h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

Darmstadt University of Applied Sciences

– Faculty of Mathematics and Natural Sciences –
– Faculty of Computer Science –

Real-time Game State Detection on a semi-automatic Foosball Table

Submitted in partial fulfilment of the requirements for the degree of

Master of Science (M.Sc.)

by

David Hagens

Matriculation number: 755866

Supervisor	:	Prof. Dr. Elke Hergenröther
Co-Supervisor	:	Prof. Dr. Andreas Weinmann

Date of registration	:	June 1, 2023
Date of submission	:	December 1, 2023

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 29. November 2023

David Hagens

Zusammenfassung

Fußball stellt bekannte Herausforderungen in der KI-Forschung dar. Eine Abstraktion mit kleinerem Parameterraum liegt im Tischfußball. Bei einem halbautomatisierten Kicker wird ein Team mit Industriemotoren durch einen KI-Agenten gesteuert, während das andere Team von Menschen gespielt wird. Für eine dynamische Spielweise benötigt der KI-Agent den Zugriff auf so viele Daten wie möglich, insbesondere auf den Zustand des Spiels in echtzeit. Bei einem Kickerspiel wird der Spielzustand durch die Position und die Rotation der einzelnen Figuren sowie die Position des Balls definiert. In dieser Arbeit wird ein System konzipiert, welches die Erkennung des Spielzustands auf Basis von Computer Vision und Convolutional Neural Networks ermöglicht. Das System wird auf einem halbautomatisierten Tischkicker getestet, trainiert und evaluiert. Es wird ein End-to-End-Bildregressionsmodell verwendet, um die Position und die Rotation der einzelnen Stangen vorherzusagen. Fünf verschiedene Architekturen, nämlich ResNet18, ResNet50, MobileNetV3, EfficientNetV2 und eine eigene Architektur, werden als Feature-Extractor für das Regressionsmodell verwendet und evaluiert. Für das Training der Modelle wird ein Datensatz erstellt, welcher die Position und Rotation der einzelnen Stangen enthält. Die Daten der schwarzen, automatisierten Figuren werden direkt aus den Motoren ausgelesen. Im Gegensatz dazu müssen die Daten für die weißen Figuren manuell erhoben werden. Dazu wird die Rotation mit Beschleunigungssensoren gemessen, während die Position auf Basis des Bildmaterials berechnet wird. Das System zur Erkennung des Spielzustands wird als Prototyp entwickelt und evaluiert. Es wird gezeigt, dass der Prototyp eine hohe Genauigkeit bei der Vorhersage der Position und Rotation der Figuren erreicht, welche den Anforderungen entspricht. Speziell die ResNet-basierten Modelle erreichen vielversprechende Ergebnisse. Die geforderte Vorhersagegeschwindigkeit von 60 Frames pro Sekunde konnte das System aufgrund hoher Inferenzzeiten und sequentieller Ausführung jedoch nicht erreichen. Am Ende dieser Arbeit wird ein Überblick über mögliche Lösungen für dieses Problem präsentiert.

Abstract

The game of Football poses well-known challenges in AI research. A smaller parameter space abstraction can be found in the game of Foosball. In a semi-automated Foosball game, one team is controlled using industrial motors through an AI agent while the other team is played by humans. For dynamic gameplay, the AI agent requires access to as much data as possible, namely the real-time game state. In a Foosball game, the game state is defined by the position and rotation of the individual figures as well as the position of the ball. In this work, a concept for a real-time game state detection system based on Computer Vision and Convolutional Neural Networks is presented. The system is tested, trained and evaluated on a real-world semi-automatic Foosball table setup. An end-to-end image regression model is employed to predict the position and rotation of the individual rods. Five different architectures, namely ResNet18, ResNet50, MobileNetV3, EfficientNetV2 and a custom architecture, are utilized and evaluated as feature extractors for the regression model. For the training of the models, a custom dataset is created containing the position and rotation of the individual rods. The data for the black, automated figures is directly received from the motors. The rotation of the white, human played figures is measured by using accelerometer sensors while the position is calculated based on image data. The game state detection system is implemented as a proof-of-concept. It is demonstrated that the prototype achieves high accuracy in predicting the position and rotation of the figures, meeting the requirements. Especially the ResNet-based models show promising results. However, the system could not achieve the required prediction speed of 60 frames per seconds due to high inference times and sequential execution. An overview over possible solutions to this is presented at the end of this work.

Contents

List of Figures

List of Tables

List of Acronyms

ADC	Analog Digital Converter
AI	Artificial Intelligence
CNN	Convolutional Neural Network
CV	Computer Vision
DOF	Degrees of Freedom
DRL	Deep Reinforcement Learning
FPS	Frames per Second
MCU	Microcontroller Unit
MPS	Metal Performance Shader
MRI	Magnetic Resonance Imaging
MSE	Mean Squared Error
OCR	Optical Character Recognition
OPC UA	OPC Unified Architecture
PLC	Programmable Logic Controller
ROI	Region of Interest

Part I

Thesis

Chapter 1

Introduction

The usage of Artificial Intelligence (AI) and Deep Reinforcement Learning (DRL) for the automation of board games like Go [53] or computer games like Dota 2 [46] enables beating high level players or even world champions. Using a game is a good example for Deep Reinforcement Learning (DRL) approaches as it provides a defined set of rules and actions. Therefore, a game provides safe and reproducible environments for the development and testing of new algorithms. The transfer from a game environment into the real world is another field of research. The usage of robotics and AI in the well-known RoboCup challenge [37] enables the combination of a defined set of rules and real world challenges.

The training process of DRL agents which should control robots is (1) compute intensive and (2) needs to be trained on the actual robots. To circumvent this, the usage of simulations for the training and a transfer of the trained agent into the real world robot enables training in a safe space [39]. While a simulation needs to follow the physical constraints of the real world, it cannot cover all possible events and therefore leaves a sim-to-real gap which must be addressed. De Blasi et al. [19] introduced a DRL system for the automation of a Foosball table. In contrast to Football, the game of Foosball provides less variables as the game state can be defined by only using the position and rotation of the figures and the position of the ball. They also used a simulation for the training and adapted the agents to the real world.

The sim-to-real gap can include physical constraints which are not implemented in the simulation but can also include the absence of data in the real world. While the Foosball table is automated on one side, the other side is played by humans. The table communicates the game state of the automated side but not of the human side. In contrast, this data is available in the simulation. A DRL agent which is trained inside the simulation using this data could learn strategies based on the opponents moves which are not applicable in the real world as this data is not available. Another point is the quality of the available data in the real world. While the game state is highly accurate in the simulation, the measurements in the real world can potentially introduce some error margins. Furthermore, the agent should have as much information as possible available in the training process. Therefore, the detection of the game state in the real world enables the DRL agent to use this data for its decision making. In this work, a game state detection system for the Foosball table is designed and presented.

1.1 Introduction to the KIcker Project

The semiautomatic Foosball table which is used in this work (cf. Figure 1) was originally build by Bosch Rexroth and presented by De Blasi et al. [19]. They used an Ullrich Sport U4P table with black and white teams and four rods per team. The black rods are automated using IndraDyn S MS2N03 synchronous motors for the rotation and IndraDyn L MCL020 motors for the translation of the rods. The motors are driven by IndraDrive Cs drive converters which are connected to the Programmable Logic Controllers (PLCs) IndraControl XM21 and XM22. The PLCs are connected via an Ethernet interface and can communicate using the OPC Unified Architecture (OPC UA) protocol [44], an industrial communiction standard for secure data exchange between different devices in industrial automation. A major benefit of OPC UA is its platform and programming language independency.

A Logitech Brio 4K camera is mounted above the table and enables a top-down view. The camera is able to capture 1080p (1920 ×1080 px) in 60 Frames per Second (FPS). The motor drives are utilized as sensors and report their rotation or lateral position through OPC UA. For safety reasons a miniTwin4 light curtain by Sick is added above the playing field which stops all rod movements if the light grid is interrupted, e.g. by a human hand or another object getting too close to the playing field.

Several studies researched the automation of the Foosball table using DRL. De Blasi et al. [19] aimed for the automation of the striker rod. The DRL system was trained in a simulation to score goals. The ball was manually placed on pre-defined positions. Their aim was to bridge the reality gap of such a system which arises from unavoidable deviations in the real world which cannot be properly addressed in a simulation. The steps for bridging this gap are (1) ensuring that the simulation is as close to the real world as possible and (2) Applying randomization to the parts of the simulated physical system which suffer from imprecision. In contrast, Rohrer et al. [51] aimed their research at the automation of the goalkeeper using the same physical table and simulation. While the camera would be able to capture at higher frame rates, the authors only captured the game state in 30 FPS. Therefore, the position of the ball is calculated approximately every 33 ms. Using two consecutive detected ball positions, the direction of the ball is calculated and the DRL system is trained to move the goalkeeper accordingly to block the shot. Gashi et al. [25] aimed also at the automation of the goalkeeper but additionally included the opponents striker rod using multi-agent DRL. The agent for the striker rod was trained to score goals similar to De Blasi et al. [19] while the goalkeeper was trained to prevent the scoring of a goal similar to the approach of Rohrer et al. [51]. Gashi et al. solely trained the system inside the simulation since the white opponent is not automated on the physical table. As the position and rotation data of the white figures is available inside the simulation, the DRL agents can use this information to predict their actions. In the real world, this data is not available. The authors note the possibility of imitation learning in the future which would require a game state detection system to capture a Foosball game between two human players. Using this captured game state, the DRL agent could learn actual human

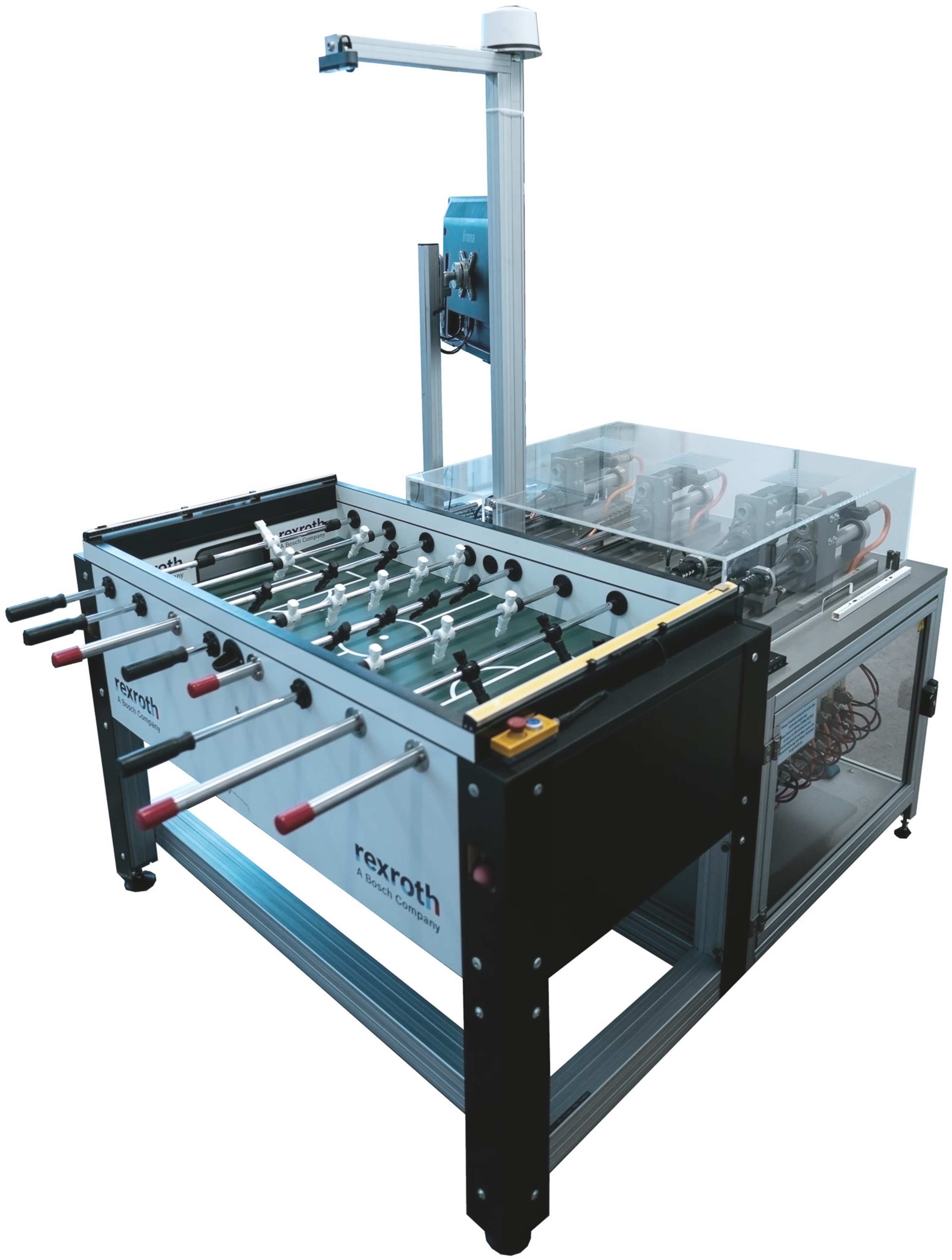


Figure 1: The semi-automated Foosball table used in this work.

playing strategies and adapt those in the own action prediction.

1.2 Objectives

The goal of this work is a concept of a game state detection system for a Foosball table which can detect the current game state in real-time to enable DRL approaches in the automation which can act on the opponents actions and therefore enable a true dynamic Foosball game. Additionally, the game state detection system could be used to capture a human-played game and enable an imitation learning approach for the automated agent as proposed by Gashi et al. [25]. In the following section, a short definition of the term *real-time* is presented followed by a description of the defined requirements to reach the overall goal.

The Definition of Real-Time

Most studies in the field of real-time computer vision or computer graphic systems lack a clear definition of the term *real-time*. The Oxford Dictionary of Computer Science [16] defines a real-time system as follows:

"Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness." [16]

Using this definition, it is clear to say that the term *real-time* cannot be defined without a given context and without defining the *acceptable timeliness* in this context. In computer graphics, the acceptable timeliness is usually defined in FPS. In general, the acceptable amount of FPS is defined by the ability of the human eye to detect differences between two images. While humans can detect flicker artifacts of up to 500 Hz, the rate at which a stable image without flickering can be seen is much lower at around 50-90 Hz [18]. Using this rate, Akenine-Moller et al. [11] define an upper limit for real-time rendering at 72 FPS. Their main definition of real-time is based on the ability of the user to focus on action and reaction. Therefore, a system can be considered real-time at as low as 15 FPS.

In the Computer Vision (CV) context, this definition cannot be used directly since a CV system often depends on other systems like a camera. Otterness et al. [48] define real-time as 30 FPS since most cameras support the capturing of videos at this frame rate. Pulli et al. [50] report a response time of 30-40 ms as real-time which corresponds to around 25-30 FPS. In other studies [30,54], real-time is defined between 24-60 FPS. In the previous work by Horst et al. [31], real-time was considered to be 60 FPS which is the capturing frame rate of the used camera. This definition is also used in this thesis as the same physical hardware is used. Therefore, a resulting acceptable response time is around 16.6 ms.

Another important aspect of real-time is the latency of a system. If the processing time is below 16.6 ms and the system is therefore able to process input at 60 FPS it would be considered real-time. If this system introduces some static latency, e.g. for networking or other communication, it would still meet the 60 FPS requirement but cannot be considered real-time, as the processing lags behind the actual time. Floridi [23] focuses his real-time definition solely on the latency aspect and describes a system with lower than 100 ms latency as real-time. Additionally, he described real-time as “something that is simulated, represented, communicated, interacted with, shown etc., at the same time or at the same rate as it happens” [23]. This more general approach to defining real-time adapts better to reality, as the real-time requirement is always depending on the use case. Claypool [17] researched the influence of latency in computer gaming and specially in real-time strategy games. His studies showed that added latency does not impact the result of the games indicating that real-time is clearly not as tight defined as it is in other games. Janssen et al. [34] described a maximum velocity of a ball in a Foosball game to be 10 m/s or 1 cm/ms. Considering the size of the ball with a diameter of around 35 mm it needs 3.5 ms to completely leave its current position. Therefore, a system latency of 10 ms would result in a high chance of missing the ball at maximum velocity. Considering the much lower average velocity in a Foosball game, a latency of 10 ms can be good enough.

Requirements

The overall goal of the work on hand is a system which can detect the game state of a Foosball game in real-time. As already mentioned, real-time is defined per use case and lacks a generally valid definition. Considering that the DRL algorithm, which should use the detected game state for the automation of the black rods, and the actual control of the rods through the motors and the PLCs also introduce some latency, the overall reaction time of the computer controlled rods can only be as fast as the whole system starting with the image capturing, game state detection, action prediction and ending with the actual movement. Therefore, the latency of the game state detection should be as low as possible. Additionally, the utilized webcam can only record 60 FPS at 1080p which is the desired image resolution. Albeit the documentation of the webcam stating that the camera is able to record 90 FPS at a 720p resolution this was not reproducable during the implementation of the game state detection system. Therefore, the real-time requirement is met if the system is able to achieve 60 FPS and does introduce a maximum static latency of 10 ms.

Furthermore, the system should be able to detect the game state which is defined as the position and rotation of all figures and the detection of the ball. In the previous work, the ball is already detected by De Blasi et al. [19]. A system for the detection of the white, human played figures is conceived by Horst & Hagens et al. [31]. While the position of all white figures could be detected, the rotation detection was only implemented for the white midfield figures. This approach should be extended and verified to the other white rods and to the black rods. Therefore, the game state detection system must be able to detect the position and the rotation of the black and white figures.

As the DRL system should use the detected game state as input features, the game state must be detected with a high accuracy. Overall, the position of the figures is more important than the rotation as the rotation has a much higher range in which the ball is stopped. For an easier estimation of the rotation range it is assumed that the figure does not move when hit by the ball. While this is the case on the black figures as the motors are stronger than the balls impact, the white figures will rotate a few degrees on impact. Therefore, an offset of 5 degrees is deducted from the rotation range. In Figure 2 the rotation range is illustrated. Considering the length of a figure between the center of the rod and the end of the feet which is 72 mm, the distance between the playing field and the end of the feet of 12 mm and the diameter of the ball which is 35 mm, the angle α is approximately 47 degrees. Deducting the 5 degree offset, a resulting rotation range of ± 42 degrees or 23 % of the overall rotation range of the white figures is calculated.

To perform a straight shot, the ball must be hit in the center. Considering the width of the feet of 22 mm, a position range of ± 11 mm is defined. It is noted that a small deviation from the center would probably still result in a straight-enough shot. The positional limits of the rods vary between ± 55 mm on the midfield rods and ± 180 mm on the defender rod. Therefore, the position range would be approximately 20 % of the midfield limits and 8.5 % of the defender rod. Since a player would want to block a ball, it can be assumed that the figures are standing upright most of the time while they are not always in the center of the ball. Therefore, an accurate position detection is more important than an accurate rotation detection. The requirement for accurate detection is met if the average error is inside the presented ranges.

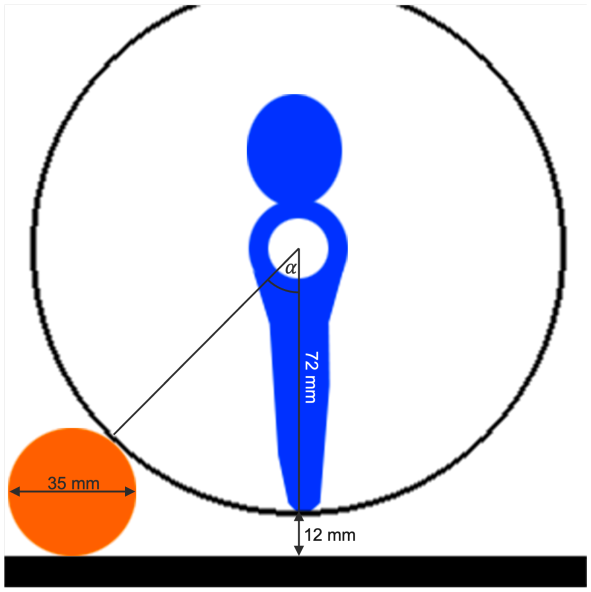


Figure 2: Rotation range of a figure which would stop the ball.

The system should be used by future students to enhance the automation of the Foosball table using DRL. Therefore, the hardware requirements should be low, and the system should be runnable on commodity hardware and laptops. Furthermore, hardware modifications on the Foosball table itself are not possible.

1.3 Structure of this Thesis

First, a short introduction to Convolutional Neural Networks (CNNs) and the network architectures ResNet, MobileNet and EfficientNet is presented. Following this, an overview of related and previous work describing the state-of-research on game state detection in Foosball games is given. Following this, a concept is presented to capture and detect the game state of such a Foosball table in real-time, which is implemented as a prototype in the consecutive chapter. Afterwards, an evaluation of the different network architectures and the overall system is presented including a discussion of results. The thesis ends with a conclusion and a brief overview of future research fields based on the presented concept.

Chapter 2

Convolutional Neural Networks

The first predecessor of Convolutional Neural Networks (CNNs) is considered to be the Neocognitron in 1980 [24]. The groundbreaking work was LeNet, a CNN to classify handwritten numbers, by LeCun et al. [41] in 1989. Albeit the early work and due to the lack of computing power, successful CNN models were only available after the work by Krizhevsky et al. [38] in 2012. Since then, CNNs are widely used in image-based machine learning problems like object recognition, image classification or text recognition [cf. 12,13,40,42,66].

In contrast to traditional neural networks which utilize hidden layers with fully connected neurons, the neurons in a convolutional layer are only connected to a subset of neruons from the previous layer. This behavior enables an implicit learning of features. Each convolutional layer consists of a set of spatial feature maps with learnable kernels. Neurons of the same feature map share the same weight parameters which reduces the number of learnable parameters and therefore the complexity of the whole network. In a basic CNN architecture, each convolutional layer is followed by a pooling layer which reduces the dimension of the feature maps which thereby also reduces the complexity and the number of parameters. After extracting the features through convolutional and pooling layers, at least one fully connected layer such as in traditional neural networks is used as a loss layer where the error between the desired and actual output is calculated [cf. 13].

In the work on hand, the following three well-known CNN architectures are used:

- ResNet [28];
- MobileNetV3 [33]; and
- EfficientNetV2 [59].

2.1 ResNet

He et al. [28] introduced the usage of residual blocks in deep CNNs. A residual block, also called skip-connection block, learns a residual function which references the input of the block. While a traditional neural network would learn the direct mapping between the input and the ouput, a residual block learns the difference between the input and the output. If the input is denoted as x and the desired output is $H(x)$, a traditional network would directly try to optimize $H(x)$. In contrast, the residual network would optimize another mapping $F(x)$ with $F(x) := H(x) - x$. Therefore, the network only learns the difference between the input (x) and the desired output ($H(x)$). The actual output is calculated by transposing the formula to $H(x) = F(x) + x$ as illustrated in Figure 3a. The authors describe the residual mapping as easier to optimize than the unreferenced mapping in a traditional neural network. Additionally, the utilization of the original input inside the network reduces the vanishing gradients problem.

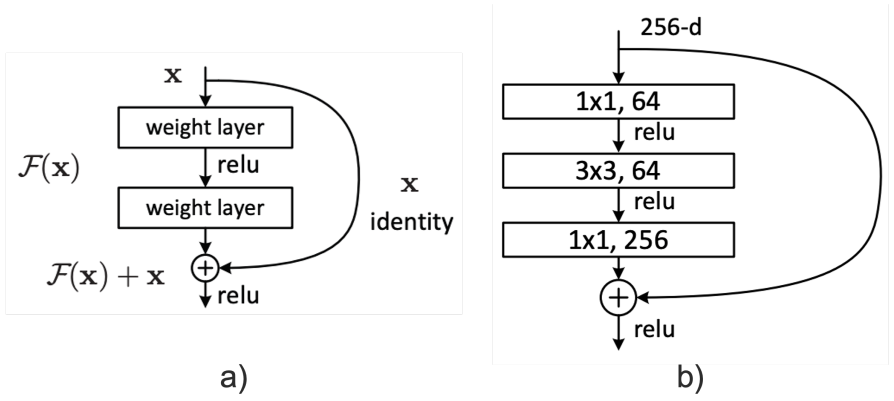


Figure 3: Illustration of a residual block and a bottleneck residual block [28].

