Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Image Based Robot Localization and Orientation Classification Using CGI and Photographic Data

Eric Claus Bergter

MIN-Fakultät
Fachbereich Informatik
Knowledge Technology
Studiengang: Bachelor Informatik
Matr.-Nr. 6791895

Erstgutachter: Dr. Matthias Kerzel
Zweitgutachter: M.Sc. Marc Bestmann

Abgabe: 29.06.2020

A distributed system is one where the failure of some
computer I've never heard of can keep me from getting my work done.
– *Leslie Lamport*

1 Abstract

In this thesis the neural network architecture YOLO is trained with CGI to estimate the rotation of an object around one axis. The neural network is used to localize a specific robot in an image and estimate its rotation relative to the camera in the RoboCup Soccer environment. Current methods rely on specific geometry on the robot to determine its orientation. Determining the rotation of a robot is important for any tactical maneuver and helps to orient itself. For the training a mix of CGI rendered models and real recordings was used. The results show that convolutional neural networks such as YOLO can be used to do basic pose estimation and is able to use CGI images during the training for real world predictions.

2 Zusammenfassung

In dieser Arbeit ist die neuronale Netzwerkarchitektur namens “YOLO” mit CGI trainiert worden, um die Drehung eines Objekts um eine Achse zu bestimmen. Das neuronale Netzwerk wurde verwendet, um einen Roboter in einem Bild zu lokalisieren und seine Drehung relativ zur Kamera in der RoboCup Umgebung abzuschätzen. Aktuelle Methoden basieren auf einer bestimmten Geometrie des Roboters, um daran seine Ausrichtung zu bestimmen. Die Bestimmung der Rotation eines Roboters ist für jedes taktische Manöver wichtig und hilft sich im Spiel zu orientieren. Für das Training wurde eine Mischung aus CGI gerenderten Modellen und echten Aufnahmen verwendet. Die Ergebnisse zeigen, dass Neuronale Netze wie YOLO zur grundlegenden Ausrichtungsbestimmung verwendet werden kann und in der Lage ist, CGI Bilder im Training für reale Vorhersagen zu verwenden.

Contents

List of Figures

Glossary

Cam Video camera or webcam. 36

CGI Computer-generated imagery. i, vi, 2, 3, 13, 16, 20, 25, 29, 30, 33, 36

CNN Convolutional neural network. 16, 18

EEVEE Blenders new physically-based real-time rendering engine. v, 17

IoU Intersection over union. vi, 23, 27, 30, 33

JPEG Joint Photographic Experts Group, a lossy image file format. 17

PNG Portable Network Graphics, a lossless image file format with an Alpha channel. 17

ROS Robot Operating System. v, 18, 19

1 Introduction

The RoboCup Humanoid Soccer aims to imitate real soccer with robots on a smaller scale. In the kid sized league, the robots have to act autonomously by making decisions based on the data they receive from human like sensors. Due to their human posture and the limited perception of their surroundings, they are clumsy and frequently fall over or lose the ball. On a basic level, the robots from the same team are able to communicate with each other and use the information of their teammates rotation to direct them towards the ball, or to help them get a sense of direction should they be lost. Furthermore, on a higher tactical level, the information of robots rotation, friend or foe, is crucial for tactical planning. Since the robots vision and movement are primarily limited to the forwards direction, knowing who is looking where is needed for more complex maneuvers such as passing the ball, or intercepting it. The motivation behind this thesis is explained in 1.1, the goal and expected results are explained in sec. 1.2 and 1.3.

1.1 Motivation

Machine learning has made great progress in the recent years and even found its way into our daily lives. Every time you type something into Google, get a product recommendation or ask a Virtual Personal Assistant for help, you will get customized results optimized by machine learning algorithms.

Availability of newer and better hardware, large amounts of data, more sophisticated and specialized network architectures made this possible. However, the fact that a new problem often requires a custom tailored neural network and custom data to train it remains unchanged. In the RoboCup Soccer domain, for example, it is common practice to create a custom network for each task which requires unnecessary computational power and memory space. Reshaping the problem of the pose estimation for the robots, in such a way that a custom network becomes unnecessary and a commonly used network could be used instead, would allow for the combination of multiple problems into a single network, leaving additional resources for other tasks. The post estimation of a robot, is currently a problem that is only approached with complex algorithms and hand crafted to specific robots. A neural network and a training process that would allow a robot to be quickly trained and be able to changes, such as training to detect a different robot, could save many hours of work and reduce computational requirements. The biggest issue with this approach, is that for a different robot, you usually also need to create a new data set by hand which is time consuming and work intensive. To reduce the workload

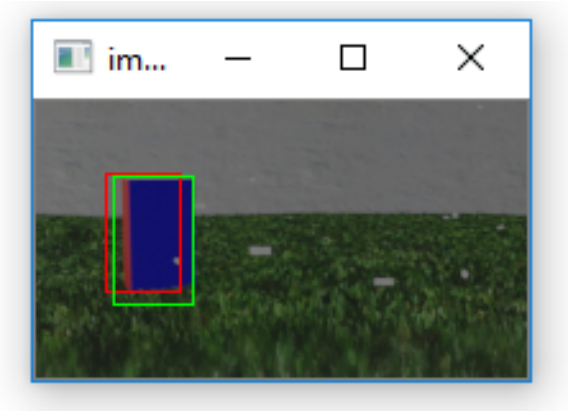

Figure 1.1: Green is the bounding box, red is the prediction by the model. The object to detect is a box with a different color on each side.

a 3D software is used to generate the data artificially. This has multiple advantages, as it allows the quick generation of many images without much effort and makes it possible to change the position, rotation, lightning and surroundings for the rendered images. The software itself can then directly provide the correct labels for each frame. The initial training data will be generated using the 3D Software Blender, see sec. 2.4, with the 3D model of the Robot being provided by the RoboCup Team of the University Hamburg. Blender is capable of rendering photo realistic images and itself is an open source program written in c++/python which allows for the reading of position and rotation data for the label generation. To test this approach, a simple scene was created consisting of a white wall, grass (particle hairs) and a cuboid that has a different color on each side. In the initial test 4800 images (158mb) were generated at 192x108px in about 10min. Those images were then used to test different network architectures. One of the early results, see Figure 1.1 was made with the Mobilenetv2 [SHZ+18] network.

1.2 Thesis Goal

This thesis has two goals. The primary goal is to evaluate if it is possible to re-purpose object classification for post estimation. The trained neural network should be able to localize and classify a robot and its vertical rotation based on a single image in real time and on limited hardware. The secondary goal was to evaluate the feasibility of using computer-generated imagery (CGI) for training robots, as a way to simplify the training data generation process. The neural network trained with these generated images should be able to transfer and apply this knowledge on real images taken by a robot.

1.3 Expected Results

The successful detection of a Robot in an image is to be expected due to the existence of networks capable of similar tasks. However, whether it is possible to accurately determine the rotation of a 3D object with an object classification network remains to be tested. Training the networks using CGI data instead of real images should be possible as long as the virtual scene is realistic and matched to the target domain [PSAS14].

2 Fundamentals

The following sections present the fundamental concepts and tools. At first in sec. 2.1 the context of the RoboCup and the limitations of the robot are shown. The mathematical concept of neural networks is explained in sec. 2.2 with the architecture Yolo being detailed in sec. chap:YOLO. The software used to generate images is presented in sec. 2.4 and to track real objects in sec. 2.5.

2.1 RoboCup

RoboCup is an annual international robotics competition which aims to promote robotics and AI research. The declared goal is to be capable of winning against the human soccer World Cup champions by 2050 [Roba]. From the first competition of soccer robots with just eight teams in 1996, it has grown to 3,500 dedicated scientists and developers from more than 40 countries [Robd]. Since then, multiple leagues have been founded in the RoboCupSoccer competition like the "Humanoid KidSize Size", "Standard Platform", "Small Size" and "Simulation" league. The Simulation League focues on artificial intelligence and team strategy. Virtual players play soccer on a virtual field inside a computer [Sim]. The Small Size League with small robots moving on wheels, focuses on intelligent multi-robot/agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system [Sma]. The Standard Platform League requires all teams to compete with identical robots. With this restriction, the software is the important part and allows for a direct comparison between the teams [Robc]. The KidSize Size League requires fully autonomous robots with their own sensors on board. These robots can be freely designed and are only limited by a maximum size and weight. This league focuses on the hardware design of the robots, as well as controlling and coordinating them [Robe]. It requires the robots to play against each other completely autonomously with a human-like body and human-like senses. Since the Hamburg Bit-Bots play in the Humanoid KidSize League, this league will be used as the reference. Similar to the Standard League the rules in this league specify a field size of 9 by 6 meters [Robb]. The Hamburg Bit-Bots robots participating in this league are based on the Wolfgang robot platform (see fig. 2.1) [Wol].

