Summary

This paper presents the research of using Multi-View Learning (Multi-View Learning), Multi-Task Learning (MTL), and dueling architecture based on deep double-Q network (D3QN) to design a network for obstacle detection and dirt prediction, which makes it possible for a robot to predict dirt without hitting obstacles. The training should make it superior to robots using random navigation, but using only a monocular camera it should make it cheaper than robots from existing companies.

The system, , consists of two individual parts, FCRN network with 2 outputs (FCRNO2) and D3QN network with 2 inputs (D3QNI2). FCRNO2 uses a modified version of the Fully Convolutional Residual Network (FCRN) presented in the paper "Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning" by L. Xie et al. It has been modified to use MTL to do both depth prediction and dirt detection. This is done using hard parameter sharing which uses a number of shared layers to represent common features that can be learned at the same time and then some task specific layers for each task. The images in the "Depth Image Stack" comes out in the same size as in the paper we base our solution on, but the images in the "Dirt Image Stack" have the top 40% cut off to focus on the ground and remove noise from the walls. D3QNI2 consists of two separate parts Convolutional Neural Network (CNN) and dueling architecture based on deep Q network (DDQN). CNN is the part of that uses Multi-View Learning to work on the layers using a one-view-one-network strategy until flattening them. The DDQN takes this as input and chooses the best action.

The reward function that is used consists of multiple parts. This is the final reward function, but others were tried with e.g. different amounts of punishment. The first comes from D3QN and is about driving, the second is about hitting objects, and the third comes from the added dirt detection.

$$r_{drive} = v \cdot \cos(\theta) \cdot 0.2 - 0.01 \tag{1}$$

It gives higher rewards if the robot drives fast and in a straight line, but therefore also will get lower for turning. The one for hitting objects sets the entire rewards to $r_{total} = -10$. The last is calculated based on the earlier found reward (Equation 1) and returns the total reward if the robot has not hit an obstacle. The reward is calculated based on 3 scenarios:

1) Getting closer to the closest dirt spot
2) Getting further away, after having been close enough
3) Getting further away from the closest dirt spot

In scenario 1 the robot is getting closer to the dirt tile that the system has found to be the closest, and the reward returned is doubled.

$$r_{total} = r_{drive} \cdot 2 \tag{2}$$

In scenario 2 the robot is getting further away from the dirt that the system has found to be the closest in the start, but is still getting closer to some dirt, and the reward is added 1.

$$r_{total} = r_{drive} + 1 \tag{3}$$

In the final scenario, the robot is getting further away from the dirt that the system has found to be the closest in the start, but is not getting any closer to any other dirt. Here we have tried to multiply it with e.g. -2 and 0.5 to see the effect but found -1 to be the optimal of the values compared.

$$r_{total} = r_{drive} \cdot k_a \tag{4}$$

All of the training and testing was done using the Gazebo Simulator and Robot Operating System (ROS). All the experiments have been trained and tested on an Intel i9-9900 CPU, 16GB RAM, and an NVIDIA GeForce RTX 2070 Super 8GB GPU. We have tested FCRNO2 with 1, 2, 3, and 10 task-specific blocks, where 2 was the best of the 4 tested, showing the least, most evenly distributed noise.

For the navigation part, the comparison that has been made is between a robot using random algorithm and a robot using . While testing we found that D3QN and FCRN trained with dirt pictures instead of depth pictures showed the best test results, with fewer hit obstacles on average compared to the random robot. As for D3QNI2, the test configuration where the reward is negated by -1 when driving away from the closest dirt, reached the highest total rewards, with fewer bumps than the random robot on average.

We also discuss multiple ways to improve FCRNO2, e.g. using different types and sizes of dirt where as in this paper it has been static; and different loss functions for dirt and depth prediction.

Another interesting improvement CleanNav would be to give the D3QNI2 network another input from a downward-facing camera, to tell whether or not it is driving over dirt or not. This could improve the training by making the training signal less noisy with regards to where the dirt is.

In this paper a multi-purpose network for dirt detection and depth prediction is proposed.

The FCRNO2 network seems to give some good results for dirt detection. It were seen that when driving only having trained FCRN using dirt images, the robot drove for the dirt, which means it is likely this part of CleanNav works. When being trained together (FCRNO2 and D3QNI2) the results do not show any positive results.

When looking at task-specific blocks we saw that L2 was better to the others in the case of noise. Single task did also have less noise than e.g. L1, but L2 was still better because of the noise distribution.

This suggests there might be some potential for the dirt detection and that the problems seems to originate in the way that the D3QNI2 network uses the dirt detection and depth prediction results. The depth prediction and dirt detection seems promising, however, it does not seem like D3QNI2 is the right choice in this setup.

Our final conclusion is therefore that FCRNO2 shows potential, however, CleanNav with D3QNI2 and the current setup does not seem to give a positive outcome.

CleanNav: Dirt Detection and Depth Prediction with Multi-Task Learning and Multi-View Learning

Emil Johan Taudal Andersen, Peter Fogh Bugtrup, and Jonas Rechnitzer Eriksen

***Abstract*—Autonomous cleaning robots are becoming more popular in the domestic sector and business sector. Many companies, schools, airports, and such are using them to keep large surfaces clean while freeing up employees to clean other places that the robot cannot. Dirt detection is one of the newer features in autonomous cleaning robots that enables smarter cleaning. Dirt detection has the potential to make cleaning robots more efficient both in time, but also in resources like water, detergents, and energy. We try to use Multi-Task Learning (MTL) for extending Fully Convolutional Residual Network (FCRN) to make a single network for both depth prediction and dirt detection. Furthermore, we use Multi-View Learning (Multi-View Learning) in a dueling architecture based on deep double-Q network (D3QN) network for making a robot able to learn to navigate based on depth prediction and dirt detection. We use the Gazebo Simulator combined with Robot Operating System (ROS) to test the proposed solution on a simulated robot. We test our solution against a random navigation algorithm as well as some variations of our own solution. We conclude that our solution does not perform better or worse than the random robot, and that the D3QN network with 2 inputs (D3QNI2) architecture might not be suitable for this kind of task.**

***Index Terms*—Deep Reinforcement Learning, Multi-Task Learning, Multi-View Learning, Depth Prediction, Dirt Detection, Gazebo, Simulation**

I. Introduction

Autonomous cleaning robots are getting more popular among companies. As a result the algorithms used in the robots are becoming more advanced. Two important tasks for autonomous cleaning robots to solve are obstacle avoidance and dirt detection. Obstacle avoidance is important for the robot to be able to navigate safely around in static and dynamic environments. Obstacle avoidance often relies on sensors like Light Detection and Ranging (LiDAR), depth camera, and more.[3] It can also be a combination of the aforementioned sensors, however, some of these solutions are expensive and consume a lot of power. A solution to this could be to use monocular RGB cameras as they are cheap, easy to use, and dont consume as much power. They also tend to give richer information than the other sensors, but they also have the disadvantage that they do not give any depth information, and that makes it difficult to navigate based on the data. We will explore the use of deep neural networks to predict the depth in a 2D image from a monocular RGB camera and the use of Deep Reinforcement Learning (DRL) for navigating without hitting obstacles. The other task, dirt detection, is about detecting different amounts and types of dirt. Dirt detection can be done in different ways with different sensors, however in this paper we are focusing on using a camera for dirt detection. We will also be focusing on locating the dirt, and we will not focus on detection of the amount of dirt. Dirt detection has multiple purposes. One purpose is cleaning on demand, which can be used to keep certain areas clean. This can make the cleaning more efficient because the robot does not necessarily have to clean everywhere. Another purpose is to optimize the usage of water, detergents, and power. By estimating the level of dirt on the surface it should be possible to estimate the level of resources needed to clean the area. Usually dirt is spread unevenly on a surface and so there is no reason to use equal amount of resources on every square meter. This should also make it cheaper for the customer to run the robot. Dirt detection could also be applied on non-autonomous cleaning machines, but will not be explored further in this paper as the focus is on autonomous robots.