The residual blocks are utilized by He et al. [28] to create the ResNet architecture. The authors define a residual layer as multiple residual blocks with multiple convolutional layers with the same output size. Each residual block uses a skip-connection between the blocks input and the output. These residual layers are utitlized to create different sized ResNet models. Each model starts with one convolutional layer using a 7×7 kernel and a stride of two thus dividing the input size in half. Following this are four residual layers finishing with one fully connected classification layer. At the beginning of each residual layer, the input size is halved again. In case of ResNet18, each residual layer contains two residual blocks with two convolutional layers per block, resulting in 18 layers in total. The larger ResNet50 model uses a total of 50 layers but introduces a different residual block design called *bottleneck residual block*. This block uses three convolutional layers as shown in Figure 3b. Starting with a 1×1 convolutional layer which reduces the input dimension of the following 3×3 layer and ending with an additional 1×1 layer to restore the original output dimensions. This technique leads to more efficient and deeper networks. ResNet50 uses three of those bottleneck residual blocks in the first residual layer, four blocks in the second, six blocks in the third and three blocks in the last residual layer. He et al. report a inferior performance of the residual based networks in contrast to similar architectures without skip-connections.

2.2 MobileNetV3

The aim of the MobileNet series is the creation of a deep CNN tuned to mobile and embedded devices. MobileNetV1 by Howard et al. [32] introduced the usage of depthwhise separable convolutions enabling the creation of lightweight deep CNNs suitable for mobile and embedded devices with limited resources. In contrast to traditional convolutions, the depthwhise separable convolutions are applied separately over each channel thus lowering the resource requirements. Sandler et al. [52] extended this design in MobileNetV2 by introducing inverted residual blocks. The skip-connections are between bottleneck layers to reduce the complexity of the network. Howard et al. [33] created MobileNetV3 which is especially tuned to mobile phone CPUs by utilizing hardware-aware network architecture search. Additionally, the NetAdapt [65] algorithm was utilizied to shrink the network. NetAdapt shrinks existing pretrained networks based on resource budgets. In contrast to other network simplification algorithms which often optimize indirect metrics like the number of parameters, NetAdapt optimizes for the direct metrics of latency and energy consumption. Additionally, Howard et al. created a new efficient network design and used new efficient versions of nonlinearities. The authors presented two different Versions of MobileNetV3, MobileNetV3-Large and MobileNetV3-Small. The small version aims for low latency on slower hardware while the large version uses a deeper network architecture and is suitable for higher performing hardware.

While latency was also an important parameter in the creation of ResNet by He et al. [28], MobileNet prioritizes the computational efficiency. Therefore, MobileNet architectures tend to have a smaller number of parameters than ResNet. Through the adaption of inverted residual blocks in MobileNetV2, MobileNetV3 is build on comparable techniques as introduced by He et al. [28].

2.3 EfficientNetV2

In 2020, Tan et al. [58] introduced the first iteration of EfficientNet, a CNN architecture and network scaling method. In contrast to previous network scaling by scaling the depth, width and resolution of the network independently, EfficientNet uses a compound coefficient to scale these factors uniformly. The underlying intuition that a network needs to be deeper if the input image is bigger justifies the uniform scaling. The network is based on the inverted bottleneck residual blocks of MobileNetV2 [52]. Later, Tan et al. [59] advanced this architecture to EfficientNetV2 achieving faster training speed and better paramter efficiency by combining neural architecure search and scaling to optimize training speed.

In contrast to ResNet [28], the scaling strategy of EfficientNetV2 results in achieving better accuracy with fewer parameters and enables its usage in resource constrained environments. Albeit achieving high accuracy on the ImageNet dataset [20], the scaling generally prioritizes efficiency. Similar to MobileNetV3 [33] and ResNet [28], EfficientNetV2 is available in small and large versions with different depth and therefore resource requirements.

Vdovjak et al. [60] conducted a comparison of ResNet, MobileNet and EfficientNet by creating a fire detection classifier based on the different networks. Their comparison was partitioned into multiple model tiers based on model size. In their findings, The ResNet models performed equally or worse than the MobileNet and EfficientNet classifiers. Especially the large ResNet with 101 layers performed the worst of all compared models. In contrast, the smaller ResNet models with 18, 34 or 50 layers performance was closer to the results of MobileNet and EfficientNet. The authors describe the ResNet architecture as outdated as the newer network architectures achieve the same or better results with less computational power needed. Albeit this conclusion, all models required approximately the same inference time independent to the model size with 5 to 6.5 ms. A correlation between the number of trainable parameters and the inference time is not visible which is clearly indicated by the largest ResNet101 model needing the same inference time as the smallest ResNet18 model. Therefore, the higher computational requirements of ResNet do not directly influence the inference time in the presented use case. It is noted that the authors used high performing hardware, namely an AMD Ryzen 9 5900X processor with 64 GB RAM and an NVIDIA RTX 3080 GPU which is similar to the hardware used in this work.

Chapter 3

Related and Previous Work

3.1 Related Work

Several studies were conducted around the creation of a semi-automatic Foosball table. A key part of the automation is the subprocess of game state detection. Most studies address both fields [27,29,62] while some address only the state detection [14,31]. Studies which only address the automation without mentioning the game state detection usually extend an already existing automated Foosball table and use the established state detection methods [25,51,67].

The automation hardware is usually built around linear and rotary motors to control the rods with additional sensors to measure the position and rotation, cf. [19,34,45,62]. While the controlling of those motors in older approaches was implemented using rule based algorithms [62,64] the research shifted towards DRL based machine learning models [19,25,51]. In between those, Zhang et al. [67] used imitation learning methods to improve the rule based agents by Weigel et al. [62]. Regardless of the approach to automating a Foosball table, the game state detection or at least the detection of the ball is needed for a reactive system.

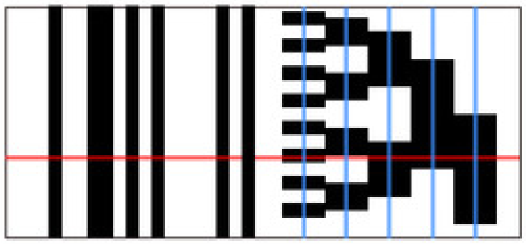
The game state of a Foosball Table is generally defined as:

- The position of the ball; and
- The position and rotation of the figures.

Additionally, the velocity and direction of the ball is a key part. Those values can be calculated by capturing multiple game states over a short time. Therefore, the real-time detection of the game state is important. The position of the figures is addressed in some studies while the rotation of the figures is mostly omitted or only rudamentary addressed. In the most cases, the game state is retrieved via a CV system consisting of a camera to capture the playing field and traditional CV algorithms to retrieve the game state.

The only commercially available automated Foosball table, *StarKick*, was developed by Weigel et al. [62,64]. To this date, the commercial version was stopped and no further development seems to be planned. In the early non-commercial version, called *KiRo*, they used a top down camera to capture the game state including the individual figures. Their Foosball table had the player figures in red and blue colors with a yellow ball. Therefore, the figures and the ball could be detected using classical color segmentation. Since the position of the individual figures on each rod is known, this *a priori* knowledge can be used to average out possible segmentation errors. The authors calculate the rotation angle of the figures using the bounding box width and a measured minimum and maximum value. Since only the width of the bounding box is used, a distinction between a left or right / up or down rotated figure is not possible. Therefore and because of limited image quality they only detect the figures as an *up* and *down* position without extracting the actual angle.

In later versions [61,63] they switched to a bottom up camera which is mounted inside of the Foosball table. They used a one-sided transparent playing field, so the field appears to be green for an outside player while it is transparent for a portion of the infrared light spectrum. Using additional mounted infrared LEDs to illuminate the field, the camera could get an image of the playing field without any figures or other objects obstructing the view of the ball. Due to the new camera setup, the figures could no longer be detected and the resulting game state only included the position of the ball.


(a) Marker pattern graphics with illustrated L_m line data (red) and bit spaces sampling points (intersection of blue and red lines)

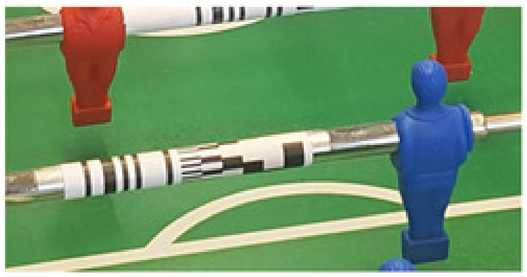

(b) Marker, as installed on the playing rod

Figure 4: Visual marker pattern used to detect a rods rotation [15].

Bošnak et al. [15] proposed a method for the detection of the rotation of a rod by using a camera and visual marker patterns on the rods. Figure 4a shows the used pattern. For positioning reasons, each pattern begins with a barcode on the left side followed by a 5 bit gray code pattern. Each rotation angle is encoded in this pattern corresponding to 32 distinct values and a usable bucket width of $\frac{360}{32} \approx 11$ degrees. The authors proposed an interpolation method to increase the accuracy of the measurement resulting in an angular resolution of 0.7 degrees. Figure 4b shows the pattern in use on a rod on the Foosball table.

Mohebi [45] discussed several approaches for the game state detection of a semi-automatic Foosball table. To detect the position of the rods, linear positioning sensors were proposed. Those include inductive linear sensors and hall effect based sensors using the change in magnetic fields. Other mentioned sensor types include resistance-based, optical or capacitive positioning sensors. Another approach is the algoritmic calculation of the position and rotation of the motor-controlled rods. This has a major disadvantage in the accumulation of small errors over the span of a game to large errors in the assumed position and rotation, creating the need for some error correction. Another approach for the rotation detection is the usage of potentiometer sensors by attaching the shaft of the potentiometer to the rod. The rotation can be calculated by measuring the resistance of the potentiometer. The author points out that several safety features need to be applied to prevent permanent damage on the sensors. Additionally, the use of potentiometers can increase the drag of a rod. A third option includes the usage of a CV system for the player detection utilizing a top down camera mounted above the Foosball table. This system would use a similar rotation detection as described by Bošnak et al. [15].

Mohebi [45] also proposed several strategies for the tracking of the ball:
First, a touch screen can be used for the ball detection. While several touch screen technologies are compared, an infrared based touchscreen is the only available option due to the light, non-conductive ball with no electrical ground connection during the game. Thus, capacitive and resistive touch screens as used in mobile phones cannot be used for the ball detection.
The second discussed option is the tracking of the ball via a Bluetooth transmitter inside the ball and multiple Bluetooth antennas around the playing field. The position can be calculated by measuring the time difference between the antennas. While theoretically working, this method is not usable in a semi-automatic Foosball table due to the following reasons. The data processing needs to be very fast, considering the time difference between one antenna with 9000 mm distance and a second antenna with 300 mm distance to the ball is only about 2.9×10^{-9} s. Furthermore, the position of the sender inside the ball and interference with other Bluetooth enabled devices can introduce errors in the position detection. The Bluetooth transmitter would also need a battery inside the ball which must be changeable. The author notes that the game could be impacted if the mass of the ball is not uniformly distributed. Therefore, the Bluetooth transmitter and battery must be precisely positioned with almost no margin for errors.
The third option is the use of a grid of light emitters and detectors. The main advantages of this system are the flexibility as a higher resolution can be added afterwards by increasing the number of emitters and detectors and the easy handling of malfunctions as only the broken parts need to be replaced while the rest of the system is not impacted. The addressed disadvantages include the requirement of programming for the ball detection and the limitations in detection accuracy due to the space needed between two detectors to not interfere with each other.
The last option is the usage of a computer vision system like already proposed for the figure detection. In addition to the top-down camera which has the disadvantage of possible hiding of the ball under the figures, a bottom up camera under the Foosball table is addressed. Similar to the ball tracking approach in the second iteration by Weigel et al. [61], the detection of the figures is not possible with the bottom up camera setup. Additionally, the author notes possible problems in the detection if light sources are directly above the Foosball table.

In contrast to the other presented authors, Bambach et al. [14] only researched the CV based state detection of a standard Foosball game with no automation involved. While the real-time detection was a key part, the authors used captured videos instead of live footage. Their approach starts by finding the rods of the Foosball table. Since this step is compute heavy and the rods themselves do not move over the span of a game, the rods only need to be found once at the start. The rods are found using *a priori* knowledge about the table and the Hough transform line detection algorithm [10]. Since the camera orientation is known, only detecting horizontal lines is sufficient. Lines in the center part and at the end of the image can also be ignored to suppress the detection of the background lines on the playing field and the walls of the table. Afterwards, the eight highest peaks in the Hough space are detected as the rods. To accomodate camera distortions, multiple lines are tested for the difference of each pixel with the expected gray value. The line with the least median difference is defined as the center line of each rod. To detect the individual figures, the negative mean distance between the pixel colors and the known player color for each row is calculated in a 10 pixel wide vertical strip around the calibrated center. Resulting is a likelihood distribution for the containing of a part of the figure for each row. After smoothing this distribution, the resulting peaks are the center locations of each figure. The rotation of each rod is calculated by searching the endpoints of the individual figures using a similar color likelihood distribution. Using the length between the endpoints, the rotation angle is computed using predefined min and max values between -90 and 90 degrees. Therefore, the rotation of the rods does not accommodate for overhead states where the feet of the figures are above the horizontal line. The detection of the ball was implemented by masking the background and utilizing a template matching algorithm supported by a Kalman filter. The background is defined as everything outside a specific color range around the red ball. The following template matching is supported by predicting the next location of the ball using a Kalman filter and searching for the template of the ball inside a 40×40 px square on the predicted location. Lastly, occlusions of the ball are detected by a linear thresholding function based on the velocity of the ball. If an occlusion is detected, the balls state is not updated for the specific frame. As no ground truth was present, the approach was not evaluated quantitatively but the authors describe the results as “fairly good with some noise due to the video quality” [14].

Janssen et al. [34] proposed a ball tracking system using CV components for a semi-automatic Foosball table build by the University of Eindhoven. Their setup included a top down camera which could only capture monochrome images. In contrast to other research, the ball is white thus simple color segmentation could not be used as the figures were yellow but appeared white through the monochrome camera. The authors note that even advanced circle detection algorithms were not applicable in the setup of a Foosball table since the ball is often hidden below a figure or rod and other objects, like circles in the background, could be misinterpreted as the ball. Their approach for tracking the ball starts by defining a Region of Interest (ROI) of 100×60 px. At a maximum observed velocity of 10 m/s of the ball, the minimum framerate needed for the defined ROI is reported as 97.3 Hz while other research [14,21,31] report lower minimum frame rates. To reduce the possibility of errors during the ball detection, static background objects are removed by subtracting the background. Additionally, the yellow figures need to be masked as their color is not differentiable from the ball in the grayscale image. The detection of the figures is done using a Magnetic Resonance Imaging (MRI) scanner to create a 3D scan of the figures and converting their position to the cameras pixel space using the known rotation angle as the yellow figures were connected to motors. The dark figures of the other team are simply removed by thresholding. Therefore, albeit detecting the yellow figures, this information is not further used. Since everything left in the frame is either black or the ball, the position can be detected by finding any bright pixel.

In a further development [35], they replaced the monochrome camera by a colored one. As the white ball and the yellow figures are now distinguishable, the ball can be detected using color segmentation algorithms. Therefore, the MRI scanner was no longer neccessary but can improve the performance of the color segmentation. To accommodate different lighting situations which could result in non-ball and ball pixels having a similar color, a calibration tool was implemented. In this approach, the detection of the figures was not considered.

The California Polytechnic State University conducted several experiments for the automation of a Foosball table. Gutierrez-Franco et al. [27] initialised the project introducing two different aproaches for the detection of the ball. The first approach is a CV system using color segmentation with a distinguished colored ball or a detection based on the ball's shape. They note a major drawback in this system in the occlusion of the ball by the figures and the rods while additionally noting the need for a high processing power to process the individual photos. As a second approach, the authors present a laser-grid based system similar to the light grid system by Mohebi [45]. The main drawback of this system is the accuracy of the ball detection, which is directly related to the amount of lasers used to detect the ball which cannot be infinitly increased due to interfering between the lasers if placed to close to each other. It is also noted that the system will be unable to detect the ball if it bounces off the playing field. Overall they decided to use the vision system as the laser grid would introduce the need for adjustments to the Foosball table itself in contrast to almost no adjusments needed for the usage of a camera. Additionally, the vision system can deliver a higher resolution.

In the next iteration, Stefani et al. [55] enhanced the vision system by adding additional lighting to the Foosball table to prevent shadows and reflections. They note the need for further development of the vision system as the existing solution could only deliver around 11 FPS. Considering the maxium velocity of the ball at 10 m/s as reported by Janssen et al. [34], this would result in the ball moving potentially by around 90 cm per frame. Otherwhise, the paper focused more on the actual hardware development and mechanical engineering of the Foosball table.

Another semi-automatic Foosball table was built by the University of Central Florida by Enos et al. [21]. In contrast to Janssen et al. [34], they reported a maximum ball velocity of only 5 m/s and define 30 FPS as sufficient as the ball would need ~30 ms to travel between two rods. Since the defense rod is between the goalkeeper and the striker of the opponent, the computer will have at least ~60 ms to block a goal shot of the opponent. Therefore, the 30 FPS requirement would result in at least two samples of the ball which is sufficient to detect the direction and velocity. The authors note that a higher frame rate would be desirable but not achievable considering their budget constraints. Additionally, they note the importance of a short exposure time (or fast shutter speed respectively) to prevent motion blur. While noting that the detection of the opponents figures can be beneficial for the automation process, the authors do not include detection for them as it can be assumed that a shot will be blocked.

Aeberhard et al. [9] from the Georgia Institute of Technology also researched a semi-automatic Foosball table. While the authors focus mostly on the hardware of the Foosball table, they describe a way for detecting the game state using a webcam for the detection of the ball and the human played figures. Their approach relies solely on color matching with a calibration step in which the user needs to select the color of the background, the figures and the ball. The figures are detected by finding the first similar colored pixel as the calibrated figure color. Then, the center of mass near that pixel is calculated concluding in the calculation of the distances between individual figures on the found rod. The ball is initially found by color comparisons on the whole table and with a 40×40 px ROI centered around the last known location afterwards. It should be noted that the used camera only had a resolution of 320×240 px, so the ROI is a relatively large field. If the ball is not found in this ROI, the whole table is rescanned and, if the ball is not found in 10 consecutive frames, a goal is registered. Each frame, the positions of the human controlled figures is recalculated by scanning the pixels of the calibrated rod until the top figure is found. As the distance of the figures is already known, the other figures can be calculated without further image processing. The rotation of the figures was not addressed.

3.2 Previous Work

The work on hand uses the same physical Foosball table as [19,25,31,51]. De Blasi et al. [19] used the top-down webcam mounted above the table to detect the position of the ball using a standard color-based detection algorithm. As their focus lay on reinforcement learning and sim-to-real transfer the detection of the human played figures was not addressed. However, the authors note the possibility of detecting the figures in the future. The current implementation of the Foosball table uses a similar approach but added a ROI for performance reasons [4].

In a previous study of Horst & Hagens et al. [31] which is included in Appendix B, the figure detection for the white, human playable figures was addressed. They implemented a proof of concept which can detect all white figures and predict the rotation angle of the white midfield figures. Instead of predicting the position of the whole rod, each individual figure was detected using the visual object detection network YOLOX [26]. The rotation was afterwards predicted for each individual figure of the midfield rod by a modified ResNet18 [28] network. The final rotation of the midfield rod was calculated as the mean rotation of all five predicted figures.

3.3 Own Contribution

Fundamentally, this work is based on the previous work of Horst & Hagens et al. [31]. While their approach showed promising results, the rotation detection is limited to the white midfield figures. In this work, the rotation should be detected on all figures (including the black, motor-controlled figures). Furhtermore, a trustworthy ground truth should be established as the previous training data contained some faulty data points. The main contributions can be summarized as:

- Verifying Accelerometers to establish a trustworthy ground truth;
- Creating a concept for the real-time detection of the position and rotation of the black figures and extending the previous concept for the white figures;
- Implementing a proof of concept to verify the approach;
- Evaluating different Feature Extractor neural networks; and
- Evaluating the implemented prototype.

As already stated in [31], the color based approaches, cf. [9,14,55,62], are not applicable since the figures are not colored with a distinguishable color. While the black of the motor-controlled figures is often confused with shadows in the goals or at the site of the table, cf. subsection 4.2.1, the white of the human opponents figures is not distinguishable as the walls and lines on the playing field are also white. The usage of hardware sensors as described by Mohebi [45] looks promising but is also not applicable on the Foosball table on hand due to a non-permanent modification constraint. Additional hardware sensors in the form of accelerometers are only utilized during the creation of training data. In contrast to the classical CV based approaches, cf. [15,27,29,61], state of the art deep CNNs are utilized for the game state detection thus removing the need for a calibration step before the game. The use of deep CNNs also introduces a high robustness against deviations in lighting or shadows which could occur in the camera image as the light in the room of the Foosball table is not a controlled environment.

Chapter 4

Concept

The game state of a Foosball table is defined by the position of the ball and the position and rotation of the individual figures. While the ball is already detected by De Blasi et al. [19], the position and rotation are only detected in a proof of concept by Horst & Hagens et al. [31]. In the prototype, only the midfield figures of the white team were detected. In this thesis, the prototype should be extendet to include all white figures. Additionally, the black figures should also be detectable. Furthermore, the objectives, as defined in section 1.2, should be achieved. Overall, a game state detection pipeline should be implemented with the camera image as input and the resulting game state data as output. This game state data will be used for reinforcement learning purposes to automate the Foosball table in the future. The following chapter describes a concept of a system which can achieve those objectives.

While the black rods are connected to motors which provide the rotation and position data, the white rods are not connected to any sensors. For the whole purpose of game state detection, it would be sufficient to only detect the white figures. To enable an end-to-end system, which could probably be used for imitation learning on other Foosball tables in the future, the black figures should also be detected using the camera image. The chapter is structured as followed. First, training data needs to be captured. Therefore, temporary sensors should measure the rotation of the white figures. Afterwards, a software should be implemented to capture the image and read the measurements by the motors and the temporary sensors. With the captured training data, a prediction model for the position and rotation should be trained. Two approaches will be presented. The first is a further development of the previous proof of concept, the second is a new approach using only one end-to-end regression model. To finish the concept, an output system is presented to provide the predicted game state for external systems.

4.1 Collection of Training data

The collection of the training data underlies the following requirements. The training data must consist of

- the rotation angle of the rods;
- the position of the rods;
- the camera frame; and
- a time indicator for synchronization.

Additionally, the dataset should be saved in a standard data format and should use widely used en- and decoders.

As described in section 1.1, the black figures are connected to motors and sensors which can measure the positional shift and the rotation angle. More problematic are the white, human playable figures which do not have any sensors connected to them.

In the previous work, Horst et al. [31] used accelerometers connected to a Microcontroller Unit (MCU) to measure the rotation of the white figures. The measurements were displayed on small displays connected to the same MCU which were visible by the camera. The measurements could be collected using the Tesseract OCR engine [7].

In this work, an optimized version of this approach was used. First, the displays were discarded in favor of direct readings from the MCU. Horst et al. [31] reported several issues with the character recognition. Two examples for those issues are shown in Figure 5. The cause for those errors was a missing synchronization between the camera and the displays resulting in display changes while the camera would take an image.

