Reinforcement Learning with LEGO Mindstorms

Project Report

Per Steffen Czolbe (wjq874)

Handed in: January 2, 2019

1 Abstract

Contents

1 Abstract **1**

2 Introduction **2**
2.1 EV3 robot . . . 2

3 Development Setup **2**
3.1 Ev3dev Platform . . . 3
3.2 Control by external Computer . . . 3

4 Sensors and Motors **3**
4.1 Time Delays . . . 3
4.2 Motors . . . 3
4.3 Gyroscope accuracy . . . 6
4.4 Ultrasonic Sensor . . . 6
4.5 The Colour Sensor . . . 6
4.6 Calibration . . . 6
4.7 Building Techniques . . . 7

5 Reinforcement Learning with LEGO Mindstorms **7**

6 Colour Detection **7**
6.1 Problem . . . 7
6.2 The Robot . . . 7
6.3 The Learning . . . 7
6.4 Results . . . 7
6.5 Discussion . . . 7

7 Crawl-Robot **8**
7.1 Robot Design . . . 8
7.2 Formulation of the Learning Problem . . . 8
7.3 Results . . . 9
7.4 Discussion . . . 11

8 Swing-Robot **13**
8.1 Problem . . . 13
8.2 The Robot . . . 13
8.3 The Learning . . . 13
8.4 Results . . . 13

8.5 Discussion 13

9 Future Work **13**

10 Individual contributions **13**

2 Introduction

scientific goal, educational goals, agenda by lego

2.1 EV3 robot

The EV3 system is the third version of the LEGO Mindstorm family of robot toys. It contains a central processing component, called ”Brick“, to which motors, sensors and other peripheral devices can be attached. All these components are embedded into plastic casings, which make them compatible with other LEGO parts. The LEGO Group markets the EV3 system to educational facilities and private persons in a package with build instructions and parts for multiple robots. The current product offering consists of 2 different packages to build a total of 9 robots, including the self-balancing two wheeled “Gyroboy” and a sorting machine that orders LEGO bricks based on their colour. The playful development of further robots by end users is encouraged. A large community of engaged hobby robot builders has formed around the EV3 platform.

Hardware

The Brick is the central processing component of each EV3 robot. It can be controlled via multiple buttons, features a small LCD screen and can be connected to a computer via Wifi, Bluetooth or USB. For peripherals, up to 4 motors and 4 Sensors can be connected. The specifications of the embedded system are [1]:

CPU: 32bit, 300MHz ARM9 Processor
Memory: 64Mb DDR RAM
Storage: 16Mb Flash, up to 32Gb SD-Card
Communication: USB, Bluetooth, Wifi
Peripherals: 4 motors, 4 Sensors

Software

The LEGO company ships the EV3 platform with a software system centred around its own proprietary graphical programming language based on LabVIEW. The graphical interface enables people with limited programming experience to quickly develop simple programs that can be executed on the Brick. The graphical language is supported by an editor available as a program for Windows and Mac, and as an app for Android and iOS.

While the default software is well supported by the editor and operating system of the brick, more experienced programmers might prefer a more capable language. Multiple community driven software packages exist for this purpose. These support languages such as matlab, python, java, javascript, C++, C, GO, Ruby, Perl, R, Lua, even direct control of the robot via command line is possible [4]. We discuss a specific development set up in the next section.

3 Development Setup

In this section we present the development set up chosen for this project. It consists of the open source operating system ev3dev to power the brick, the ev3dev python library to control motors and sensors, and remote procedure calls to communicate with the robot from a host computer the via jupyter notebook.

3.1 Ev3dev Platform

Ev3dev is a Debian Linux-based operating system that runs on several LEGO Mindstorms compatible platforms including the LEGO Mindstorms EV3 and Raspberry Pi-powered BrickPi. It is maintained by a community of enthusiasts and offers libraries to interact with LEGO Mindstorms motors and sensors in many programming languages, such as Java, Ruby and Python [4]. The Linux disk image can be downloaded at www.ev3dev.org and flashed on a SD card. Once the SD card is inserted into the LEGO Mindstorms brick, the brick will boot into the ev3dev operating system. A set up guide with detailed explanations on installation and guides on how to get started with development are available at the ev3 website.

