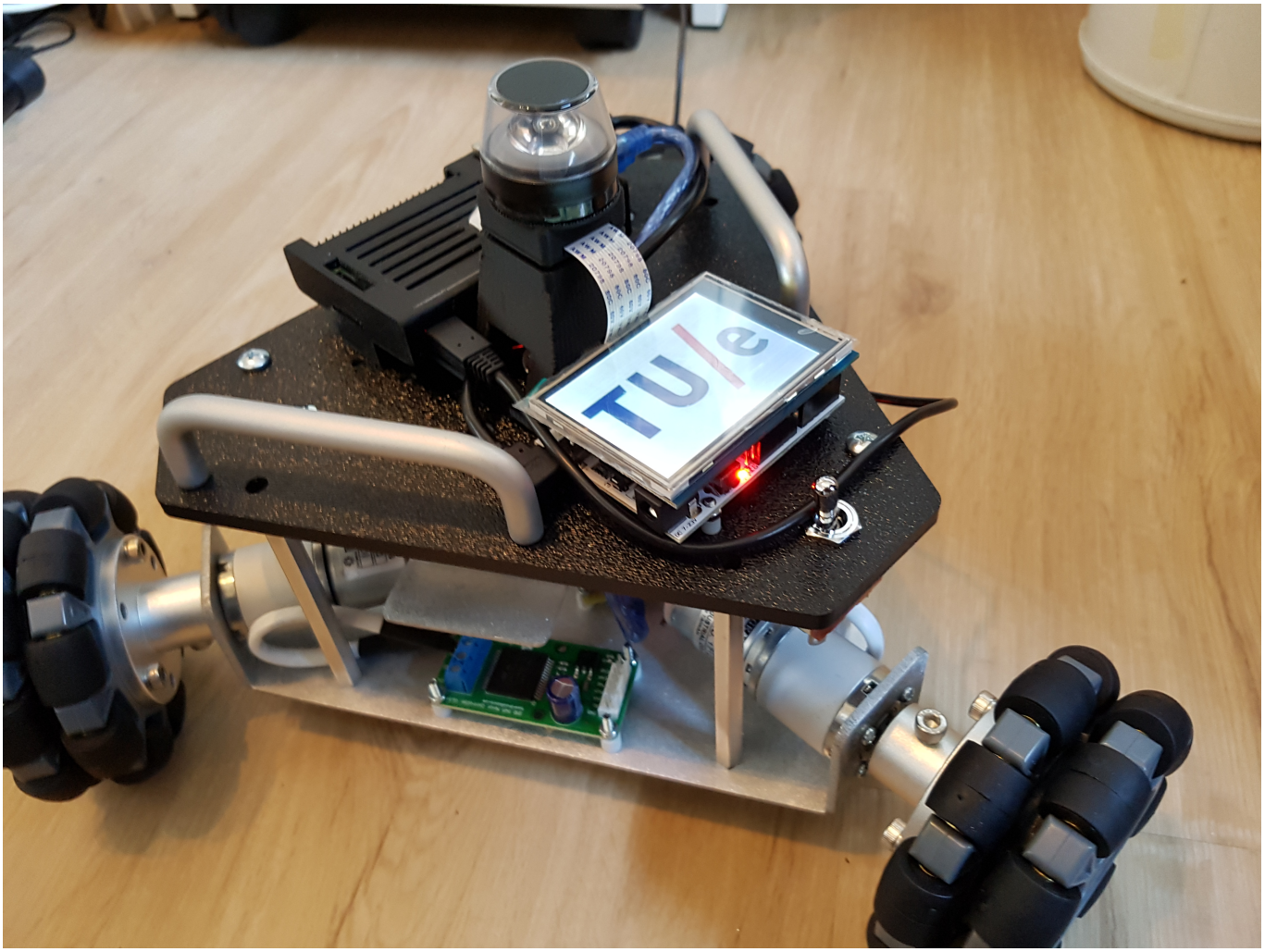
TU/e

Building the TU/e Mini-Turtle Robot

Peter van Lith
Technische Universiteit Eindhoven
vanlith.peter@gmail.com

June 23, 2019

Abstract

In 2016 we started a project to recognize robots with the Tech United MSL team during a competition using Neural Networks. During work on this project the idea to build a smaller version of the Turtle robots emerged to serve as a testbed. This document describes the development of this platform. Gradually this platform grew into an independent robot that was intended to learn by itself to recognize obstacles and dribble with a ball in a room with unpredictable obstacles.

1. Introduction

This project is a combination of an autonomous soccer playing robot that uses a Neural Network both for image classification and localization and a learning environment in which a GPU-equipped laptop is used to train a Neural Network used for object recognition and learning simple soccer-playing behaviors.

2. The Hardware

2.1. The Basic Robot

The robot is a modified SuperDroid 3-Omniwheel Vectoring Robot. Normally this robot has the motors and drive electronics under the bottom layer. This gives the robot a high clearance but has the disadvantage that any obstacles that are on the ground can damage the electronics or the wiring. So we decided to make the underside of the robot flat and mount the motors and electronics on the inside. This makes the clearance lower, but protects all mechanical and electronic parts.

The top layer is reserved for all other electronics like the power supply, batteries, fuses and wiring, which are all mounted at the bottom of the top layer. On top of the robot are the processors, which are easily accessible. Two handles on the sides are used to carry the robot and also to protect the processors.