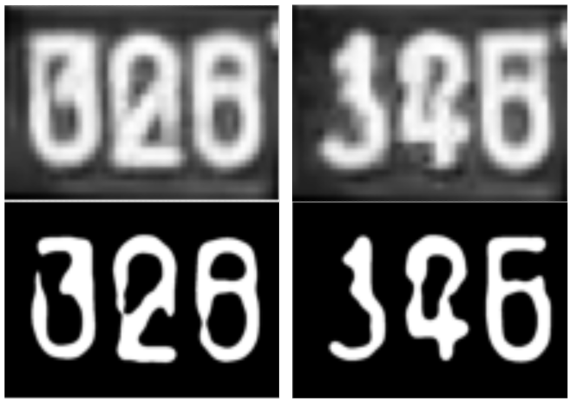


Figure 5: Faulty display images resulting in erroneous OCR readings [31].

Secondly the measurement hardware in form of the accelerometers should be evaluated to get an overview over possible miscalibrations, misreadings and deviations.

Thirdly, a software should be implemented to measure the sensors of the black figures, the accelerometers of the white figures, a time indicator and the camera image.

Those optimizations should result in a robust dataset with trustworthy ground truths. The desired dataset should be in an easily readable data format which does not require special decodings or non-standard format readers.

4.1.1 Calculating the rotation angle using Gravitational Acceleration

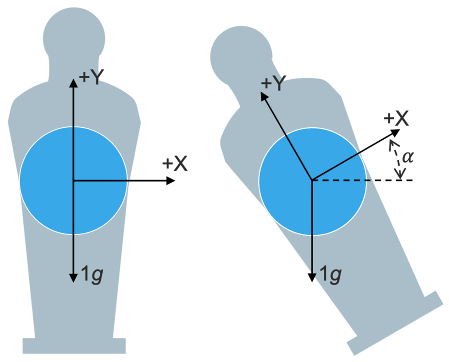


Figure 6: Measuring the tilt angle using two axis acceleration.

An accelerometer measures the linear acceleration along different defined axis. In this thesis, three-axis accelerometers with the orthogonal axis X, Y and Z are used. Since the gravitational force is a linear acceleration for a given point, the relative tilt of an accelerometer to the ground can be measured. Figure 6 shows the setup to measure the angle α using the X and Y axis and the gravitational acceleration. Since the individual rods are fixed to the frame of the Foosball table, the need for a third axis is eliminated as only a one-axis rotation is possible.

As Fisher [22] points out, the X-axis of the accelerometer is proportional to the sine of the rotation angle. Additionally, the Y-axis acceleration is proportional to the cosine of the angle due to the orthogonal alignment of the two axes, which is shown in Figure 7. Therefore, the ratio of the accelerations of X and Y can be defined as

$$\frac{A_{X,OUT}}{A_{Y,OUT}} = \frac{1g \times \sin(\alpha)}{1g \times \cos(\alpha)} = \tan(\alpha)$$

which results in

$$\alpha = \tan^{-1}(\frac{A_{X,OUT}}{A_{Y,OUT}})$$

with A as the measured acceleration on the specified axis. The resulting angle can be converted afterwards from radians to degrees.

Fisher [22] also points out that small inclines on the XZ or YZ planes do not introduce a relevant bias into the calculated angle. Therefore, a perfect leveling of the whole Foosball table, while necessary for the actual game, is not necessary for the measuring of the rotation of the rods.

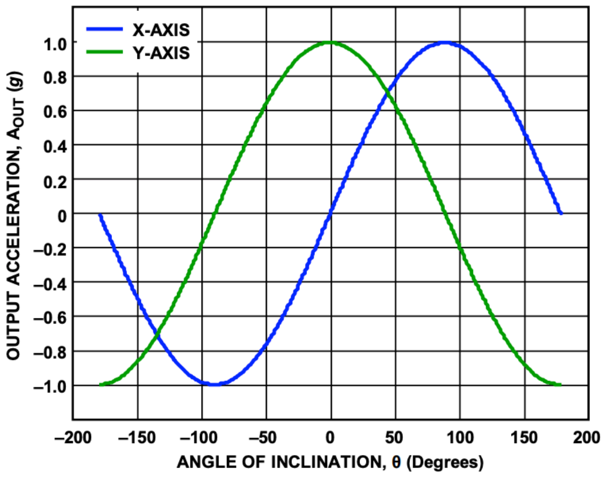


Figure 7: Output acceleration vs Angle of Inclination for Dual Axis Inclination Sensing [22].

4.1.2 Sensor Evaluation

Choosing an accelerometer

The accelerometers used to measure the rotation angle should:

- measure correct values with a low error rate;
- be widely available; and
- be cheap.

In the previous work [31], MPU6050 accelerometers by TDK InvenSense on a GY-521 breakout board, a 3-axis accelerometer and gyroscope providing 6 Degrees of Freedom (DOF) [1], were used. Another widely used accelerometer in the Arduino community is the ADXL345 by Analog Devices which does not contain an integrated gyroscope, therefore providing only 3 DOF [2]. Both the GY-521 and the ADXL345 are available by third party vendors costing around 3-5€ each on Amazon Germany, so they comply with the widely available and cheap requirement. In the following sections, the GY-521 and MPU6050 will be referred to as MPU and the ADXL345 will be referred to as ADXL.

Both sensors work with a supply voltage range of 2.0V to 3.6 V and communicate via an I^2C interface bus. While the ADXL supports only a 12 bit resolution at $\pm 8g$ range, the MPU Sensor uses a full scale 16 bit Analog Digital Converter (ADC). Both sensors support high output polling rates at up to 1000Hz for the MPU and up to 3200Hz for the ADXL. Both report an absolute maximum rating of 10,000g acceleration in an unpowered state [1,2].

To validate the accuracy of the measurements, the sensors were tested on the black figures where the real rotation angle could be set and read from the motors. The rotation of the rod can be set to a number between -100 and 100 which corresponds to an angle range between 120 and 240 degrees with 0 at 180 degrees. All Sensors were mounted to the same rod for comparable results. The accuracy testing procedure will take a defined number of measurements while altering the rotation of the tested rod using a defined sequence of movement. This sequence contains the following steps:

1. Alternate Direction: The Rotation will start at -100 and increment each iteration by 1 until it reaches +100. After this, it will decrease by 1 until the rotation is back at -100.
2. Alternate Sign: The rotation starts at -100, increments each iteration by 1 and toggles the sign between + and -, resulting in -100, 99, -98, 97, ...
3. Sine: The rotation follows the sine of the current iteration using $\text{rot\_val} = \sin(\frac{i}{100}) \times 100$ with i as the count of the current iteration.
4. Random: The rotation will be randomized to an integer between -100 (incl.) and 101 (excl.).

In each iteration, the measurements of the accelerometers and the motor is taken. All accelerometers were reset to zero before testing. The procedure was iterated 10,000 times with a mode change every 600 iterations. To accommodate faulty sensors, two MPU and two ADXL modules were tested simultaneously resulting in a dataset with 10,000 rows of two measurements per sensor type and the real rotation measured by the motor.

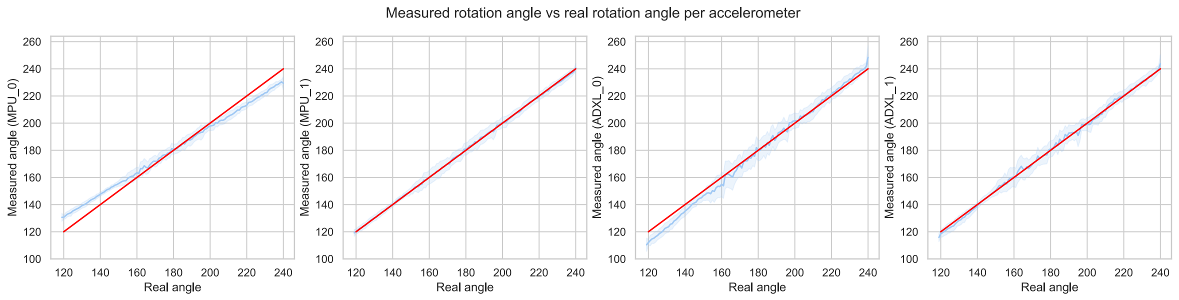


Figure 8: Measured rotation angle vs real rotation angle per accelerometer.

Figure 8 shows the measured rotation and the real rotation converted to an integer. The red line indicates a perfect measurement. The blue line indicates the average measured rotation per real rotation and the light blue area indicates the deviation in the measured data.

The plot clearly shows a miscalibration error in the first MPU while the second measured on average accurate results while containing some deviation. The first ADXL Sensor could also be miscalibrated but shows it less clearly. The second ADXL is on par with the real measurement while having a bigger deviation.

Figure 9 shows the absolute error between the sensor measurement and the real rotation by different movement modes. As expected, the first MPU shows a bigger error on the outer angles which is explained by the observed miscalibration. The second MPU had a mean error of 2.839° and a median error of 2.252° resulting in accurate readings.

The first ADXL sensor has, as also expected, a bigger error than the second one. Again, this can be explained by the observed miscalibration, even if it's slightly lower than the MPU error. The second ADXL sensor has a mean error of 4.026° and a median error of 3.401° which is

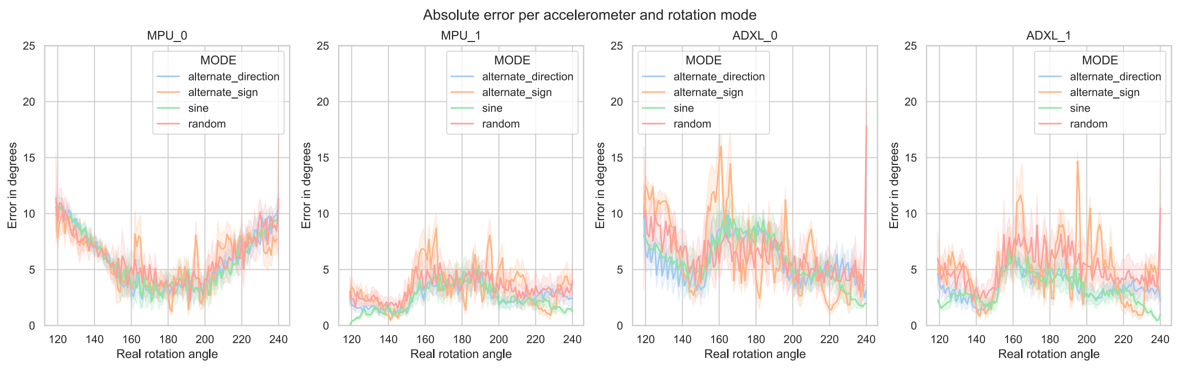


Figure 9: Absolute error between the measured and real rotation angle per accelerometer and rotation mode.

slightly worse than the MPU sensors.

In all sensors, the *alternate sign* (orange) and *random* (red) modes introduce bigger errors seen by the increased amount of higher spikes in Figure 9. This shows a clear influence of the rotation intensity, as the other two modes only rotate about 1-2 degrees per iteration. On the other hand, the *alternate sign* and *random* modes can rotate about 120 degrees in one iteration. Therefore, to get good measurements, sudden and intense rotations with high acceleration should be avoided when capturing the training data.

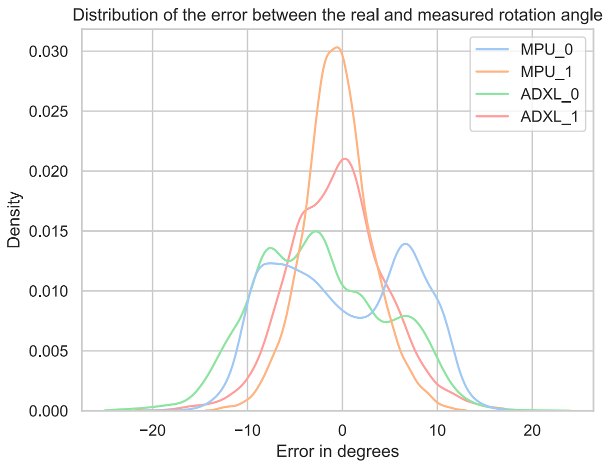


Figure 10: Distribution of the error between the measured and real rotation angle per accelerometer in degrees.

As seen in Figure 10, the error between the measured and real rotation angle of the second MPU sensor follows a normal distribution. The error distribution of the second ADXL can also be approximately described by a normal distribution. On the other hand, the first MPU and first ADXL show clear deviations from a standard distribution, indicating problems such

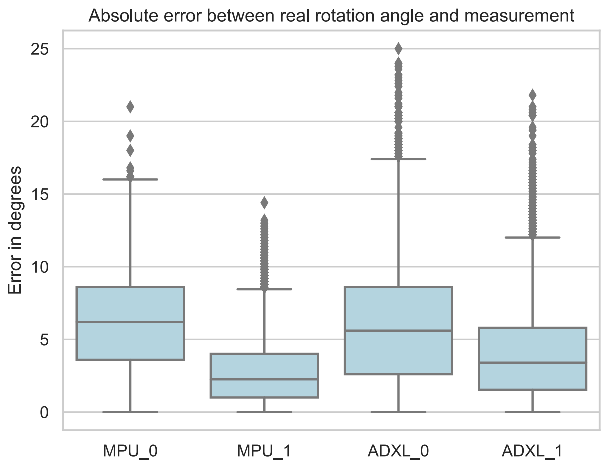


Figure 11: Absolute error between the measured and real rotation angle per accelerometer in degrees.

as the miscalibration of MPU_0. By looking at the absolute error in Figure 11, it is clear that the second MPU generates the best results with the lowest outliers if only by a small margin. The second ADXL also generates good results but measured bigger outliers of up to 22.5°. Overall, the MPU-based sensor delivers better measurements. While the MPU theoretically delivers a higher precision given the 16 bit resolution in contrast to the 12 bit resolution of the ADXL sensors, this difference is insignificant.

As a result of this evaluation, the MPU6050 or GY-521 respectively was chosen for the creation of the training data. Since the taken measurements show a clear indication of possible calibration issues, further testing of different GY-521 modules was conducted.

Testing different GY-521 modules

As already seen, The GY-521 modules can have calibration errors. As the sensors are factory calibrated, those errors couldn't be resolved by hand. To get reliable readings with low error rates, 8 individual modules were tested. The results of those tests are shown in the following section.

Figure 12 shows the measured vs real rotation angle. The red line marks a perfect measurement and the blue line the actual measurement of the tested module. The light blue area marks the standard deviation of the measured angle. Any measurement with a visible systematic drift between the real and measured angle could be the result of a miscalibration.

Out of the 8 tested modules, MPU_6 shows a clear miscalibration with further possible miscalibrations of MPU_0, MPU_2 and MPU_4. The measurements of MPU_2 result in a higher standard deviation. This can further assure the miscalibration hypothesis but could also be the result of a general hardware error in this module. This is also clearly visible in the absolute error as shown in Figure 13.

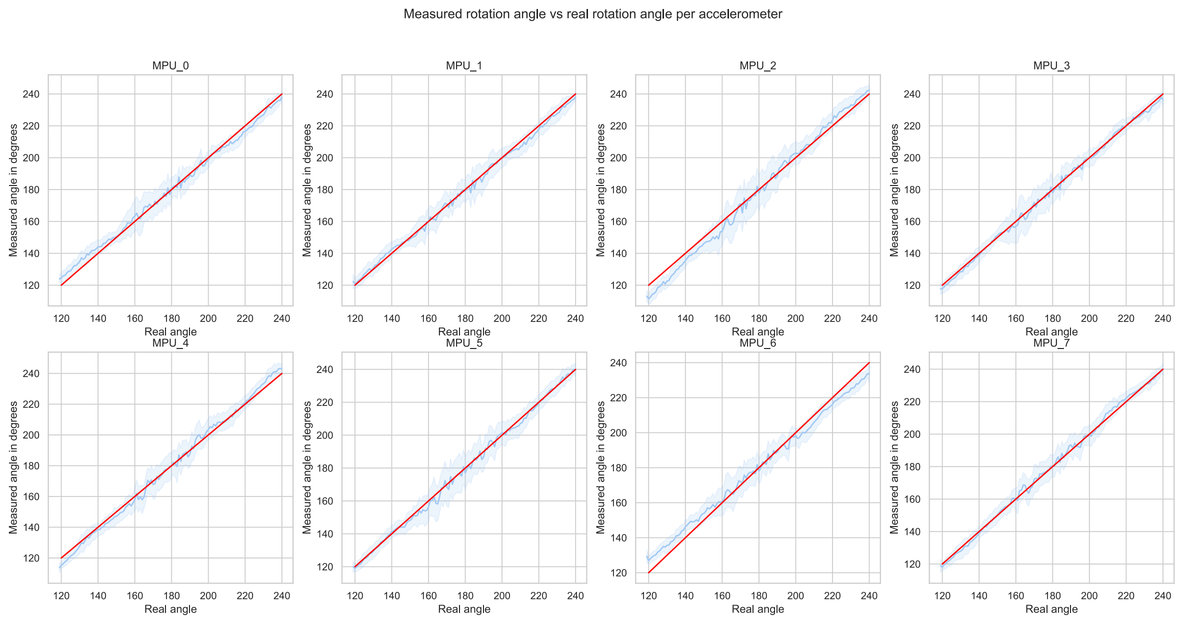


Figure 12: Measured vs real rotation angle per different GY-521 modules.

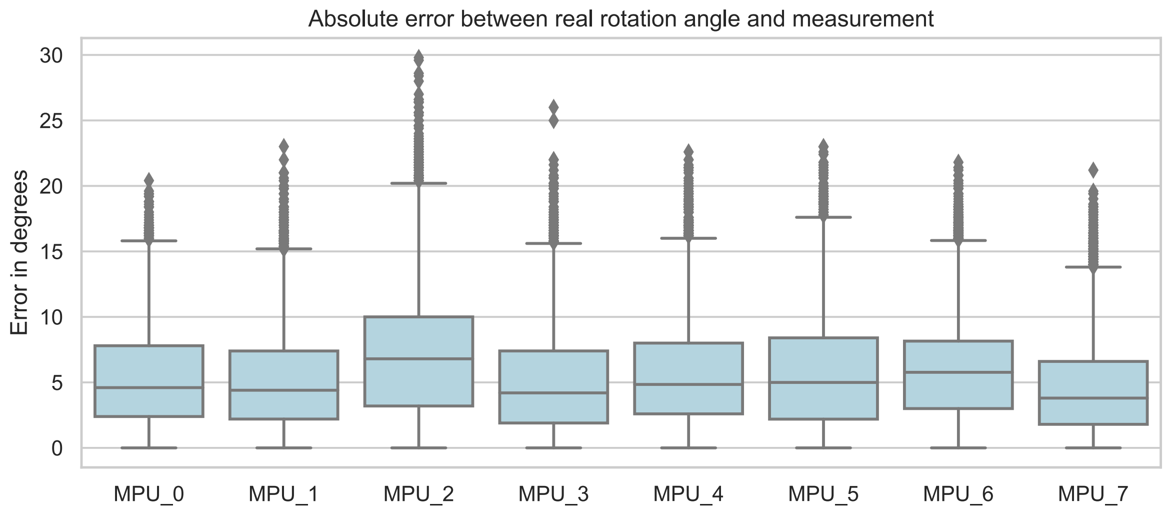


Figure 13: Absolute error between the measured and real rotation angle per module.

Except for MPU_2, all tested modules measured a mean absolute error of about 5 degrees while getting maximum outliers between 20 and 25 degrees. As already stated in the last section, a larger rotation between two measurements and therefore a higher angular acceleration results in a higher error. Since the acceleration can be controlled during the capturing of the training data, a low mean error is more important than low outliers. Therefore, the modules MPU_0, MPU_1, MPU_3 and MPU_7 were chosen for the training data creation process.

4.1.3 Measurement Hardware

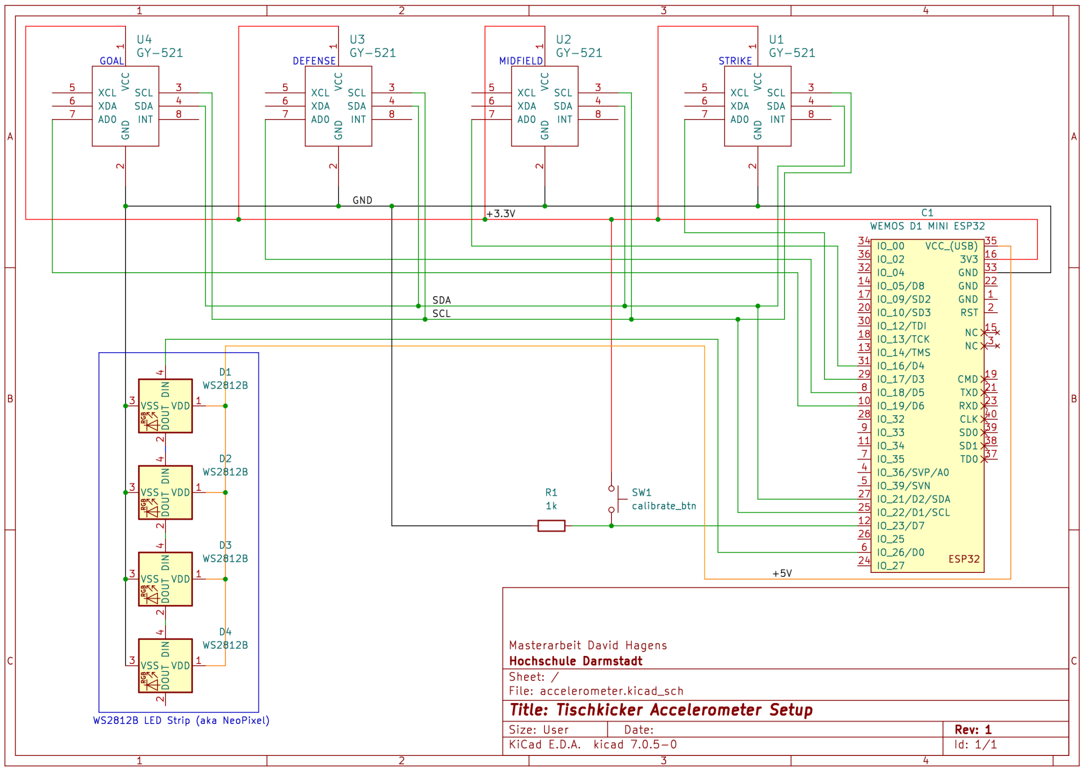

U4 GY−521 GOAL
U3 GY−521 DEFENSE
U2 GY−521 MIDFIELD
U1 GY−521 STRIKE
VCC XCL XDA AD0 SCL SDA INT GND
GND
+3.3V
SDA
SCL
C1 WEMOS D1 MINI ESP32
IO_00 IO_02 IO_04 IO_05/D8 IO_09/SD2 IO_10/SD3 IO_12/TDI IO_13/TCK IO_14/TMS IO_16/D4 IO_17/D3 IO_18/D5 IO_19/D6 IO_32 IO_33 IO_34 IO_35 IO_36/SVP/A0 IO_39/SVN IO_21/D2/SDA IO_22/D1/SCL IO_23/D7 IO_25 IO_26/D0 IO_27
VCC_(USB) 3V3 GND GND GND RST NC NC CMD TXD RXD CLK SD0 SD1 TD0
ESP32
D1 WS2812B
D2 WS2812B
D3 WS2812B
D4 WS2812B
VSS VDD DIN DOUT
R1 1k
SW1 calibrate_btn
+5V
WS2812B LED Strip (aka NeoPixel)
Masterarbeit David Hagens
Hochschule Darmstadt
Sheet: /
File: accelerometer.kicad_sch
Title: Tischkicker Accelerometer Setup
Size: User Date:
Rev: 1
KiCad E.D.A. kicad 7.0.5-0
Id: 1/1

Figure 14: A schematic plan of the used hardware setup.

As stated in the last section, the GY-521 module was chosen to measure the rotation of the white figures. Four of those modules are connected to an ESP32 [3] MCU so all rods can be measured simultaneously. A LED Strip based on WS2812B digital RGB-LEDs is added for time synchronization. The LEDs show a timer with an eight second time window which is binary encoded, so three of the four connected LEDs are used to display the current time. A fourth LED is added to allow the display of general information, like a signal for the zeroing of the accelerometers.