3.2 Control by external Computer

While the brick, after flashed with the ev3dev image, runs a Debian Linux system that can execute general purpose programs, the processing and memory constraints imposed by the hardware limit the complexity of programs executed on the brick severely. Installing a python package takes 2 minutes. Executing more computationally expensive tasks like deep learning or real time video processing on the brick is not feasible.

To circumvent these problems, a setup of the main program running on a computer and interacting with the brick to move motors or read sensor data has been employed by multiple community projects, such as a rubics cube solver [3]. We follow the design approach pioneered by these projects. The brick runs a remote procedure call server, implemented by the python3 library `rpyc`. This allows a host computer to connect to the brick and remotely interact with motors and sensors. All program logic is executed on the computer, the brick is listening to commands. Next to the performance improvements, another benefit is that the program can be executed in a jupyter notebook on the computer, which allows to better integrate documentation and visualization into the development process.

4 Sensors and Motors

In this section we examine the peripheral devices of the EV3 platform. We examine the behaviour of motors and sensors, design experiments to measure accuracy and time delays. We discuss the need for calibration and give general advice on building solid and functional robots.

4.1 Time Delays

Asger

4.2 Motors

The LEGO product line includes a variety of 9V DC electrical motors, which can be used to power a wide range of LEGO models, such as cars, trains and robots. The current product offering consists of two main families: “Power Functions” motors and “EV3” motors. The most common sizes of these product lines are shown in figure 1.

Power Functions Motors

The “Power Functions” line of motors is designed to work with a battery pack and is intended to be controlled either with a physical switch on the battery pack or a remote control. Adapters for the power cables are required to use them in combination with the EV3 platform. [10]

By changing the current and polarity of the DC power supply, these motors can spin in both directions and different speeds. Because ‘Power Functions” motors do not measure rotation or angle, robots utilizing them often require other means of measuring mechanical movements. Compared to the EV3 motors, they offer similar torque and mechanical power, have faster reaction speeds but offer no feedback. [6]

Figure 1: Overview of current (2018) LEGO motors. Upper row: Power Functions large and X-large motor. Bottom row: EV3 medium and large motor. Images from [6].

EV3 Servo Motors

The EV3 robot kit contains a medium and two large servo motors. The EV3 motors use tacho feedback to measure their alignment and rotation, which allows for more precise control compared to "Power Functions" motors. [5]. In strength and speed they are comparable to their "Power Functions" counterparts. The large EV3 motor is roughly twice as powerful as the medium one, and comes in a bigger case with different mounting points. Detailed comparisons of the mechanical and electrical properties of all LEGO motors is available at [6].

The ev3dev programming environment offers functions for low level interactions with the EV3 motors. Available are methods to start and stop the motor, reading the angle, setting the speed and 3 different stopping patterns (coast, brake and hold). The medium and large EV3 Motor offer identical programming interfaces. For angle measurements, the motor saves the rotation at system start as 0 degrees. During operation, the current rotation angle offset to the starting point can be read by the motor. [2]

Motor Accuracy

Reinforcement learning algorithms rely on repetition of the same motions many times. To assume a non-adversarial environment and ensure reproducible results, actions taken by the motors need to be accurate and consistent. We design an experiment to measure the accuracy of the motors after a series of repeated actions.

Experimental Setup

The experiment consists of an arm attached to a motor. We rotate the arm in a 90 degree angle back and forth. After every 25 iterations we measure the offset from the starting position externally with a set square. To measure the influence of weight on the results we use two independent arms, one without a weight and the other one weighted by three heavy metals balls. The experimental set-up is shown in figure 2.

The ev3dev python bindings offer two different movement patterns: `'on_for_degrees()'`, which rotates the motor relative to the current position, and `'move_to_pos()'`, which rotates the motors to an absolute value. Further, three stopping patterns exists: `'brake'`, `'hold'` and `'coast'`. In the experiment we will test different combinations of movement and holding patterns, to determine the configuration with the highest amount of accuracy and reproducibility. Since we want high accuracy, we do not test the coasting setting.