This robot kit comes in three versions: a basic kit with only the layers and motors, a version with mounting brackets and a programmable version that includes an Arduino. We selected the kit with mounting brackets and provided our own electronics. We also modified the layout, so all components could fit inside the body of the robot. It is smaller than our MSL turtles and therefore easier to carry along for tests and demonstrations.

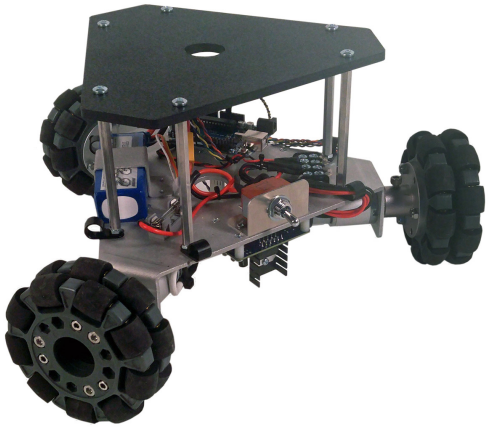

Figure 1: *SuperDroid standard TP-251-003 Robot Kit*

2.2. The Bottom Layer

The bottom layer consists of an aluminium frame on which the motors are mounted. We inverted the base plate, so that the motors and electronics are on the inside of the robot and the bottom is entirely flat.

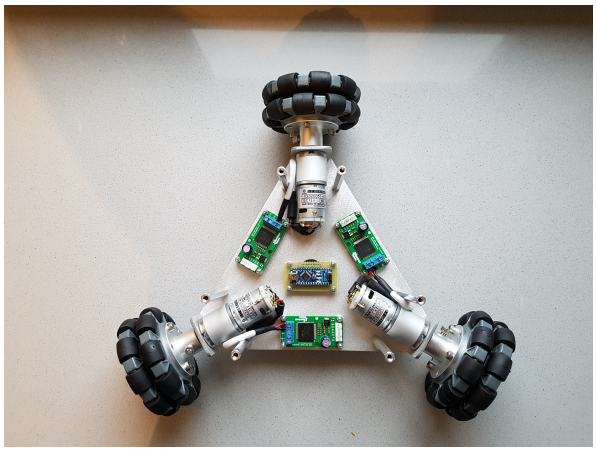

Figure 2: *The bottom layer of the robot*

The three motors each have a driver board that is connected to the Arduino Nano in the middle. This processor is connected through a USB cable with the Main Processor and controls the three motors. It also monitors the battery levels.

2.3. The Top Layer

The Top layer has both a Top and Bottom part. The Bottom part hosts the batteries, power control and all power wiring. In the picture you see the various components from bottom to top:

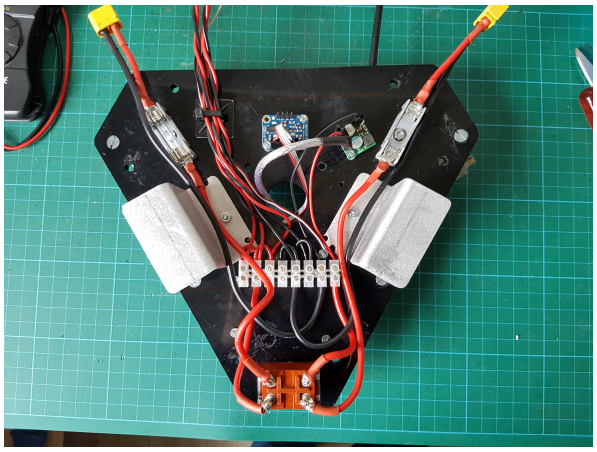

Figure 3: *The bottom of the toplayer with wiring*

1. The Power Switch.
2. The cable connector rail.
3. The battery holders.
4. The fuses with battery connectors.
5. The blue PCB is the Inertial Movement Unit.
6. The step-down buck converter for the RPi. This was removed in 2019 and replaced with a 4 Amp buck converter with display, located on the top layer of the robot, next to the main processor.
7. The power cables leading to the bottom layer.

The upper part of the top layer houses the main processors:

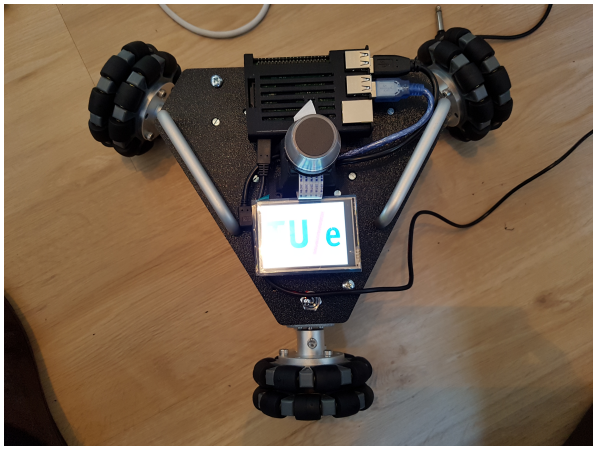

Figure 4: *The toplayer with processors*

1. The Power Switch.
2. The Arduino with touch screen.
3. The Omnidirectional Camera.
4. The Raspberry Pi.
5. The Movidius NCS with power unit. This was replaced in 2019 with the NVidia Jetson Nano, with 128 Cuda cores.
6. The handle bars.