The need for autonomous cleaning robots has already been demonstrated by several public research projects like FLOBOT [4] and BakeR [5]. Furthermore a couple of commercial companies like Tennant [6] and Nilfisk [7] are also selling autonomous cleaning robots for industrial use. Most of these robots have an array of sensors to be able to navigate safely around in dynamic or unknown environments. Some of the sensors are mandatory because of safety regulations and these regulations will vary depending on which situation the robot is deployed in.

In this paper we propose a solution where dirt detection and depth prediction is built into the same neural network. That means that the only sensors needed to operate is a standard monocular RGB camera. Our main contribution is a deep learning system that use dirt detection and depth prediction which can potentially be used to optimize the resource usage, such as water, energy, and detergents, of autonomous cleaning robots.

Our method is based on the method used in [1], where they use a neural network for predicting the depth in an image and a DRL network for navigating based on the predicted depth. We will expand the two networks to do Multi-Task Learning (MTL), such that the solution is capable of learning dirt and depth detection at the same time.

II. Related Work

This section is based on the related work section from our previous paper [2] with extended related work about MTL and dirt detection. In this section we will discuss work related to obstacle avoidance, MTL, and dirt detection.

a) Reinforcement Learning and Depth detection: Obstacle avoidance is a widely explored field with many different solutions to the problem such as using LiDAR, RGB-Depth (RGB-D) [8], stereo cameras [8], and monocular camera

[9], [1]. Before deep Q-network (DQN) [10] was proposed, reinforcement learning was used to solve the obstacle avoidance problem [11]. A solution using a monocular camera to solve the problem has also been presented [9]. A Visual SLAM approach with only one camera as the primary sensor has also been suggested [8]. When navigating using only a monocular camera, scale is difficult to get right, as there are no depth information [1], [8]. Various solutions to the problem has been suggested, e.g. [8] proposes to use stereo cameras or an RGB-D camera. A more advanced solution is depth prediction, where neural networks is used to predict the depth in a 2D image. Depth prediction only makes sense to use when only a monocular camera is available. In [1], an Fully Convolutional Residual Network (FCRN) network [12] is used for depth prediction in combination with a dueling architecture based on deep double-Q network (D3QN) network to teach a robot to navigate without hitting obstacles. When using a monocular camera, the method proposed in [13] seems to be the best solution for depth prediction according to [14]. As mentioned, there are many solutions to the obstacle avoidance problem, where some requires more expensive hardware than others. Monocular RGB cameras are generally cheap and easy to use, and they have seen more use for obstacle avoidance in recent times [1], [8], [14].

b) Multi-Task Learning:
MTL has been experimented with in many different settings and domains. An overview of MTL with deep neural networks can be found in [15] and [16]. In [17], [18], and [19] are examples of neural networks used for MTL. In [20], they use a network called MultiNet to do Multi-Modal MTL for autonomous driving. MultiNet is a technique for learning multiple different behavioral modes through a single deep neural network. It is tested by controlling a model in unstructured environments like unpaved roads. In [21], MTL is used for combining the learning of object detection and distance detection into one model. It is shown that the model actually performs better than a single-task model. Another example is [22], where a single-stream two-task network is used to do both semantic segmentation and depth prediction.

c) Dirt detection:
Dirt detection is a fairly new research area investigating how to detect dirt on surfaces. In [23] and [24] an overview of what advanced cleaning robots consists of can be found. It provides an overview of the concept and its components. Dirt detection based on vision approaches are presented in [25], [26], and [27]. In [27], a solution based on the YOLOv3 framework is used to create an algorithm for dirt and office item detection. The algorithm can be trained to 2 class task classification to distinguish between dirt and office items. The algorithm can also be trained to classify the office items into specific categories with classification of 10 classes. In [26], a solution that can detect dirt without any previous training is presented. The solution is based RGB-D cameras and achieves dirt recognition rates of 90% and with a false positive rate of 45%. In [25] they present a dirt detection solution based on Gaussian Mixture Models. They use unsupervised online learning and approach the problem as a single-class classification problem. The solution performs particularly well on complex floor textures.

Our solution will not be using a predefined framework for dirt detection, but will be using a modified version of the FCRN and D3QN from [1]. The solution will require pre-training of the FCRN before training the D3QN.

III. Method Overview

When referring to the entire system it will be called Clean-Nav, when referring to the FCRN it will be called FCRN network with 2 outputs (FCRNO2), and when referring to the D3QN part it will be called D3QN network with 2 inputs (D3QNI2).

A. Problem Definition

This section is based on the problem definition section from our previous paper [2]. We are using DRL for navigation, and therefore the problem can be formalized as a Markov Decision Process (MDP). We are using a D3QN network with 2 inputs to solve the problem. The robot can choose between 7 different actions, 2 to control linear velocity, and 5 to control angular velocity. The reward function is based on the dirt detection and depth prediction (see subsection III-G). A state s_t will be comprised of two images, the dirt image di_t and the depth image de_t, so $s_t = (di_t, de_t)$. So the next state will be: $s_{t+1} = (di_{t+1}, de_{t+1})$ For every state s_t at timestamp $t \in [0, T]$, the robot will choose an action a_t. Based on action a_t, the robot will receive a reward, and thereafter it will transition to the next state s_{t+1} based on the reward. The goal of DRL is to maximize the accumulated future reward, which is the maximum future reward that the algorithm can get from the current state s_t. The accumulated future reward can be formalized like: $R_t = \sum_{t=0}^{\infty} \gamma_t \cdot r_t$, in which $\gamma \in [0, 1]$ is the discount factor. If we then follow a policy starting from a state s and an action a, it can be defined as in Equation 1:

$$Q(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma_t \cdot r_t | s_0 = s, a_0 = a, \pi] \tag{1}$$

To always choose the optimal action we can use the Bellman equation which describes the optimal Q-value function. The Bellman equation can be seen in Equation 2. In Equation 2, s' and a' signifies the next state and action following state s and action a.

$$Q^*(s,a) = \mathbb{E}[r + \gamma \cdot \max_{a'} Q^*(s',a') | s,a] \tag{2}$$

The Bellman equation states that the optimal Q-value in a timestamp t is the immediate reward r added with discounted future reward at timestamp $t + 1$. This makes it possible to approximate the optimal Q-value function with neural networks, which makes it possible to use larger state spaces than is possible with traditional reinforcement learning. As a consequence DRL can be used to solve more complex problems than traditional reinforcement learning algorithms can.