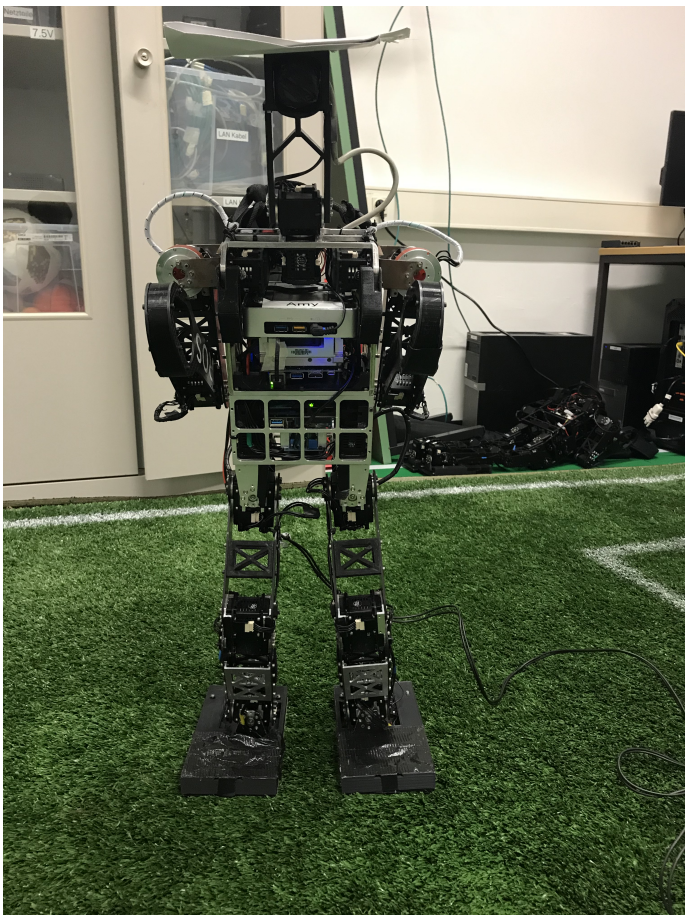

Figure 2.1: Robot "Amy" with an AprilTag attached to the head.

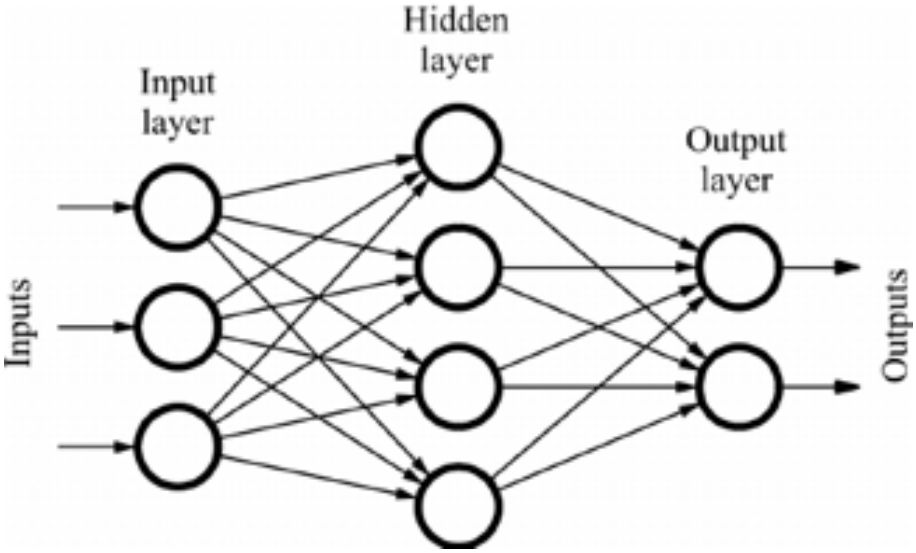

Figure 2.2: The input layer on the left is connected to the hidden layer. The hidden layer then connects to the output layer (a.k.a. the prediction) [War].

2.2 Neural Networks

A Neural network in the context of computer science is a mathematical model in the form of a computer program, that tries to emulate the way the human brain analyzes and processes information. The concept for the first neural network was developed in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts [MP88]. However, the concept could not be tested until 1954 due to the limited technology at the time. An example of a basic neural network is visualized in 2.2. Most neural network follows the same structure of an input layer, multiple hidden layers and an output layer. The layers consist of neurons that activate based on the input they receive. The activation is determined by an activation function, a bias, and a weight. All neurons in the same layer have the same activation function. This collection of neurons can be trained by giving it an input and determining the difference to the desired result, also called forward propagation. This error is then passed backwards through the network and the bias of each neuron is adjusted with error and the learning rate to get closed to the desired result. In computer vision neural networks are often used to classify images.

Overfitting in the conext of neural networks means fitting the parameters too closely to the specific training data [Kan11]. This leads to a model that can accurately predict the output for the training data but becomes inaccurate for data that was not included in the training set.

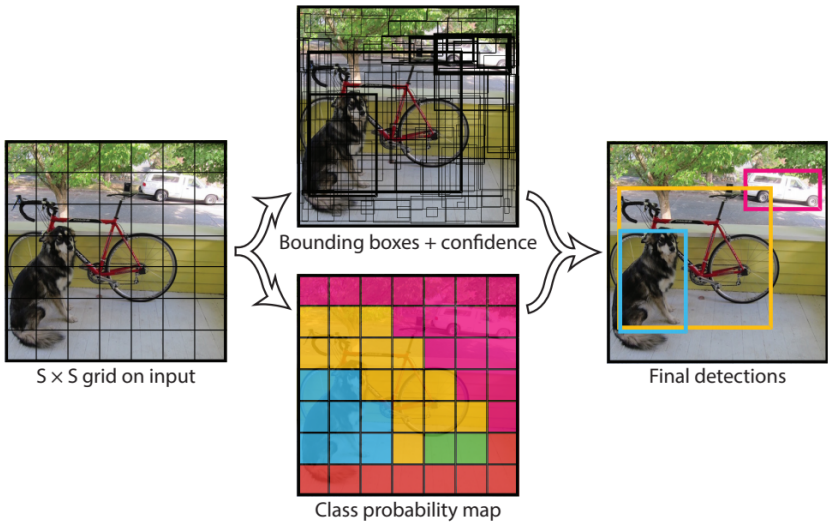

Figure 2.3: The Yolo system models detection as a regression problem. It divides the image into an S x S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

2.3 YOLO

YOLO is a convolutional neural network based model that detects objects in real time using the "You Only Look Once" framework. It is based in darkflow [Tri18] and can detect over 9000 different objects with 70% accuracy. Compared to other region proposal classification networks (fast RCNN) which perform detection on various region proposals and thus end up performing prediction multiple times for various regions in a image, Yolo architecture is more like FCNN (fully convolutional neural network) and passes the image (nxn) once through the FCNN and the output is (mxm) prediction. This the architecture is splitting the input image in mxm grid and for each grid generation 2 bounding boxes and class probabilities for those bounding boxes. The basic workings of the model can be seen in fig. 2.3. In this thesis Yolo refers to the Yolov3-Tiny configuration as seen in fig. 2.4. Yolov3-Tiny focuses on a small number of classes and is much faster than Yolos normal sized configuration. [RF16]

2.4 Blender

Blender is a free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and 2D animation pipeline. [Blea] In addition to that, all of those tools can be accessed using a python API. With the recent addition of the the EEVEE engine, the gap between offline and real-time rendering is being bridged. It Previsualizes Cycles shading with great accuracy in real time, in the viewport, and significantly speeds

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figure 2.4: Layout of the tiny-YOLOv3 network. The input is a 416x416pixel image with RGB colors. The output depends on the number of classes, in this case it is a 13x13x24 grid. [HHW+19]

up the shading and texturing process. [Bleb]

2.5 Apriltag

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera. The AprilTag library is implemented in C with no external dependencies. It is designed to be easily included in other applications, as well as be portable to embedded devices. Real-time performance can be achieved even on cell-phone grade processors. [Ols11]

3 Related Work

The following sections present work related to the approach introduced in sec. 4.1. Section 3.1 and sec. 3.2 give an overview of general approaches for computer vision, specifically computer vision with neural networks. The approaches for the task of detecting the pose of humanoid objects are detailed in section 3.4 and for object in sec. 3.3. The method used by the RoboCup team B-Human for University Bremen is explained in sec. 3.5.