Figure 14 shows the hardware schematic of the measurement hardware. The ESP32 is connected via USB to a computer and communicates using the Arduino Serial protocol. On the top of the schematic, the four accelerometers are seen. Since ESP32 modules can potentially break if a 5V signal is connected to an IO pin, the sensors are connected to 3.3V. While the MPU6050 sensor itself has a voltage range from 2.375V to 3.46V, the GY-521 breakout board has a rated supply voltage of up to +6V. The LED Strip uses 5V power but does not send anything back to the MCU. Therefore, it is connected to 5V power while only getting a 3.3V data signal. The WS2812B LED chip is not rated for only 3.3V data input voltage but does not generate any problems because of this. For a more permanent implementation, the addition of a voltage step up could be considered. Furthermore, it is noted, that an input signal could be potentially stepped up to 5V which is not the case on the utilized LEDs.

The ESP MCU communicates with the GY-521 modules via the I^2C bus. The individual modules all share the same I^2C address (0x68) but can change this address to 0x69 if the AD0 pin is connected to VCC (+3.3V) [1]. The AD0 pin can also be connected to an IO pin of the ESP32 enabling a dynamic address change at runtime. This dynamic address change is used to be able to read all four sensors over the same I^2C bus with only two addresses by switching the addresses of three sensors at a time and read the fourth one. The AD0 pins of the sensors are connected to the digital IO pins D3-D6 of the MCU as shown in Figure 14.

4.1.4 Data Capturing Software

To generate training data, a software should be implemented. This software should:

- Control the black figures for different positions and rotation angles;
- Read the data of the beforementioned accelerometers and the motor sensors; and
- Capture a camera image of the top-down camera.

The resulting data should be saved in a easy-to-read format without the need of complicated data readers. Therefore, a combination of an MPEG-4 encoded video in a .mov container and a corresponding CSV file with the extracted data was chosen.

Figure 15 shows the data capturing process. In each iteration, the black figures are moved and rotated randomly. Before reading the measurements, the software waits for the black figures to stop moving. Since this can't be read from the Foosball Table system, the maximum possible movement time was measured which is approximately 0.4 seconds. After this wait time, the accelerometer readings are retrieved from the ESP32 MCU, the sensor readings of the motors are retrieved via the OPC UA protocol and a camera image is taken. When all data is collected, it is appended to the CSV file and the image is appended to the video. After a configurable amount of iterations, the process is finished. The timing information which is displayed by the LED strip can be used to synchronize the video and the captured data to ensure accuracy between the image and the sensor readings.

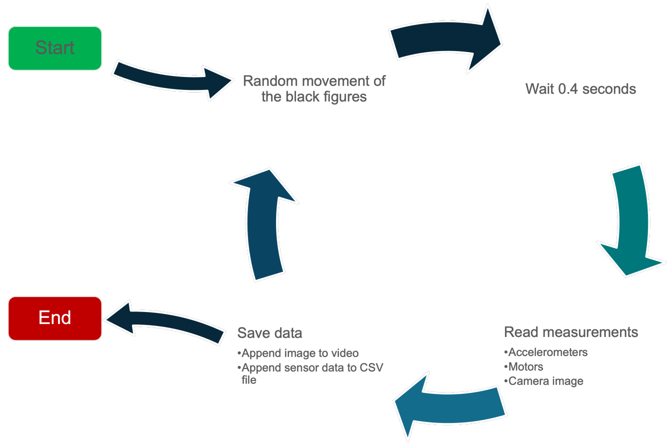


Figure 15: The process for the capturing of training data and video.

4.2 Prediction of the position and rotation of the rods

The prediction of the position and rotation of the different rods will be split into two approaches. The first approach is equivalent to the previous work by Horst et al. [31] which could not be used for the present work. Examples and reasons for this will be presented. In the second approach, the object detector step is discarded.

4.2.1 Approach 1

In the previous work by Horst et al. [31], the position and rotation prediction was separated into the detection of each individual figure using the object detector network YOLOX [26] and the prediction of the rotation using a modified ResNet18-based [28] regressor network. The position of the rod could be calculated using the individual figure positions, as those are fixed on the rod with distinct spacing. The regressor network was used to predict the rotation of a cutout of the frame of each individual figure. The rotation of the rod was then calculated as the mean of the individual figure rotations.

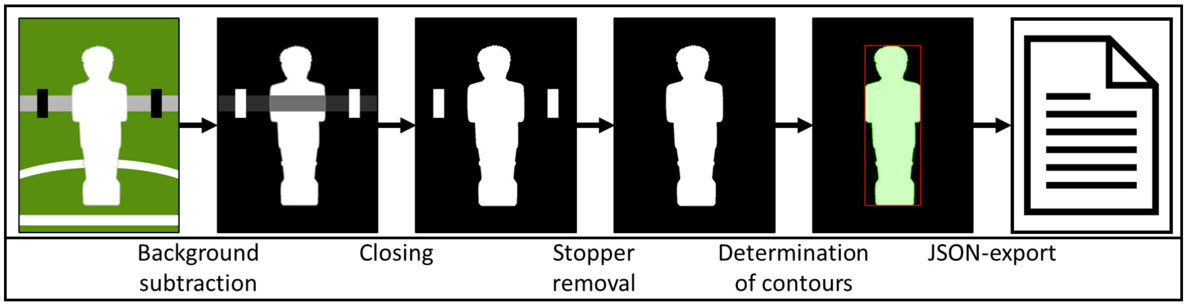


Figure 16: Existing semiautomatic labeling pipeline [31].

For this approach, two training datasets need to be created. First, a dataset containing the whole images of the Foosball table with individual labeled figures and second, a dataset consisting of individual figures labeled with their rotation angle. For the first dataset, a semi-automatic labeling process was implemented while the second dataset was created using the labeled individual figures. The semiautomatic labeling pipeline, which is shown in Figure 16, consists of five sequential steps utilizing traditional image processing techniques. First, the background of the video is extracted by calculating the median image of a random selection of frames. This technique is also called *temporal median filtering* [49]. Using morphological transformations like opening and closing in combination with a binary thresholding, a mask of the foreground of one frame at a time is created. Using this mask, the rubber stoppers on the white rods are removed using the Otsu thresholding algorithm [47]. Afterwards, only the white figures are included in the resulting mask and the bounding box of each figure can be detected by utilizing the border following algorithm of Suzuki et al. [57]. The resulting bounding boxes are saved in the COCO annotation format [43] which is used by the object detector YOLOX [26].

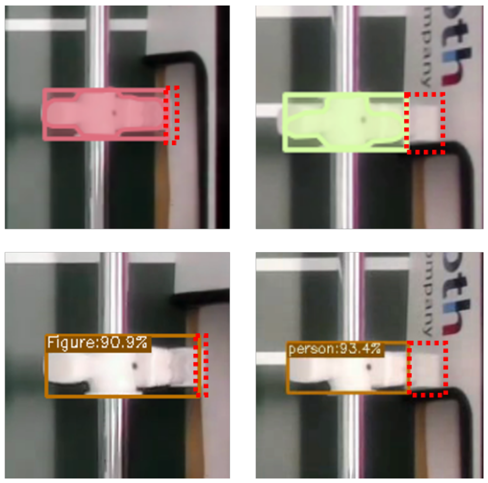


Figure 17: Errors in the labeling (top) and prediction (bottom) of the goalkeeper in the previous work [31].

In the previous work, some errors had been reported regarding the feet of the goalkeeper being cut off as seen in Figure 17. These errors were present in the labeling process and were later learned by the object detector. In this thesis, the labeling process should be extended to also include the black figures (which are controlled by the DRL agent) and be adapted to avoid the shown errors. Several experiments were conducted to comply with those requirements. While the white figures were detected correctly in most cases (except for the goalkeeper feet), the black figures could not be detected with a sufficient accuracy, as some examples show in Figure 18.

In Figure 18a, a simple color segmentation was used to select the black players. While the bottom figure is selected accurately, the top figure segmentation includes different artifacts resulting in a faulty bounding box. Those errors were the result of difficult lighting near the

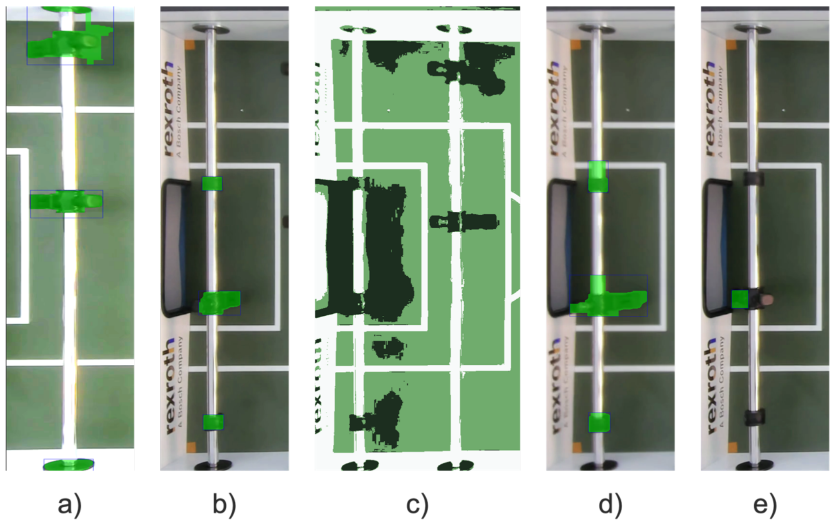


Figure 18: Examples of different approaches to automatic labeling of the individual figures.

top side wall. Different color ranges were tested but resulted in either the same result or less accurate bounding boxes around the bottom figure.

Figure 18b shows a combination of background subtraction and color segmentation for the goalkeeper rod. While this approach shows promising results, the head of the goalkeeper is cut off in this example and other manually verified examples also include smaller errors like this.

Figure 18c uses a KMeans clustering algorithm to separate the image into three different color clusters. Since the main colors of the Foosball table are the white figures, lines and rods; the black figures; and the dark green playing field, a clustering using three clusters should potentially separate the image into those colors. As seen in the example, the playing field was not clustered completely as dark green but includes some black portions, the black figures were not separated correctly. While this method works mostly fine on the midfield figures, the other figures couldn't be segmented reliably.

Figure 18d shows a similar approach as in Figure 18b using background subtraction and color segmentation with an additional thresholding and a small adjustment of the saturation. Since the background (or playing field) is very low saturated, the increment of saturation can improve the color segmentation step as the background color would be more defined. As seen in the image, this is not the case. Since the black figures and the metallic rods are mostly unsaturated, their color is undefined and can be the same as the background. An adjustment of the saturation can therefore result in the background color included in the figures. In the example, the goalkeepers bounding box also includes a small portion of the background itself at the feet of the figure.

In Figure 18e, a combination of thresholding and background masking was used. The median background image, which was also used for the background subtraction in the preceding approaches, was converted into a binary mask including only the playing field without the rods. This prevents the error of inclusion of the rods itself in the figure segments as seen in Figure 18d. The masking approach creates accurate results on the defense and striker figures most of the time. On the other hand, errors like a missing half of the figure as seen in the example are common on the goalkeeper and too wide bounding boxes were found commonly on the midfield figures.

Overall, the bad image quality, a difficult lighting environment and the rather difficult colors of the figures (black and white) resulted in a semiautomatic detection of the figures which did not generate accurate results. A further usage of this semiautomatic labeling would have resulted in the need for manual adaptions on most of the images. Therefore, the images could also be completely manually labeled without a much bigger effort. Since a labeled dataset is the main requirement for the approach using an object detector and this dataset could not be created automatically, the object detection approach was not further progressed. The already mentioned errors in Figure 17 were also not further evaluated, as they are only applicable to the object detector.

4.2.2 Approach 2

Instead of the two-step model consisting of the object detector and a following regressor, the second approach utilized an end-to-end CNN-based regression model for each rod. Each frame, the individual rods are cut out using pre-defined boxes to predict the rotation and position, thus omitting the object detection step. Consequently, the individual figures are not detected. This is also the case for the sensory data of the black figures, which also report only the positional shift of the whole rod. However, the individual position of each figure can be calculated using the *a priori* knowledge of the individual positions on each rod, as those are fixed. The discarding of the object detector also reduces the complexity of the overall system.

As mentioned before, the training data does not contain any information about the position of the white rods, so those need to be calculated. In the first approach this was not necessary, as the position of the whole rod was not addressed and could be calculated from the single figure positions. Unlike Bambach et al. [14], who used the Hough lines transform to find the rods and calculated the positional shift using a pixel-wise algorithm, the rod-position is assumed to be known, as only the pre-captured video is used. Therefore, the Hough lines transform is not needed while the position is detected based on a similar, but adjusted algorithm. First, one column in the center of each rod was cut out of the frame. As the walls of the Foosball table cast shadows on the outermost part of the rods, the cut out is a few pixels smaller than the actual rod. This does not influence the position detection in general but needs to be addressed when converting the pixels to millimeters. The pixel-column was then transformed into black and white pixels using a binary thresholding. Afterwards, groups of connected white or black pixels were built. To prevent small shadows and other errors in the thresholding, small groups were discarded based on an observed minimum length. Now, the black groups represent the figures on the black rods or the rubber stoppers on the white rods. The white groups contain the metallic rods and the white figures. To calculate the middle of a rod, the outermost black groups can be used, as the center of those groups is automatically the center of the rod if the groups are detected correctly. This works also on the black goalkeeper rod, as this rod, in contrast to the other black rods, contains the same rubber stoppers which are used on the white rods, resulting in 3 groups of black pixels (the two stoppers and the goalkeeper figure). Utilizing the calculated center of the rod and the known center of the playing field, a shift of pixels between the two can be calculated. Through the known numbers of pixels per rod and the width of the table, which is 680mm, the shift in pixels can be converted into millimeters. Here, the before mentioned removal of the outermost pixels on each rod needs to be deducted from the width of the table.

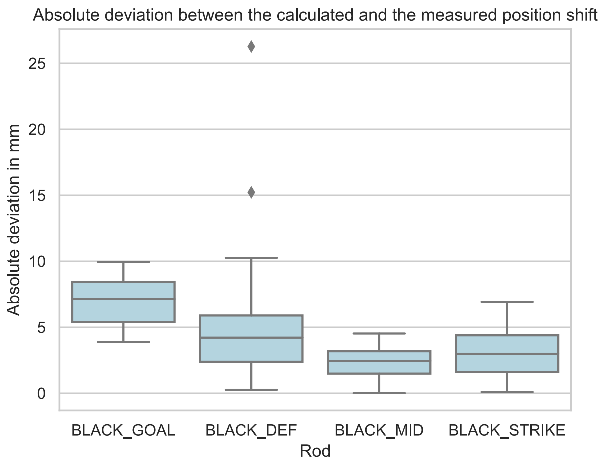


Figure 19: Absoute error between the calculated and measured position shift of the black rods.

Figure 19 shows the deviation between the calculated position shift and the reported position shift of the motors in millimeters for the black rods. While the overall deviation is less than 10mm, the black defense rod calculated a maximum deviation of 26.27 mm which is a lot considering the diameter of the ball of only 35 mm. On further inspection, those outliers seem to be the result of an incomplete movement of the rod. The motors report the destination position even if the destination is not reached by the time the sensor is queried. The biggest outlier with a deviation of 26.27 mm is shown in Figure 20a. In the image, the blue line represents the column size but drawn off-center. The red line represents the calculated shift in the center and the green line represents the reported shift by the motor. The black horizontal bar was added manually in the center between the two figures. While the red line ends exactly at the center, the green line with the reported position shift ends a few pixels above the center. On closer examination the figures are vertically blurred while notable sharper in the horizontal direction. Therefore, it is rather likely that an ongoing movement occurred while the image was taken. Figure 20b shows the other outlier of the black defense rod as observed in Figure 19 with a deviation of 15.22 mm. Here, the overall image is blurry. Again, the calculated position shift shown by the red line ends at the manually added center while the reported position shift in green ends above the center. In a further inspection of other images with high deviations between the calculated position and the reported position, the same could be observed. In conclusion, the calculated positions seem to be more accurate than the reported positions. Therefore, those calculations were used in the further training process.

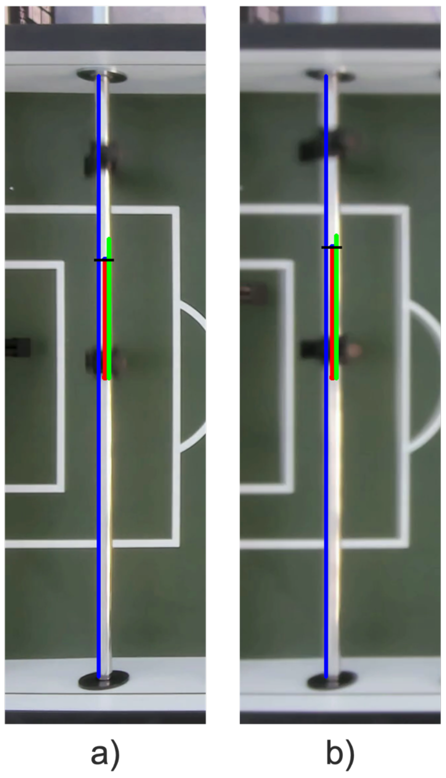


Figure 20: Outlier of the black defense rod with a deviation between the calculated and measured position shift of 26.27 mm.

The prediction of the ball position was considered during the implementation as the usage of the object detector would enable the detection of the ball position without the need for further detection systems. Since the object detector was discarded in favor of the end-to-end regressor based game state detector, the detection of the ball position was also discarded.

Image-based Regressor Networks

Utilizing the before mentioned approach to calculating the positional offset of the white figures, an annotated dataset with position and rotation information for the individual rods can be created. This resulting dataset can be used in the end-to-end CNN-based regression models. Most of the commonly available deep CNN are built for the use case of image classification [12,13,40,42,66]. Since the rotation and position are continuous variables instead of discrete variables, the problem on hand is defined as a regression problem. Therefore, a classification network cannot be used directly. The structure of those image classification networks usually consist of a feature extractor backbone using convolutional layers and a classification head using fully connected layers. The feature extractor uses multiple sequential layers while the classification head can contain only one fully connected layer in ResNet [28] or multiple layers with pooling layers in between in EfficientNetV2 or MobileNetV3 [33,59].

The use of already existing networks is a good practice for new applications as those networks are already proven to deliver good results. Additionally, this practice enables the use of transfer learning to minimize the amount of needed training epochs and therefore the overall training time. To use existing networks for the prediction of continuous variables, the classification layer needs to be customized. To circumvent the circular continuity in angle degrees, the network should predict the sine and cosine of the rotation. Additionally, the position should be predicted in a scaled range from -1 to 1. The scaling prevents a higher influence of the position on the training process, as the sine and cosine are also in a -1 to 1 range. Therefore, a linear layer with an output dimension of three is used to predict those values. This layer applies a linear transformation to the incoming data: $y = xA^T + b$. Any activation function on the classification layer of the base network is removed to prevent any transformation of the output of the last layer.

In the previous prototype [31], a ResNet18 [28] feature extractor backbone was used to predict the rotation angle of individual figures. In this work, different backbones were used and evaluated. In addition to ResNet18, ResNet50 [28], EfficientNetV2 [59] and MobileNetV3 [33] backbones are implemented. Furthermore, a simple CNN implementation is used using five convolutional layers and a linear layer for the regression.

4.3 System Output

To reach the overall goal of creating a usable system for reinforcement learning experiments conducted by future students, the serving of the predicted data is an important factor. The system output underlies the following requirements:

- It must be easily readable in standardized formats;
- It must be easily accessible with a minimal need of further Hard- or software; and
- It shouldn't introduce high latency into the system.

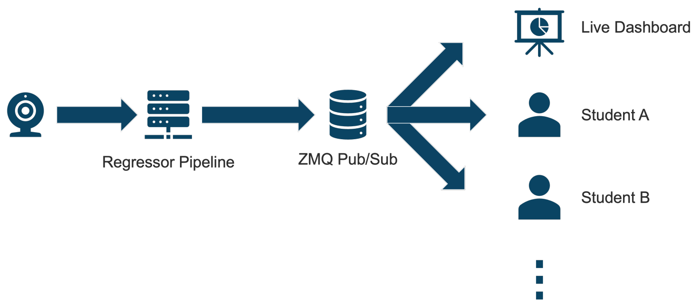


Figure 21: Data Pipeline from the camera input to different Clients.

With the use of a ZeroMQ Publish and Subscribe messaging system, those requirements could be achieved. The data producer pipeline, in which the rotation and position is predicted, opens a TCP socket using the ZeroMQ protocol and publishes JSON-formatted results. Clients can then connect to this socket, subscribe to the messages and decode the JSON data. Figure 21 shows the overall pipeline using the ZeroMQ server with different (example) clients.

The use of ZeroMQ introduces some latency but also the ability to easily deploy the system on an integrated server with a dedicated GPU. This server could be accessed by future students and therefore reduce their hardware requirements and / or frees hardware resources for reinforcement learning experiments. Since one ZeroMQ Server can serve multiple clients, a group of students would also be able to get the data without the need of additional resources.

Chapter 5

Proof of Concept

In this chapter, a proof of concept of the presented system architecture from chapter 4 is implemented. The system is developed for the already described semi-automatic Foosball table which was also used in the previous work by Horst & Hagens et al. [31]. The described prototype can be used inside an Anaconda virtual environment and is implemented using the Python programming language. The software which controls and reads the accelerometers is written in C++ using the Arduino framework. First, the implementation of the data capturing system with the included measurement hardware is described. Afterwards, the end-to-end regressor pipeline is portrayed. Furthermore, the implementation of the output system and two example clients is shown. The section ends with the implementation of a pipeline including the regressor and the output system.

5.1 Data Capturing

Measurement Hardware

The measurement hardware, as shown in Figure 22 and assembled from the schematic in Figure 14 was built using GY-521 modules and an ESP32-WROOM-32 from AZ-Delivery. The accelerometers are screwed onto a 3D printed mount which itself is screwed onto the rods of the Foosball table. The mounting hardware is printed in PETG[1] and needs two M3 $\times$20 mm, two M3 $\times$4 mm screws and two M3 nuts per rod. The LED strip is mounted to the wall of the Foosball table using double-sided tape which can be removed without residue. Each accelerometer is connected through a five-wire cable for VCC, GND, SDA, SCL and AD0 for changing the I^{2}C address as illustrated in Figure 14.

The software for the ESP is written in C++ using the Arduino Framework. The GY-521 sensors are polled utilizing the Adafruit_MPU6050 library. The WS2812B LEDs are controlled using the Adafruit_NeoPixel library. Figure 24 illustrates the loop of the accelerometer hardware. Each rod (and therefore each accelerometer) needs to be activated by setting the specific

[1]Polyethylene Terephthalate Glycol, a widely used 3D-printing material.

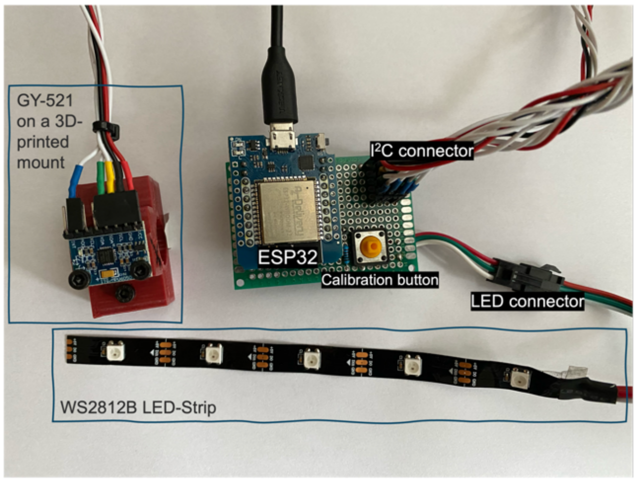


Figure 22: Image of the assembled measurement hardware.