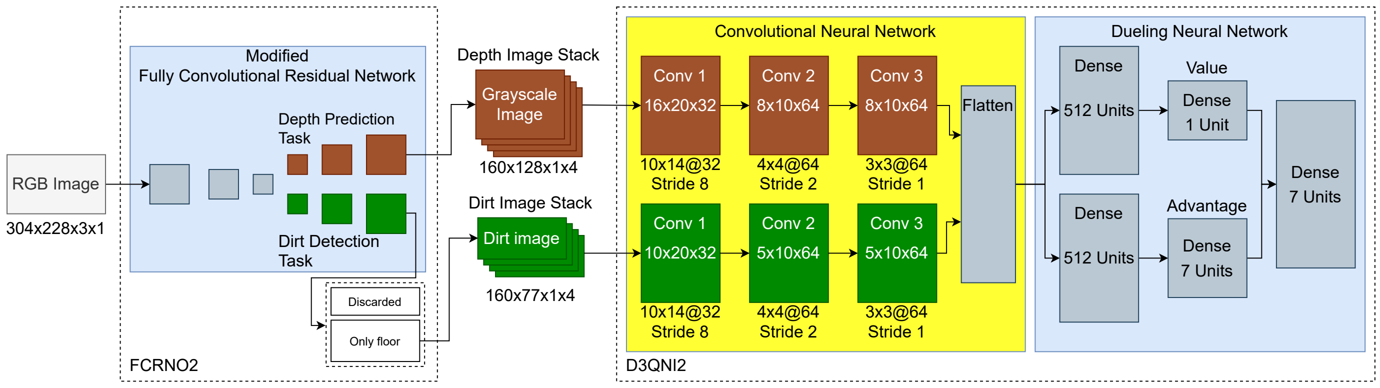


Figure 1: The network that the robot uses to decide on an action based on an image.[1] Notice the cropping process right below the FCRNO2 network.

B. D3QN

This section is based on the D3QN section from our previous paper [2]. We use a D3QN network to decide which action the robot should take. A D3QN network consists of both a dueling architecture based on deep Q network (DDQN) network and a Deep double-Q network (Double DQN) network, and is an improvement of the DQN network as well as the two aforementioned networks. The D3QN network we use is a slightly modified version that has two inputs and one output as is explained subsection III-C.

The DDQN architecture was proposed by Wang. et al [29]. The idea is to use two streams of fully connected layers to estimate the value of a state and the advantage of taking an action separately. The two values are lastly added to get a Q-value. The Q-function used in a DDQN can be seen in Equation 3, where $A(s,a)$ is the advantage function and $V(s)$ is the state value function.

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{||a||}\sum_{a'} A(s,a') \tag{3}$$

In many reinforcement learning problems it does not make sense to evaluate all actions in all states. In our case the robot could drive into an dead end where it can only drive backwards in order to continue. Since driving backwards is not one of the actions it can choose, every action will make the current episode terminate, thus the current state is overall bad and should be avoided. The DDQN get around this problem by evaluating the state value and advantage values separately, and consequently this can actually make the network learn faster in some situations.

Van Hasselt et. al proposed the Double DQN architecture, which is an improvement of the original DQN [28]. A DQN tends to overestimate the rewards because the network both has to select the best action in the current state as well as calculating the target Q-values of taking that action in the subsequent state. This problem can also cause the training to become unstable because the same network chooses and evaluates the actions. To solve this problem, two DQN networks can be used to decouple these two tasks. So one network will be used for selecting the best action to take in the next state, which will be the action with the highest Q-value. The other network is used for calculating the target Q-value of taking the aforementioned action in the subsequent state. The networks are called Online Network and Target Network respectively. The formula used can be seen in Equation 4, where Q' is the Target Network and Q is the Online Network.

$$Q(s,a) = r(s,a) + \gamma Q'(s', \text{argmax} Q(s',a')) \tag{4}$$

The Online Network chooses the best action to take in the next state s' using the argmax function as can be seen in Equation 4. This action is then used by the Target Network to calculate the Q-value of taking that action in state s'. With this method the network is less prone to overestimation and the learning should also be more stable. All of this should also contribute to faster learning.

C. Network architecture

In this paper we use a network similar to the one we used in our previous paper [2]. Due to the similarity in the overall structure of the network, the following section is largely based on the network architecture section from our previous paper [2] with new content added about MTL and Multi-View Learning. In Figure 1, the network architecture is outlined.

The network consist of two building blocks, FCRNO2 and D3QNI2.

The first building block, FCRNO2, consists of a modified FCRN which takes an RGB image as input and learns multiple tasks, which is depth prediction and dirt detection. The depth output of the FCRN is stacked as is, while the dirt output is cropped as explained in subsection III-F and then stacked. They are then used as inputs in the next block, with the 4 latest images in each stack at any time. At $t = 0$, all images in each stack are the same.

The second building block, D3QNI2, consists of two Convolutional Neural Network (CNN) and a DDQN. In the two CNN, the two image stacks are run through the convolutional layers and lastly combined with a flatten layer. The dimensions shown in Figure 1 in the middle of each layer, e.g. Conv 1, is the dimension of the image after it has been run through the given layer. For the depth image CNN, the image size after the first layer is 16x20x32. As for the dirt image, the CNN produces a smaller output, 10x20x32, due to the smaller input size. The numbers below each layer are the size and number of filters in the given layer, including the stride. So in the first convolutional layer in each CNN “10x14@32” means that this layer uses 32 filters of size 10×14 with a stride of 8.

The DDQN consists of 3 dense layers, which creates two streams. The numbers of units in each layer of the two streams can be seen in the layers, so the first layer of the two streams has 512 units. The output of the D3QNI2 network is a combination of linear and angular actions, whichever has the highest Q-value. There are 7 actions in total, 2 for linear velocity and 5 for angular velocity.

D. Multi-Task Learning

In this paper we will be using MTL to take a single input from a monocular RGB camera and provide two outputs from one model: an output for dirt detection and another for depth prediction. There are two types of MTL, which are called hard parameter sharing and soft parameter sharing. The principle of hard parameter sharing can be seen in Figure 2. In hard parameter sharing a network contains shared layers and task-specific layers. It is called hard parameter sharing because the weights of the shared layers should generalize to work for all tasks. The network also contain task-specific layers for each task, and here the network will learn specific features for each task. An advantage of this method is that it reduces the risk of overfitting, because the network will have to learn general features between the different tasks. So the risk of overfitting will be reduced in relation to the number of specific tasks [16]. The other type of MTL is soft parameter sharing, and the principle of this can be seen in Figure 3. This works a bit differently from hard parameter sharing as we use a specific model for each task, instead one model for all tasks, and instead of shared layers it uses constrained layers. The constrained layers all have their own weights, but to enable MTL the weights are constrained usually by minimizing the difference between the weights. This way the models can learn common attributes in the data. As with hard parameter sharing the models also contain task-specific layers [16].

We have chosen to use hard parameter sharing in FCRNO2 because depth prediction and dirt detection has some common features that can be learned by the shared layers. In both tasks the network should learn about the brightness and contrast in the image. So by that we mean, that the network should learn to create a depth image from an RGB image, which can be used later to learn about distance in the D3QNI2 network. The network should also learn about the placement of dirt, which can also be used by the D3QNI2 network to about the distance to the dirt. Thus there is a good chance the two tasks has some common features. Furthermore, by using hard parameter sharing we can reuse most the previous network for both tasks.

For depth prediction and dirt detection, we make use of an FCRNO2 network. We observe that FCRN, without modification, is able to be trained to detect dirt, exactly in the same way it is trained to predict depth. The input RGB images used for training FCRN for dirt and training FCRNO2 for dirt detection and depth prediction are the same. However the target images

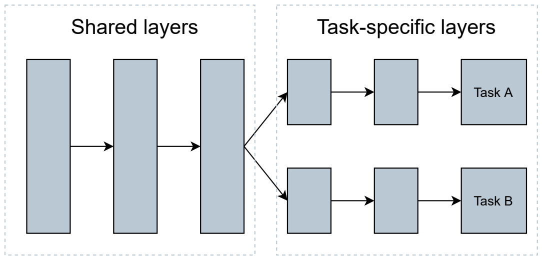


Figure 2: Multi-Task Learning hard parameter sharing [30].