3.1 General Approaches

For neural Networks image classification is a supervised learning problem which attempts to create a model that is able to identify objects within an image.
Recently deep neural networks, more specifically convolutional neural networks (CNN) are gaining more and more popularity, as they are increasing in accuracy. [KSH12a] On the other hand conventional approaches, that do not make use of neural networks make, use a combination of handcrafted filters. These filters are very time consuming to create and adjust.

3.2 Object detection

The convolutional neural networks AlexNet [KSH12b] won in the ImageNet challenge in 2012. With an error rate of only 15% on their top 5 guesses, it showed the feasibility of using a CNN for object detection. The VGG Net [SZ14] showed that increasing depth by using smaller filters improves the results of the network. The VGG Net used smaller 3x3 filters unlike e.g. AlexNet which used larger filters but fewer layers. In medicine, the U-Net [RFB15] has been very successfully used for image segmentation. In this case, not only image classification is needed, which the AlexNet already performed well for the ImageNet challenge, but also the image has to be segmented into several object regions. This means the location of different objects has to be detected. The task given is an image of cells. The cells have to be segmented in the image and then they have to be classified. Another challenge in this domain is the very low amount of training images. Thus data augmentation was heavily used for the training of the U-Net. YOLO [RF18] as explained in section 2.3, puts a grid of predefined size on the image. Then the architecture is split to fulfill two tasks. One of these tasks is the object localization. This part does not classify what the bounding box contains, it only finds the most probable positions for objects. The classification is done in the other task that is run simultaneously. Each grid element

is classified as one object. The bounding boxes passing a confidence threshold are then evaluated and a class is chosen for each bounding box. This approach is faster than other approaches [RDGF15] while maintaining very high accuracy. Additionally, a model called Tiny YOLO has been published [RF18] which makes the trade-off of accuracy for speed of detection. This architecture would be the most suitable to achieve real-time detection on the limited hardware available on the robot.

3.3 Pose estimation

PoseCNN is a convolutional neural network for 6D object pose estimation. The model estimates the 3D translation of an object by localizing its center in the image and predicting its distance from the camera. For that it first generates semantic labels, which are then used to predict the 3D postion. This however, requires a depth camera to generate accurate results. [XSNF17]

3.4 Orientation Estimation

Deep Convolutional Neural Networks (DCNN) have been proven to be an effective solution for various computer vision problems. The paper [HVC17] demonstrates the application of the proposed architecture on a continuous object orientation estimation task, which requires prediction of 0 to 360 degrees orientation of the objects. It compares three continuous orientation prediction approaches designed for the DCNNs. The first two approaches work by representing an orientation as a point on a unit circle and minimizing either L2 loss or angular difference loss. The third method works by first converting the continuous orientation estimation task into a set of discrete orientation estimation tasks and then converting the discrete orientation outputs back into the continuous orientation using a mean-shift algorithm. It is evaluated on a vehicle orientation estimation task and a pedestrian orientation estimation task, and it was demonstrated that the discretization-based approach not only works better than the other two approaches but also achieves state-of-the-art performance.[HVC17]

3.5 Approaches in RoboCup

The paper by B-Human from the University Bremen presents a vision-based approach for determining the orientation of humanoid NAO robots over short and medium distances. It is based on the idea of analyzing the alignment of other robots foot sides using canny edge detection. Under ideal conditions it can accurately determine the direction of the robot. [ML18]

4 Approach

This chapter explains the approach to the experiment and the reasoning behind the decisions that were made. In sec. 4.1 the format of the Labels for the images and the neural training is explained. Sec. 4.2 defines the label classes and their derivation from the 3D data. The different methods for labeling recorded image data can be found in sec. 4.3 and how CGI was generated and labeled in sec. 4.4.

4.1 Label Format

In the initial phase of the project, it was not clear in how many different classes the rotation of the robot could be split. At the time the large majority of data was 3D generated images. Therefore, there was plenty of additional information that could be stored together with the bounding box in the label file. The exact details of this XML label file are explained in sec. 8.4. This label file together with various information contains the exact rotation of the robot relative to the camera in degrees. This allows it to quickly separate the data into different classes based on the rotation. As the project progressed further, it became clear that Yolo [RF18] would be the central neural network to be used. It was also decided that there are 8 rotation classes to be differentiated, the reason for this decision is explained in 4.2.
With this information the labels for Yolo were generated in the darknet format. This format remains unchanged from the original paper and is explained in detail in sec. 2.3

4.2 Label Classes

In an ideal world, a detection algorithm that can determine the orientation of an object down to a single degree would be the ultimate goal. However, there are numerous difficulties in the real world that make this goal difficult to achieve. A perfect solution for this task is not required. In a real soccer game, for example, player "A" does not need to know the exact absolute orientation of player "B" if he wants to pass the ball. Instead player "A" only needs to know the general direction player "B" faces. This could be represented as the classes "facing" and "not facing", where the class is "facing" when player "B" is looking at player "A". However for a proper cooperation it is necessary to know: Play the ball left, right or center towards player "B"? The two classes can not represent that, and neither can four properly, as the margin between "front" and "left" would be still too

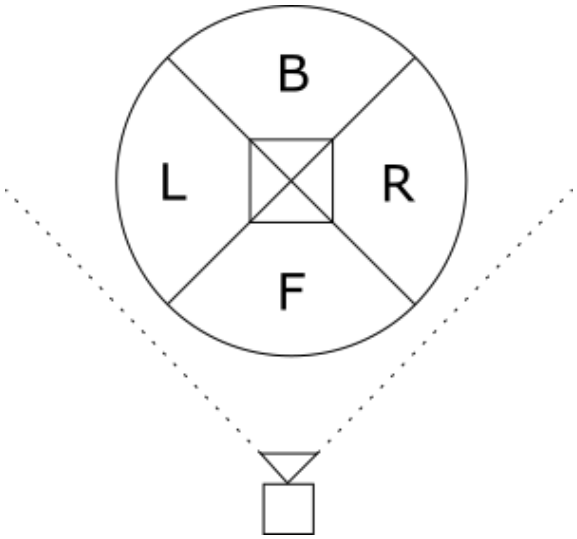

Figure 4.1: Simple separation of the different direction that can be recognized. Top down perspective with the robot in the center of the circle. Split into 4 classes F (front), L (left), B (back) and R (right), each 90 degrees. The robot is "front" if he is looking directly at the camera.

big. As seen in fig. 4.1 an image labeled "left" could almost be 180 degree from an image labeled "front".

As four classes are still insufficient, the number of classes is increased to eight. The eight classes are, "front", "front left", "left", "left back" back", "right back", "right" and "front right" as seen in figure 4.2.

An increase to 16 classes would require more than double the required training data as it makes it more difficult for the CNN to differentiate between the classes. With more classes it also becomes increasingly difficult for a human to differentiate between them, which would further prolong the labeling process and decreases the quality of the training set. When manually labeling images, it is simple to determine the rotation simply based on image. When generating the image, it is important to accurately determine the rotation relative to the camera and not in a global absolute value. In figure 4.4, the method for calculating the rotation for generated images is shown. The angle $\measuredangle ba$ is the rotation that is denoted in the XML file as described in 8.4.

Knowing this basic rotation of also allows the robot, with additional communication to his teammates to better orientate itself on the soccer field.

4.3 Image generation using manual labeling

The first step in training a Convoluting Neural Networks is the data generation. This, in many cases, is the biggest challenge for a supervised network, as the lack of reliable training and test data is a frequent occurrence.