2.4. The Drive Unit

The drive unit consists of the three motors with their motor drivers and the Arduino Nano which controls the three motor drivers. The Nano also has an analog input for the two batteries, so it can measure the power level of each battery. Every motor driver receives its power by a cable from the upper deck.

3. The Processors

The mini Turtle robot is equipped with four processors:

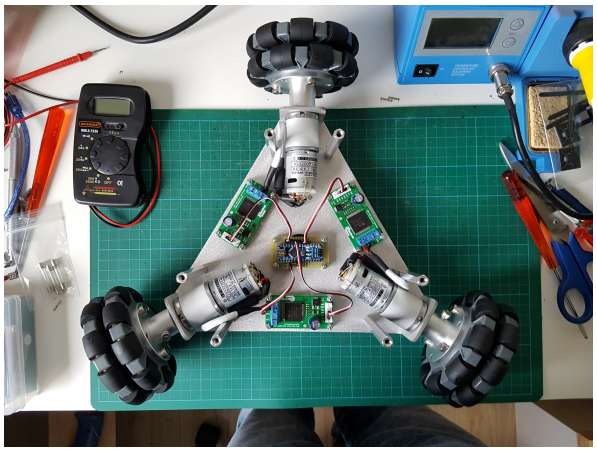

Figure 5: *The bottom layer of the robot with wiring*

Figure 6: *The Raspberry Pi 3B*

1. The Main Processor, a Raspberry Pi 3B, located at the top layer. this processor was replaced in 2019 with the new NVidia Jetson Nano.
2. The
3. The Control Processor with an LCD touch screen, located at the top.
4. The Motor Processor, located at the center of the bottom layer and
5. A Movidius Neural Compute Stick (NCS) also located at the top layer.

The processors are all powered from a 11.1V Lipo battery with a capacity of 2200 mAh through a 5V step-down buck converter that supplies 2.5A.

Figure 7: *The Jetson Nano processor*

3.1. The Main Processor

Initially the Main Processor was a Raspberry Pi 3B with a 32G SD card, connected via a USB cable to the three other processors: This processor is running Raspbian Stretch and has an optional connection with the portable System76 Oryx-Pro, which is equipped with a powerful NVidia GTX-1070 GPU.

In March 2019 this was replaced with the NVidia Jetson Nano, which is an AI version of the Raspberry Pie but has a move poweful processor, 4 Gb of RAM, a 64 G SD card and 4 USB-3 ports. It main advantage is that is has 128 Cuda cores and can run TensorFlow natively. It is a more powerful machine, running Ubuntu 18.04 LST, the same as our main Oryx-Pro processer.

3.2. The Control Processor

3.3. The Motor Processor

The Motor Processor is an Arduino Nano V3, connected to the Raspberry-Pi through USB. The Communications Protocol is used to send and receive messages over the USB port, consisting of a pre-defined sequence of commands. The processor is connected to three motor controllers, whjch send one PWM and two digital signals to each motor controller. The three motor controllers A, B and C are connected through the pins, shown in table X according to the schema in Figure 11

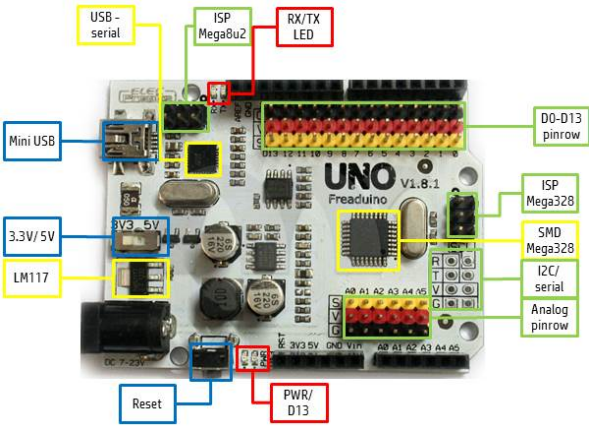

Figure 8: *The Freaduino Uno Pin-out*

Figure 9: *The TFT Touch Screen*

3.4. The Neural Network Processor

In march 2019 the Movidius was replaced with the NVidia Jetson Nano processor which houses 128 Cuda cores and runs standard Ubuntu 18.04 LTS.

Table 1: *Motor Controller Pins*

Connections			
Motor Signal	Blk InA	Red PWM	Wht InB
A	D10	D09	D08
B	D07	D06	D05
C	D04	D03	D02