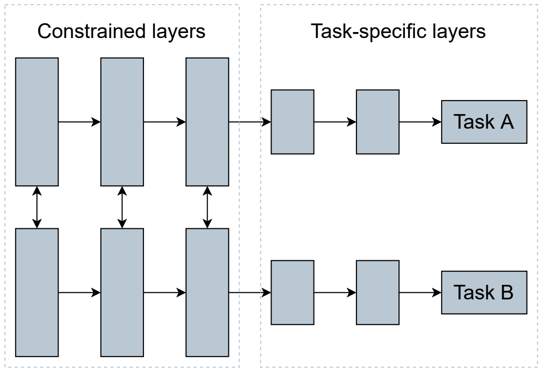


Figure 3: Multi-Task Learning soft parameter sharing [30].

are different, where FCRN only use dirt images as target and FCRNO2 use both dirt and depth images as target. Thus, to implement MTL, the last few blocks are duplicated such that there are specific blocks for both tasks. With these duplicated blocks, the network should be able to learn more task-specific features to predict and detect both depth and dirt, with the first part of the network being the shared layers. With duplicated blocks it is meant that it is the same layers that already existed in the network, but using data specific for each task.

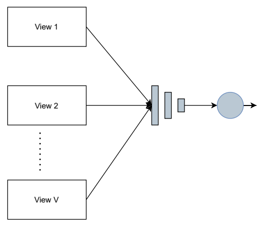
The output from training this network gives both a distinguishable depth prediction and dirt detection, with the exception of the top part of the dirt detection output since the top 40% of these are cut of as explained in subsection III-F. The process for overcoming this is described in subsection III-F.

E. Multi-View Learning

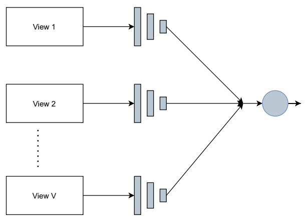
The idea behind Multi-View Learning is to learn one function behind each of multiple different views which can describe the same problem [31], [32]. In our case as it can be seen on Figure 1 our first view is the “Depth Image Stack” and our second is “Dirt Image Stack” which will then run through the same function: CNN.

As described in [33] the CNN multi-view architecture uses either one-view-one-net mechanism or multi-view-one-net mechanism as shown in Figure 4. The multi-view-one-net mechanism takes multiple views and feeds the input to same network which then gives the final representation, where as the one-view-one-net mechanism has one network for each view and then extracts each feature separately and at last all the representations are fused.

Our system will be using a “One-view-one-network strategy” since we have two CNNs that are later fused, one for each view.



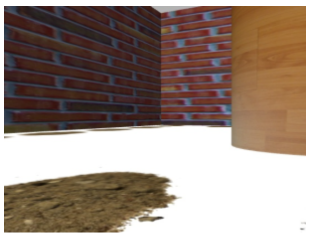
(a) Multi-view-one-network strategy.



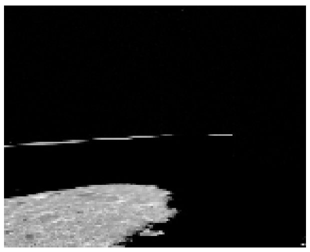
(b) One-view-one-network strategy.

Figure 4: Multi-view CNN [33].

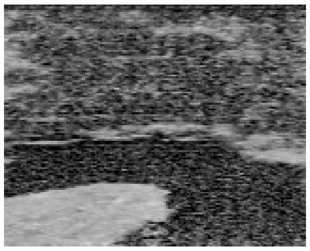
F. Preparing dirt prediction images



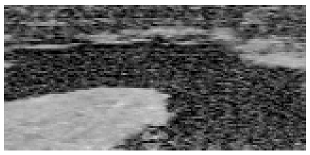
(a) The input RGB dirt image.



(b) Dirt truth. Note the empty top part.



(c) Dirt prediction output of modified FCRN.



(d) Cropping the image to bottom 60% of the dirt prediction output.

Figure 5: Dirt predicted compared to the dirt truth and cropped image.

The dirt prediction output of FCRNO2 has a few issues. The top part of the output, around 50%, is mostly noise, as can be seen in Figure 5c. The content in this part of the output falls into two categories:

1) Walls of the simulation environment
2) Floor of the simulation with dirt spots 1 to 2 pixels high

As we are aiming to make a robot to clean floors, and not walls, we can ignore the top part all together, as we do not place dirt on the walls during training. Thus, even if it were to correctly predict dirt on a wall, it would not matter as the robot only drives on the floor.

As for the second category of content in the top part, D3QNI2 might be able to learn something from these 1-2 pixel thick dirt spots, though it seems questionable at best.

Based on the above, all dirt prediction output images are cropped to bottom 60% of the image, preserving the floor part and allowing D3QNI2 to focus on finding features in this part alone.

Removing the top does not limit the future improvement of FCRNO2 either in training or network structure.

The end result of this cropping is shown in Figure 5d. The raw RGB image can be seen in Figure 5a, and it is clear that the top 40% of the image is indeed walls. The cropping process is also included in the figure showing the network (see Figure 1).

G. D3QNI2 training

To train D3QNI2, we first train the FCRNO2 and lock the weights.

a) Reward:
The reward consists of multiple parts. The first comes from D3QN and is about driving, the second is about hitting objects, and the third comes from the added dirt detection. The first part lays the ground for the total reward and is the same reward function used in [1]. It can be seen in Equation 5, where v is the linear velocity and θ is the angular velocity.

$$r_{drive} = v \cdot \cos(\theta) \cdot 0.2 - 0.01 \tag{5}$$

This means that the robot gets higher reward for driving fast and in a straight line. It also means that the robot gets a lower reward for turning. The maximum value attainable in a single step is thus $0.4 \cdot 1 \cdot 0.2 - 0.01 = 0.07$.

The second part is also from the paper [1] where if the robot hits an obstacle, the current episode is terminated, and given a reward of -10.

$$r_{total} = -10 \tag{6}$$

The last part uses the calculated reward from Equation 5 to return the total reward. This part relies on a constant, k_f, and two factors, k_c and k_a for which different values will be tested and compared to find the best combination. The distance to the closest dirt spot at step t noted as d_t, is used to calculate the the difference, m, between d_t and d_{t-1}.

$$m = d_{t-1} - d_t \tag{7}$$

It is defined such that the robot has visited a dirt spot when the robot is within $0.5\,\mathrm{m}$ of its center, $d_t < 0.5$. The total reward r_{total} is calculated based on three different scenarios that depend on the values of m and d_t.

1) $m > 0$: Getting closer to the closest dirt spot
2) $m < 0$ and $d_t < 0.5$: Getting further away, after having been close enough
3) $m < 0$ and $d_t > 0.5$: Getting further away from the closest dirt spot

In scenario 1 the robot is getting closer to the dirt tile that the system has picked as the closest and the reward returned is multiplied by k_c.

$$r_{total} = r_{drive} \cdot k_c \tag{8}$$

In scenario 2 the robot was close enough to the closest dirt tile, and the return reward is added k_f.

$$r_{total} = r_{drive} + k_f \tag{9}$$

In scenario 3 the robot is getting further away from the first picked tile, but is not getting closer to another one the dirt tiles, the reward from Equation 5 is multiplied by k_a.

$$r_{total} = r_{drive} \cdot k_a \tag{10}$$

IV. Experiments and Results

In this section we will present the experiment setup and results. To test our algorithm we use the Gazebo simulator and Robot Operating System (ROS). We will also report on the results of the FCRNO2.