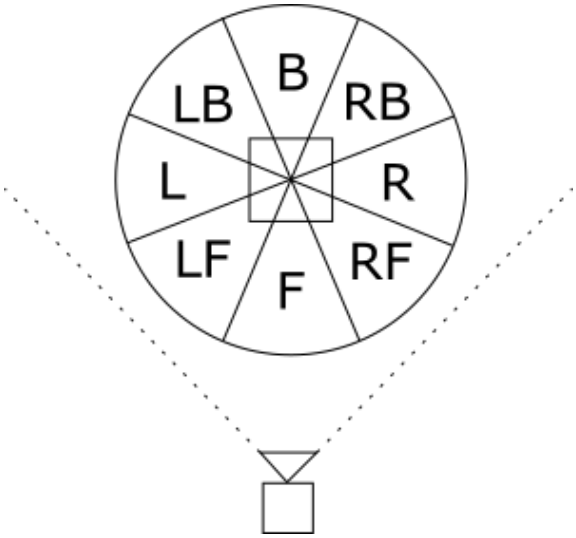

Figure 4.2: Similar to fig. 4.1, but with four additional sectors, each 45 degrees

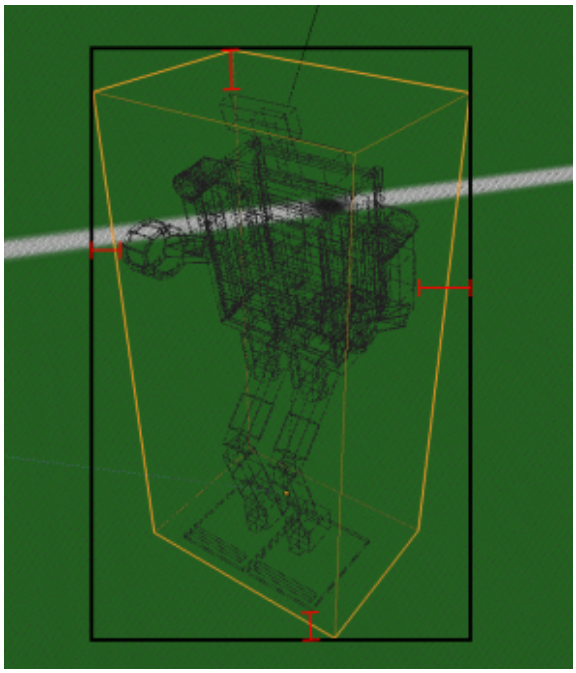

Figure 4.3: Transformation of the 3D bounding box (orange) to the 2D bounding box (black). Red bars highlight that the 2D bounding box is too large.

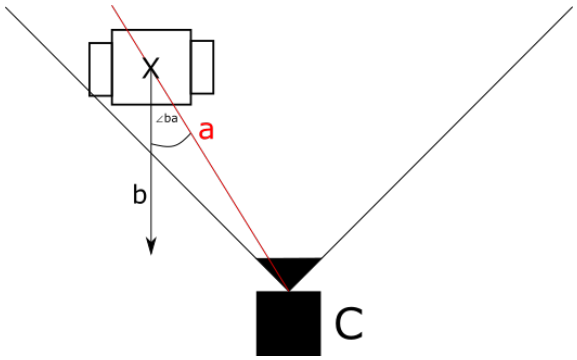

Figure 4.4: Computation of the rotation $\measuredangle ba$ relative to camera C.

Our team Bit-Bots at the University Hamburg has developed a tool "imagetagger: An open source online platform for collaborative image labeling" [ima] and a database for our robots. However, it does not contain any data for the new robot models or has any of the labels required for training. A common approach when training a CNN is to take a lot of images and label them manually. This usually means taking a lot of photos, upwards 1.000 images per class of the desired object. Each of these images then has to be labeled by a human. A label can be a name, a number or a shape on the image. In this case, a label is required that contains the rotation information and the position of the object in the image as a bounding box. The manual labeling was done using a modified version of OpenLabeling [Ope] storing the labels in the Yolo format (<object-class> <x> <y> <width> <height> sec. 8.5). The modifications include the ability to directly read Yolo's label format and some modifications to the UI for easier use.
The difficulty of creating manual labels did not only lie in the large number of labels required, but especially in the hard to decide edge cases. For the rotation class, edge cases were occurring when the object was just in between two rotational sectors, in which case it was decided at random. It was also decided that at least 50% of the robot has to be visible for it to be labeled. In total, ~12.000 images in the sets 10, 13 ,14 and 15, see table 8.1, were hand labeled and remaining missing labels were complemented with the automatic generation by the trained Yolo network.

4.4 Image generation using 3D rendering

The advantage of CGI is that they can be quickly created and adjusted to one's needs. The initial renders consisted of a simple colored box on a flat textured plane with a very low resolution of just 192x108 pixels, see fig. 4.5. This allowed for very fast render times of around 0.07s per frame or just 4min for a full data set of 3.000 frames. Not all of these data sets are listed in the table 8.1, as most of them were not significant enough to be considered relevant.
These quickly generated and compact data sets also allowed for a fast training and test-

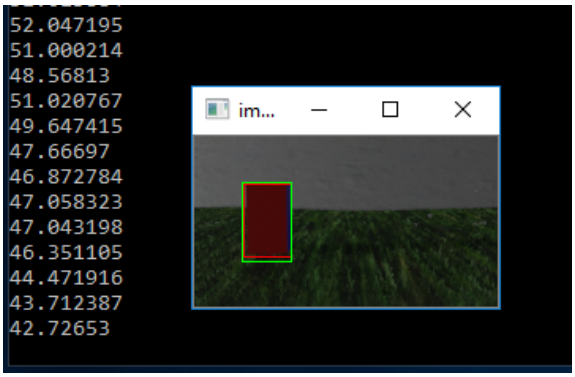

Figure 4.5: 192x108 pixel Blender EEVEE render. The box, is currently facing the camera with its red side, and is marked with a red bounding box for the ground truth and a green box for the prediction. On the left in the console are the rotation predictions in degrees from the recent frames.

ing of neural networks, as they contain little noise and are very small in file size. The renders were also changed from the PNG to the JPEG format, due to the much smaller file size. The image resolution also exactly matched the resolution of the input layer for the neural networks, and this removed yet another step in the training process.
The entire image render process is handled by the python script blender_get_labels.py, it takes a blender scene and automatically creates the desired amount of images with labels in the PASCAL VOC xml format [PAS]. Additional scripts also allow for the fast conversion of an existing data set to other label formats.
From there the complexity of the scene was increased step by step. The background was replaced by walls with texture, the ground at first just a plane evolved later into a group of fully simulated grass particles. The robot was also adjusted to the requirements, initially just a box, later a detailed model very similar to its real counterpart, see fig. 4.6. The robot was rigged, set up with key frames to simulate common poses and to dynamically change the color of certain textures, such as the colored arm bands. However some objects, such as the light and the grass was later reduced in complexity to maintain low render times.
During testing of the trained network, various flaws became obvious, such as the inability to detect the robot in front of a black background. This was addressed by adding different backgrounds into the virtual scene which solved the issue. Other tweaks included changes of light, additional noise with random background objects, tilting of the camera and the addition of motion blur.

4.5 Image generation using April Tag

AprilTag, see sec. 2.5, was also used for the data generation. The idea was that a ceiling camera C_a is used to track the position of both the recording camera C_b and the target

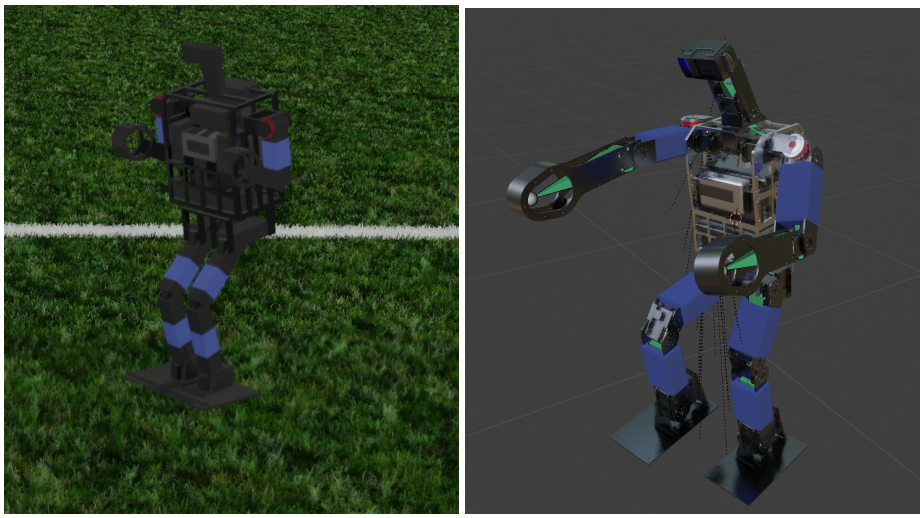

Figure 4.6: On the left the initial low-poly model of the robot using basic geometry, on the right the complex model of the fully rigged robot.

robot R as shown in fig. 4.7 and fig. 4.8. With that data it should be possible to calculate the position of the Robot relative to camera to create the bounding box label using the vectors V_c and V_r.
However, during the evaluation of the recorded data it became apparent that the Laptop running ROS was unable to properly handled the data of the 4k ceiling camera C_a and ended up generating only one data frame per second. Normally it would not be an issue, however, due to the long computation time of the data frame it became out of sync with the data recorded by the camera C_b. Another issue was the camera C_b was not calibrated and did not provide a camera matrix, which made it impossible to transform the 3D vectors of the data frame into the desired 2D bounding box. A makeshift solution was to feed the data into blender to approximate the camera matrix and calculate the labels using the blender_get_labels.py script. The result was barley usable, and still required manual corrections. As there was already sufficient manually labeled data, the recording of AprilTag data was not repeated.