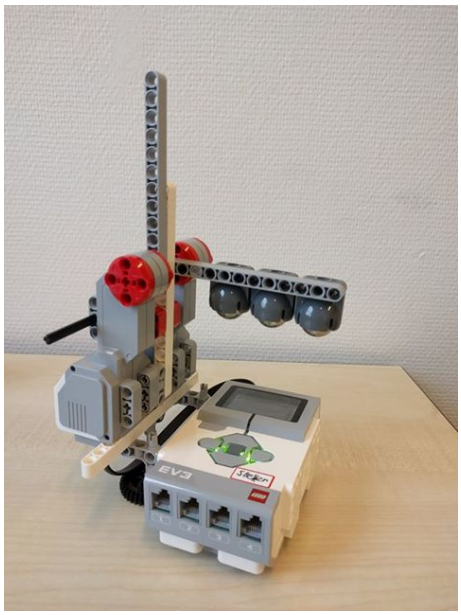


Figure 2: Experiment to measure motor accuracy. The two arms (gray) swing back and forth in a 90 degree angle. One arm is weighted down by three heavy metal balls.

Move Pattern	Stop Pattern	Weight	offset after 25 iterations	after 50it	after 75it	after 100it
relational	hold	no	1	1	2	2
relational	hold	yes	9	11	35	45
relational	brake	no	1	1	1	2
relational	brake	yes	35	50	>50	>50
absolute	hold	no	0	1	2	1
absolute	hold	yes	2	1	4	3
absolute	brake	no	1	2	2	3
absolute	brake	yes	4	6	5	4

Table 1: Experimental result. Angle offset in degrees after 25, 50, 75 and 100 iterations for different configurations.

Results

The angle offsets measured after 25, 50, 75 and 100 iterations for different motor control settings and configurations are shown in table 2. As we can see, the non-weighted arm is very accurate in all scenarios. The results for the weighted arm differ drastically between configurations. We can observe two trends:

1. The `'brake'` stopping pattern creates more offset than the `'hold'` pattern. Further observations of the experiment let us conclude that `'brake'` stops the motion of the motor, but does not block the motor for further movement after the initial momentum has been eliminated. This leads to the weighted arm pulling slightly further down between the breaking phase and the next swing of the arm. The `'hold'` stop motion holds the arm in place even after the initial momentum is eliminated. It leads to more accurate results under load.

2. Movements ot absolute positions are more precise than relative movements. While inaccuracies of relative movements accumulate over time, movements to absolute positions only carry the inaccuracy of the last movement.

During the experiment we also measured the range of motion that is possible without receiving any push-back from the motors. This amount of slack is consistently measured at about 12 degrees. No configuration lowered this inaccuracy. If it causes a problem later on, mechanical solutions such as a worm gear could be used to reduce the slack.

In conclusion, to attain the highest level of accuracy and reproducibility, motors should be moved to absolute positions and the stop pattern `'hold'` should be used. We note however that motors still allow for free movement of about 12 degrees without providing any breaking or pushback.

4.3 Gyroscope accuracy

4.4 Ultrasonic Sensor

The EV3 Ultrasonic Sensor generates sound waves and reads their echoes to detect and measure distance from objects. It can also send single sound waves to work as sonar or listen for a sound wave that triggers the start of a program. Distances mesured range from 0-250cm. While the python bindings of this sensor return a distance measurement with an accuracy of up to 1mm [2], LEGO advertises an accuracy of "up to +/-1cm". [9]

While working with the sensor on the crawling robot, we noticed the importance of mounting the sensor on a stable frame. It should not tilt vertically or horizontally, as any change in angles also changes the distance to a target. To attain accurate and reliable distance measurements, the target should be a rather large, solid plane, oriented orthogonal to the ultrasonic waves emitted by the sensor. The accuracy is sufficient to measure sub-centimetre movements. Roughly 0.1% of measurements were faulty and reported the maximum distance value of 255cm. No further experiments to evaluate the accuracy of the ultrasonic sensor have been performed.

Figure 3: The EV3 ultrasonic sensor. Image from [9].

4.5 The Colour Sensor

Asger

4.6 Calibration

Servo motors and gyroscope sensors are initialized when the brick is restarted. All other sensors require no calibration.