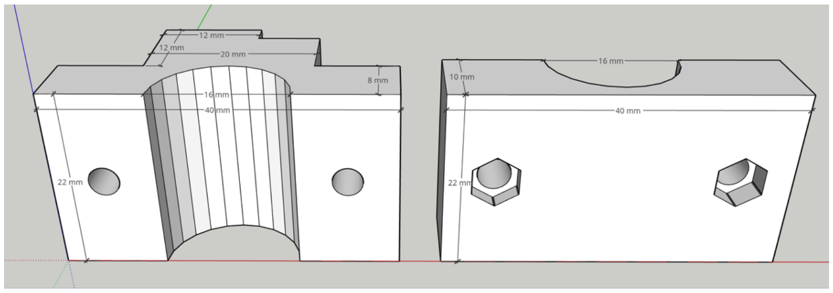


Figure 23: Model of the 3D-printed mount for the accelerometers.

AD0 pin of the accelerometer to LOW (0V), i.e. the AD0 pin is utilized as a chip select. At startup, those pins are set to HIGH (3.3V), so the accelerometers do not respond on the default I^{2}C address, which is 0x68. The ESP communicates via a serial connection using the on-board USB port with 1,000,000 Bd. As an on-board voltage converter is available and the 3.3V and 5V pins of the ESP deliver regulated voltages when powered via USB, the USB port is also used for power. Therefore, no external power supply is necessary.

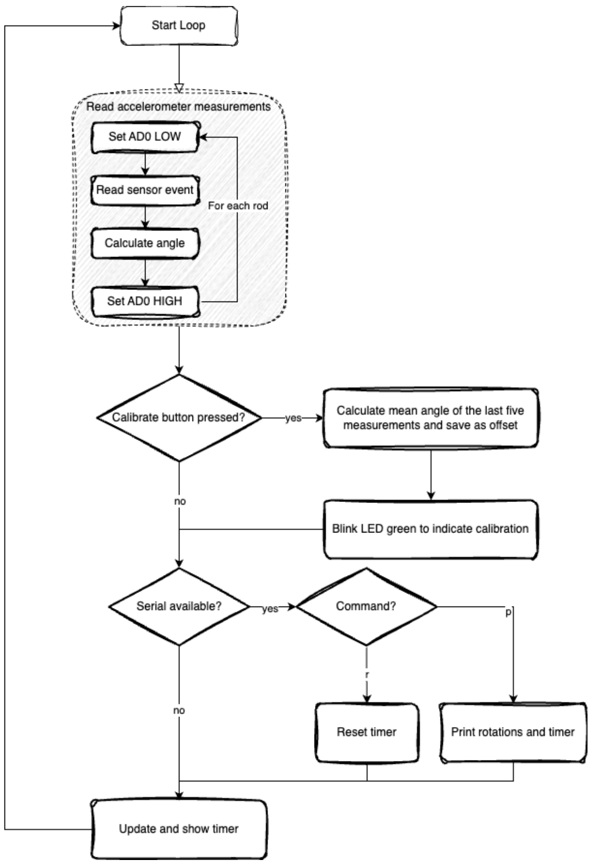

Start Loop
Read accelerometer measurements
Set AD0 LOW
Read sensor event
For each rod
Calculate angle
Set AD0 HIGH
Calibrate button pressed?
yes
Calculate mean angle of the last five measurements and save as offset
no
Blink LED green to indicate calibration
Serial available?
yes
Command?
p
r
no
Reset timer
Print rotations and timer
Update and show timer

Figure 24: Flowchart of the ESP loop.

Reading the data

The overall data-capturing software is implemented in Python using the AsyncIO library, which enables asynchronous computing. This is especially useful when relying on communication with other services, as they can be polled simultaneously instead of sequentially. Since the software needs to capture data of different devices, mainly the ESP with the accelerometers and the Foosball table itself, the asynchronous polling of both enables a shorter time period between the measurements in contrast to a defined lag if the measurements were taken sequentially.

1. ESP

As mentioned before, the rotation data from the white rods is read from the ESP using a serial connection over USB. On the python side, the Serial_AsyncIO library is used which enables asynchronous communication. The serial port can be configured through command line options in the python application.

2. Motors
   The motors and sensors of the Foosball table itself are controlled and polled via the OPC UA protocol [44], an open source networking protocol for communication between sensors, controllers and other devices. The Foosball table uses the protocol via TCP-IP as already implemented by Bosch Rexroth and described by De Blasi et al. [19].
   On the python side, the open source project AsyncUA is utilized. The original code [4] of the Foosball table uses the predecessor of this library, which is officially deprecated and, in contrast to the AsyncUA library, does not support asynchronous computation.

3. Camera
   The camera image is captured using the OpenCV [6] framework with a custom build AsyncIO wrapper, as OpenCV does not provide an asynchronous version of the `VideoCapture` class.

Combined, the training data contains a dataframe with the white rods rotation, the black rods position and rotation and the ESP's timer information. Additionally, the camera image is capture as a video file for data compression and portability reasons. The dataframe is saved as a `.csv` file while the video is saved in an MPEG-4 encoded `.mov` container. The timer information of the ESP is used for manual time synchronization between the captured video and the accelerometers executed by skipping a few frames at the beginning in a video editing software. In this work, DaVinci Resolve[2] was used but any software capable of simple video editing can be used for the synchronization.

5.2 Prediction of the Position and Rotation of the Rods

Creating a dataset

Since the captured data only consists of the whole frame, the rotation of the white rods and the rotation and position of the black rods, the data needs to be further processed before training a regressor network. In contrast to the accelerometers which measure 0° in an upright position, the sensors on the black rods measure this position as 180°. Therefore, the measured rotations of the white rods need to be shifted by 180° to match the zero point of the black rods. Afterwards, the positions of the white rods must be calculated as described in subsection 4.2.2.

As each rod is predicted by an individual regressor network, this individual network should only receive a portion of the whole frame which includes only the rod with its figures. Those slices are defined by a color-coded mask which was manually created and is shown in Figure 25. The mask includes some padding per rod so small adjustments of the camera position or field of view should still result in a correct slicing of the image. A modification of the camera by

[2]https://www.blackmagicdesign.com/de/products/davinciresolve/

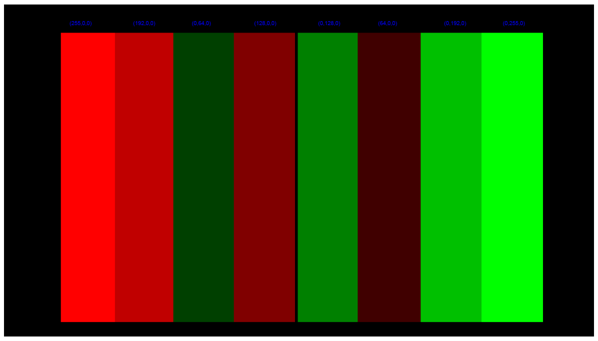


Figure 25: Color-coded rod mask to create individual image slices for each rod.

a higher margin would result in the need for creating a new rod mask. Each rod in the mask uses a distinct color with different intensities of red for the black rods and different intensities of green for the white rods. The color codes are written as RGB codes in a blue text color above the slices. Therefore, simple color segmentation methods such as OpenCV's `inRange` function can be used to create a binary mask of the specific rod. It is also possibly to generate direct masks for all black or all white rods by selecting the whole red or green color spectrum. The resulting image slices are resized to 224 ×224 px and converted to tensors for direct use with the regressor networks.

The shown process results in a data directory containing eight individual datasets with each containing the image slices of a specific rod. Each dataset is available as a `torch.utils.data.DataLoader` which can be directly used to train a PyTorch-based neural network without further processing. To counteract overfitting, those datasets are split into 80% training data and 20% validation data.

Regressor Training

As described in subsection 4.2.2, the position and rotation of the rods should be predicted using an end-to-end CNN-based regressor network. This is implemented as eight individual regressor networks for each rod to circumvent errors through the different colors and number of figures of the rods. In addition, problems through the perspective distortion introduced by the camera are avoided. This results in eight sequential regressor networks which are implemented utilizing the PyTorch [56] library. Additionally, the Torchvision [5] extension library is used as it has a wide variety of pre-built models and features for image processing.

Five different regressor models are implemented while four of which are based on existing CNN-based image classifiers. Two models of the ResNet [28] series are used, the smallest ResNet18 and the larger ResNet50. Additionally, the small versions of MobileNetV3 [33] and EfficientNetV2 [59] are utilized. Since the existing models are built for image classification tasks while the work on hand is a regression problem, the classification layers are replaced by linear layers without an activation function. Furthermore, a simple CNN feature extractor is implemented using the architecture illustrated in Figure 26. In contrast to the other networks this is the only network without the usage of residual blocks. Except for the own implementation, the weights of all CNNs are initialized using transfer learning with the base networks weights which were trained on the ImageNet [20] 1K dataset. The new regression layers and the own implemented network are initialized using random weights.

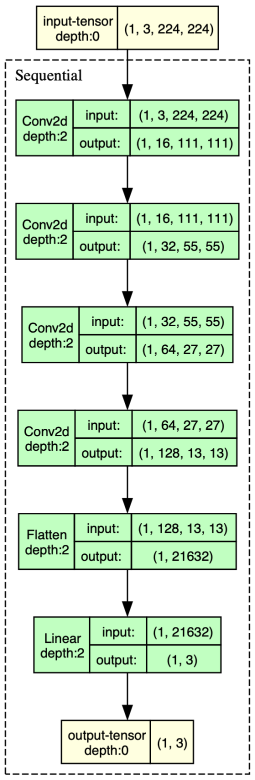


Figure 26: Architecture of the own implemented CNN.

Each regressor is custom trained with the before mentioned dataset and configuration. To enable an easy-to-use platform independent training process, the best possible device is automatically chosen. If a CUDA enabled GPU is available, the training process should utilize this GPU. Otherwise, an Metal Performance Shader enabled device is used. If no GPU (either CUDA or Metal Performance Shader (MPS)) is available, the training process is executed on the CPU. The training logic is implemented in a small python application which creates a dataset for each rod and trains a regressor using the configurable feature extractor backbone. For each regressor the best checkpoint is saved per default. Additionally, the checkpoints of each epoch could be saved which greatly increases the needed disk space. The regressors are trained over 50 epochs utilizing the Mean Squared Error (MSE) loss function and the Adam optimizer [36] with a learning rate of 0.001.

Table 1 contains an overview of the overall training time per model (i.e. containing all eight individual rod regressors) and the file size of the checkpoint file which contains the trained weights. The training was conducted on a M1 Pro based Apple MacBook Pro with Metal Per-

Table 1: Comparison of the checkpoint file size and training time on a M1 Pro (MacBook Pro 2021) with MPS acceleration.

Backbone	ResNet18	ResNet50	MobileNetV3	EfficientNetV2	Own Implementation
Checkpoint size	44.8 MB	94.4 MB	6.3 MB	81.7 MB	656 KB
Training time	00:19:25.5	01:02:21.9	00:22:35.6	01:30:54.6	00:13:12.3

formance Shader (MPS) GPU acceleration. Of the four existing feature extractor networks, the ResNet18-based model finished the training process in the fastest time but is closely followed by the MobileNetV3-based model. As expected, the ResNet50-based model finished after a significantly longer time period due to its larger architecture compared to the smaller ResNet18-based model. Additionally, the ResNet50-based model resulted in the largest checkpoint file with 94.4 MB per rod or 755.2 MB for all rods. In contrast, the ResNet18-based model only needs less than half the disk space with 358.4 MB or 44.8 MB per rod. The EfficientNetV2-based regressor completed the training in 01:30:54 and therefore needed the longest time. With 81.7 MB per rod or 653,6 MB overall for the checkpoint files it also needs the second largest disk space. Considering that the own implemented regressor network is the smallest of all used network architectures, the fastest training time of only 13 minutes and the smallest checkpoint file size of 656 KB is expected.

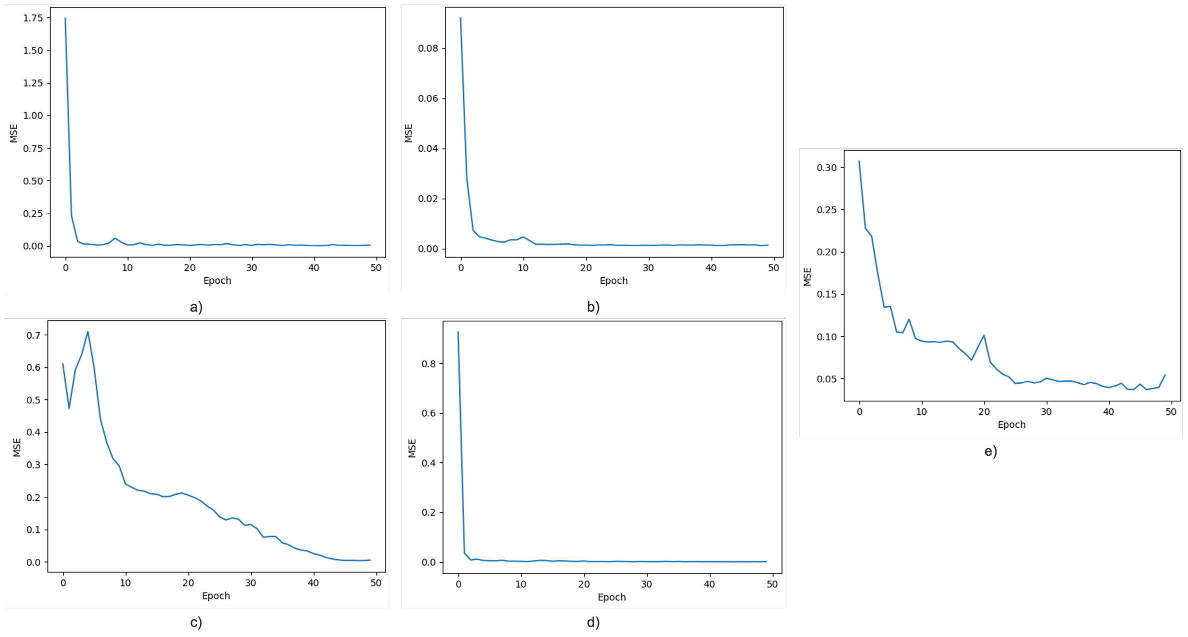


Figure 27: MSE loss per epoch and Regressor Architecture.

Figure 27 illustrates the loss function of the training process per epoch and regressor architecture exemplary for the black goalkeeper. The ResNet-based (a & b) and the EfficientNetV2-based (d) models have a good learning start with only minor improvements after a few epochs. In contrast, the MobileNetV3-based (c) model and the own architecture (e) show visible improvements over the whole 50 epochs which indicates that an extension of the training process over more epochs would potentially improve the prediction with those networks.

5.3 Output

The system outputs its data through a ZeroMQ publish and subscribe system. This enables the usage of multiple clients on one game state detector. To showcase this output system, two different clients are implemented. First a simple image viewer application and second a web interface which plots the position and rotation for each rod.

The implemented Image Viewer client uses ZeroMQ subscribe to receive the individual frames and the corresponding predictions. The position and rotation predictions for each corresponding rod is printed above the rod using pre-defined center positions which were manually derived from the training data. The frame is updated each time a new frame is sent by the pipeline. The Image Viewer utilizes the OpenCV [6] library to print the values to the frame and show the final image in a window. Figure 28 illustrates two sample frames as displayed by the Image Viewer client.

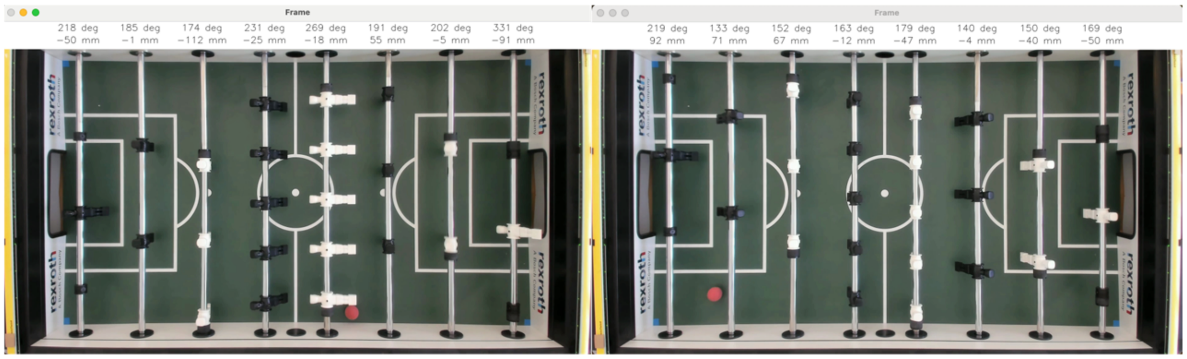


Figure 28: Example output of the Image Viewer client.

The second implemented client is a web-based interface which visualizes the predicted position and rotation of each rod over time. Two sample screenshots of this interface are presented in Figure 29. The top portion displays the default view, where the first plot shows the predicted position and the second plot the predicted rotation. Additionally, the average FPS are calculated and printed on the top of the page. The plots and the computed average FPS are updated once per second. The web interface allows users to filter each plot for one or more specific rods as demonstrated in the bottom part of Figure 29. The web interface is implemented using the Dash library by Plotly. The integrated use of Plotly enables the filtering while the usage of Dash enables the web server and the updating of the plots.

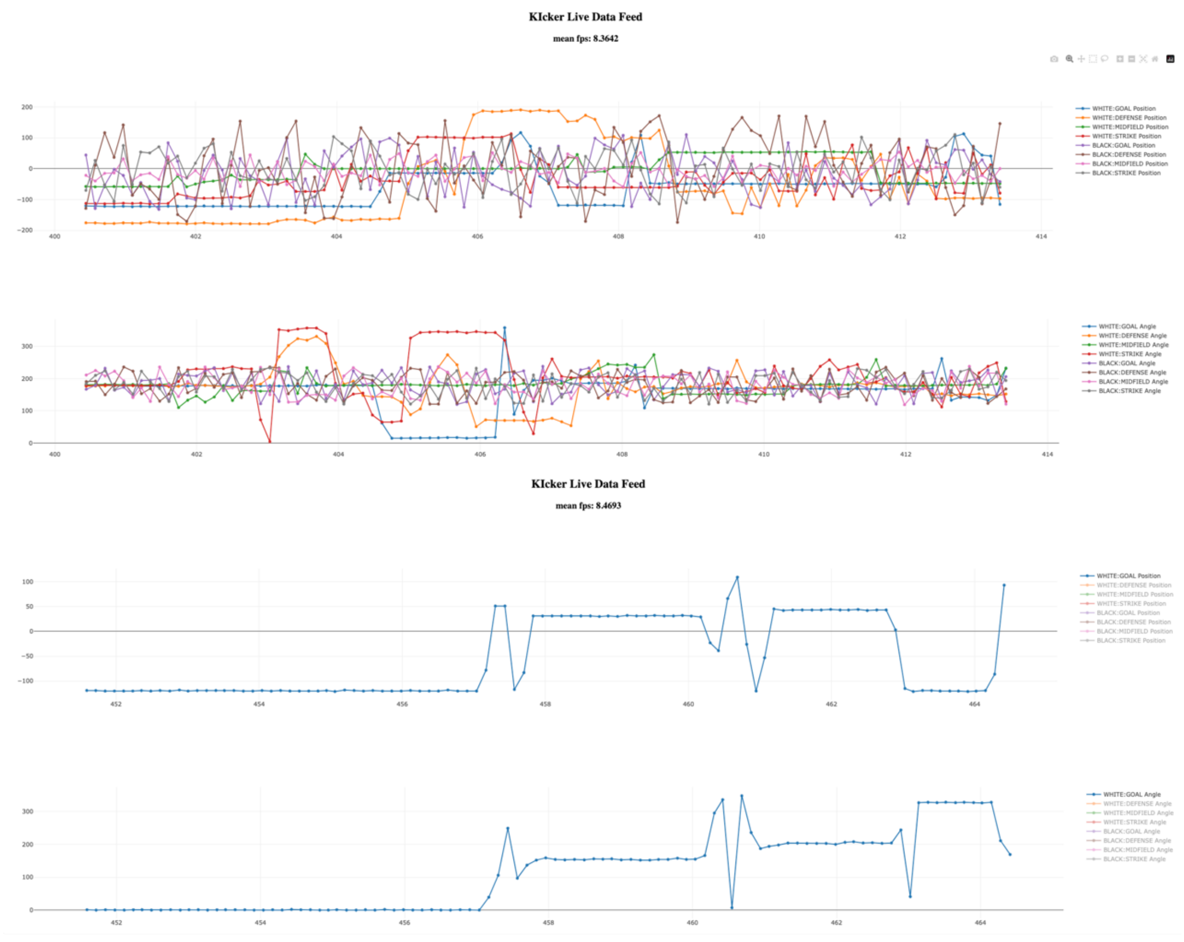


Figure 29: Example output of the web interface client.

5.4 Game State Detector Pipeline

To use the whole game state detector, a simple-to-use python application was implemented. This includes the detector pipeline which provides the individual regressors for the game state prediction and the output system. The pipeline can either use a video as input data or capture a camera stream. If the input data is a video file, the pipeline will run infinitely until a keyboard interrupt is triggered and the video is looped.

The pipeline can be configured using the following startup arguments:

- `--model`: configures the model to use. This must be a directory with 8 subdirectories for each individual rod. Each subdirectory must include a *best_chk.pth* file with the actual trained weights of the model and a *config.yml* containing the regressor configuration. While the configuration includes a *device* entry, the best available device will be used automatically.
- `--input`: can be either a file path to a video file or a camera device index. For video files, any file type which can be read by OpenCV is supported. The camera index needs to be an integer and is typically either 0 or 1 depending if an internal webcam is available or not.
- `--include-frame`: controls the output system. By default and for performance reasons, the frame itself is not included in the data output stream. If the frame is needed for further processing or viewing, e.g. by the Image Viewer client, this option needs to be activated. The frame will be included as a Base64 encoded JPEG image.
- `--include-infer-time`: also controls the output system. By default, the inference time is not included. If the inference time is needed for further processing, e.g. by the web interface client to calculate the mean FPS, this option needs to be activated.

On startup, the pipeline will load the defined regressors for each rod and open a ZeroMQ publisher TCP socket on `tcp://localhost:5555`. Afterwards the video or camera stream will be opened. For each frame, the models predict the position and rotation for each rod and an output message in JSON format is created. If the frame should be included, the image is converted to JPEG format, encoded in Base64 and appended to the output data. Afterwards, the overall processing time per frame is calculated and logged. The process is illustrated in Figure 30.