4.6 Training

Initially, different networks such as a modifed CNN [SBB18], the architecture Mobilenetv2 [SHZ+18] and Yolo-tiny [RF18] were used for training. However, various factors, described in sec. 5.3, shifted the focus entirely onto the yolo-tiny framework.
The network was trained in the default configuration with anchors calculated on based on the given data set, as proposed in the original paper [RF18]. The networks hyperparameters can be found in the projects data sets as "yolov3-tiny.cfg". Following configuration is used across all data sets:

- batch = 64

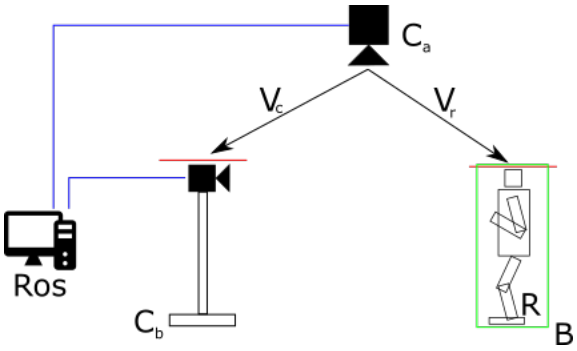

Figure 4.7: AprilTag Setup. Computer ROS connected to two Cameras, ceiling camera C_a and field camera C_b. AprilTags (red) are ontop of Camera C_b and Robot R. Over the robot is bounding box B (green).

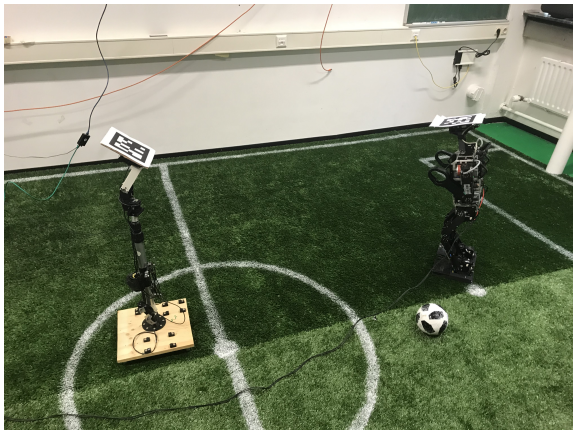

Figure 4.8: On the left, the camera used to record the robot. On the right, the robot. Not the AprilTag atteched ontop of both of them.

- subdivisions = 8
- width and height = 416
- learning_rate = 0.001
- classes = 8
- num = 6
- validation with data set nr. 15

Yolo-tiny was trained on an Nvidia GTX 1080, see Hardware sec. 8.1, and took between 1 and 6 hours depending on image resolution and training set size.
Early stopping was not used, as the network saved its weights separately every 1000 epochs. This made it possible to select and evaluate the desired iteration based on the loss value in the iteration-loss chart, see fig. 4.9. All sets used for evaluation were trained for at least 15.000 iterations. As seen in fig. 4.9, the graph of the loss function starts to flatten out near the 12.000th iteration. Different image sets were used for the training process and testing to prevent over-fitting.

4.7 Training with combined sets

A training set only consisting of CGI might perform poorly in real world conditions. The combination of training sets is important as each can substitute for the others lack of data in specific fields, it also increased the amount of data that can be used for training. During early testing it became obvious that combining different data sets increased the variety and performance of the trained network.

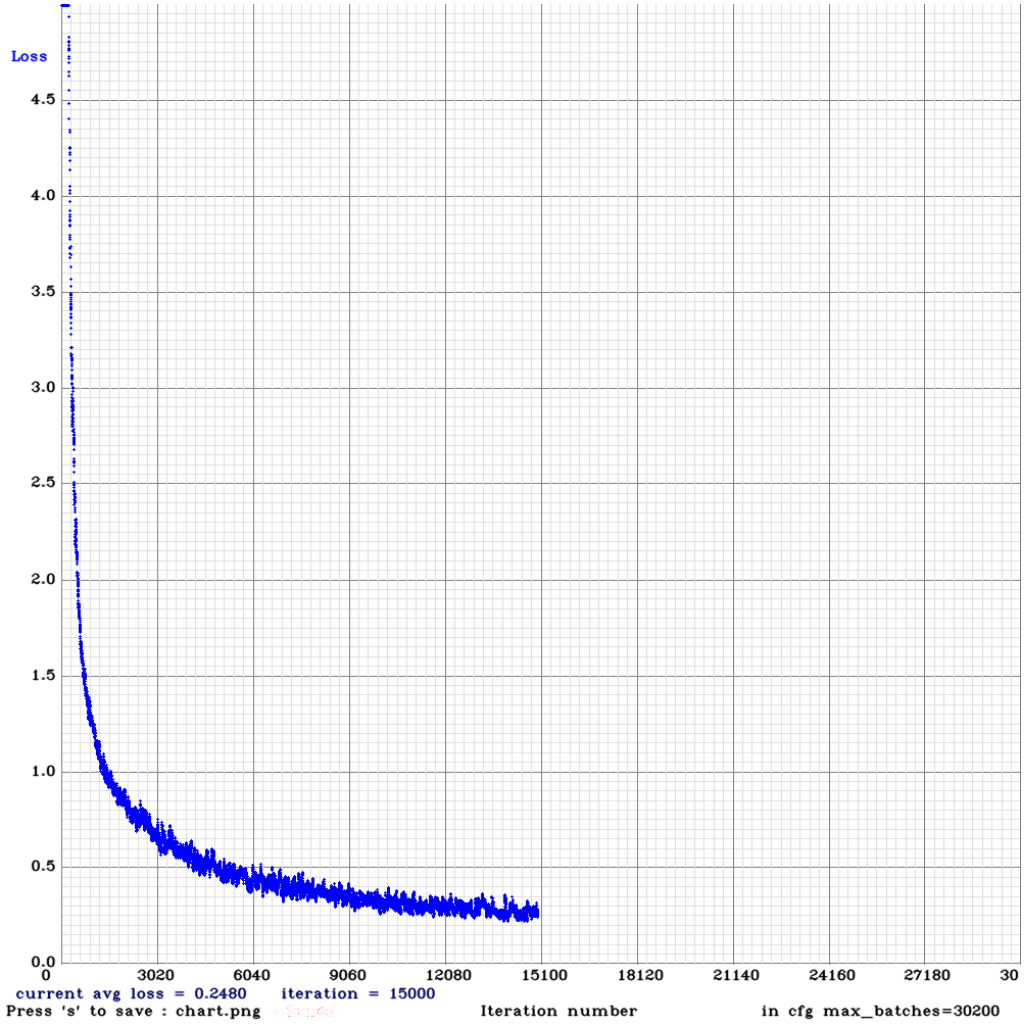

Figure 4.9: Training: Y-Axis: Loss, X-Axis: Iteration. Data set 9 for 15.000 iterations.

5 Evaluation

This chapter evaluates the results of the presented approaches by calculating the intersection over union for the objects detected. How to calculate intersection over union is explained in sec. 5.1 and the definition of class error in 5.2. The neural network used for the evaluation is presented in 5.3. Data about the training sets, including the amount of labels used for the training, is described in 6.2. The data of the different training methods will be compared and evaluated in 5.6.

5.1 Intersection over Union Calculation

The Jaccard index [Jac01], commonly known as Intersection over Union (IoU), is an evaluation metric used to mesure the relative size of the overlap of two finite sets. [RTG+19] In machine learning, the IoU is used as a metric to measure the accuracy of an object detector on a particular data set [Kos16]. The output of a neural network is often a predicted bounding box (P), which is then compared to a manually labeled ground truth bounding box (T). The IoU is then calculated by dividing the area of overlap and the area of union of P and T.

$$IoU = \frac{\mid P \cap T \mid}{\mid P \cup T \mid}$$

An IoU of 1 means that the prediction P perfectly overlaps with the ground truth T, while an IoU of 0 means that there is no overlap. The general threshold for the IoU is 0.5, though this varies from problem to problem. Normally, IoU > 0.5 is considered a good prediction.