On system start, the position of the motor is initialized as 0 degrees. If the robot is dependent on absolute positional values, it is important to calibrate the 0-point. This can either be done by moving the motor forcefully into the correct position before starting the brick, or using more elaborate constructions of moving the motor against a button, until the button is pressed. This gives certainty over the 0-position of the motor and can be used to calibrate the rotation.

The gyroscope initializes its orientation at system start as 0 degrees. Further, it assumes to be not in motion during system start. All acceleration measurements are in relation to this value taken during system start. Since the rotational angle returned by the gyroscope is calculated as the integral over acceleration measurements, even

a slight inaccuracy in acceleration will accumulate to a significant drift in angle over time. This drift has to be monitored and adjusted for. This can be done either programmatic, or by restarting the system to re-calibrate the sensor.

4.7 Building Techniques

During the building process, multiple key takeaways for building with LEGO were:

- To improve reproducibility of experiments, the LEGO builds should be as solid as possible. Big, complicated constructions tend to flex and bend. Small, simple and solid designs are most easy to work with.
- Avoid using gears and other moving parts as much as possible. These introduce additional complexity, points of failure and tend to make the build less rigid.
- Account for plenty of space for cables. Motors and sensors need to be connected to the brick with stiff cables. Additionally, leave space for the charging cable and a USB connection to a computer.

5 Reinforcement Learning with LEGO Mindstorms

general issues arising when working with actual robots vs simulation

6 Colour Detection

Asger

6.1 Problem

what task solved

6.2 The Robot

description of the robot build

6.3 The Learning

implementation of learning, choices made in picking and implementing algorithm

6.4 Results

final results

6.5 Discussion

includes issues / future work

7 Crawl-Robot

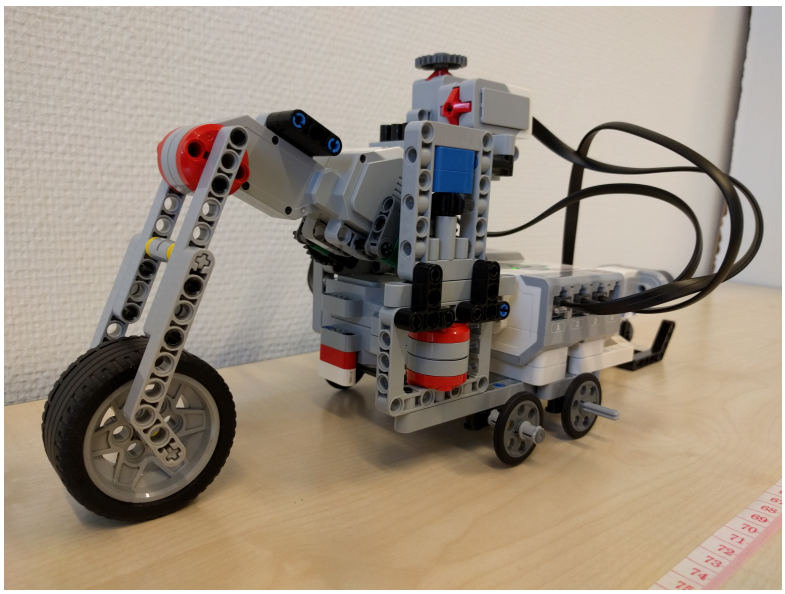


Figure 4: Crawl Robot, frontal view.

We seek to demonstrate reinforcement learning on a simple robot. While many robot designs and learning goals are possible, we selected a crawling robot as a suitable object for demonstration. The concept of a robot using one arm to crawl, or drag itself along a linear track has been explored by multiple other studies [7] [11]. We selected this task first, as it is a comparatively simple design and we know the problem can be solved. Realizing this robot will equip us with the experience and knowledge to solve more complex problems in the future.

7.1 Robot Design

To facilitate the crawling motion, the robot needs a movable arm to pull itself forward. The arm consist of two elements: the base arm, which is attached to the robot, and a pivot arm attached to the base arm. Both arm joints rotate around the same axis, allowing the hand at the far end of the pivot to move on a plane. To allow the arm to pull the robot, the ground friction of the arm has to be maximized, and the ground friction of the robot minimized. This is achieved by a big rubber tyre firmly attached (not spinning) to the robots hand. The robot body sits on wheels which can spin freely. See figure 4 for a photo of the robot arm, hand and wheel basis.