A. Data collection

To train the FCRNO2 network we use a dataset consisting of 3 three types of images. The three types of images will be an RGB image, a dirt image, and a depth image which can be seen in the following figures respectively: Figure 6a, Figure 6b, and Figure 6c. Our datasets will be collected in the Gazebo Simulator. We collect the data by letting the simulated robot drive 2 identical runs, where RGB images and depth images are captured. The first run will be in a world with dirt spots on the surface, and the second run will be in the same world without dirt spots on the surface. The dirt image is then retrieved by subtracting the RGB images from the first and second run. To train the network, the RGB image will be given as input and the two outputs will be the depth image and the dirt image. This way we should be able to get a network that can predict depth in an image as well as detecting dirt. So the two outputs of FCRNO2 will be images similar to Figure 6c and Figure 6b.

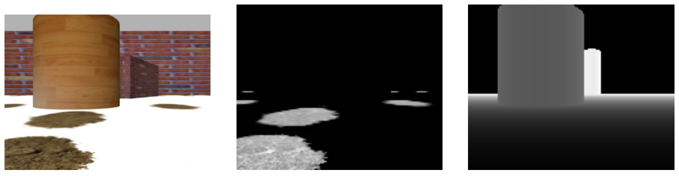

(a) RGB image. **(b)** Dirt image. **(c)** Depth image.

Figure 6: Example images from a dataset.

B. Simulation setup

In the simulation setup we use broadly the same Gazebo worlds as in our last paper [2]. We have made some modifications to the Gazebo worlds to be able to train our new algorithm. The Gazebo worlds are squares of 10×10 meters and contain several simple shapes like cylinders, boxes, and triangles. In this project the ground surface of the worlds are composed of tiles that can have custom textures. We use the textures to simulate dirt as it is a simple way to represent something similar to dirt. We only use one type of dirt and thereby only one type of texture to represent the dirt. The size of the tiles can be varied in order to simulate different amount of dirt on the ground surface. When the robot drives over a tile with dirt on it, the dirt is hidden until the world is reset (e.g. between episodes). All of this is the same for testing.

The training and test worlds can be seen in Figure 7. In the figures the dirt spots on the ground surface can be seen.

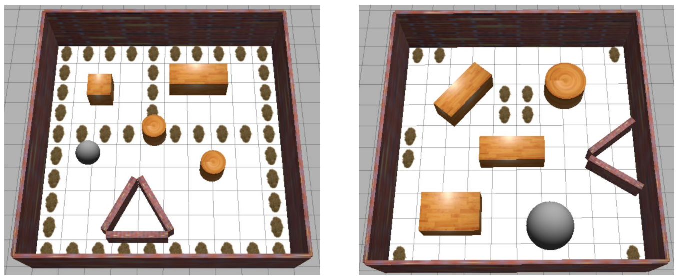

(a) Training world. **(b)** Testing world.

Figure 7: The environments used for training and testing the robot.

For the experiments we use a computer with an Intel i9-9900 CPU, 16GB RAM, and an NVIDIA GeForce RTX 2070 Super 8GB GPU for training and testing.

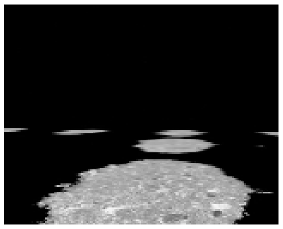
C. Dirt and depth prediction results

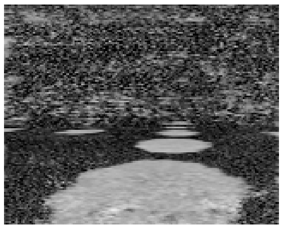
To test the precision of the dirt detection and depth prediction network, we have conducted some experiments where we vary the number of task-specific blocks in order to observe what difference it makes. We have conducted experiments with 1 layer and 2 task-specific blocks. Furthermore we have trained the network both to do only dirt and depth prediction to test if both tasks are actually possible.

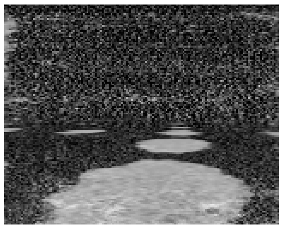
In the network, an AdamOptimizer is used with a learning rate of 0.00005. During the training the batch size is 16 and it is run for 150 epochs.

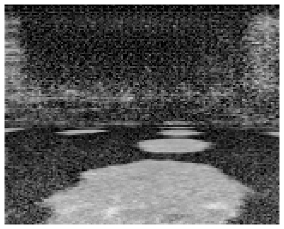
As previously mentioned the network for dirt detection and depth prediction consists of 21 blocks that contain some convolutional layers. We have tested the network in three different configurations for both dirt detection and depth prediction. These three configurations are as multi-task network with both 1 task-specific block and also with 2 task-specific blocks, and lastly training the network as single-task. We trained the network with both 1 task-specific block, and 2 task-specific blocks to see if we could improve the prediction/detection by allowing more precise weights through less shared layers. The network with 1 task-specific block will be referred to as L1, and the network with 2 task-specific blocks will be referred to as L2. The results can be seen in Figure 8. The ground truth for dirt and depth can be seen in Figure 8a and Figure 8e respectively.

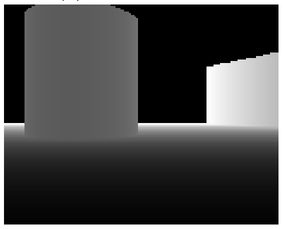
The results for the L1 network can be seen in Figure 8b and Figure 8f. In Figure 8c and Figure 8g the results for the L2 network is shown. Lastly the results of the networks trained for only dirt detection and depth prediction is shown in Figure 8d and Figure 8h. The loss graphs from the training can be seen

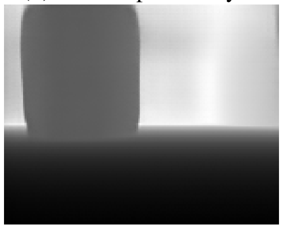

(a) Ground truth.

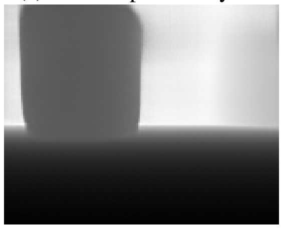

(b) 1 task-specific layer.


(c) 2 task-specific layers.


(d) Single task.


(e) Ground truth.


(f) 1 task-specific layer.


(g) 2 task-specific layers.

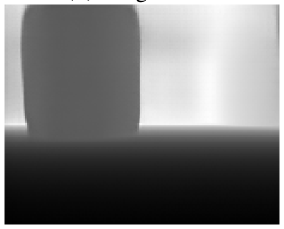

(h) Single task.

Figure 8: Dirt images above and depth images below.

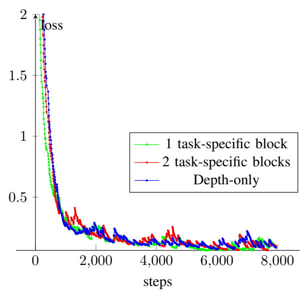
in Figure 9, with depth test shown in Figure 9a and dirt test in Figure 9b.

The dirt detection images are a lot more different when comparing between the configurations unlike the depth images. If one compares Figure 8a to both Figure 8b and Figure 8c, it can be seen that the network actually detects most of the dirt spots correctly. There are some problems with the small dirt spots in the background where there is a lot of noise in the predicted images from the L1 network. In the image from the L2 network, it is clear that there is considerably less noise in the image.

