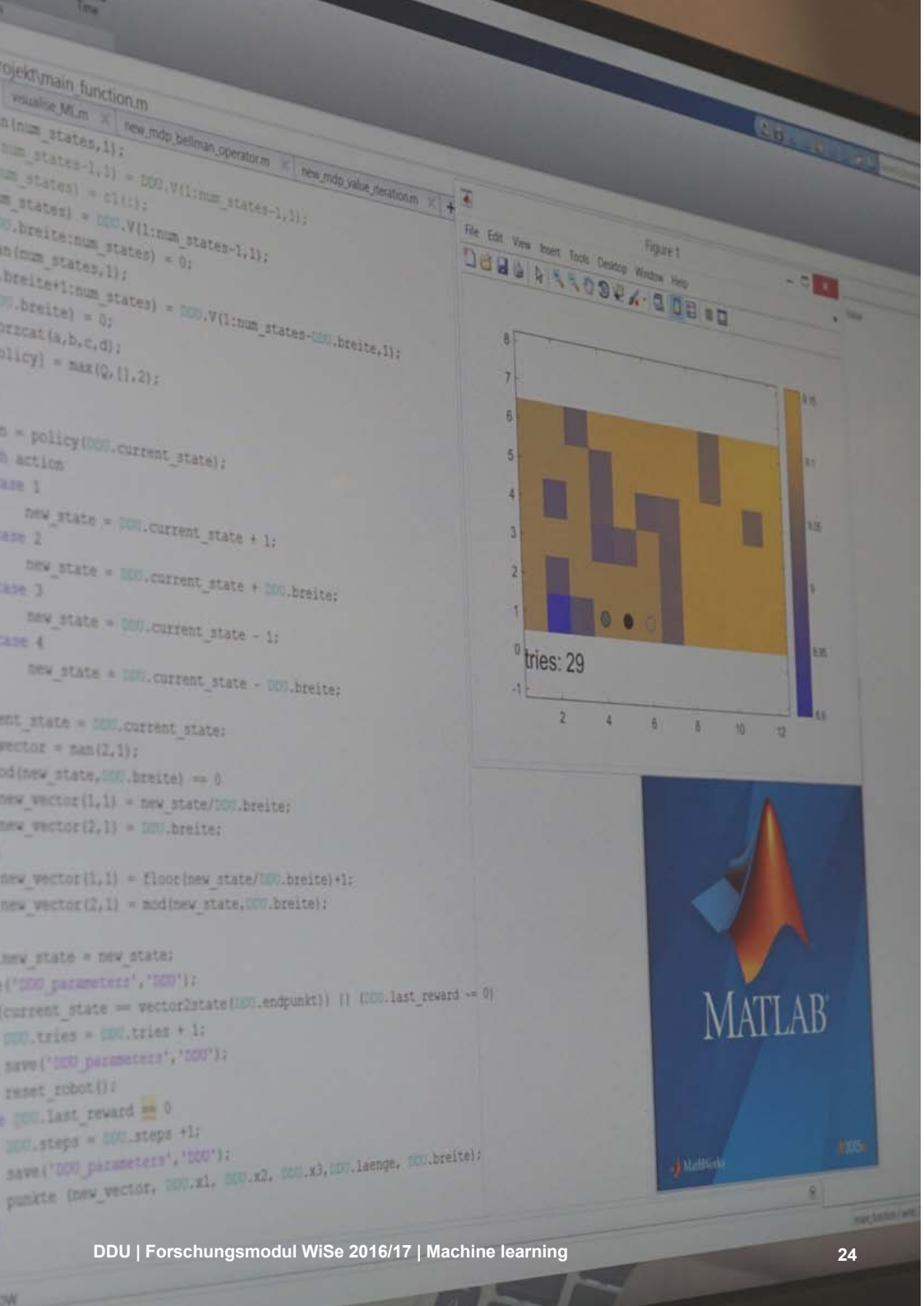


The ARDUINO checks constantly for changes in the values from the two potentiometers. If the change exceeds a certain amount, the Arduino sends an integer via USB corresponding to the deflection of the rod. If no obstacle blocks the movement, the value equals zero.