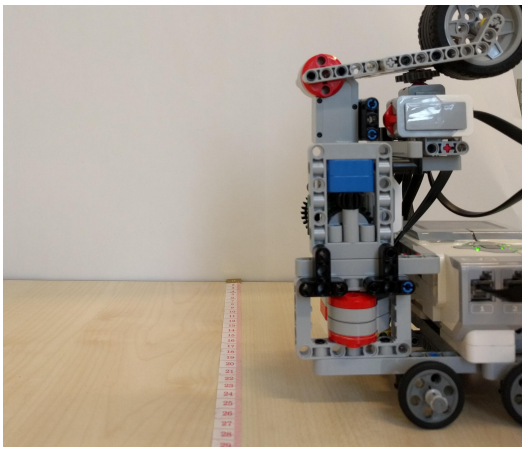
To measure distance travelled, the ultrasonic sensor is used. This sensor is attached to the back of the robot and records the distance to a target from the rear end of the robot. As the wheel basis of the robot might tilt when the arm is pressing into the ground, while the ultrasonic sensor requires a stable angle towards the target for consistent measurements, the sensor is mounted on a sled which is pulled behind the robot. The sled is visible in figure 5a.

To calibrate the motors of the arm, two buttons are attached to the top of the robot. During the start up process, the arm is moved all the way back, until base arm and pivot arm hit their respective buttons. This motion is depicted in figure 5b. The certainty about the arm position attained by this action is used to initialize the 0-angle of both motors.

7.2 Formulation of the Learning Problem

To train the robot how to crawl, we use Q-learning. O-learning is a form of reinforcement learning. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions. To learn, an agent tries an action at a particular state, and evaluates its consequences in terms of the

(a) Ultrasonic sensor on a sled at the back of the robot.



(b) calibrating motion of the crawl robot.

Figure 5: Designs of the Crawl Robot

immediate reward or penalty it receives and its estimate of the value of the state to which it is taken. By trying all actions A in all states S repeatedly, it learns which are best overall, judged by long-term discounted reward [12].

The Q-learning algorithm fits the domain of the crawl robot. The implementation chosen is shown in Algorithm 1 and follows [8]. We model an action as moving the arm. The immediate reward or penalty received is proportional to the distance traveled by performing the action, as measured by the ultrasonic sensor. As learning speed is of importance, we define a discrete state space with 9 states. These states are displayed in figure 6. We could have sub divided the arm movement plane into more states, but learning is expected to take longer as more states have to be explored by the learning algorithm. More formally, the choices for the action-space A and state space S are:

$$A = \{ \text{ move base arm up}, \text{ move base arm down}, \text{ move pivot arm far}, \text{ move pivot arm closer}\}$$
$$S = \{\text{base up pivot far}, \text{ base mid pivot far}, \text{ base down pivot far},$$
$$\text{base up pivot mid}, \text{ base mid pivot mid}, \text{ base down pivot mid},$$
$$\text{base up pivot down}, \text{ base mid pivot down}, \text{ base down pivot down}, \}$$

The intial state for each epoch is given by $s_0 =$ base up pivot far, which is the furthest extension of the arm. We select learning rate $\alpha = 0.5$, discount rate $\gamma = 0.8$. The exploration factor ϵ is initially set to 0.5 and declines exponentially throughout training. We train for a maximum of 100 epochs with 35 steps per epoch.

7.3 Results

The crawl robot was trained using tabular Q-learning with the parameter choices stated in the previous section. In this section we discuss the training process and the learned result. For training, the robot is placed on a linear track. At the back of the robot, a solid target is placed to act as a point for range measurement by the ultrasonic sensor. During training, the robot explores actions and moves back and forth on the linear track.