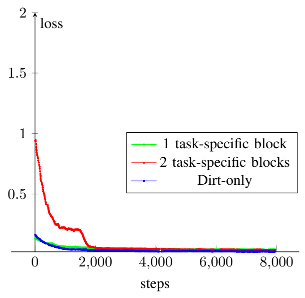
The depth prediction images in Figure 8f, Figure 8g, and Figure 8h are all pretty similar. There seems to be a tiny difference between Figure 8f and Figure 8g, where the latter image looks a tiny bit more clear. Other than that, using more task-specific blocks does not seem to make much difference on the depth images. However it does actually make a noticeable difference on the dirt images.

When looking at the test loss graphs for depth, it can be seen that the loss graphs for the L1 network, L2 network, and depth-only network are very similar. They converge at pretty much the same rate, where as the L2 network converges a little slower. However all three network ends up with the same results, which is also clear from the predicted depth images. So it seems like the network is capable of accurately predicting the depth both with 1 and 2 task-specific blocks, when trained with 10,000 images. When looking at the test loss graphs for dirt, it is a little bit different. The L1 network converges faster at the early stages than the L2 network, however after 2,000 images the graphs are very similar. The L2 network ends up with a slightly smaller loss than the L1 network, and therefore the images predicted with the L2 network contains less noise which can be seen in the predicted images. It can also be seen that the dirt-only network has the lowest as expected, however the L1 and L2 network are very close to having the same loss, so the practical difference is likely to be small. We also experimented with L3, and L10. The L3 network has same precision on depth prediction as L2, however on dirt prediction it was slightly better than the L2 network. The L10 network also has same precision on depth prediction as L2 and L3, but on dirt prediction it performed slightly worse than the L3 network. Because of time and resource constraints the experiments with the D3QNI2 network was run with the L2 network for dirt and depth information.

One thing to notice about the depth images in Figure 8 is that the background on the ground truth image in Figure 8e is black, where as on the predicted images the background is white. The black background in Figure 8e seems to be caused by how it is represented or some wrongly configured camera setting in the Gazebo simulator where the image is captured. The background should be white as in the predicted images, since the white color indicate a farther distance and black indicates closer distances. So the whiter an object is the further away it is, and the darker an object is the closer it is. Theoretically it should also not matter whether the background is white and the objects black or the other way around, as long as there is a clear difference between the background and objects.



(a) Depth test.



(b) Dirt test.

Figure 9: Test loss graphs.

D. Navigation experiments

We will use one experiment to test two different navigation methods. The experiment will be testing how effectively a robot can find and visit all dirt spots in the environment. For this purpose we will use the worlds presented in Figure 7. We will use the following navigation methods:

- Random navigation
- CleanNav
- CleanNav with dirt detection and without depth prediction

Since the papers we would be able to compare CleanNav to is from companies and do not contain any information about how well the robot performed, we will create our own solution to compare against. Furthermore, there does not seem to be much information on how more advanced robots navigate, so it is not possible to compare to those solutions. There is also time constraints that does not allow implementing more advanced methods to compare against.

We train the robot in two different types of dirt layout. One is a static world where the dirt layout is the same for every episode (for layout see Figure 7a). For the other the layout, random, it begins changing after the first 100 episodes and a new random layout will be used every 5 episodes to try to avoid overfitting. The layout will contain between 15 and 25 dirt spots.

We are hindered by hardware constraints therefore all training runs for D3QNI2 will be using a replay memory of 25,000 and will run for 800 episodes. During the first 10 episodes, the robot will only make random actions and after that the robot will make actions based on what it sees. Each of these episodes will run until it hits an obstacle, drives for a maximum of 500 steps, or all dirt spots have been visited. The starting point will be the same for each run, but the orientation will be different. The robot that is used in Gazebo Simulator is a modified version of the robot presented in [34].

E. Metrics

We will be using four metrics to evaluate our solution. The first one is how well dirt is identified, and the metric for this will be amount of steps used to visit all dirt spots. A step corresponds to an action made by the robot, which could be to turn left or drive straight. The second metrics is the efficiency of the training, which will be the time it takes to train in relation to the reward results. The third is obstacle avoidance, which will be measured on the amount of obstacles hit while testing. The fourth and last metric is how long it takes to reach all dirt spots and finish the test. Obstacle hits will also be called to bump into something or bumps.

F. Navigation results

We first train D3QNI2 with different configurations for the reward function. Afterwards, the algorithms is tested in a world with 11 dirt spots distributed across the whole simulation world, where the grid used in the test is 10×10, same size as when trained.

a) Training configurations:
To find the values for the constant and factors described in subsection III-G, we set up multiple different training configurations to cover a variety of different settings. The different configurations can be seen in Table I, and are split into three parts.

#	Solution	Layout	k_c	k_f	k_a	Training time
1	Random	-	-	-	-	-
2	CleanNav	Static	2	$+1$	-1	5h 04m
3	CleanNav	Static	2	$+1$	-2	3h 25m
4	CleanNav	Static	2	$+1$	0.5	3h 33m
5	CleanNav	Random	2	$+1$	-1	3h 04m
6	CleanNav	Random	2	$+1$	-2	2h 52m
7	CleanNav	Random	2	$+1$	0.5	3h 42m
8	Dirt only	Static	2	$+1$	-1	4h 04m

Table I: Training configurations including how long it took to train each configuration. Training time has been normalized to a realtime factor of 2 for the Gazebo simulator, i.e. for every 1 'real' second, 2 seconds pass in simulation time.

The first is a robot built using a random algorithm (random robot), that we will test and use as a base case. It is not be trained, as it drives randomly, but is included as test configuration number 1.

Test configurations 2-7 cover the second part of the configurations. They consist of training D3QNI2 in 2 types of layout. The random layout starts switching dirt layout every 5 episodes after the first 100 episodes, while the static layout uses the same throughout all episodes. For all test, k_c is set to meaning, the reward is doubled for driving closer to the closest dirt spot. We vary k_a, to find the best value for punishing the robot for driving away from the closest dirt spot. We try halving the reward and negating it by a factor of -1 and -2. From testing, we found that for k_f, a value of $+1$ seemed good based on the idea that it is good to drive over dirt (giving $+1$), but not so good as it is bad to crash into an obstacle (giving -10).

Test configuration #8 is to test how good the robot trained only with dirt prediction (e.g. no depth information) is compared to the rest. This setup is based on test configuration #2, as the training graph for that config shows the highest rewards among the other configurations.

The total reward over steps collected by the robot during training is show in Figure 10. We see that #8 reaches the highest rewards, while #2 drives the most steps during the 800 episodes they each were trained. It can also be observed that despite #4 halving the reward for driving away from the closest dirt, #2 and #8 still reach the higher total rewards, suggesting the punishment works as intended.

b) D3QNI2 testing:
In Figure 11 the route for the random navigation test (#1) can be seen. The red lines are the route that the robot has driven, the blue spots are the bumps, the orange spots is the starting point of the robot, and the brown spots are the dirt spots.

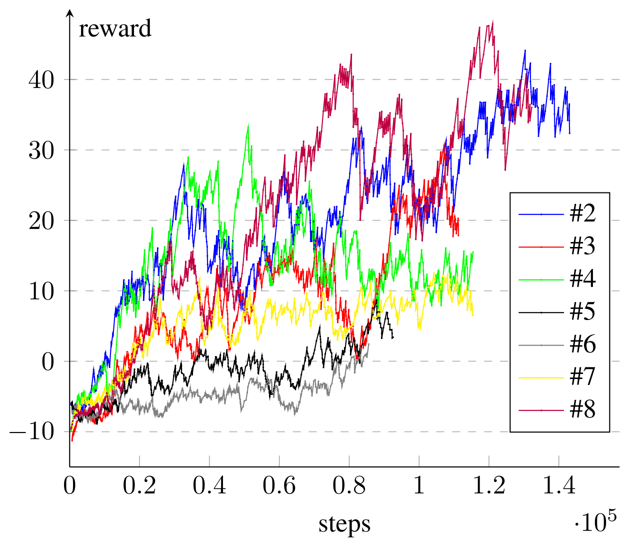


Figure 10: Total reward for the different reward function configurations. Smoothed with a factor of 0.95.