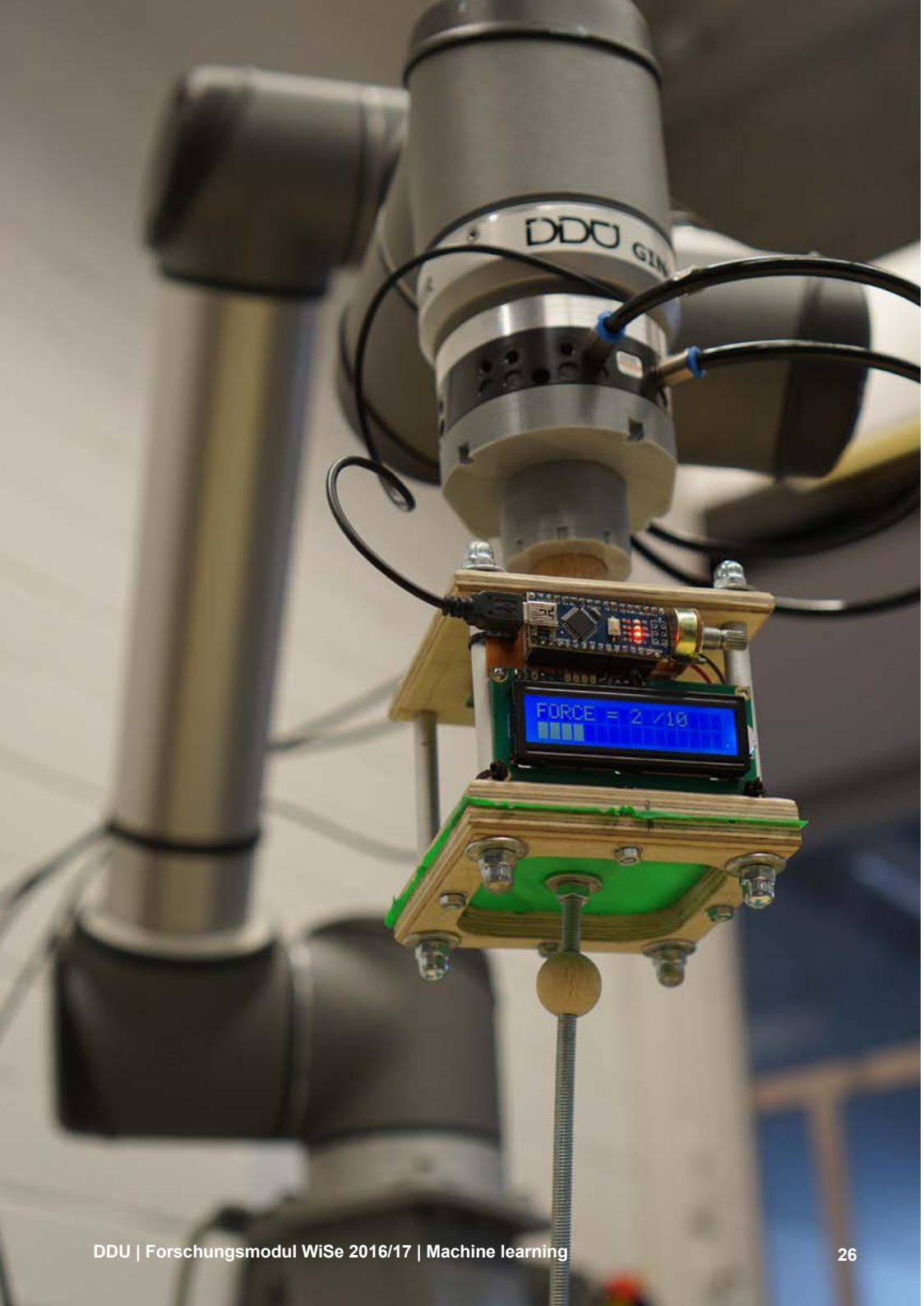
A PYTHON SKRIPT, embedded in a dedicated Grasshopper file, checks the incoming values from the serial port the Arduino writes to. Besides that, it keeps track of the difference between the actual position of the robot and its destination. They are read from the ".txt-files" Grasshopper and Matlab write into.