After 20 minutes of training, the learned motion appeared near perfect and training was stopped. This corresponds to 65 epochs of training, for a total of 2275 actions. The reward gathered in each epoch is plotted in figure 7a. We can see that the reward gathered in each epoch has a clear upward trend, indicating that the robot

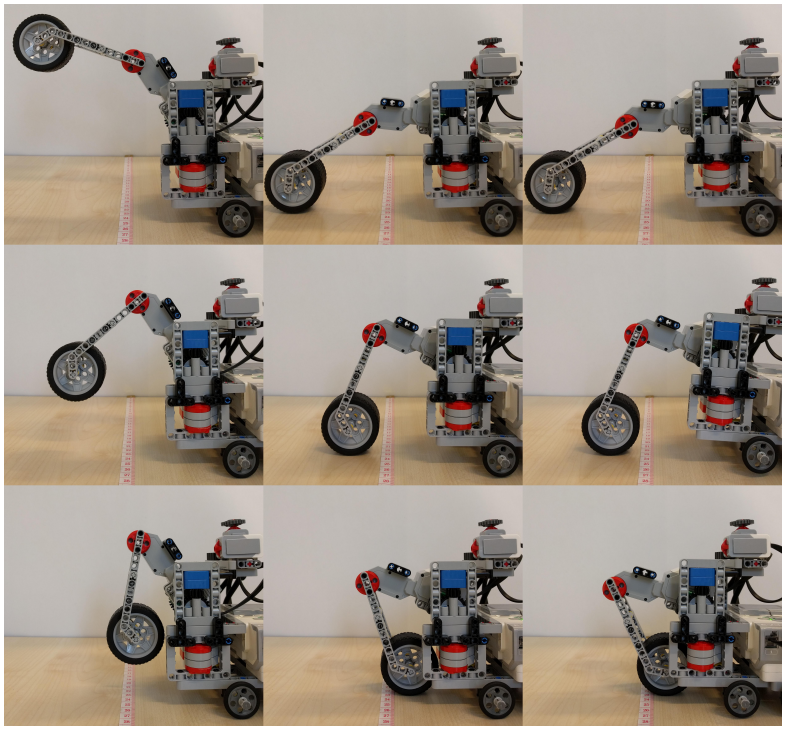


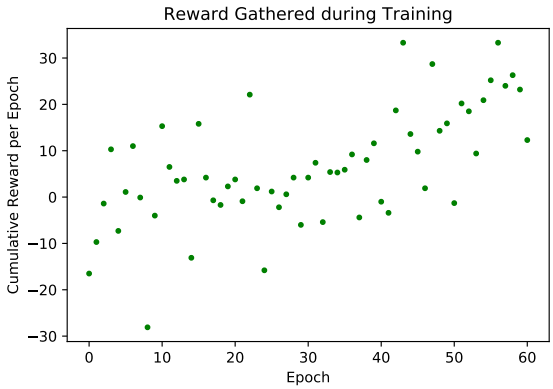
Figure 6: Arm positions (states) of the crawl robot. From top to bottom: Pivot arm in positions far, mid, close. From left to right: Base arm in positions up, mid, down. The tilt of the robot base in the right column is caused by the arm pressing into the floor.

Algorithm 1: Q-learning with ϵ-greedy exploration

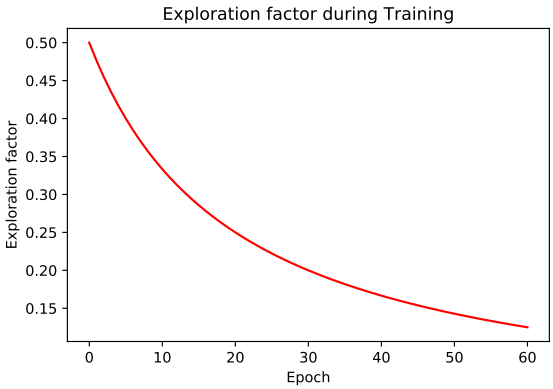
Input : learning rate $\alpha \in]0, 1]$, discount rate $\gamma \in [0, 1[$

initialize $Q(s, a)$ with 0 for all s, a
foreach *epoch* **do**
  return to initial state s_0
  repeat
    sample x uniform at random for $[0, 1]$
    if $x < \epsilon$ **then**
      sample a at random
    else
      select $a \leftarrow \arg\max_a(Q(s, a))$
    end
    take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma\, max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  until *epoch end*;
end

Output: state value function Q



(a) Reward gathered per Epoch. An improvement is clearly visible.



(b) Exponential decline of the exploration rate.