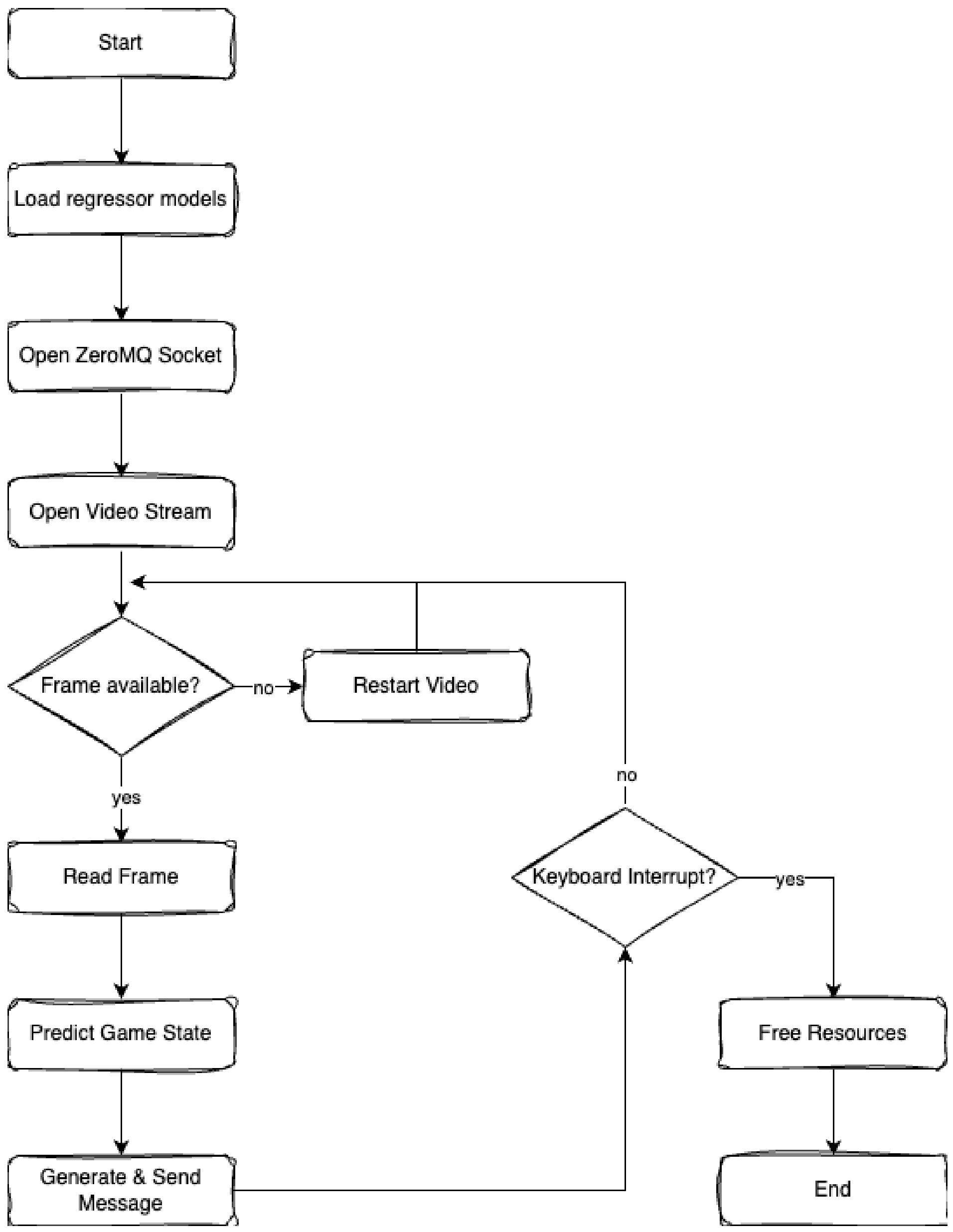


Figure 30: Flowchart of the game state detection pipeline.

Chapter 6

Evaluation and Discussion of Results

Derived from the requirements as defined in section 1.2, the system is evaluated considering:

1. The accuracy of the system; and
2. The real-time objective.

In the following, a quantitative evaluation of the system is described. Additionally, a qualitative inspection on a random sample of images was executed but generated no further insights. To meet the real-time objective the system should be able to detect the game state with 60 FPS on commodity hardware. Especially this should be achieved on a laptop. Therefore, the inference time of the proof of concept is evaluated on four different systems including two laptops, one desktop PC and one cloud VM. Furthermore, the overall results considering the defined objectives are discussed.

6.1 Evaluation of different Feature Extractor Backbones

6.1.1 Accuracy of the Prediction

As mentioned in section 1.2, the accuracy objective is met if the average error of the position and rotation prediction is inside the position and rotation range. The rotation range was defined as ± 42 degrees or around 23 %. The position range is defined as ± 11 mm. As every rod has different positional limits, the percentual range is calculated per rod. On the midfield rod, which has the least movement range, the acceptable range would be around 20 %. The goalkeeper rod has a movement range of ± 120 mm resulting in an acceptable range of approximately 9.2 %.

Figure 31 shows the quantitative evaluation result of the different feature extractor backbones for the black goalkeeper. Each Column shows the result for one feature extractor network. The first row shows the predicted sine compared to the actual sine of the rotation of the rod. The

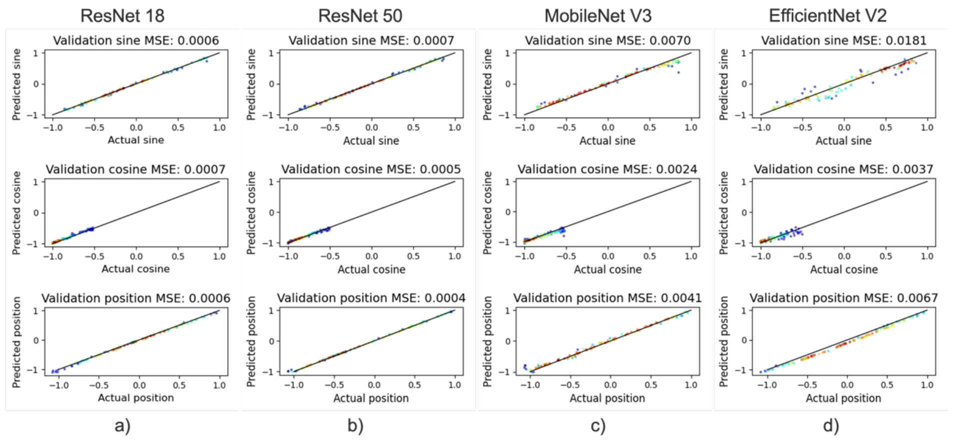


Figure 31: Prediction metrics of the different feature extractor backbones on the black goalkeeper.

second row shows the cosine predictions and the third row the comparison of the predicted and actual position. The results were not further processed so the position is in the -1 to 1 scale and not the actual positional shift in millimeters. The plots in the image show the results of the black goalkeeper as an example. In the plots, data points are scattered on the actual and predicted values. The amount of data points on a single point in the plot is mapped to the color where a red point indicates a high density of data points and a blue point indicates a low density. A perfect fit between the actual and predicted values would result in each point in the plot laying directly on the diagonal black line. Additionally, the MSE for each value is presented above the plot. Since the rotation of the black rods is capped between 120 and 240 degrees, the actual cosine is also capped between the corresponding $-\frac{1}{2}$ for 120 and 240 degrees and -1 for 180 degrees.

Overall, the ResNet-based regressors in Figure 31a and Figure 31b show promising results with a low MSE and mostly correct predicted values. Albeit using a larger network, the results of the ResNet50 feature extractor in Figure 31b are only slightly better than the results of the ResNet18 base in Figure 31a. In contrast, the results of MobileNetV3 (Figure 31c) and EfficientNetV2 (Figure 31d) are visible inferior which is further underlined by a higher MSE on all predicted values.

In comparison, Figure 32 shows the same metrics on the white goalkeeper. Here, the cosine is not capped as the white rods can rotate freely. While the position is again predicted accurately by all regressors, the rotation is not. Albeit the majority being predicted correctly as indicated by the red points on the diagonal line, some outliers are present in each feature extractor. This resulted in a converted mean absolute error of 12.64° for the ResNet18-based network as seen in Table 2. Again, the ResNet-based networks showed the most promising results of the evaluated feature extractor backbones.

The self-implemented feature extractor, as shown in Figure 33, generated promising results

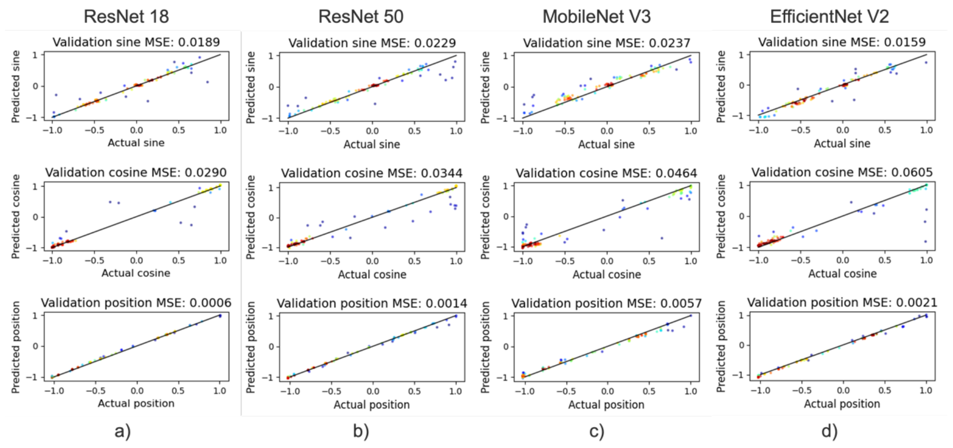


Figure 32: Prediction metrics of the different feature extractor backbones on the white goalkeeper.

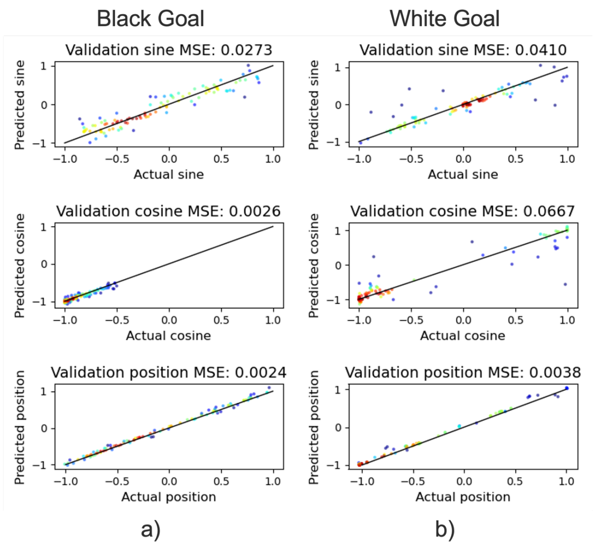


Figure 33: Prediction metrics of the own architecture on the black and white goalkeeper.

Table 2: Mean absolute error of the predicted rotation angle in degrees of each rod and each feature extractor.

Feature Extractor	**Black Rods**				**White Rods**			
	Goal	Defense	Midfield	Striker	Goal	Defense	Midfield	Striker
ResNet18	1.23	1.47	0.88	1.38	12.64	13.93	4.96	10.93
ResNet50	1.34	1.43	0.97	1.33	8.33	5.44	4.06	9.91
MobileNetV3	3.46	2.18	1.72	4.35	17.86	22.96	7.69	21.21
EfficientNetV2	6.31	2.14	2.29	1.69	14.26	25.32	13.59	28.68
Own Implementation	6.54	6.84	3.2	5.76	35.73	31.23	13.09	15.25

Table 3: Mean absolute error of the predicted position in mm of each rod and feature extractor.

Feature Extractor	**Black Rods**				**White Rods**				**Average**
	Goal	Defense	Midfield	Striker	Goal	Defense	Midfield	Striker	
ResNet18	2.94	4.02	1.23	1.6	2.94	7.42	2.2	8.68	3.88
ResNet50	2.4	4.02	1.35	2.77	4.5	6.49	1.91	7.83	3.91
MobileNetV3	7.68	9.35	3.01	7.32	9.06	11.94	3.89	6.78	7.38
EfficientNetV2	9.82	5.97	2.06	3.75	5.5	12.73	4.47	8.53	6.6
Own Implementation	5.88	8.44	1.23	3.2	7.39	21.97	2.46	6.78	7.17

for the position prediction while being worse at predicting the rotation. The overall trend of a worse prediction on the white rods is also visible. The evaluation metrics of the other rods are available in Appendix A.

As illustrated in Table 2, the rotation of the black rods is predicted with more accuracy than the rotation of the white rods by all regressors. Firstly, the rotation measurement of the black figures is more accurate than the accelerometer-based measurement of the white figures. The accelerometers have shown a mean absolute error of about 5 degrees as shown in subsection 4.1.2. Therefore, each captured data point is on average ±5 degrees off of the actual rotation. Secondly, the captured black rod rotations are evenly distributed due to the random movement between each captured frame. In contrast, the white rods where moved manually and not simultaneously resulting in consecutive frames without any change in rotation and a non-uniform distribution as illustrated in Figure 34. Additionally, it should be noted that the uncapped prediction of the full possible 360° is more complicated than the capped rotation between 120° and 240° degrees. Overall, all regressors achieve the required maximum average error of ±42 degrees.

Illustrated in Table 3 are the mean absolute errors of the predicted position converted to mm of each rod and feature extractor. The defined position range was ±11 mm, so an average error needs to be below 11 mm for the objective to be met. On average, each feature extractor passed the objective. With an average error of 3.88 mm over all rods, the ResNet18-based model achieved the most accurate position prediction. The least accurate prediction with an average error of 7.38 mm over all rods was achieved by the MobileNetV3-based model. By looking at each individual rod, only the MobileNetV3, EfficientNetV2 and the own implemented model did not meet the requirement. The own implementation is the least accurate on the white

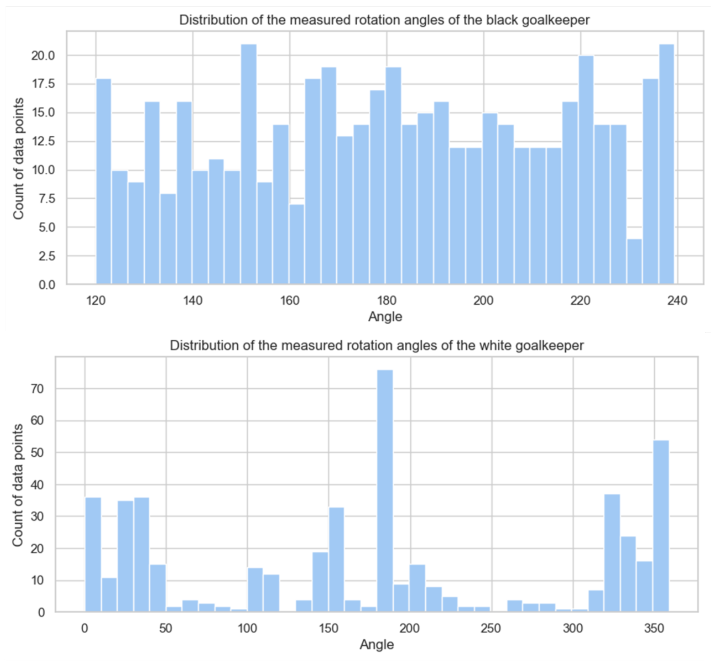


Figure 34: Distribution of the measured rotation angle of the black (top) and white (bottom) goalkeeper.

Table 4: Testsystems on which the regressor models have been evaluated.

System	CPU (Cores / Threads)	GPU	RAM	OS
A	Intel Core I7 @2.2 GHz (6C/12T)	AMD Radeon Pro 560X with MPS	16 GB	MacOS 14.0 Sonoma
B	Apple M1 Pro (10C)	Apple M1 Pro (16C) with MPS	32 GB	MacOS 14.0 Sonoma
C	AMD Ryzen 9 5900X @3.7 GHz (12C/24T)	NVIDIA RTX 3080 with CUDA 12.2	64 GB	Windows 11 Pro 22H2
D	AMD EPYC-Milan @ 2.7 GHz (8VC)	NVIDIA A100 80G PCIe with CUDA 12.0	64 GB	Ubuntu 22.04.1 LTS

midfield rod with an average error of 21.97 mm.

A qualitative inspection of random samples showed similar results with the ResNet-based regressors generating the most promising predictions of the tested CNNs. except for the white midfield rod, the own implementation predicted the position equally or better than the MobileNetV3- and EfficientNetV2-based regressors with worse results in the rotation prediction.

6.1.2 Inference Times

Benchmark Systems

Table 4 summarizes the four different systems which were used to evaluate the inference times of the game state detection system. Systems A and B are Notebooks (Apple MacBook Pro 2018 and Apple MacBook Pro 2021) while System D is a cloud VM. All systems are equipped with dedicated GPUs which are either available through NVIDIA CUDA (System C & D) or

Apple MPS (Systems A & B). MPS also enables the usage of the PyTorch Library on AMD-based GPUs. On the software side, each system uses Python 3.9, PyTorch 2.1.0, Torchvision 0.16.0 and OpenCV 4.8.0.76.

According to the TechPowerUp GPU-Database [8], System D theoretically delivers the most GPU performance with 155.92 TFLOPS in TensorFloat32 Precision. In contrast, System C has a reported theoretical performance of 29.77 TFLOPS in Float32 Precision. System A delivers 2.056 TFLOPS in Float32 Precision. This should be viewed with caution as Apple does not communicate direct hardware specs. If the GPU is modified, e.g. overclocked, the theoretical performance would be higher. Therefore, System B, which uses an Apple-built SoC, reports no theoretical performance. Considering the hardware specs, a lower performance than System C and D is expected.

Measured Inference Times

Table 5 summarizes the mean and median inference times of the different feature extractor backbones on the different test systems. The times are split into the overall times and the inference time per rod which should be around 1/8 of the overall time, as each rod is inferred sequentially. Overall, the ResNet18-based regressor performed the best with 86.18 ms median inference time per frame which is significantly higher than the required approximate 16.6 ms which would result in a real-time game state detection with 60 FPS. The ResNet18-based regressor could only achieve 11.6 FPS in the median. The other regressors were at least 29.75 ms slower on the same system. Albeit utilizing a higher performing GPU, system D could not generate a median inference time below 105 ms. In contrast to the other systems, system D also showed a much higher difference between mean and median inference times. This indicates an asymmetric distribution which is probably caused by slow outliers thus introducing latency in the prediction cycle of the whole system.

Interestingly, as shown in Table 6, the ARM-based MacBook (System B) delivers the fastest inference using the own implementation. While the overall quality of prediction, as shown in subsection 6.1.1, is not on par with the other networks especially considering the rotation prediction, the own implementation could be further examined and improved in the future.

The prediction seems to be heavily CPU-bottlenecked considering an observed average GPU utilization of only 40% on the NVIDIA RTX 3080 in system C and 7% on the NVIDIA A100 on system D during testing of the ResNet18-based regressor. This is further highlighted by the faster inference time on system B and C which both have worse performing GPUs than system D but faster CPUs with more processing cores. To circumvent this in the future, a parallelization of the system could be researched. When looking at the inference times per rod, the ResNet18-based regressor would satisfy the real-time requirement with 10.77 ms median inference time per rod on system C. Even on the lower performing laptop (system B) the requirement would be met at 13.38 ms median. A parallelization would introduce some latency which must be considered. System C would have an additional 5.83 ms free to still met 60 FPS while the laptop would have 3.22 ms free. Therefore, some latency can be introduced without breaking the requirement. Additionally, the added latency e.g. for networking would

Table 5: Mean and Median inference time for different feature extractor backbones on the different systems from Table 4.

Backbone	**System**	**Inference Time (ms)**		**Inf. per Rod (ms)**		**FPS**	
		Mean	Median	Mean	Median	Mean	Median
ResNet18	A	288.51	286.27	36.06	35.78	3.47	3.49
	B	108.03	107.11	13.50	13.38	9.26	9.34
	C	88.88	86.18	11.11	10.77	11.25	11.60
	D	126.23	106.94	15.78	13.36	7.92	9.35
ResNet50	A	581.58	573.85	72.69	71.72	1.72	1.74
	B	182.79	182.45	22.84	22.80	5.47	5.48
	C	120.83	119.32	15.10	14.91	8.28	8.38
	D	130.77	114.60	16.34	14.32	7.65	8.73
MobileNetV3	A	497.21	501.49	62.14	62.58	2.01	1.99
	B	156.76	156.44	19.59	19.55	6.38	6.39
	C	117.00	115.93	14.62	14.29	8.55	8.63
	D	122.81	105.14	15.35	13.14	8.14	9.51
EfficientNetV2	A	1538.96	1496.68	192.36	187.08	0.65	0.67
	B	402.34	400.27	50.29	50.03	2.49	2.50
	C	249.31	245.97	31.16	30.74	4.01	4.07
	D	235.70	215.24	29.46	26.90	4.24	4.65

Table 6: Mean and Median inference time of the own implemented CNN on the different systems from Table 4.

System	**Inference Time (ms)**		**Inf. per Rod (ms)**		**FPS**	
	Mean	Median	Mean	Median	Mean	Median
A	113.06	112.98	14.13	14.12	8.84	8.85
B	56.27	55.57	7.03	6.94	17.77	18.0
C	71.58	71.38	8.94	8.92	13.97	14.01
D	130.16	106.65	16.27	13.33	7.68	9.38

probably be mostly linear. This would result in a longer processing time between the capturing of the frame and the data output but would possibly not influence the actual inference time of the system.

Considering the overall performance including the quality of prediction and the inference speed, the ResNet18-based regressor network shows the best results of the tested feature extractor backbones. The short training time as shown in section 5.2 is an additional benefit. The slightly better prediction quality of the ResNet50-based regressor is not enough to justify the higher inference time of an additional 33.14 ms on the same hardware. Additionally, the ResNet50-based regressor had a three times longer training time than the smaller ResNet18.

These findings lay in direct contrast to Vdovjak et al. [60] who compared ResNet, MobileNet and EfficientNet in the image-based fire classification. They conclude that the ResNet-based classifier produced worse predictions and was slower than the other two networks. Instead of a classification task, a regression task is the aim of this work. Therefore, the direct findings and used metrics by Vdovjak et al. are not comparable. While they observed worse results in the classification, the ResNet-based regressors produced the lowest MSE for all prediction tasks in this work. A possible explanation for the different inference times could be the disability to perform batch predictions. While the authors in [60] used datasets and predicted in batches, the real-time requirement forbids the usage of batching. This could potentially lower the inference speed of the MobileNet and EfficientNet CNNs.

6.2 Discussion of Results

The demonstrated results in the last section have shown that the concept is able to capture the position and rotation of the rods of a Foosball table with room for further improvement. The following limitations of the system are noted:

- The concept only works on a Foosball table with an identical setup, i.e. the figures must be black and white and include rubber stoppers at the end of the rods;
- The quality of the detection is heavily dependent to the image quality of the camera;
- The rotation of the black rods is limited to the range between 120° and 240° due to the hardware limitation on the physical Foosball table; and

- The system does not include the ball into the game state.

Due to the lack of other Foosball tables with different color configurations the system could not be tested with different colored figures. Since the manual position detection is mostly reliant on the rubber stoppers it could be sufficient to only conduct minor adaptions of thresholding values to calculate the correct positions. The color of the actual figures should not influence this procedure as long as rubber stoppers are installed. The color of the rods, which are not colored on most of the commercially available Foosball tables, is expected to have a higher influence on the position detection. If the rods were painted black, major adaptions need to be conducted. Possibly a new approach would be the better option in this case. Since the detection of the rotation is done using accelerometer sensors, the color of the figures does not influence the rotation detection.

The image quality of the camera and especially the shutter speed shows a strong influence on the detection quality. With a long shutter speed, the images will contain motion blur which results in indistinct figures as illustrated in Figure 35a. Most common webcams including the used *Logitech Brio 4K* use the shutter speed to control the lighting due to the utilization of a static aperture. The third option to control the lighting in a digital camera is the ISO which controls the luminous sensitivity of the camera sensor. A high ISO results in brighter images but introduces noise in the image. In a bad lighting scenario, a webcam will therefore extend the shutter speed and will not use an ISO level above a certain threshold. In the normal usage of a webcam where high-speed changes of the image are not common, this behavior is expected. In contrast, in a high-speed environment like an automated Foosball table, this results in too long shutter speeds and motion blur.