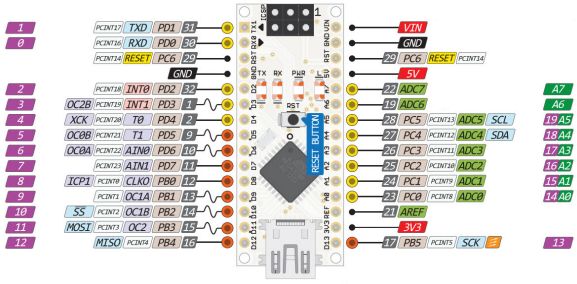

Figure 10: *The Arduino Nano Pinout*

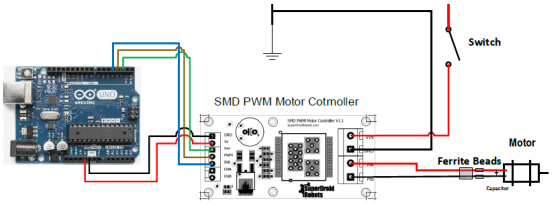

Figure 11: *The Motor Controller Connections*

3.5. The System76 Oryx-Pro computer

4. The Peripherals

4.1. The Camera

4.2. The IMU

The robot uses an absolute Inertial Measurement Unit, the Bosch BNO055. This unit is a 9DOF unit with integrated processor that uses sensor fusion to deliver an absolute orientation, using a compass, accelerometer and gyroscope. All processing to deliver an absolute orientation is done in the sensor itself and it delivers the output over a standard I2C connection. The sensor is connected to the Jetson Nano on the top layer, which will deliver this information to the main processor on request.

If you've ever ordered and wire up a 9-DOF sensor, chances are you've also realized the challenge of turning the sensor data from an accelerometer, gyroscope and magnetometer into actual "3D space orientation"! Orientation is a hard problem to solve. The sensor fusion algorithms (the secret sauce that blends accelerometer, magnetometer and gyroscope data into stable three-axis orientation output) can be mind-numbingly difficult to get right and implement on low cost real time systems. Bosch is the first company to get this right by taking a MEMS accelerometer, magnetometer and gyroscope and

Figure 12: *The Movidius Neural Compute Stick (NCS)*

putting them on a single die with a high speed ARM Cortex-M0 based processor to digest all the sensor data, abstract the sensor fusion and real time requirements away, and spit out data you can use in quaternions, Euler angles or vectors.

Rather than spending weeks or months fiddling with algorithms of varying accuracy and complexity, you can have meaningful sensor data in minutes thanks to the BNO055 - a smart 9-DOF sensor that does the sensor fusion all on its own! You can read the data right over I2C and Bob's yer uncle.

The BNO055 can output the following sensor data:

1. Absolute Orientation (Euler Vector, 100Hz) Three axis orientation data based on a 360° sphere
2. Absolute Orientation (Quaterion, 100Hz) Four point quaternion output for more accurate data manipulation
3. Angular Velocity Vector (100Hz) Three axis of 'rotation speed' in rad/s
4. Acceleration Vector (100Hz) Three axis of acceleration (gravity + linear motion) in m/s^2
5. Magnetic Field Strength Vector (20Hz) Three axis of magnetic field sensing in micro Tesla (uT)
6. Linear Acceleration Vector (100Hz) Three axis of linear acceleration data (acceleration minus gravity) in m/s^2
7. Gravity Vector (100Hz) Three axis of gravitational acceleration (minus any movement) in m/s^2
8. Temperature (1Hz) Ambient temperature in degrees celsius

Handy, right? So we placed this very nice sensor on its own breakout, complete with 3.3V regulator, logic level shifting for the Reset and I2C pins, an external 32.768KHz crystal (recommended for best performance),

Figure 13: *The Koget Dot Omnidirectional lens*

and breakouts for some other pins you might find handy. Comes assembled and tested, with a small piece of header. Some soldering is required to attach the header to the breakout PCB, but its pretty easy work. Best of all you can get started in 10 minutes with our handy tutorial on assembly, wiring, CircuitPython and Arduino libraries, and Processing graphical interface, and more!

5. Power Management

Power is provided to the computer by two 11.1V Lipo batteries with a capacity of 2.2 Ah, controlled by a switch and protected by a glass fuse under the top layer. One battery is used to power the motors, the other one is powering the electronics. There is one 5V 2.5A step-down buck converter (Pololu D24VF5) powering the Raspberry Pi 3B, which in turn powers the two Arduinos through its USB connections.

The USB ports of the Raspberry Pi can deliver a total of 1200 mA to the 4 USB ports, while the RPi itself uses roughly 1500 mA. Because the RPi is rather sensitive to power fluctuations, a 2.5A supply is recommended.

However the Movidius NCS consumes between 200 and 500 mA, which the RPi cannot provide when power-

ing both Arduinos as well. So the Movidius has its own step-down converter, housed in a small custom made USB 2.0 to Mini connector unit.

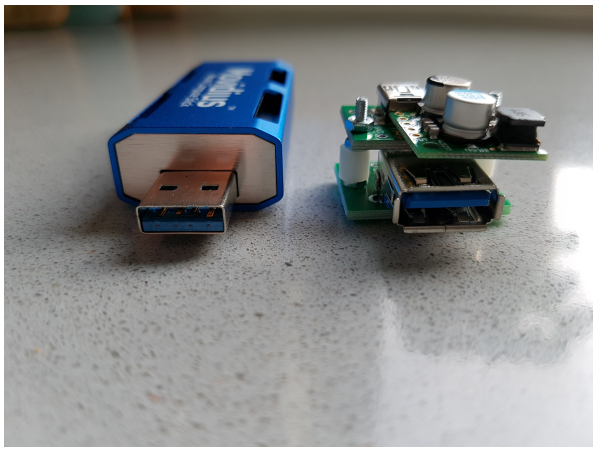

Figure 14: *The Movidius NCS with its own power supply*

6. The Software

6.1. Installing the Jetson Nano

The Jetson Nano runs the ARM64 version of Ubuntu 18.04 LTS, which comes pre-installed on a micro-SD card. In order to run our software a number of packages need to be installed. Also some small errors in the default configuration need to be corrected.