Figure 7: Q-learning of the Crawl Robot

got progressively better at the task. The exploration factor is exponentially declining, as shown in figure 7b. This leads to less exploration in later epochs. The result of this is visible in the reward plot, as in later epochs the results get more consistent.

The robot is not fully autonomous during training. It requires human intervention in three cases:

- The ultrasonic sensor has a maximum range of 250cm. If the robot moves further away from its range measuring target, it can no longer track rewards. Similarly, if it pushes into the range finding target, it can not move further into the direction of the target. Both cases disrupt assessing the rewards of actions. To handle these cases, the robot will interrupt training when it gets too close to either end of the spectrum. Training is resumed once the robot is manually placed on a valid part of the training track.
- To properly assess the range to it's range measuring target, the ultrasonic sensor needs to be orthogonal to the target. If the robot rotates during training, the angle to the range finding target can change. An angle significantly different from 90 degrees leads to skewed reward tracking, and in special cases the sensor can no longer find the target, which prohibits range measuring entirely. If the orientation of the robot changes significantly, it has to be manually rotated to its initial position.
- In rare cases, the motors of the arm have slightly too little torque to push against resistance caused by the hand hitting the floor. The motors become stuck. In these cases the robot needs to be slightly pushed to overcome the resistance and release the motors.

A recording of the full training and a presentation of the learned motion is accessible as a video online at youtu.be/NUTv-oNWEYo. The video shown the training process at 7 times normal speed. Manual interaction is shown as well.

7.4 Discussion

With the presented framework, the crawling robot is able to learn a forward crawling motion in 20 minutes. As the training process requires constant human interaction for resetting the robot, we think this amount training time is appropriate. Especially in educational settings it should be considered as an upper limit. Since the training is interactive, observing how the robot develops an increasingly stronger forwards drive during training is quite rewarding to the observer.

The learned motion is not smooth and appears abrupt, which results from the use of a discrete state space. Learning a crawling motion with this robot on a continuous state space has the potential of leading to a smoother, and possibly faster movement. A possible drawback is increased training time. Moving the arm on a continuous state space is a topic for future research.

The robot construction can be improved in the future. The current design has some mechanical issues, such as the strength of the motors in the arm and a tendency to slightly drift off the linear direction of movement. These problems could be addressed by re-constructing the robot and track of movement. A guide rail could constrict sideways the movement and gear conversions in the arm can be used to increase torque. The need for human supervision could be reduced by introducing a third motor to automatically move the robot back to its starting position after every epoch.

In conclusion, we build a one-armed robot using LEGO parts, motors and sensors. We formulated a tabular q-learning problem with the goal of teaching the robot how to crawl. With some human interaction, the robot learned a forward crawling motion in 20 minutes.

8 Swing-Robot

8.1 Problem

what task solved

8.2 The Robot

description of the robot build

8.3 The Learning

implementation of learning, choices made in picking and implementing algorithm

8.4 Results

final results

8.5 Discussion

includes issues / future work

9 Future Work

Shared/independant future robots/ multi actor learning

10 Individual contributions

Table 2 shows the individual contributions to the report by each of the graded students.

Steffen	Asger
Introduction	
Development Setup	
Motors	
Ultrasound Sensor	
Calibration	
Building techniques	
Crawl Robot	

Table 2: Sections of the report written by each of the students

References

[1] Lego mindstorms ev3 hardware developer kit. *ev3dev.org*, 2013.

[2] EV3 python bindings. *Github Repository*, 2018.

[3] LEGO Mindstorms Rubics Cube Solver. *Github Repository*, 2018.

[4] Ev3dev plattform. *ev3dev.org*, 2019.

[5] T. L. Group. EV3 Large Servo Motor, shop descitption.

[6] Philippe Hurbain. LEGO® 9V Technic Motors compared characteristics. *philohome*, 2018.

[7] V. Rousseau. Lego mindstorms: One-legged robot learning how to crawl with q-learning.

[8] R. S. Sutton and A. G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[9] The LEGO Group. EV3 Ultrasonic Sensor - Product details. *LEGO.com*, 2018.

[10] The LEGO Group. Products and Sets - LEGO® Power Functions. *LEGO.com*, 2018.

[11] T. Timm. Reinforcement learning with lego mindstorms - crawling robot.

[12] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.