The results of the test in relation to the metrics described in subsection IV-E can be seen in Table II. We ran each test 3 times and Table II shows the average steps, bumps, and testing time for each configuration. The random robot (#1) used on average 8,410 steps to visit all dirt spots, and it on average made 211 bumps. As an example, one test run from the random robot is shown in Figure 11. The random robot is not very effective as it has visited most dirt spots multiple times while doing an average of about 32 steps between bumps. This was also expected, as the robot is using the simplest form of navigation.

In comparison, we see the best configuration (#8) in Figure 12, where the driving is more structured, with far fewer bumps. It uses on average 12,085 steps and only bumps on average 160 times during each test, meaning it drives about 75 steps between each bump on average. Still, it has the same problem as the random navigation, where it "visits" the same dirt spots multiple times.

G. Never-ending tests

In Table II, configuration #6 has missing data due to never-ending test runs. We noticed during testing that the robot would drive around just like the other configurations for a few minutes, after which it started driving in circles. The same behaviour has been observed three times, two of which it drove in circles for 2 hours each. As a result, we assume that it would continue that way forever. Combined with the total reward graph for #6 presented in Figure 10 that did not even cross 0 in total reward, we have decided to omit the results because of a lack thereof.

H. D3QNI2 results compared to random navigation

Based on the results in Table II, we can see that the best solution is #8, based the fact that it has the fewest bumps in relation to the testing time while being a smaller network all-together. Though this is not the result we hoped for, it shows it can follow the dirt and find all spots with fewer bumps than the other configurations. Specifically, it is also better than #1, the random navigation, tough not faster on average. We also see that configurations trained with random layouts, on average, use more steps, bumps more, and takes longer, than the rest of the configurations. This might be because the total

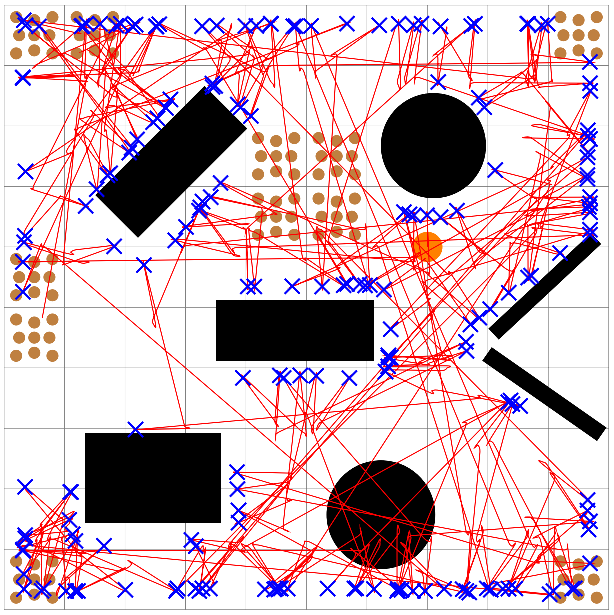


Figure 11: Showing test #1. The orange circle is the robot's starting position, the blue shows bumps, the red shows the route, and the dirt is shown by the brown dots. The dirt layout is the same as shown in Figure 7b.

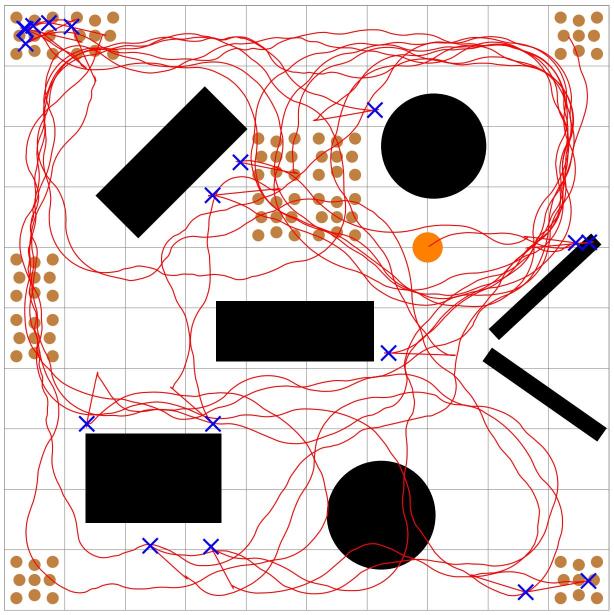


Figure 12: Showing test #8. The orange circle is the robot's starting position, the blue shows bumps, the red shows the route, and the dirt is shown by the brown dots. The dirt layout is the same as shown in Figure 7b.

#		Layout	k_a	Avg. steps	Avg. Bumps	Avg. Testing time
1	Random nav	-	-	8,410	211	28m 35s
2	CleanNav	Static	-1	9,891	190	29m 24s
3	CleanNav	Static	-2	15,245	293	49m 26s
4	CleanNav	Static	0.5	32,533	615	1h 42m 52s
5	CleanNav	Random	-1	14,685	461	44m 37s
6	CleanNav	Random	-2	-	-	-
7	CleanNav	Random	0.5	66,768	455	3h 01m 15s
8	Dirt only	Static	-1	12,085	160	29m 31s

Table II: Results of steps, bumps, and training time for Random navigation, CleanNav, and Dirt only. Missing results for #6 are explained in subsection IV-G.

reward reached for the configurations in Figure 10 were not very good compared to the static configurations and the dirt only configuration.

V. Discussion

In this section we will discuss the results of the depth prediction and dirt detection. We will also discuss future possible research directions that can improve the CleanNav solution. The FCRNO2 network seems to have some potential for dirt detection and depth prediction, so therefore we will discuss possible improvements. While we do not think that the D3QNI2 network has potential to work well in this problem, we will discuss some possible future improvements.

a) Fully Convolutional Residual Network:
In this section we will discuss the results of FCRNO2 as well as future research directions. The FCRNO2 network seems to still predict depth images as accurately as FCRN and also detect dirt accurately. As can be seen from the results section, the L10 network has the lowest loss of the network configurations that we tried. However the practical value of the dirt images could be limited because of the distribution of the noise. So the practical limit for the FCRNO2 could seem to be somewhere between 3 and 10 task-specific blocks. Since we are using MTL there would also be a practical limit in terms how of many task-specific blocks that makes sense to use. As it would not make sense to have 20 out of 21 blocks being task-specific, because then we might as well just use two separate networks.

One future direction to investigate would be to generalize the dirt detection to work on variable sized and types of dirt. In this paper we have chosen a base case with a static test world as well one type of dirt with one size in order to proof the principle. To generalize the solution, there is probably a need for images from additional environments, so that it can work in different settings. With that it should be possible to make the FCRNO2 network, work in different settings such as different floor textures, dirt sizes, and dirt types.

Another direction to investigate could be to use different loss functions for dirt detection and depth prediction. Right now we use the same loss function for both branches of the network. But it is possible that there is a loss function that works better for dirt detection than for depth prediction.