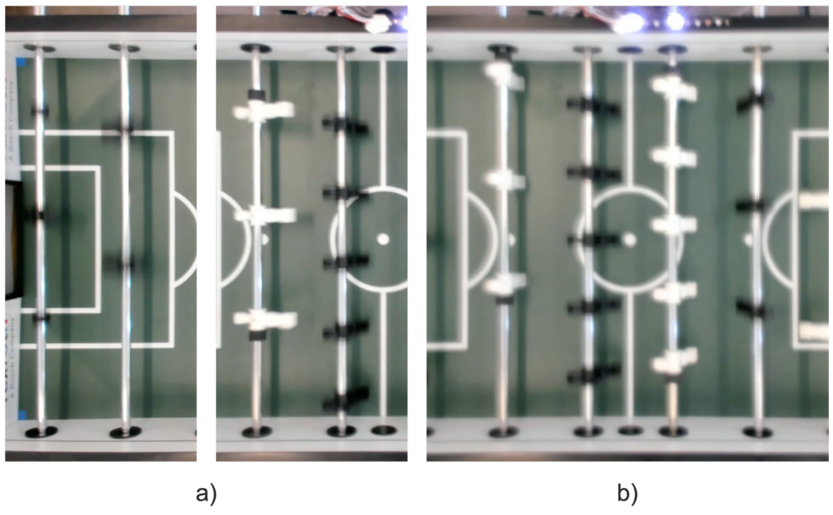


Figure 35: Example camera images with motion blur (a) due to long shutter speed and overall blur (b) due to the autofocus.

Additionally, the autofocus of the camera can introduce entirely blurred images as illustrated in Figure 35b. This is a consequence of a re-focusing of the image which could potentially be the result of the introduced motion blur and the camera thinking that the image is unclear. Again, the re-focusing is expected for the normal usage of a webcam but should not be applied on the Foosball table. In the webcam, this behavior is not controllable due to the lack of a manual focus or the ability to lock it. Therefore, it is recommended to discard the webcam and instead use a professional camera with the ability to manually control the lighting and focus.

The limitation of the rotation detection of the black rods to the range between 120° and 240° is not problematic when using the automated Foosball table on hand. Instead, this could be a problem if the system is applied to full-manual Foosball tables without the same limitation in the hardware. This can be fixed by generating a new training dataset without this limitation and re-training the regressor networks for the black figures. In the generation process, the rotation of the black rods can also be measured using the accelerometers.

As already mentioned, the detection of the ball position was not pursued due to the discarding of the object detector. In addition, the ball is already tracked by the previous work of De Blasi et al. [19] and Rohrer et al. [51].

Albeit these limitations the overall system shows promising results with room for future improvements. In section 1.2 the following main objectives were defined:

- The system should detect the game state of a Foosball table in real-time.
- The system should detect the position of all figures with high accuracy.
- The system should detect the rotation of all figures with high accuracy.
- The system should provide the data for further use in a defined and easy-to-use way.

As shown in the evaluation of the different feature extractor backbones, the implemented proof of concept could not comply with the real-time requirement as the ResNet18-based regressor, which was chosen as the overall best performing feature extractor, could only achieve 11.6 FPS. This would also be too slow for the minimum 30 FPS reported by Enos et al. [21].

The system is able to detect the position of all figures with a high accuracy on all rods. Individual figures are not detected but can be calculated using *a priori* knowledge of the rods and the detected positional shift in relation to the center of the Foosball table. This is sufficient to meet the requirement and provides the same data as the motors would provide.

The system is also able to detect the rotation of all figures with a slightly lower accuracy especially on the white figures. On average, the white goalkeeper had an absolute error of 12.64 degrees. In contrast, the black goalkeeper could achieve a mean absolute error of only 1.23 degrees. One possible reason for the difference in the prediction accuracy between the white and black figures is the error in the actual measurement during the data capturing. The black figures were measured through permanently installed sensors inside the motors without any added errors. In contrast, the white figures were measured using accelerometers which were mounted on the rods. These accelerometers introduced a mean absolute error of around 5 degrees in the data capturing process. Additionally, the white figures are not restrained to a minimal rotation range but can rotate freely. This could also introduce possible errors due to multiple rotations looking similar on the camera image.

The utilization of a ZeroMQ-based publish and subscribe system for the data output enables the usage of the detected game state by further systems with minimal effort. The whole detection system could also be outsourced to a permanently installed server to further improve the ease-of-use. Since ZeroMQ supports multiple simultaneous clients, one detection system can be used by multiple students at the same time in the future without the need for each student to set up the detection application.

Chapter 7

Conclusion & Future Work

In this work, a system was designed to capture the game state of a Foosball game on a semi-automated Foosball table. The Foosball table on hand has a white team which is played by humans and a black team which is played by a computer through DRL agents. While the position of the ball is already detected, the position and rotation of the individual figures is not. The motors controlling the rods of the black team provide position and rotation information through the OPC UA standard. The position and rotation of the white figures is not available. The game state which should be detected included the position and rotation of the white figures but also the black figures as the system should be usable on other Foosball tables to enable imitation learning.

The presented concept included the usage of MPU6050 accelerometers on GY-521 breakout boards to measure the rotation angle of the white figures and create a trustworthy ground truth. Additionally, the ADXL345 accelerometer was examined but turned out to be less accurate in the angle measurement. The accelerometers were connected to an ESP32 MCU via the I^2C bus which communicated the measured rotation angles via USB. To create a training dataset for the prediction of the position and rotation, the accelerometer values, the values reported by the motors and the image of the camera are captured and saved in separate `.mov` and `.csv` files. In previous work, the position detection of the white figures was addressed by utilizing the YOLOX object detector network with a semiautomatic labeling step using traditional CV methods. This approach showed good results on the white rods but was not adaptable to the black figures. Therefore, a new approach was presented which utilizes an end-to-end CNN based regressor network for each rod to predict the position and rotation. As the captured training data contains no information about the position of the white rods, this position is calculated using the presented color-based algorithm. Utilizing thresholding and *a priori* knowledge of the color of the rods, the position is calculated by finding the center of each rod based on either the black figures on the black rods or the black rubber stoppers on the white rods. This algorithm proofed to calculate the position of the black rods with a higher accuracy than the reported position of the motors. The most probable explanation of this is that the motors report the desired position of the rods even if the movement is not finalized and the camera image was taken before the movement was finished.

The end-to-end regressor network was implemented using five different feature extractor backbones. First, the existing networks ResNet18, ResNet50, MobileNetV3 and EfficientNetV2 were utilized with each having the classification layer replaced by a linear regression layer. Additionally, a simple CNN was self-implemented. The models were trained using transfer learning and random initialization of the replaced and self-implemented layers using the before mentioned dataset. Overall, the ResNet18-based model showed accurate results with fast inference and training times. While the ResNet50-based model predicted more accurate results on the most rods, the drawback of a longer inference and training time does not justify the minor accuracy improvements. In terms of accuracy of the position and rotation prediction, the requirements were met. The inference time however was not satisfactory as the goal of 60 FPS were not met by a huge margin. The ResNet18-based model achieved only 11.6 FPS median inference time on high end hardware (AMD Ryzen 9 5900X CPU, NVIDIA RTX 3080 GPU, 64 GB RAM).

Furthermore, an output system based on ZeroMQ was presented. The usage of a publish and subscribe messaging model enables the connection of multiple clients to the same game state detector. The game state detector does not need to run locally as ZeroMQ uses a TCP-based protocol. Therefore, a game state detector could be permanently installed on the Foosball table while mobile hardware can be used in the further automation of the table using DRL.

The presented system is able to detect the game state with high accuracy but low inference times. Therefore, the real-time requirement needs to be further addressed in the future. One possible enhancement would be the parallelization of the different regressor networks as especially the GPUs of the higher end computers were not fully utilized. Another approach in contrast to local parallelization would be the clustering of the system to multiple computers. One possible solution would be the usage of multiple NVIDIA Jetson single board computers which contain CUDA enabled GPUs. While this would introduce some networking latency, the overall performance gain should be high enough to still meet the 60 FPS real-time requirement. Another option for faster game state detection would be the usage of a tracking system where the regressor networks can use the knowledge of the last position and rotation of each rod.

In this work, a top-down approach was used to detect the game state, i.e. the hardware was set and not permanently modifiable. In the future, the game state detection can also be examined using a bottom-up approach by improving the actual hardware of the Foosball table. One example would be the switch of the camera which currently has clear drawbacks. A new camera should be able to use a manual focus and a short shutter speed. This would prevent motion blur and overall blurring which currently occurs occasionally. Additionally, the use of a high-speed camera could be tested but would also create the need for an improvement of the detector models as the current 60 FPS could not be achieved with the current setup.

Overall, the detected game state achieves good results which contribute to DRL approaches and enable imitation learning if used on another Foosball table. While this is not tested, the system should work without adaptations if the setup of the Foosball table is the same, i.e. the figures are black and white on metallic rods. The proof of concept showed limitations which should be addressed in future work. Especially the inference time of the system needs to be improved as a real-time detection is currently not possible.

Part II

Appendix

Appendix A

Evaluation metrics

In section 6.1, the evaluation metrics of the black and white goalkeeper where shown. In the following images, the evaluation metrics of the other rods are illustrated.

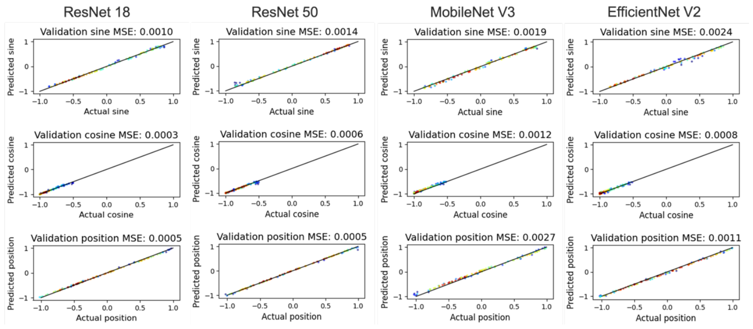


Figure 36: Prediction metrics of the different feature extractor backbones on the black defense.

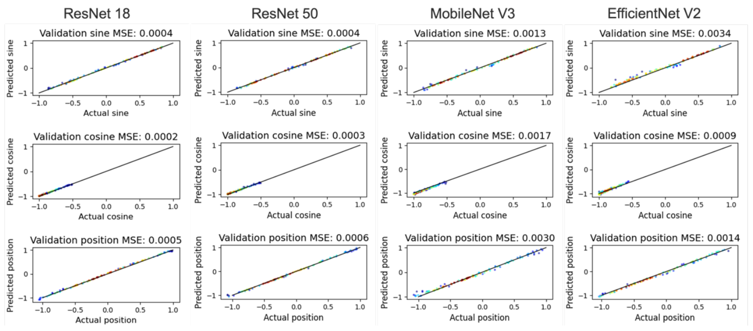


Figure 37: Prediction metrics of the different feature extractor backbones on the black midfield.

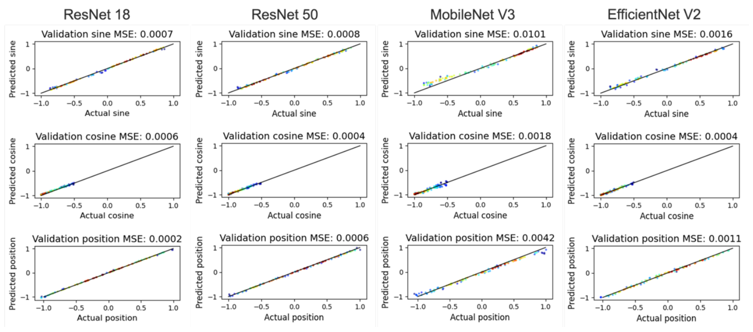


Figure 38: Prediction metrics of the different feature extractor backbones on the black striker.

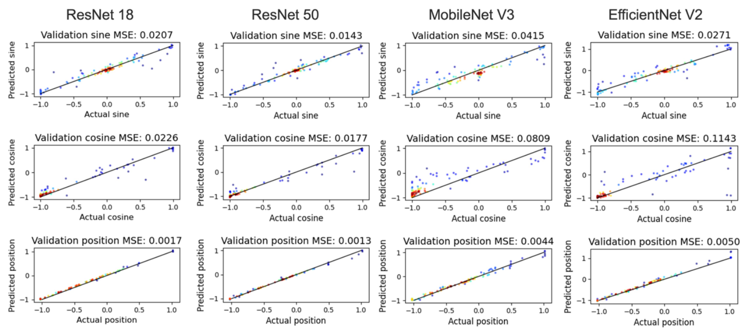


Figure 39: Prediction metrics of the different feature extractor backbones on the white defense.

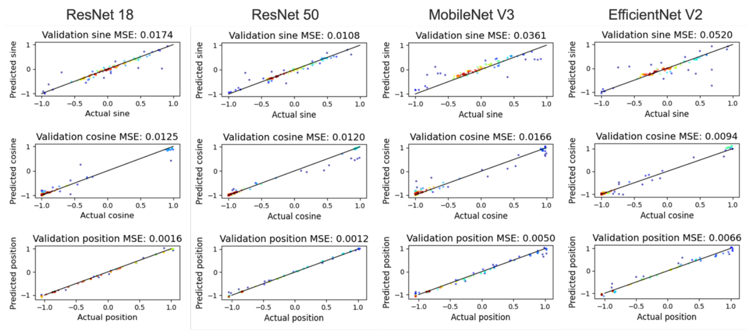


Figure 40: Prediction metrics of the different feature extractor backbones on the white midfield.

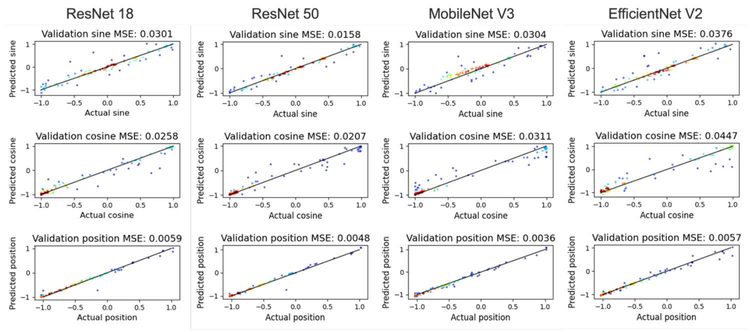


Figure 41: Prediction metrics of the different feature extractor backbones on the white striker.

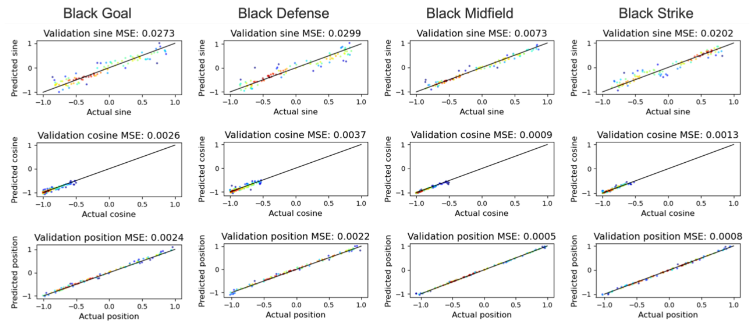


Figure 42: Prediction metrics of the own CNN architecture on the black rods.

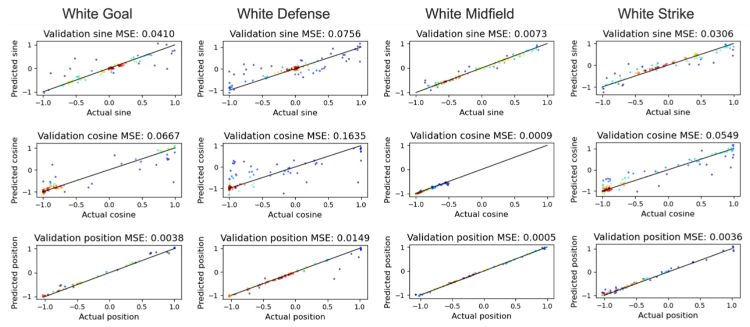


Figure 43: Prediction metrics of the own CNN architecture on the white rods.

Appendix B

Real Time State Detection of a Foosball Game Using CNN-Based Computer Vision [31]

The previous work by Horst et al. [31] is accepted for the SAI Computing Conference 2024 but not yet published. The full text of the paper is appended on the following pages.

Real Time State Detection of a Foosball Game Using CNN-Based Computer Vision

Ronny Horst, David Hagens, Elke Hergenröther, Andreas Weinmann

Darmstadt University of Applied Sciences
Schöfferstraße 3, 64295 Darmstadt
Germany

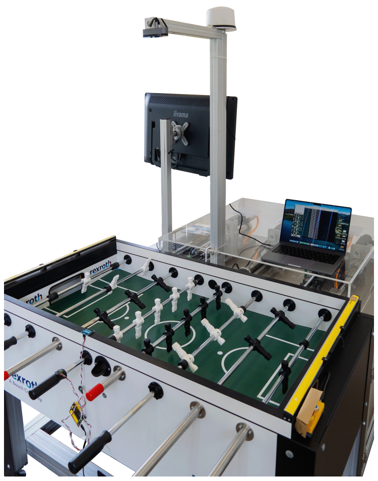
Abstract. Abstractions of the game of football serve as well-known challenges in AI research. A particularly accessible abstraction is the game of Foosball where one team is operated by an AI agent while the other side is controlled by humans. In Foosball, the dynamics can be described by a few descriptive parameters, namely the shift and rotation of the corresponding rods plus the position of the ball. In this work, we present a Computer Vision based real time game state detector in a real-world setup with an automated Foosball table (constructed by Bosch Rexroth AG). More precisely, we train an object detector network based on YOLOX to detect the positions of the figures and an image regressor network based on ResNet18 to predict the rotation angles. For the derivation of the training data we propose a semi-supervised labeling approach based on classical Computer Vision. We evaluate the proposed approach and find that our methodology works in the sense of a proof of concept.

Keywords: Foosball, game state detection, object detection, image-based regression

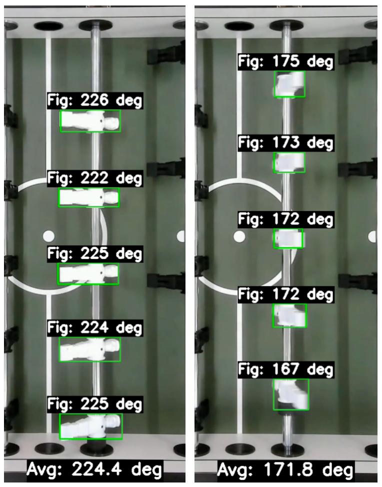
1 Introduction

Abstractions of the game of football have been used as challenges in AI research. One example is the well-known RoboCup challenge [10,14]. Another example is the game of Foosball which is the topic of the present paper. Here, one team is operated by an AI agent while the other side is controlled by humans. In our case, the AI operates the black figures; cf. Fig. 1a. In recent years, in particular deep reinforcement learning strategies have been used to train the AI agent; cf. Section 1.1 on related work below. In order to train the agent, it is essential to have access to as much information as possible on the present game state. It is important to note the whole game state can be described by the shift and rotation of the rods carrying the figures plus the position of the ball.

In this work, we aim at a Computer Vision based real time game state detector in a real-world setup with an automated Foosball table. Concretely, we use the following environment setup (see Figure 1a). (a) The black team is automated using industrial motors; cf., e.g., [3]. In particular, the position and



(a) The Foosball table setup.



(b) Game state detection result.

Fig. 1: *(a) Setup.* The Foosball table used in the present paper (constructed by Bosch Rexroth AG). The black figures are controlled by an AI agent while the white figures are controlled by humans. A camera above the table provides a top down view. *(b) Game State Detection.* Two random samples of the detected game state for the midfield rod. For each figure, both bounding box and rotation angle is predicted using deep convolutional neural networks as detailed in subsection 2.2 and subsection 2.3. Additionally, averages for the whole rod are displayed at the bottom.

rotation of the black figures are provided by sensor data of the motors and are therefore known to the AI agent. (b) The white figures are operated through human controllable rods. (c) A camera is positioned in the center above the table such that the playing field is captured in a top down perspective. (d) Accelerometers (cf. Figure 6a) are employed for angle detection of the white figures in the training stage. We note further, that there is already work on real time tracking to supply the the AI agent with the position of the ball [3,22]. In contrast to the black team and the ball, the state of the human controlled team, represented by the white figures, is not determined in our setup yet. We contribute to this issue by detecting position and angles of the white figures using a deep neural network approach.

1.1 Related Work

Game State Detection. Numerous studies have examined the utilization of computer vision techniques for real time game state detection in Foosball [1,9,23]. These experimental setups employ distinct colorized figures. They use classical color image processing exploiting the distinct color to capture the figures. Weigel et al. [26,27,28,29] describe a prototype of a semi automated Foosball table which advanced commercially as *Star-Kick*. In earlier experimental setups, these authors used a top-down camera. They track the ball and segment the distinct yellow figures controlled by the human player using classical color image processing as well. The rotation of the rods was (rudimentary) addressed by switching between an *up* and *down* position using the bounding box sizes of the figures. In later configurations, they use a bottom-up camera mounted inside the table with a half transparent playing field. They address tracking the ball, but do not consider the detection of the figures. Mohebi [16] captures the position and rotation of the figures using hardware potentiometer sensors. In contrast, Bambach et al. [2] captured the position of the figures via the Hough lines algorithm and color distributions on the rods. There, the rotation of the figures was addressed by detecting the length of the figures using pixel wise color detection and calculating an angle using pre-measured minimum and maximum length values. Janssen et al. [11] use a magnetic resonance imaging (MRI) scanner to mask the figures and track the ball position. At first, a monochrome camera is used to capture the ball position by masking the figures in real time and subtracting a static background. Despite the detection of the figures through the MRI scanner, these data were not extracted and only used to detect the ball. In [12], these authors switch to a color camera and refine their algorithm omitting the MRI scanner. In particular, they use the color information to detect the white ball against to the black and yellow figures. The detection of the figures was not considered.

Automation of Foosball. The detected game state defines a feature space which can serve as a basis for the automation of a Foosball table, in particular for training and executing an AI steering one team. Classical approaches entail rule based methodologies; e.g., [27]. Zhang et al. [30] improved the rule based automated agent of [27] using imitation learning techniques. Recently, research has shifted towards deep reinforcement learning methods to train the agent. De Blasi et al. [3] propose a simulation-based method for applying Deep Reinforcement Learning (DRL). They used a simulation to train a DRL agent to score goals with the striker rod which was afterwards tested on a physical Foosball table. Their key contribution is the demonstration of sim-to-real transfer of DRL models. In contrast, Rohrer et al. [22] aim to automate the goalkeeper using the same simulation and physical table (constructed by Bosch Rexroth AG). The DRL training was performed inside the simulation and transferred to the physical table for execution as well. Gashi et al. [5] proposed a multi-agent DRL approach for the automated goalkeeper and the opponents striker rod in the before mentioned simulation. Instead of relying on pre-defined scenarios in the training process, the two separate DRL agents were trained simultaneously while playing against each other. A transfer of the simulation-trained agent to the physical table was not performed.

1.2 Our Contribution

In this work, we propose a deep convolutional neural network based real time game state detector in a real-world setup with an automated Foosball table. In

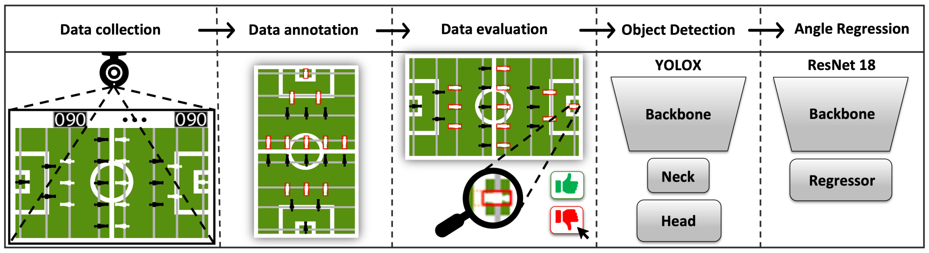


Fig. 2: Proposed training process for the detection of figures and the prediction of their rotation angles. The dataset creation process, composed of data collection, data bounding box annotation and data evaluation, is described in detail in subsection 2.1. The figure detection using YOLOX is described in subsection 2.2 and the angle regression is further described in subsection 2.3.

particular we determine the state of the human (white) team without using a hardware framework distinguishing the color of the figures and the background (as previous approaches do.) Tracking the ball was considered in [3], information on the black team is derived by sensor information of the robot. In detail, our contributions are as follows: *(i)* We propose an semi-automatic labeling pipeline using classical image processing. As a result, we obtain an annotated dataset for object detection and regression of the rotation angle for the white, human team. *(ii)* We fine-tune an object detector based on YOLOX [6] and a regressor based on ResNet18 [8] using the derived annoted data (cf. (i)) to obtain a detector for the game state of the white, human team. *(iii.)* We evalute the derived algorithms.