5.2 Class Error

Usually, IoU is the go to metric when evaluating image localization networks, however in this case, the neural network was re-purposed for post estimation. This means that the classes should not be evaluated on their own, but instead be seen as a directional vector. The "class error" measures the average error between the ground truth and a prediction, only if they overlap with greater than a 0.5 IoU. The error is the distance between the class "A" and class "B", following bidirectional along the circle as seen in fig 4.2. A class error of 0 means that all classes are predicted perfectly, while a class error of 4.0 (for the

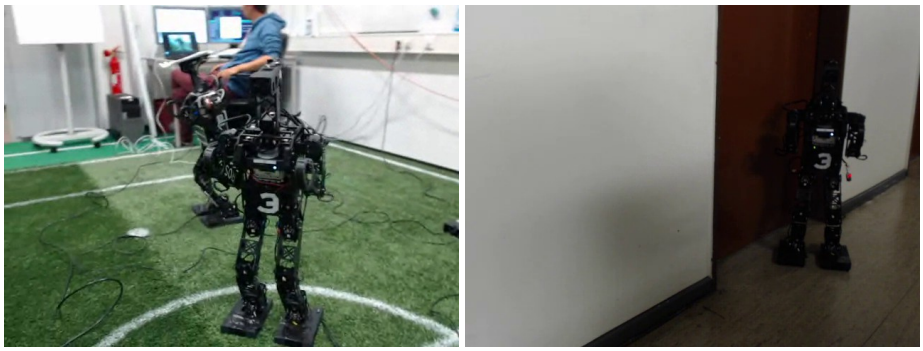

Figure 5.1: Evaluation set nr. 15. First scenario on the left, second scenario on the right.

eight classes) means all predictions where off by 180 degree. For example the class error for the prediction "front", with the ground truth "front_left", would be 1.0.

5.3 Evaluation Neural Network

In the initial phase of this project, there were different neural networks that were tested for the task of real time object detection on a mobile platform. Due to the hardware limitations and the task requirements of localization and classification, only the neural networks with the MobileNetv2 [SHZ+18] and YOLOv3-Tiny [RF18] remained. In the end YOLOv3-Tiny proved to be superior in both, performance and accuracy and was chosen for the evaluation.

5.4 Evaluation Image Set

The choice for the evaluation image set is limited to the sets with complete labels (see table 8.1), therefore it is not possible to use any sets from the ImageTagger of the Hamburg Bit-Bots [ima]. For the evaluation, the selected image set is nr. 15 "Duo Robots", because it was recorded using the same hardware and under similar conditions as the robots during a RoboCup soccer game. While the original set contains 5.000 images, the evaluation set is reduced to 957 images by removing almost identical frames. The set is broken down further by distinguishing between two different scenarios. The first half, roughly 35% of the evaluation set focuses on a scenario that is similar to the training sets, in order tests how good the network managed to learn. It consists of two robots standing close to each other on a soccer field, as seen in fig. 5.1. The second scenario is much more challenging, it is situated in an environment that the neural network has never seen during the training and only contains a single much darker robot.

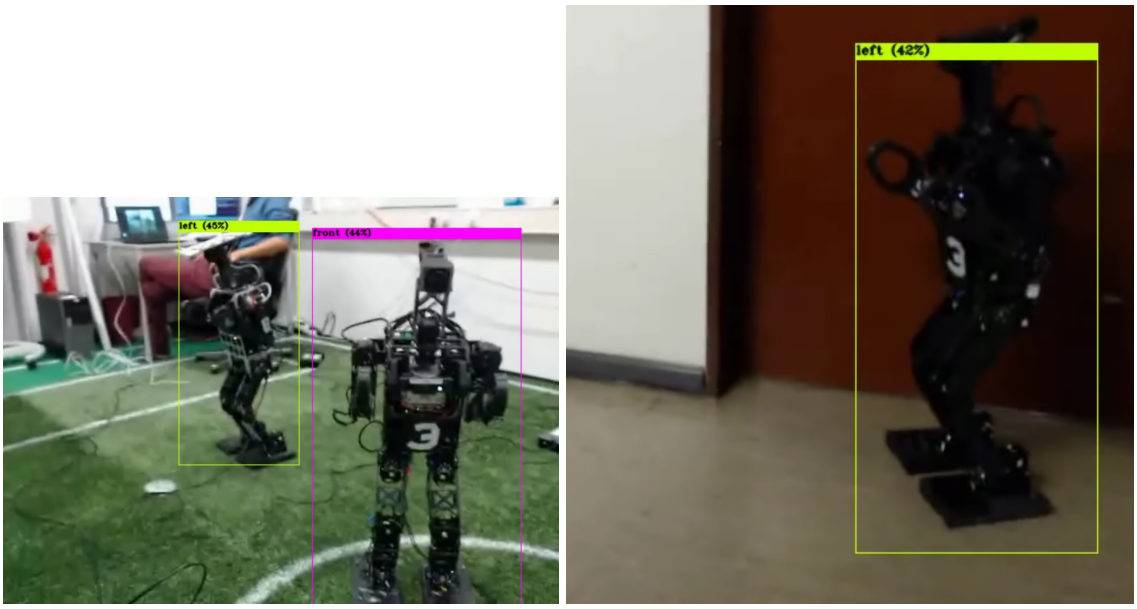

Figure 5.2: Predictions by set 12 on the evaluation set

5.5 Training Image Sets

The central training set is a combination of multiple data sets 9, 11, 12 and 13 (see Table 8.1). The foundation of this training set is formed by set 9 and 11, with a total of 18.000 CGI generated images of a virtual RoboCup Soccer field and robots. This is complemented by the sets 12 and 13 with 4.300 real images to teach the network the appearance of the robots in the real world. In section 5.6 other combinations of the training sets are analyzed on their accuracy.

5.6 IoU Accuracy and Evaluation

The performance of the Yolov3-tiny network trained with different data sets, see sec. 5.5, is evaluated on the evaluation set 15.
In scenario one, see table 5.1, the set 12 provides the most accurate results, with a true positives rate of 31.7% and a class error of just 1.2. For set 12 about one third of the predictions are correct and that on average the rotation is off by $f(x) = 360/8 * 1.2 = 54$. The equation to calculate avrg rotation error:

$$f(x) = 360/num\_classes * class\_error$$

Set 9, despite being a pure CGI set, has a true positives rate of 25.7% and a class error of just 1.93.

While set 9 and 12 individually showed good results, the combination of the two lead

to a drop in accuracy. The same is true for combined set 9, 11 ,12 and 13. Set 11 appears to be a case of over-fitting, as it was unable to make any meaning full predictions.
The labels right and left have a high class error of 2.23 and 2.95 respectively, indicating that the neural network has trouble keeping them apart.
In the second scenario, see table 5.2 the situation is similar, however, this time the prediction ratio is 1.0 meaning there are no false positives, if you ignore the class labels.

Figure 5.3: Tables with the evaluation results evaluation set 15. The true positives rate is the percentage of images that have both, a correct localization and classification label. On the other hand, the prediction ratio describes the percentage of correct localizations regardless of their class, as long as the IoU is > 0.5. As explained in in sec. 5.1, the IoU gives the accuracy of the bounding box. The class error, see sec. 5.2, indicates how accurate the label of the prediction is.

Table 5.1: Evaluation results for the first scenario of the evaluation set 15.

Training Set	Set 9	Set 11	Set 12	Set 9, 12	Set 9,11,12,13
True Postives	0,257	0,022	0,317	0,143	0,164
Prediction ratio	0,781	0,711	0,785	0,728	0,791
IoU	0,8	0,75	0,83	0,85	0,8
class error	1,93	1,77	1,2	1,28	1,61
class error per class					
Class 0 front	0,85	1,03	0,8	0,42	0,5
Class 1 front_right	1,71	0	1,71	3	2,25
Class 2 right	2,23	1	1,33	3	3
Class 3 right_back	n/a	n/a	n/a	n/a	n/a
Class 4 back	n/a	n/a	n/a	n/a	n/a
Class 5 left_back	0,58	2,5	0,1	0,16	0,38
Class 6 left	2,95	2,53	1,56	2,03	2,19
Class 7 front_left	1	2	0,9	0,62	1,05

Table 5.2: Evaluation results for the second scenario of the evaluation set 15.