Implementing an extra output branch in the FCRNO2 network to indicate if the robot is directly on top of a dirt spot is also a possible future improvement. Right now the training of the D3QNI2 network could have a problem with delayed rewards. We have observed that the robot might get a reward for visiting a dirt spot after it has visited the dirt spot. The problem with this, is that the robot might then associate a wall or something else with the reward instead of the dirt spot. And because this maybe is not a problem every time, the training signal might contain some noise. A way to fix this could be to give the robot a signal to indicate that the robot is now directly on top of a dirt spot. This way we can make the robot clearly associate the reward with the dirt spot. A possible way to do this could be to install a second monocular RGB camera facing directly down, such that it gives images of the surface in a perpendicular angle to the robot. And then we could add one more input and output to the FCRNO2 network, which could then give a true or false to indicate if there is a dirt spot or not. This could maybe improve the training by making the training signal less noisy.

VI. Conclusion

In this paper a multi-purpose network for dirt detection and depth prediction is proposed.

The FCRNO2 network seems to give some good results for dirt detection. It were seen that when driving only having trained FCRN using dirt images, the robot drove for the dirt, which means it is likely this part of CleanNav works. When being trained together (FCRNO2 and D3QNI2) the results do not show any positive results.

When looking at task-specific blocks we saw that L2 was better to the others in the case of noise. Single task did also have less noise than e.g. L1, but L2 was still better because of the noise distribution.

This suggests there might be some potential for the dirt detection and that the problems seems to originate in the way that the D3QNI2 network uses the dirt detection and depth prediction results. The depth prediction and dirt detection seems promising, however, it does not seem like D3QNI2 is the right choice in this setup.

Our final conclusion is therefore that FCRNO2 shows potential, however, CleanNav with D3QNI2 and the current setup does not seem to give a positive outcome.

Acknowledgment

The authors would like to thank Professor Bin Yang and Assistant Professor Dalin Zhang for supervision during the creation of the paper and for topical and intellectual discussions. The authors would also like to thank IT Services at AAU and Aalborg University Library for access to CLAAUDIA, and IT Services at AAU for access to physical hardware.

References

[1] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning" in CoRR, vol. abs/1706.09829, Jun. 2017

[2] E. Andersen, J. Eriksen, and P. Bugtrup. "Combining D3QN and Reward Backfill for obstacle avoidance"

[3] https://www.braincorp.com/technology/autonomy/

[4] https://www.baker-projekt.de/

[5] https://www.flobot.eu

[6] https://www.tennantco.com/en_us/solutions/robotic-cleaning-machines.html

[7] https://www.nilfisk.com/global/products/floor-cleaning/autonomous-floor-cleaning/liberty-sc50-x51d-obc-ph/p_56104508/

[8] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.

[9] Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. "High speed obstacle avoidance using monocular vision and reinforcement learning" In Proceedings of the 22nd international conference on Machine learning (ICML '05). Association for Computing Machinery, New York, NY, USA, pp. 593-600, Aug. 2005 , DOI: https://doi.org/10.1145/1102351.1102426

[10] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Human-level control through deep reinforcement learning," Nature 518, pp. 529-533, 2015, doi: 10.1038/nature14236

[11] Tony Prescott and John Mayhew "Obstacle Avoidance through Reinforcement Learning" In Advances in Neural Information Processing Systems, vol. 4, 1992, pp. 523-530.

[12] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari and N. Navab, "Deeper Depth Prediction with Fully Convolutional Residual Networks," 2016 Fourth International Conference on 3D Vision (3DV), Stanford, CA, 2016, pp. 239-248, doi: 10.1109/3DV.2016.32.

[13] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow, "Digging into Self-Supervised Monocular Depth Prediction," The International Conference on Computer Vision (ICCV), Oct. 2019.

[14] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," Science China Technological Sciences, vol. 63, no. 9, pp. 1612-1627, Jun. 2020, issn: 1869-1900, doi: 10.1007/s11431-020-1582-8.

[15] M. Crawshaw, "Multi-Task Learning with Deep Neural Networks: A Survey" in arXiv 2009.09796, Sep. 2020

[16] S. Ruder, "An Overview of Multi-Task Learning in Deep Neural Networks" in arXiv 1706.05098, Jun. 2017

[17] Y. Fang, Z. Ma, Z. Zhang, X. Zhang, X. Bai "Dynamic Multi-Task Learning with Convolutional Neural Network" Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17) pp. 1668-1674 2017 doi: 10.24963/ijcai.2017/231

[18] S. Li, Z. Liu, A. B. Chan "Heterogeneous Multi-task Learning for Human Pose Estimation with Deep Convolutional Neural Network" in arXiv 1406.3474 Jun. 2014

[19] X. Liu, P. He, W. Chen, J. Gao "Multi-Task Deep Neural Networks for Natural Language Understanding" ACL 2019

[20] S. Chowdhuri, T. Pankaj, K. Zipser "MultiNet: Multi-Modal Multi-Task Learning for Autonomous Driving" 2019 IEEE Winter Conference on Applications of Computer Vision (WACV)

[21] Y. Chen, D. Zhao, L. Lv, Q. Zhangab "Multi-task learning for dangerous object detection in autonomous driving" Information Sciences Vol. 432 Mar. 2018 P. 559-571 doi: 10.1016/j.ins.2017.08.035

[22] M. Aladem, S. A. Rawashdeh "A Single-Stream Segmentation and Depth Prediction CNN for Autonomous Driving" May 2020 doi: 10.1109/MIS.2020.2993266

[23] R. Bormann, J. Hampp, M. Hägele "New brooms sweep clean - an autonomous robotic cleaning assistant for professional office cleaning" 2015 IEEE International Conference on Robotics and Automation (ICRA) May 2015 doi: 10.1109/ICRA.2015.7139818

[24] J. Hess, M. Beinhofer, W. Burgard "A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots" 2014 IEEE International Conference on Robotics and Automation (ICRA) Jun. 2014 doi: 10.1109/ICRA.2014.6907682

[25] A. Grünauer, G. Halmetschlager-Funek, J. Prankl, M. Vincze "The Power of GMMs: Unsupervised Dirt Spot Detection for Industrial Floor Cleaning Robots" TAROS 2017 pp. 436-449

[26] R. Bormann, F. Weisshardt, G. Arbeiter, J. Fischer "Autonomous dirt detection for cleaning in office environments" 2013 IEEE International Conference on Robotics and Automation Oct. 2013 doi: 10.1109/ICRA.2013.6630733

[27] R. Bormann, X. Wang, J. Xu, J. Schmidt "DirtNet: Visual Dirt Detection for Autonomous Cleaning Robots" 2020 IEEE International Conference on Robotics and Automation (ICRA) Sep. 2020 doi: 10.1109/ICRA40945.2020.9196559

[28] H. van Hasselt, A. Guez, D. Silver, "Deep Reinforcement Learning with Double Q-learning" AAAI'16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094-2100, Feb. 2016

[29] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning" ICML'16: Proceedings of the 33rd International Conference on International Conference on Machine Learning, vol. 48, pp. 1995-2003, Jun. 2016

[30] S. Ruder, "An Overview of Multi-Task Learning in Deep Neural Networks", arXiv, eprint: 1706.05098, 2017.

[31] Jing Zhao, Xijiong Xie, Xin Xu, Shiliang Sun, "Multi-view learning overview: Recent progress and new challenges", Information Fusion, Volume 38, 2017, Pages 43-54, ISSN 1566-2535, https://doi.org/10.1016/j.inffus.2017.02.007.

[32] A. Labroski, "Multi-view versus single-view machine learning for disease diagnosis in primary healthcare", Dissertation, 2018.

[33] Xiaoqiang Yan, Shizhe Hu, Yiqiao Mao, Yangdong Ye, Hui Yu, Deep multi-view learning methods: A review, Neurocomputing, Volume 448, 2021, Pages 106-129, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2021.03.090.

[34] https://github.com/HumaRobotics/mybot_gazebo_tutorial