Update and install Desktop sharing

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get autoremove
sudo apt-get install nano
```

See Getting started woth the NVIDIA Jetson Nano Developer Kit from Alasdair Allan about fenabling desktop sharing error in Vino files. Also set Desktop sharing as indicated in this document.

Install Samba

Samba has its own password system. Install this too.

```
sudo apt-get install samba
sudo smbpasswd -a peter
sudo /etc/init.d/smbd restart
```

Add shares for Desktop, Public and Documents

Install PIP and PIP3

Table 2: *Opcode Structure*

Instruction		
Opc	Length	NParms
0b11110000	0b00001100	0b00000011
0xF0	0x0C	0x03

Table 3: *Instruction Set*

Instruction		
Sync	Opcode	Parms
0xFF	0xFF	0xFF
0xFF	0xFF	0xFF

```
sudo apt install python3-pip
sudo apt install python-pip
```

Install Thonny

`pip3 install thonny (not working)`
Install Arduino
Set up file sharing
Synchronize MiniTurtle with other computers.

6.2. Communication Protocol

The Raspberry Pi is the main processor and communicates with the two Arduinos and the Movidius using the USB serial connection. Although the Movidius is a USB 3.0 device, it is used in USB 2.0 mode, because the RPi only has USB 2.0 ports.

To be able to control the two Arduinos a special protocol is developed that makes the execution of remote procedure calls (RPC) in the Arduino possible. This protocol is implemented as a Virtual Machine, where the Arduinos are given variable length instructions and are returning data when executing every function. Every instruction begins with a synchronization byte to make sure that both processors are in sync. When a connection is lost, or a cable is removed, data can still be floating and the RPi will send sync commands until the Arduino responds with an acknowledgment. When synchronization is established the variable length instruction is sent and the RPi waits for the response. The communication with the Movidius is handled by the API, provided for the NCS unit.

7. Learning

The mini-Turtle project aims at providing a real-time, one-shot, continual learning environment for a simple robot that learns continually to play with a ball. To this end it needs to learn to recognize arbitrary balls, to explore its environment, to dribble with the ball and deliver it to a designated area, while navigating its environment. All this needs to happen online and in real-time in a real live environment, without the use of any simulator. Existing learning environments are not well-suited to this task, since they are lacking the following capabilities:

1. They cannot learn sequentially but in batches of large numbers of examples. In our case there is a continuous stream of image- and sensor input that serves as input to the learning agent.
2. Existing one-shot learning algorithms suffer from catastrophic forgetting as new concepts come in. We need a way to let the network grow dynamically.
3. One-shot learning, using iterative, back-propagation learns in small increments and needs many cycles to learn something, while some events may represent dramatic occurrences that should be remembered with much more strength than other, less dramatic events. Current networks do not make such distinctions.
4. As new events occur, new classes need to be learned and some kind of structure or hierarchy needs to grow that also is grounded in real-world objects. There needs to be a correspondence between learned concepts and their real-wprld symbolic representation or meaning.

While in old-school AI systems, like Expert-Systems and Data-Driven approaches everything was built around symbolic concepts, the era of Neural Networks seems to have done away with symbols and merely concentrates on self-learning and model-driven or model-less systems, that bear no relevance to our symbolic world. This makes communication with and understanding of the way these systems operate very difficult. If self-learning systems are to become acceptable to our society, they need to be able to explain why and how they solve problems in our living space and must therefore speak and understand our language and way of live. We will not feel comfortable with systems that alienate themselves from our world of feelings and impressions.

Therefore some fundamental shortcomings in current AI technology will be required before we can allow them to take control of part of our lives on our behalf. Impressive as contemporary deep learning systems are, they do not meet these requirements and much more research to endow them with these capabilities is required.

This project is aimed at finding out, what research has already been done on these topics and use these techniques to provide a simple self-learning platform that exhibits some of these features, while showing us, what capabilities are missing and need further development. This very simple, soccer-playing robot is meant to form a research platform for such a system, operating in a simple environment, executing simple tasks, learning these by experiments and helped by a teacher, using only simple positive or negative feedback signals in the form button pushing and symbolic grounding information.

7.1. The Neural Network

7.2. Image Processing

7.3. What the robot learns

The robot will learn a relatively small number of skills, that are gradually getting better by trial and error. We will include facilities to learn the following behaviors:

1. Finding the Ball. This part is split into recognizing the ball and finding its location.
2. Go to the Ball. When found, the robot must learn how to go to the ball and stop when it gets close.
3. Avoid obstacles. It should explore its environment and find out what obstacles do look like and learn to avoid them.
4. Dribble with the Ball.
5. Find the Goal.
6. Move to the Goal.
7. Score a Goal.

7.3.1 Recognize the Ball

Using its camera input, the system will have to learn to recognize the ball. We do this by pressing a button Ball or NoBall. The current image is then used to let the network learn to recognize the ball. Several augmented versions of the image, in which rotations and distortions

are used to let the network learn the features that determine what a ball looks like. It should also be possible to retrain the network to recognize other balls interactively.

7.3.2 Find the Ball

Once the network has learned to recognize the ball, it needs to find the location of the ball. This is done using a Class Activation Map, in which the highest activation point determines the location of the ball. It will learn to return either a vector and size or an angle and distance.