Training Set	Set 9	Set 11	Set 12	Set 9, 12	Set 9,11,12,13
True Postives	0,26	0,02	0,32	0,14	0,16
Prediction ratio	1,00	0,99	1,00	1,00	1,00
IoU	0,88	0,79	0,89	0,91	0,89
class error	1,57	1,32	1,91	2,09	1,89
class error per class					
Class 0 front	2,47	1	1,07	2,33	1,42
Class 1 front_right	1,2	n/a	1,52	2,67	2,14
Class 2 right	0,86	1,07	2,41	3,23	2,36
Class 3 right_back	1,73	3,6	1,74	2,31	1,6
Class 4 back	0,87	0,71	2,96	2,5	3,33
Class 5 left_back	1,13	3	2,12	0,23	0,75
Class 6 left	2,85	1,38	1,35	1,46	1,56
Class 7 front_left	1,7	n/a	1,21	0,4	1,38

6 Discussion

In the following chapter, the evaluation results are discussed. The sec. 6.1 describes which image sets were used in the training process as well as which problems were noticed with the generated image sets. Sec. 6.2 discusses the choice of image set for the evaluation. The combination of different training sets and the impact on the results is discussed in sec. 6.3.

6.1 Training Data Generation

The training images available in the imagetagger [ima] mostly consist of images that were taken by a human with a phone, often in bad light conditions and only from the edge of the soccer field. The images also contain mostly various older robot models, which is not ideal as the goal is to work with the current models. Therefore, instead of hand labeling older images, the focus was on acquiring a working neural network and up to date training sets.
The initial testing and training was done using mostly just basic CGI to evaluate the neural network models at the time, as it was much faster due to the automatically generated labels as described in 4.4. From there, it was an iterative process of improving the neural networks and the training sets. For the training sets, most of the improvements were achieved by increasing the realism of the scene by adding more detail and more noise, as shown in figure 6.1. An important aspect of this process was streamlining the generation process and keeping the render times low, the detailed process is explained in section 4.4.

In initial tests, the CGI trained neural networks displayed the ability to detect a real robot, which it had never seen before. At the same time, it some issues with the training

Figure 6.1: Improvements in the CGI sets. From left to right: Set 01, 09.

set became apparent. For example as seen in figure 6.1 the background in the training set is mostly white. This directly translates to the real world detection ability, as the trained model was able to detect a robot in front of a white background, but not any other color. It also initially detected humans as robots when they stood on the soccer field. This was addressed in later training sets by adding different colored backgrounds and light conditions. During the evaluation on the real data set 15 "Duo Robots", as in section 5.6, the CGI trained models showed that with a 32% true positive rate and 80% of IoU the could be applied to real footage. However it also showed that without real image data the network struggles with foreign objects. There the usage of CGI training data in conjunction with real data is recommended. The data sets containing real photos taken using the robots web-cams was recorded and labeled in different ways, as explained in section 4.3. The majority of the labels for these data sets were generated using software, instead of being labeled by hand. One method was the in section 4.5 mentioned AprilTags method mentioned in sec. 2.5, this however proofed to be unreliable as the used hardware was unable to provide accurate results. The majority of the recorded data points were out of sync with video and therefore unusable. This was due to the hardware being unable to process the video streams fast enough and the AprilTag not being properly calibrated. Instead of attempting to correct this a faster method was available. The YOLO model, trained using CGI data, was use to pre-label the video recorded using AprilTags. After manually correcting pre-label errors, ,the same Yolo model was trained to over fit on the given pre-labels. Frames without a label were not used in the training process. After over fitting the trained model, it was able to generate the labels for all remaining images in the data set. Only very minor manual corrections we necessary. The majority the of the real data sets were created in this way. The normally undesired over fitting was a useful tool to quickly label and generate a lot of accurate data, saving a lot of time.

6.2 Evaluation Image Set

The image set 15 "Duo Robots" (see Data set 8.1) was chosen as the evaluation image set. The main reason is that it was recorded simulating similar conditions, as during a real RoboCup match. This image set provides a variety of angles of two different robots in front of different backgrounds at various distances.
Training set 15 consist to 35% of 2 Robots standing on a soccer field, to 50% out of a single robot standing in a hallway and the remaining images contain no robot. Due to a large portion of the set not being the usual soccer field, the evaluation set tests the ability of the network to deal with unknown environments. However, the overall sample size of the evaluation set is rather small with around one thousand images and does not test the behavior of the neural network with other different robot models.

6.3 Combination of different Data sets

Single data sets originating from a single recording cause various issues during training. They tend to over fit on the given images and are unable to translate effectively into a real application. Combining different data sets in order for them to complement each other appeared to be the most effective way to ensure more generalization and to prevent over fitting. However, as shown in the evaluation in sec. 5.6 this is not necessarily true, as often the overall accuracy of the network was decreased when different data sets were combined.

7 Conclusion

Recent research on vision based pose estimation, especially in the RoboCup environment, has achieved some success. However, current methods such as the line detection [ML18] only work on specific visual features under the right circumstances. In this thesis a fast and robust method for training and detecting the rotation of other robots based on visual data was implemented. A pipeline consisting of a fast data generation, training and detection was developed and evaluated.
In sec. 5.6 the accuracy of the CGI set nr. 9 was directly compared to the performance of the network trained on real photos, set nr. 12. Despite never having see a real photo before, the CGI set nr. 9 was able to achieve a 0.8 IoU and a class error of 1,57. The result of the set nr. 12 was similar, as they training data was not diverse enough and unable to deal with the difficult evaluation set. Against the expectations the combination of data sets 9, 11, 12 and 13 scored overall lower than individual sets.
It was shown that the orientation of a 3D object can be predicted with a standard network used for object classification and localization. In addition it was shown that CGI generated images can be used and even replace real data during training almost completely.
The work flow used for this project, of generating data, reusing existing network architectures, training, and testing, significantly reduced the required work, especially in the area of labeling and creating that data as explained in sec. 4.4.

7.1 Future Work

In the future, the proposed process of the data generation could be made into a tool that allows the management, generation of data, training and testing of neural networks. The current implementation is just a group of lose scripts with hard coded file paths and only supports one or two label formats.
For the trained network, tiny-yolo, there are further improvements to be made. It should be tested how much the complexity of the network can be reduced, to increase FPS performance without losing detection rates. Further, in the current version the network is only able to detect of the rotation of an object around one axis, however it should be possible to expand further onto 2 or all 3 rotational axis for full 6D detection by adjusting the labels and training data. Overall the training and testing data needs to be improved. While the manually labeled photos have their own issues with inconsistency, the CGI images for example can be improved with a tighter bounding box, as the current implementation does not work well on specific angles, as seen in fig. 4.3.

8 Appendices

8.1 Hardware

OS: Linux Ubuntu and Windows 10
Python 3.6 Tensorflow
CPU: i7-6800k @3.6Ghz
GPU: GTX1080

8.2 Performance and Runtime

The performance of the trained network was not explicitly tested on the robot hardware and is 220 FPS with a desktop class GPU. [RF18]

8.3 Datasets

Relevant data sets can be seen in Table 8.1. Early data sets are not included in this list.

8.4 Data Format

Basic information for all generated images is written as a XML file in the same directory as the image and with the same name, but with a .xml-extension. The file is in a PASCAL VOC Format [PAS] with the additional field "rotation", that stores the information of the objects rotation relative to the camera.

8.5 Label Format

The label format for training is the same format used in the Yolo [RF18] network. A .txt-file for each .jpg-image-file. The file contains: object number and object coordinates on this image, for each object in new line:
<object-class> <x> <y> <width> <height>
Where:
<object-class> - integer number of object from 0 to (classes-1)
<x> <y> <width> <height> - float values relative to width and height of image, it can be equal from (0.0 to 1.0)

Table 8.1: Data sets used in this paper

ID	Name	Type	Description	Size	Resolution
1	01_simple_cube_small	CGI	Colored Cube on soccer field	1.499	192x108
2	02_simple_cube_large	CGI	Colored Cube on soccer field	2.999	1920x1080
3	03_simple_cube_small	CGI	Colored Cube on soccer field	2.999	192x108
4	04_robo1_small	CGI	Basic Robot on soccer field	2.999	192x108
5	05_robo1_small	CGI	Basic Robot on soccer field	4.499	192x108
6	06_robo1_50	CGI	Basic Robot on soccer field	4.499	960x540
7	07_robo2_100	CGI	Complex Robot on soccer field	4.498	1920x1080
8	08_robo2_100_blur	CGI	Robot - Motion Blur	8.998	1920x1080
9	09_robo2_640	CGI	Robot with realistic Environment	8.998	640x380
10	10_rosbag	Cam	Rosbag Apriltag Recording	14.179	640x380
10.1.	10_rosbag_generated	Cam	Rosbag Apriltag. Generared Labels	14.179	640x380
11	11_robo3_100	CGI	Robot with realistic Environment	8.998	1920x1080
12	12_rosbag	Rosbag	Dataset 10 - Handlabled	3.274	960x640
13	13_imagetagger_352	Cam	Close ups (Other Robots models)	786	960x540
14	14_imagetagger_167	Cam	Other Robot models - No Labels	1.596	640x360
15	15_test_recording	Cam	Duo Robots	9.570	640x480
15.1.	15_test_recording_3FPS	Cam	Due Robots, low FPS	957	640x480

for example: <x> = <absolute_x> / <image_width> or <height> = <absolute_height> / <image_height>
<x> <y> - are center of rectangle and not the top-left corner

8.6 Python Scripts

The project can be found on `https://git.mafiasi.de/15bergter/RoboRotationYolo`

8.6.1 *main.py* and *core.py*

A basic menu to start and review training with different networks.