If the difference gets sufficiently low, Python recognises this as a completed movement and writes a specific feedback into a ".txt"-file. It contains the two points between which the movement was executed as well as the maximum of the Arduino's force value during the movement. If the force value exceeds a certain maximum before the movement is completed, Python immediately overwrites the command file for the robot movement with the point it came from to prevent it from crashing. Again a feedback gets written, now containing a force value of e.g. ten instead of zero.



MATLAB constantly checks for changes in the file the Python script writes into. If a new feedback has been written, Matlab launches a sequence of functions.

In a nutshell Matlab edits the reward-matrix based on the feedback, evaluates a new value function and determines the next action to take, which is to go one step in a certain direction. The new field gets transformed into the coordinates Grasshopper works with and gets written into the ".txt"-file for robot commands.



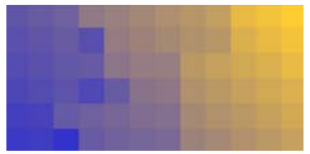
HOW DOES THE ROBOT LEARN?

For our project we decided to use Matlab due to its advantages in dealing with matrices and our familiarity with it. The whole machine learning part of our Project is implemented in Matlab. Merely gathering the feedback we had to outsource to a Python script.
The labyrinth task is an optimisation problem, which can be solved using a “Markov Decision Process“. It is specified by a reward-matrix and a transition-matrix. The reward-matrix assigns to each possible action a certain value (reward). In this case there are at most four and at least two actions possible depending on whether the current state is an edge point or not (up, down, left and right).
The transition-matrix represents the propabilities to get from one state to another performing a certain action. By using the “Bellman equation“, a „value function“ can be iterated, which requires by far the most computation time.
The value function assigns a certain value to each of the states. The higher the value of a state, the better is the state to “collect“ a maximum of rewards performing a minimum of actions. If the reward for leaving the destination state is high and the reward for leaving states with obstacles is low, the value function will reveal a solution to our problem if the policy insists on going to the highest value of the adjacent states.
The problem is that there are no rewards known other than the one for the final state. Ignoring this lack of information, the robot tries to go for the best path based on the current reward-matrix. As soon as the robot collides with an obstacle, the reward-matrix can be refreshed and a better value function can be iterated.

TWEAKING PARAMETERS

To minimize the time the robot needs to reach its destination, we changed the rewards for bumping into an obstacle and performing one step. Below you can see three different plots representing the values of each state after 15, 100 and 1000 attempts. Each row describes a new approach. Without any learning the value function is just a colored gradient which reveals the shortest path without any obstacles (similar to the bottom left mapping).

value function

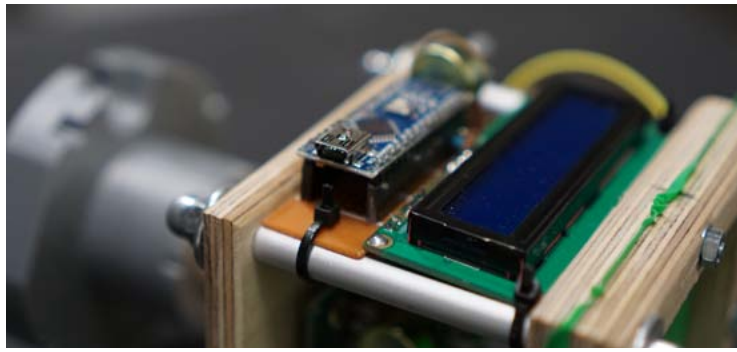
attempts	15	100	1000	rewards: obstacle	rewards: one step	
				-0.1	-0.0001	After 15 attempts the robot crashed into almost every obstacle and reached his destination already after 13 attempts. The downside is that it will not find the correct way if it collects a reward that is actually not true or the problem changes over time.
				-0.01	-0.0001	It takes way more time to finish than in the first example but on the other hand, it is able to proceed after the robot accidently collected a wrong reward. These parameters turned out to suit our problem best.
				-0.001	-0.0001	In this case it takes too long to get a proper result, since the robot could not find the right way after 1000 attempts. Therefore, the punishment for crashing is too low.