7.3.3 Go to the Ball

Using the known location of the ball, the robot should move towards the ball and learn to control the three omnidirectional wheels to let the robot move in a straight line, accelerating and decelerating as is goes. It should use the ball position and compass information to maintain direction. However, when it detects an obstacle, it should avoid it. This is a separate function that needs to be learned and integrated with Go2Ball. We will use a Q-Learning approach to let the robot learn this behavior.

7.3.4 Recognize Obstacles

To recognize obstacles, we take an approach that differs from recognizing the ball. We will let the robot explore its environment, more or less randomly. While it is driving around it will learn to associate motor forces with acceleration data. When it experiences a resistance, it knows that there is some kind of obstacle. When it detects a negative acceleration, it knows, it bumped into something. Using Q-Learning it will then learn to associate the part of the image where it collided with the class Obstacle and learn to recognize obstacles in this way.

7.3.5 Avoid Obstacles

Once it has learned to recognize obstacles it will learn to steer away from obstacles. While executing the GoTo commands, it will always keep the obstacles in sight and maintain its direction, while steering around obstacles.

7.3.6 Dribble with the Ball

In this case the robot will go to a certain location, but at the same time must keep the ball. Since the robot has no dribbler, it will have to keep the ball between two of its wheels, while navigating. We will be using Q-Learning as well to let the robot learn this behavior.

7.3.7 Recognize the Goal

7.3.8 Move to the Goal

7.3.9 Score a Goal

7.4. Complexity of the Learning Process

We have no idea yet of how well a small Jetson Nano will be able to learn these behaviors. So for each behavior there will be a separate Neural Network. If the processor is not powerful enough to process this information, we will investigate the possibility of including a Coral TPU processor, to handle more demanding tasks. This may mean that the learned networks need to be compiled and converted for the TPU Accelerator and that new behaviors are learned offline. We will see how far we get with this. As the process of learning of each behavior is similar, we may learn each behavior separately and then later on transfer them to the more powerful TPU Accelerator. Since that only handles inferencing, it can handle more tasks than the Jetson GPU's. There is also the possibility to later upgrade to the Jetson Nano board with 256 Cuda Cores, if these become available.

7.5. How the robot learns

The main aim of this miniTurtle project is to build a self-learning robot. In most cases learning is done in a simulation environment, using a large number of examples in a batch process. In this project we want the robot to explore its environment and learn interactively. That means a different approach, where real events drive the learning process in a continuous learning process. As soon as the robot is powered on, it is learning continuously.

Such an approach is known as Continual Learning and differs from the well-known Reinforcement Learning approach in several aspects:

1. Forgetting earlier events. As more and more new events occur, older events tend to be forgotten by the network. In a batch environment this is prevented by collecting all examples and training the network with these. We would have to store all earlier examples and regularly retrain the network.

This however is not the way that we want the network to grow. So we need to find a way to let the system remember earlier events.

2. Adding new classes. As new events occur, new classes or categories will be added to the network. This will change the network architecture and makes remembering earlier events more difficult. How can we grow the network and adapt it with new classes, while the learned information is retained.
3. Small number of samples. In existing networks, thousands of samples are required. In our case there will be samples, coming from the environment and they will grow with time. We could use augmentation to generate many variations of each event, so the network can be trained with more examples. This process is also known under the name of 'one-shot-learning'.
4. Slow learning rate. Examples are all learned with the same learning rate. However some events are more important than others. When an event has a large impact on performance, it should get a larger influence in the weights that are learned. So we need to find a way to include the importance of an event.

References

[1] *Continual Lifelong Learning in Neural Networks: a Review*.

[2] Jake Bruce, Niko Sünderhauf, QUT, Brisbane QUT, and Brisbane. One-Shot Reinforcement Learning for Robot Navigation with Interactive Replay. 1711.

[3] Yanwei Fu, Tao Xiang, Yu-Gang Jiang, Xiangyang Xue, Leonid Sigal, and Shaogang Gong. Recent advances and in zero-shot and recognition. 1710.

[4] Guang-Bin Huang. Real-Time Learning Capability of Neural Networks.

[5] Jaehong Joon, Eunho Yang, Jongtae Lee, and Sung Ju Hwang. lifelong learning with dynamically expandanle networks.

[6] Daniel J. Mankowitz, Augustin Žídek, André Barreto, and Dan Horgan. Unicorn: Continual learning with a universal, off-policy agent. 1802.

[7] Alexander Ororbia, Ankur Mali, C. Lee Giles, and Daniel Kifer. Continual Learning and of Recurrent and Neural Networks and by Locally and Aligning Distributed and Representations. IEEE, 1810.

[8] Anton Puzanov and Kobi Cphen. deep reinforcement and one-shot learning and for ai classification systems.

[9] Mark Woodward, Chelsea Finn, Independent Researcher, Berkeley AI, and Research (BAIR). Active one-shot learning. 2016.

8. Appendix

8.1. Previous work

In this section earlier work on learning behaviors with robots is discussed. It forms the basis for the previous work sections.

8.2. Tutorials and Overview Articles

8.2.1 Fu1710 - Recent Advances in Zero-shot Recognition (8) [3]