8.6.2 *blender_get_labels.py*

A script that has to be executed from within Blender.
It is handling both the rendering process and the generation of labels with bounding boxes and classes.

8.6.3 *blender_rosbag.py*

Similar to sec. 8.6.2 it has to be executed within Blender and will handle the rendering process and label creation. In addition, this script positions the objects in the virtual scene according to the AprilTag data.

8.6.4 *convert_to_darknetLables.py*

Converts a given data set from the Pascal VOC format to darknet labels used by Yolo.

8.6.5 *dataset_txt_to_train.py*

A simple tool to write all labeled images into a train.txt. Each row in the train.txt is the full path to an image with a label.

8.6.6 *generate_darknet_from_dataset.py*

Generates all files required to train a data set from the data sets path.

8.7 Acknowledgment

I would like to thank everyone from our local RoboCup Team Bit-Bots, with special thanks to:
Marc Bestmann
Dr. Matthias Kerzel
Daniel Speck
Jonas Hagge
Niklas Fiedler

Bibliography

[Blea] *Blender*. https://www.blender.org/, Abruf: 03.06.2020

[Bleb] *Rendering*. https://www.blender.org/features/rendering/, Abruf: 03.06.2020

[HHW+19] He ; Huang, Chang-Wei ; Wei, Liqing ; Li, Lingling ; Anfu, Guo: TF-YOLO: An Improved Incremental Network for Real-Time Object Detection. In: *Applied Sciences* 9 (2019), 08, S. 3225. http://dx.doi.org/10.3390/app9163225. – DOI 10.3390/app9163225

[HVC17] Hara, Kota ; Vemulapalli, Raviteja ; Chellappa, Rama: *Designing Deep Convolutional Neural Networks for Continuous Object Orientation Estimation*. 2017

[ima] *Bit-bots imagetagger*. https://humanoid.robocup.org/, Abruf: 03.06.2020

[Jac01] Jaccard, Paul: Etude de la distribution florale dans une portion des Alpes et du Jura. In: *Bulletin de la Societe Vaudoise des Sciences Naturelles* 37 (1901), 01, S. 547–579. http://dx.doi.org/10.5169/seals-266450. – DOI 10.5169/seals–266450

[Kan11] Kantardzic, Mehmed: *Data mining : concepts, models, methods, and algorithms*. Piscataway, New Jersey Hoboken, NJ : IEEE Press Wiley, 2011. – ISBN 978–0–470–89045–5

[Kos16] Kosub, Sven: A note on the triangle inequality for the Jaccard distance. In: *CoRR* abs/1612.02696 (2016). http://arxiv.org/abs/1612.02696

[KSH12a] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey: ImageNet Classification with Deep Convolutional Neural Networks. In: *Neural Information Processing Systems* 25 (2012), 01. http://dx.doi.org/10.1145/3065386. – DOI 10.1145/3065386

[KSH12b] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. Version: 2012. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-network

pdf. In: PEREIRA, F. (Hrsg.) ; BURGES, C. J. C. (Hrsg.) ; BOTTOU, L. (Hrsg.) ; WEINBERGER, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, 1097–1105

[ML18] MÜHLENBROCK, Andre ; LAUE, Tim: Vision-Based Orientation Detection of Humanoid Soccer Robots. In: AKIYAMA, Hidehisa (Hrsg.) ; OBST, Oliver (Hrsg.) ; SAMMUT, Claude (Hrsg.) ; TONIDANDEL, Flavio (Hrsg.): *RoboCup 2017: Robot World Cup XXI*. Cham : Springer International Publishing, 2018. – ISBN 978–3–030–00308–1, S. 204–215

[MP88] In: MCCULLOCH, Warren S. ; PITTS, Walter: *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Cambridge, MA, USA : MIT Press, 1988. – ISBN 0262010976, S. 15–27

[Ols11] OLSON, Edwin: AprilTag: A robust and flexible visual fiducial system, 2011, S. 3400 – 3407

[Ope] *OpenLabeling*. https://github.com/Cartucho/OpenLabeling, Abruf: 03.06.2020

[PAS] *PASCAL VOC Format*. https://github.com/shwars/mPyPl/wiki/Reading-PASCAL-VOC-Format, Abruf: 03.06.2020

[PSAS14] PENG, Xingchao ; SUN, Baochen ; ALI, Karim ; SAENKO, Kate: *Learning Deep Object Detectors from 3D Models*. 2014

[RDGF15] REDMON, Joseph ; DIVVALA, Santosh K. ; GIRSHICK, Ross B. ; FARHADI, Ali: You Only Look Once: Unified, Real-Time Object Detection. In: *CoRR* abs/1506.02640 (2015). http://arxiv.org/abs/1506.02640

[RF16] REDMON, Joseph ; FARHADI, Ali: *YOLO9000: Better, Faster, Stronger*. 2016

[RF18] REDMON, Joseph ; FARHADI, Ali: YOLOv3: An Incremental Improvement. In: *arXiv* (2018)

[RFB15] RONNEBERGER, Olaf ; FISCHER, Philipp ; BROX, Thomas: *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015

[Roba] *RoboCup*. http://www.wpcentral.com/ie9-windows-phone-7-adobe-flash-demos-and-development-videos, Abruf: 03.06.2020

[Robb] *RoboCup Humanoid League*. https://humanoid.robocup.org/, Abruf: 03.06.2020

[Robc] *RoboCup Standard Platform League*. https://spl.robocup.org/, Abruf: 03.06.2020

[Robd] *RoboCup Visitors*. https://2019.robocup.org/visitors.php, Abruf: 03.06.2020

[Robe] *RoboCupSoccer - Kid Size*. https://www.robocup.org/leagues/29, Abruf: 03.06.2020

[RTG+19] REZATOFIGHI, Hamid ; TSOI, Nathan ; GWAK, JunYoung ; SADEGHIAN, Amir ; REID, Ian ; SAVARESE, Silvio: *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*. 2019

[SBB18] SPECK, Daniel ; BESTMANN, Marc ; BARROS, Pablo: *Towards Real-Time Ball Localization Using CNNs*. https://www.researchgate.net/publication/334976652_Towards_Real-Time_Ball_Localization_Using_CNNs. Version: 2018

[SHZ+18] SANDLER, Mark ; HOWARD, Andrew ; ZHU, Menglong ; ZHMOGINOV, Andrey ; CHEN, Liang-Chieh: *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2018

[Sim] *Simulation League*. https://www.robocup.org/leagues/23, Abruf: 03.06.2020

[Sma] *Small Size*. https://www.robocup.org/leagues/7, Abruf: 03.06.2020

[SZ14] SIMONYAN, Karen ; ZISSERMAN, Andrew: *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014

[Tri18] TRIEU, Trinh H.: Darkflow. In: *GitHub Repository. Available online: https://github.com/thtrieu/darkflow (accessed on 14 February 2019)* (2018)

[War] WARKE, Chetan: *Simple Feed Forward Neural Network code for digital Handwritten digit recognition*. https://mc.ai/simple-feed-forward-neural-network-code-for-digital-handwritten-dig Abruf: 03.06.2020

[Wol] *Wolfgang robot platform*. https://submission.robocuphumanoid.org/uploads//Hamburg_Bit_Bots_and_WF_Wolves-specs-5c03d58ec8f93.pdf, Abruf: 03.06.2020

[XSNF17] XIANG, Yu ; SCHMIDT, Tanner ; NARAYANAN, Venkatraman ; FOX, Dieter: *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. 2017

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____