Blue fields represent states with relatively low value compared to the relatively high values of the yellow fields.

POSSIBLE IMPROVEMENTS

Due to the Matlab calculations between each step, the robot takes quite a long time to start moving again after finishing one step. So trying to start the calculations while moving from one point to another might be a way to accelerate the progress.
For now, the task of the robot is to learn the positioning of the obstacles, to avoid these and to find the right way through our labyrinth. But it is not able to recognise when it finds the fastest path to its destination. Therefore, we need an algorithm which checks if the current path is the fastest one so far and calculates the probability of a faster path in the future.
Introducing new obstacles into our labyrinth would first of all need an improvement of the force sensor. For the moment its feedback is not accurate for small deflections.
However, the most important task is to implement an user interface so everybody is capable of using the programme.

get a short IMPRESSION
through our project video



PROSPECTS

As mentioned earlier we had some other obstacles than just solid blocks in mind to do tests with. Due to the lack of time were not able to realise it. Nevertheless, testing some of those seemingly more complex labyrinths could reveal unexpected advantages or show the limits of our approach on this task.

The Arduino does already provide a usable feedback with five increments. Turning a few blocks around to make them movable and implementing a corresponding reward change would be possible without too much effort.

REFERENCE

Reinforcement Learning: A Tutorial, 1997
Mance E. Harmon, Stephanie S. Harmon
http://www.cs.toronto.edu/~zemel/documents/411/rltutorial.pdf (13/02/17 9 a.m.)

Introduction to Reinforcement Learning with Function Approximation, 2015
Rich Sutton
https://www.youtube.com/watch?v=ggqnxyjaKe4 (13/02/17 9 a.m.)
http://incompleteideas.net/sutton/Talks/Talks.html#RLtutorial (13/02/17 9 a.m.)

Markov Decision Process (MDP) Toolbox for Matlab, 1999
Kevin Murphy
https://www.cs.ubc.ca/~murphyk/Software/MDP/mdp.html
(13/02/17 9 a.m.)

Artificial Intelligence: Foundations of Computational Agents, 2010
David L. Poole and Alan K. Mackworth
http://artint.info/html/ArtInt_227.html (13/02/17 9 a.m.)

The Markov Decision Problem, Value Iteration and Policy Iteration, 2003
Cyrill Stachniss and Wolfgang Burgard
http://ais.informatik.uni-freiburg.de/teaching/ss03/ams/DecisionProblems.pdf
(13/02/17 9 a.m.)

Vorlesungsnotizen zur Veranstaltung „Einführung in das wissenschaftlich-technische Programmieren“, 2016
Technische Universität Darmstadt, Alf Gerisch

Markov Decision Processes (MDP) Toolbox, 2009 (revision 20/01/2015)
Marie-Josee Cros
https://de.mathworks.com/matlabcentral/fileexchange/25786- markov-decision-processes--mdp--toolbox
(13/02/17 9 a.m.)

SPECIAL THANKS TO

Prof. Dr.-Ing. Oliver Tessmann
Dipl.-Ing. Bastian Wibranek
Andrea Rossi, M.A.
Dipl.-Ing. Anton Savov
Alexander Stefas, Diplom Media Systems Designer