This overview article describes the most important concepts and work in LifeLong Learning and Zero-shot learning. It also addresses One-shot learning, Few-shot learning and similar concepts. This approach is also known as 'learning to learn'. The idea behind Zero-shot learning is to learn new concepts and add these to an existing network, without any new examples. This is achieved by generalizing recognition models and using existing properties to infer the existence of a new class. This is a form of transfer learning and the main problem with this is to learn new concepts, without forgetting existing concepts. The main idea is to associate learned features in an image with semantic attributes and to use a model of such attributes to recognize new classes. In our case, for instance, the features of a robot shape like color, shape of the number plate, body shape and other objects, could be used to separate the robot from one team from another one. There are several different kinds of attributes, that are use in the various approaches:

1. User-Defined attributes. Defined by experts. Easy to use, but hard to define and maintain.

2. Relative attributes. Define rankings. Singh et al [48] developed a DCNN to simultaneously localize and rank visual attributes. We could use that in recognizing robots, people, arbitrary balls and other categories.

3. Data Driven attributes. This can be achieved by creating a large body of relative attributes that help in discriminating attributes automatically.

4. Video attributes. Most attributes are taken from static images. In moving images more dynamic attributes can be found like actions and direction of movement.

5. Semantic embedding like a Concept Ontology or Semantic Word Vectors.

In most cases Zer-shot learning will be based on an embedding model. There are different type of models:

1. Bayesian models.

2. Direct Attribute Prediction Models (DAP) in shich the semantic relationship between concepts is encoded by human experts.

3. Indirect Attribute Prediction (IAP) which is built by combining the probabilities of all associated known classes. It is also called a direct similarity-based model.

4. Semantic Embeddings. Learn to associate the visual features with a semantic space.

5. Embeddings in common space. Here an existing embedding space is used.

6. Deep embeddings. Here a deep neural net learns class label embedding vectors.

The paper makes a distinction between Zero-shot recognition and One-shot learning. In One-shot learning one or a few examples are used to learn to recognize new classes. This can be done by direct supervised learning and by transfer learning. The approaches mainly differ in What knowledge is transferred and how the knowledge is transferred. There are a number of approaches: SJE, WSABIE, ALE, DeViSE and CCA.

The basic problem with all approaches is to learn to recognize new classes and integrate these with an existing network, without losing the concepts already learned. Technically the problem is to add new classes to an existing network, which changes its architecture and keep all the learned weights, while allowing new weights to be learned, without gradually eliminating the older weights. In a conventional system this is prevented by retraining the network with all known examples. In a Continual Learning environment, no examples are kept and new samples could easily erradicate older knowledge. This problem is also associated with defining the strength of an event, which should be taken into account, when remembering new or existing events.

The approach is to maintain a model to infer new classes, but no examples were given, where the actual network architecture was growing to accomodate new

classes. There must be more work where this is actually done.

8.2.2 Parisi1802 - Continual Lifelong Learning with Neural Networks: a Review (8) - [1]

In this overview article several approaches to Continual Learning are described. It explores the biological inspiratons for this work, especialliy the Hippocampus, which is associated with short-term memories and the Prefrontal Cortex (PFC) which is associated with long-term memory. Both systems need to be synchronized, which is believed to take place during REM sleep. In this process, generalization takes place as well as reinforcement of earlier known concepts, facilitating the prevention of forgetting.

Neural Networks are usually trained in batches of input data. When data is learned in a continuous stream, catastrp[jic forgetting takes place when new concepts come in and overwrite older concepts. This also seems to take place in humans, but only under restricted conditions.In the past 20 years many attempts have been made to prevent this forgetting. The simplest approach is to completely retrain the network when new examples come in, but this takes additional time and memory. The most general approach is to either store older examples, or recreate older examples from existing memory and use a replay buffer to rehearse the system at regular intervals. The three most common approaches are: 1) Retraining with regularization, 2) Training with network expansion and 3) Selective Network Retraining and expansion.

Most experiments are conducted with MNIST and CIPHAR-10 but more elaborate and realistic domains are being developed. With these simple test environments a number of approaches have been tested in order of performance:

1. GeppNet
2. GeppNet+STM
3. FEL
4. MLP
5. EWC

Themes that are important in these approaches are staged learning, generalization, transfer learning, where earlier concepts are re-used to learn newer concepts. Presenting tasks in a sequence here tasks are gradually becoming more complex and hierarchical learning are important. Other aspects that were explored are attention, intrinsic motivation, curiosity and multisensory learning. Especially the newer work of Hinton on Ensembles were mentioned as a way of separating the impact of an experience from the task policy in two different weights. The main conclusion is that much more research is required and most proposes approaches are still performing less than batch-like training, while the time and space requirements of the newer approaches are more demanding. Experience Replay, especially when based on self-generated examples seems the most promising direction.

8.2.3 Joon2018 - Lifelong Learning with Dynamically Expanding Networks (9) - [5]

Most existing networks train their data in batches. When a new class needs to be added, nem samples need to be added and the entire network must be trained again. For systems that receive their input sequentially in real time, like in a self-learning autonomous robot this is not a feasible approach. In this paper a new approach is proposed, Dynamically Expading Networks (DEN). In this work every new event is compared with existing knowlegde, not based on examples but on the weights in the activation matrix. If the new event is similar to the existing data, it is added to the training. If it is too different, the neurons in the various layers are split into two new neurons, one that keeps the existing connections and another one that connects to a new class. Then the network is retrained, using the existing weights as a basis in which Experience Replay is performed based on the learned weights of the various layers. So three different situations can occur:

1. Selective retraining. It selects the neurons that are relevant to the new task and selectively retrains these.
2. Dynamic network expansion. If the selective retraining does not improve the loss, the entire network is expanded top-down, eliminating unnecessary neurons.
3. Network split/duplication. When units drift away too much from their original values, neurons are duplicated and then retrained.

This approach eliminates catastrophic forgetting, but also creates a network that is more efficient than one with fixed sizes of layers. This not only reduces that size of the network, but also makes it more efficient. In tasks like MNIST and CIFAR-100 it proves that this approach results in better performance, while preventing drift and forgetting. The sequence in which concepts are presented are relevant in the speed of learning. Concepts that are more related help in improving the learning speed when presented gradually, while presenting them at random results in longer training time.

The method is compared to Elastic Weight Consolidation (EWS - Kirkpatrick2017) and a progressive network (Rusu2016).

8.2.4 Mankowitz1802 - Unicorn: Continual Learning with a Universal Off-Policy Agent (6) - [6]

This policy-gradient based DQN type network learns to generalize tasks using Universal Value Function Approximation (UVFA) to learn to find similar tasks in a network that uses an LSTM layer. The pool of tasks acts as a replay buffer that aims at learning a hierarchical task model. It is similar to [9] and [8]

8.2.5 Puzanov1808 - Deep Reinforcement One-Shot Learning for AI Classification Systems (5) - [8]

This article is similar to the previous one but introduces a pool of analysts that have to determine the actual class, in case the network is not certain enough. This includes an elaborate model of determining the time needed for human analists to classify and include this in the total cost of finding a solution.

8.2.6 Woodward2016 - Active One-shot Learning (6) - [9]

This paper describes a one-shot reinforcement learning approach with a deep recurrent model, Is is based on DQN but with the added condition that it is a one-shot approach. It is based on work of Santoro2016 and extends this work. It is very different from the Dynamic Extension Network [5], which deals with a CNN only. In this work, a DQN network is extended with an LSTM layer to learn to generalize its input. Because of this approach there is no dynamic network expansion and is solving another type of problem altogether. Need to further work out the differences between these two and concentrate on finding out how these two approaches may be combined.

8.2.7 Bruce1711 - One-Shot Reinforcement Learning for Robot Navigation with Interactive Replay (8) - [2]

This work describes a combination of real and simulated robot navigation learning using a DDQN with experience replay buffer, concentrating on one-shot learning. In a single pass the robot explores its environment and then a model is created in which augmented versions are used to create a replay buffer from which navigations to a given target are learned. The 360 degree image is split into four 90 degree images and using a resnet-50 transfer learning layer, finds features in the image to find significant features. It used offline interactive replay to learn the best navigation, which is then used in the real-time environment. It includes a good explanation of model-based and model-free Q-Learning and tests three different approaches, of which the bootstrapped DQN with an LSTM performs best. This work is described in Osband2016 Deep exploration via bootstrapped DQN.

8.2.8 Huang2006 - Real-Time Learning Capability of Neural Networks (7) - [4]

This curious, self-referencing article from 2006 is interesting, because it introduces a different approach to learning a sparse network that is not based on iterative back-propagation but on a single-pass analytical calculation of weights and biases in two layer neural networks. It claims to be a factor 10 faster and having a better perfomance than regular backprop networks. The approach is called Realtime Learning Algorithm (RLA) and it is interesting to investigate if this. or similar approaches have been investigated any further and found their way in deep-learning approaches.

8.2.9 Ororbia1810 - Continual Learning and of Recurrent and Neural Networks and by Locally and Aligning Distributed and Representations (8) [7]

This work Parallel Temporal Neural Coding Network (T-TNCN), based on Local Representation Alignment (LRA) as an alternative to back-propagation in recurrent networks (RNN) using LSTM. It is based on Bayesian

predictive coding. In related work, sparse attentive backtracking is mentioned and suggests that in the brain a continuous process of hypothesizing and prediction works as an intertwined top-down-bottom-up method.

The way it works is that it receives a continuous stream of input concepts, each trained individually, as a mini-batch of size 1. At each step a hypothesis is made, after which this is compared with the real class. Instead of using back-propagation, it now creates a separate instance for each class at all layers and calculates the error in parallel. This error is then used in a calculation that optimizes the prediction on the current layer as well as the forward prediction to the next layer. It uses a Hebbian decay term to prevent domination of the updated parameter. This calculates the total discrepancy as the level of mismatch within the various layers. It uses a Laplacian sparsity constraint (whatever that may be).

It mentions a number of important concepts that we need to check:

1. Bayesian learning
2. Hebbian learning
3. Bolzmann Machines
4. Hopfield Networks
5. Laplacian and Gaussian

This paper presents a seemingly important approach to one-shot continual learning of time sequences that is a good alternative to conventional LSTM based DQN learning, using back-propagation.

9. Additional topics:

1. State Detection
2. Merger between Gofai and NN
3. Learning by example
4. Subsumption Architecture and Neural Networks
5. Remote Real Time learning