In contrast to existing non-learning based approaches [2,11,12], we utilize state of the art deep neural networks (YOLOX [6], ResNet18 [8]) to ensure robustness and avoid calibration requirements. To create the data set however, we utilize a similar classical image processing approach as Janssen et al. [12]. A distinct feature of our Foosball table are the white figures, such that the color is not a unique feature of the figures. Therefore, we depart from the color based approaches [1,2,23,27]. Contrary to the hardware solution by Mohebi [16] we solely used hardware sensors to create training data.

The work contributes to the derivation of a complete feature set uniquely determining the state of a Foosball game. In particular, determining position and rotation angle features from image data allows DRL-approaches to include the human controlled figures into the decision making process. Thus, our work contributes to the line of research started by De Blasi et al. [3], Rohrer et al. [22] and Gashi et al. [5] on training an AI agent using DRL-approaches.

2 Real Time State Detection

In this section we describe our methodology for achieving real time game state detection in our experimental Foosball setup, which is visualized in Figure 2.

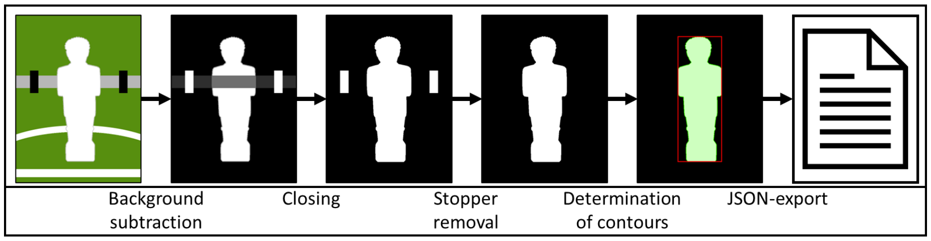


Fig. 3: Data annotation process using image processing techniques to automatically create bounding boxes for the white figures. The resulting annotations were used in subsection 2.2 to train the YOLOX detector.

To leverage modern CNN-based models, we collected image data using a 60 frames per second camera positioned above the Foosball system. We then created an annotated dataset using a preprocessing pipeline based on classical image processing techniques, followed by a semi-automatic labeling process to ensure dataset quality. Then object detection was performed using a YOLOX-network [6]. Lastly angle regression was achieved using a modified ResNet18 model [8].

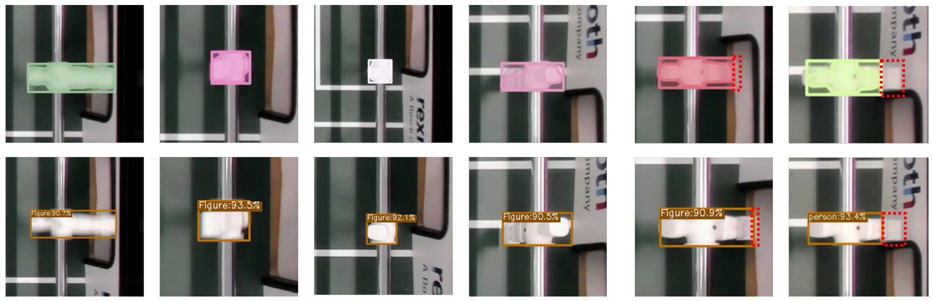
2.1 Dataset creation

The first three steps of the process, as illustrated in Figure 2, can be described as dataset creation. The resulting annotated dataset is compulsory for the later supervised figure detection.

Data collection. In the data collection step, we captured videos of the Foosball table through the mounted webcam. We recorded multiple videos on various day times and diverse weather conditions resulting in individual ambient illumination and shadows. By manually simulating various movements of the white figures in those videos, we collected a representative data set.

Data annotation. Our proposed data annotation process, as illustrated in Figure 3, employs image processing techniques and utilizes a semi automatic labeling method. The process begins by extracting the video background through temporal median filtering [19]. Morphological transformation involving closing [7] and thresholding produces a binary mask on the foreground. Utilizing the Otsu thresholding algorithm [17], rubber stoppers on the rods are removed to avoid confusion with the white figures. This process results in a foreground mask containing only the white playable figures. Lastly, bounding boxes are calculated using the border-following algorithm developed by Suzuki et al. [24]. The final annotations are saved in the COCO format [15] using the specific data structure needed for YOLOX object detection [6].

Data evaluation. An automated process can lead to incorrect annotated bounding boxes. Therefore we implemented a semi automatic labeling tool. Thus low quality video frames can be identified while simultaneously reducing human



(a) Correct bounding box prediction. (b) Incorrect prediction.

Fig. 4: Bounding box annotations (top) and the predictions by the object detector (bottom). While the top and bottom pictures are the same, the scaling of the images is slightly different. *(a) Correct Prediction.* The predicted bounding boxes surround the underlying figures precisely. *(b) Incorrect Prediction.* The erroneous bounding box annotation as shown in the top pictures were learned by the object detector resulting in incorrect bounding box predictions shown below. The red dashed square shows the missing part of the bounding box on both pictures. The object detector shows a clear sensitivity to errors if a part of the figure lies above white background like the walls of the Foosball table.

labeling costs. Furthermore, the data annotation process was improved by optimizing algorithm parameters (kernel size, threshold values etc. see section above) through multiple iterations using the insights gained through our labeling tool. Nevertheless small bounding box mistakes as seen in Figure 4b still remained.

2.2 Figure detection

Our experimental setup induces certain requirements such as real time detection and commodity hardware compatibility. Thus we used an object detection network to extract the positional information of the figures, which comply with before mentioned demands. The YOLO family demonstrated proficiency in real time object detection [20,21]. Since real time detection is a requirement for our approach, we utilize YOLOX-s [6], the smallest YOLOX model, for figure detection.

The YOLOX-s detector was fine tuned to detect the position of each white figure and create individual bounding boxes. The resulting bounding boxes were later used to extract a cutout of each figure from the original image to train the regressor as described in subsection 2.3. The network was pre-trained on the original COCO-dataset [15] and refined using approximately 4900 images for training and 1200 images for validation. To reduce the training complexity and computation expenses we solely adjusted the weights of the neck- and classification head layers, keeping the feature detector backbone weights locked.

2.3 Rotation prediction of the midfield figures

The rotation angle is a continuous variable while the YOLOX network can classify only discrete variables. Therefore we used a regressor network to predict those angles.

To estimate the angle rotation of a playable rod, an image regression network can be used to predict the sine and cosine values. As our setup includes only one top-down camera, the perspective distortion varies between the center figures and the figures at the end of the Foosball table. In order to realise a proof of concept we reduced the complexity by restricting the angle estimation to the midfield rod. To train the regressor, we created a labeled dataset of images containing the figures and their corresponding rotation angles.

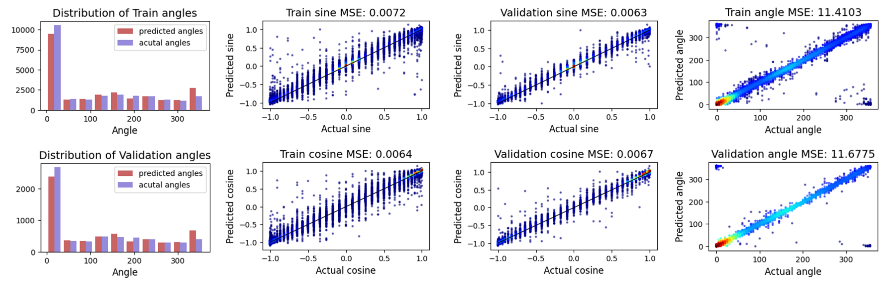
Pham et al. [18] proposed a method for calculating the tilt angle of objects using the gravity acceleration measured by an accelerometer. In our study, we employed a similar approach to measure the rotation angle of the playable figures by using a MPU6050 accelerometer connected to an Arduino Uno microcontroller. The measured angle was displayed on an OLED display, which was visible in the camera feed as outlined in Figure 6a. In this way, an optical verification of the functionality of the test setup is possible. We leveraged the Tesseract optical character recognition (OCR) engine [25] to extract the rotation values from the images. The resulting dataset consisted of 30555 images of individual figures and their rotation angle. To address misread rotation angles, we corrected outliers by calculating the mean of the preceding and following frames if the current frame deviated more than 30 degrees.

The problem of circular continuity in angle degrees was tackled through the computation of sine and cosine values of the angles. We employed a pre-trained ResNet18 model [8] and substituted the classification layer for a linear regression layer with an output dimension of two, resulting in two neurons for the prediction of the sine and cosine values of the angles. The ResNet model was used as a feature extraction backbone. We used the bounding boxes of the figures which where extracted by the YOLOX network in the previous section to cut out the figures from the whole image. Those cut outs where then used as the input of the Regressor model. To compensate for possible errors in the bounding box detection we added a padding of 20 pixels on each side of the figure which turned out to be a suitable value for the used YOLOX object detector. The model was then trained using a Mean-Squared-Error loss function and optimized with the ADAM optimizer [13].

3 Evaluation

We present the results and evaluation of our approach. Initially we perform a quantitative analysis of the object detector and rotation regressor. Subsequently, we evaluate the outcomes on a qualitative inspection.

Quantitative analysis. The YOLOX-based object detector achieved an average precision of 88.3% and an average recall of 90.7% at an Intersection-over-Union (IoU) above 0.50, which is a typical threshold for correct object detection



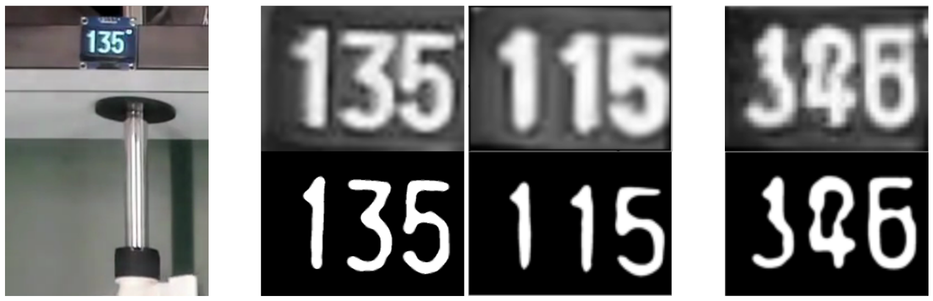
(a) Angle distrib. (b) Regressor eval. (c) Angle eval.

Fig. 5: *(a) Angle distribution.* Angle distribution of the figures in the train and test dataset. In general the data distribution is skewed towards 0°, which resembles a realistic Foosball angle distribution and explains the red spikes in the following plots. *(b) Prediction evaluation.* Comparison of the actual and predicted values of the regressor for the training and validation dataset. *(c) Calculated angle evaluation.* Comparison of the actual angles and the calculated angles based on the sine and cosine prediction. The color indicates the quantity of overlapping data points. A red coloring implies an accumulation of data points while a (dark-) blue coloring implies a less crowded region.

Table 1: Quantitative evaluation summary.

Object detection	**Avg. precision**	**Avg. recall**		**Avg. inference time**
Training data	90.1%	92.0%		4.09ms
Validation data	88.3%	90.7%		4.09ms
Rotation regression	**Angle MSE**	**Sine MSE**	**Cosine MSE**	
Training data	11.41	0.0072	0.0064	4.30ms
Validation data	11.68	0.0063	0.0067	4.30ms

[4]. On average the inference time was 4.09 ms on a Nvidia A100 GPU which conforms to the boundary of 16.6 ms to process 60 fps, which we consider as real time detection. In regards of the evaluation of the angle regression, Figure 5b presents the visualized results. The angle regression was measured as mean squared error (MSE). On the validation dataset, the sine prediction achieved a MSE of 0.0063 corresponding to an average error of 4.55° degrees. The cosine prediction achieved a MSE of 0.0067 corresponding to an average error of 4.69° degrees. The calculated angles achieved a MSE of 11.6775 corresponding to an average error of 3.417° degrees; cf. Figure 5c. However, we noted a significant spike in the number of data points with a sine value of 0 and a cosine value of 1, corresponding to an angle degree of 0°. Further investigation of the underlying data distribution was conducted, as depicted in Figure 5a, revealing that the distribution of the data was heavily skewed towards 0°, and not uniformly distributed. This can be explained by the fact that the figures are mostly aligned orthogonal to the Foosball table to defend and block the ball. On average the inference time



(a) Display pos. (b) legible numbers. (c) illegible no.

Fig. 6: *(a)* Positioning of the display of the Foosball table above the rod. In *(b)* and *(c)* example images of the displays are shown with the original gray scale images above the corresponding binaries. The majority of the images were easy to recognise as seen in *(b)*. Asynchrony between captured frames of the camera and display changes may lead to illegible input images as illustrated in *(c)*. As a result OCR errors may occur. This problem was addressed by using the mean angle between the previous and the following input images to impute missing data.

of the regressor was 4.30 ms on the same Nvidia A100 GPU. Combined with the object detector we achieved an average inference time of 8.39 ms conforming to our boundary at 60 fps. We note a significant increase in inference time if the computation was solely CPU-based at above 100 ms for each neural network thus not hitting the real time requirement. The results are summarized in Table 1.

Qualitative analysis. Concerning object detection, a qualitative inspection of random samples revealed the potential of the network to find accurate bounding boxes as shown in Figure 4a. However, it was observed that the feet of the goalkeeper are occasionally cut off. This error was also present in the training data which is illustrated in Figure 4b. This indicates that issues in the preprocessing pipeline may be the source to this challenge which should be addressed in future research as described in section 4. Manual analysis of the OCR generated annotations revealed promising results, cf. Figure 6b. The asynchronous behavior between the displays and the camera led to the occasional encountering of ambiguous or illegible angles, as exemplified in Figure 6c. This problem has been solved by imputation of the illegible angles using the average of the surrounding frames. The inspection of random samples, cf. Figure 1b, showed promising average angle results while containing only small deviations at each figure.

4 Conclusion and future work

In this paper, we have developed a CNN-based real time game state detector for Foosball; in particular, we detect the position and rotation of the human-steered (white) figures from camera data. To this end, we first developed a semi-automatic labeling scheme using a classical image processing pipeline. As a result, we obtained a verified labeled dataset consisting of 6111 images with annotated bounding boxes of each individual figure. Additionally, we obtained a dataset consisting of 30555 images of single figures with the measured rotation angles by utilizing a MPU6050 accelerometer. Using those curated datasets we fine tuned a YOLOX model for figure detection and a ResNet18 based regression model to predict rotation angles. The evaluation on the test dataset showed promising results for both target variables. Our work extends the research of De Blasi et al. [3] and Rohrer et al. [22] by providing the game state as a feature set.

A first topic of future research is to address limitations through incorrect training data; see Figure 4b. To this end, we plan to further develop the classical image processing pipeline used (cf. Figure 3) to get higher quality of the labelled data. A further next step is to combine the figure detection and rotation regression into one compact model. A related interesting point is to also apply our scheme for the black figures, and then both to compare and potentially combine the image based results with the robot-sensor results. Further speedup is also desirable since professional Foosball players have haptical perception rates of up to 1000 Hz.

References

1. Aeberhard, M., Connelly, S., Tarr, E., Walker, N.: Single player foosball table with an autonomous opponent. Georgia Tech, Elect. and Comp. Engineering (2007)
2. Bambach, S., Lee, S.: Real-time foosball game state tracking. Tech. rep., School of Informatics and Computing, Indiana University (2012)
3. De Blasi, S., Klöser, S., Müller, A., Reuben, R., Sturm, F., Zerrer, T.: Kicker: An industrial drive and control foosball system automated with deep reinforcement learning. Journal of Intelligent & Robotic Systems **102**(1), 20 (2021). https://doi.org/10.1007/s10846-021-01389-z
4. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The Pascal visual object classes (voc) challenge. Int. J. Computer Vision **88**, 303–338 (2010)
5. Gashi, A., Hergenröther, E., Grieser, G.: Efficient training of foosball agents using multi-agent competition. Computing Conference 2023, London (2023)
6. Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J.: Yolox: Exceeding Yolo series in 2021. arXiv preprint (2021). https://doi.org/10.48550/ARXIV.2107.08430
7. Gonzalez, R., Woods, R.: Digital Image Processing. Pearson (2017)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE CVPR 2016. pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
9. Hernández, N., Rebolledo, J., Torres, J., Carvajal, G., Vargas, F.: Design of an experimental platform for the automation of the goalkeeper of a foosball table. In: 2019 IEEE CHILECON. pp. 1–7. IEEE (2019). https://doi.org/10.1109/CHILECON47746.2019.8988113
10. Hong, C., Jeong, I., Vecchietti, L.F., Har, D., Kim, J.H.: AI world cup: robot-soccer-based competitions. IEEE Transactions on Games **13**(4), 330–341 (2021). https://doi.org/10.1109/TG.2021.3065410

11. Janssen, R., de Best, J., van de Molengraft, R.: Real-time ball tracking in a semi-automated foosball table. In: RoboCup 2009: Robot Soccer World Cup XIII 13. pp. 128–139. Springer (2010). https://doi.org/10.1007/978-3-642-11876-0_12
12. Janssen, R., Verrijt, M., de Best, J., van de Molengraft, R.: Ball localization and tracking in a highly dynamic table soccer environment. Mechatronics **22**(4), 503–514 (2012). https://doi.org/10.1016/j.mechatronics.2012.02.009
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
14. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for ai. AI magazine **18**(1), 73 (1997). https://doi.org/10.1609/aimag.v18i1.1276
15. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: Common objects in context. In: Proceedings ECCV 2014. pp. 740–755. Springer (2014)
16. Mohebi, D.: The study of semi-automated foosball table. Tampere University of Applied Sciences, Degree Program in Mechanical Engineering (2022)
17. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics **9**(1), 62–66 (1979). https://doi.org/10.1109/TSMC.1979.4310076
18. Pham, D.A., Pham, T.N.: Determination of a tilt angle for the automatic balancing system with the inertial measurement unit mpu6050. In: Proceedings of the AMAS2021. pp. 349–354. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99666-6_52
19. Piccardi, M.: Background subtraction techniques: a review. In: 2004 IEEE International Conference on Systems, Man and Cybernetics. vol. 4, pp. 3099–3104 vol.4 (2004). https://doi.org/10.1109/ICSMC.2004.1400815
20. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the CVPR. pp. 779–788 (2016). https://doi.org/10.1109/CVPR.2016.91
21. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint (2018). https://doi.org/10.48550/ARXIV.1804.02767
22. Rohrer, T., Samuel, L., Gashi, A., Grieser, G., Hergenröther, E.: Foosball table goalkeeper automation using reinforcement learning. LWDA pp. 173–182 (2021)
23. Stefani, J.R., Herpy, A.J., Jaeger, B.G., Haydon, K.S., Hamel, D.A.: Automated foosball table. California Polytechnic State University, Mech. Eng. Dep. (2014)
24. Suzuki, S., Abe, K.: Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing **30**(1), 32–46 (1985). https://doi.org/10.1016/0734-189X(85)90016-7
25. Tesseract Authors: GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine . https://github.com/tesseract-ocr/tesseract, accessed: 03/20/2023
26. Weigel, T.: Kiro - a table soccer robot ready for the market. Proceedings IEEE Int. Conf. Robotics and Automation **2005**, 4266–4271 (2005). https://doi.org/10.1109/ROBOT.2005.1570776
27. Weigel, T., Nebel, B.: Kiro – an autonomous table soccer player. In: Kaminka, G., Lima, P., Rojas, R. (eds.) RoboCup 2002: Robot Soccer World Cup VI. pp. 384–392. Springer (2003). https://doi.org/10.1007/978-3-540-45135-8_34
28. Weigel, T., Rechert, K., Nebel, B.: Behavior recognition and opponent modeling for adaptive table soccer playing. In: Furbach, U. (ed.) KI 2005: Advances in AI. pp. 335–350. Springer, Berlin (2005). https://doi.org/10.1007/11551263_27

29. Weigel, T., Zhang, D., Rechert, K., Nebel, B.: Adaptive vision for playing table soccer. In: Biundo, S., Frühwirth, T., Palm, G. (eds.) KI 2004: Advances in AI. pp. 424–438. Springer (2004). https://doi.org/10.1007/978-3-540-30221-6_32
30. Zhang, D., Nebel, B.: Learning a table soccer robot a new action sequence by observing and imitating. In: Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. vol. 3, pp. 61–66. AAAI Press (2007). https://doi.org/10.1609/aiide.v3i1.18784

References

1. MPU-6050 | TDK InvenSense. Retrieved October 22, 2023 from https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/
2. ADXL345 Datasheet and Product Info | Analog Devices. Retrieved October 22, 2023 from https://www.analog.com/en/products/adxl345.html
3. ESP32 Wi-Fi & Bluetooth MCU | Espressif Systems. Retrieved October 22, 2023 from https://www.espressif.com/en/products/socs/esp32
4. KickerAI MPSE / KickerAgent · GitLab. Retrieved November 9, 2023 from https://code.fbi.h-da.de/kickerai-mpse/kicker-agent
5. torchvision — Torchvision 0.16 documentation. Retrieved November 10, 2023 from https://pytorch.org/vision/stable/index.html
6. OpenCV - Open Computer Vision Library. Retrieved November 19, 2023 from https://opencv.org/
7. GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine . Retrieved November 13, 2023 from https://github.com/tesseract-ocr/tesseract
8. GPU Database | TechPowerUp. Retrieved November 13, 2023 from https://www.techpowerup.com/gpu-specs/
9. Michael Aeberhard, Shane Connelly, Evan Tarr, and Nardis Walker. 2007. Single player foosball table with an autonomous opponent. *Georgia Tech, Elect. and Comp. Engineering.*
10. N. Aggarwal and W. C. Karl. 2006. Line detection in images through regularized hough transform. *IEEE Transactions on Image Processing* 15, 3: 582–591. https://doi.org/10.1109/TIP.2005.863021
11. Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. 2019. *Real-time rendering.* AK Peters/crc Press.
12. Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. 2017. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186
13. Neena Aloysius and M. Geetha. 2017. A review on deep convolutional neural networks. In *2017 international conference on communication and signal processing (ICCSP)*, 0588–0592. https://doi.org/10.1109/ICCSP.2017.8286426

14. Sven Bambach and Stefan Lee. 2012. *Real-time foosball game state tracking*. School of Informatics; Computing, Indiana University.

15. Matevž Bošnak and Gregor Klančar. 2020. Fast and reliable alternative to encoder-based measurements of multiple 2-DOF rotary-linear transformable objects using a network of image sensors with application to table football. *Sensors* 20, 12. https://doi.org/10.3390/s20123552

16. Andrew Butterfield, Gerard Ekembe Ngondi, and Anne Kerr. 2016. *A dictionary of computer science*. Oxford University Press.

17. Mark Claypool. 2005. The effect of latency on user performance in real-time strategy games. *Computer Networks* 49, 1: 52–70. https://doi.org/10.1016/j.comnet.2005.04.008

18. James Davis, Yi-Hsuan Hsieh, and Hung-Chi Lee. 2015. Humans perceive flicker artifacts at 500 hz. *Scientific Reports* 5, 1: 7861. https://doi.org/10.1038/srep07861

19. Stefano De Blasi, Sebastian Klöser, Arne Müller, Robin Reuben, Fabian Sturm, and Timo Zerrer. 2021. KIcker: An industrial drive and control foosball system automated with deep reinforcement learning. *Journal of Intelligent & Robotic Systems* 102, 1: 20. https://doi.org/10.1007/s10846-021-01389-z

20. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. https://doi.org/10.1109/CVPR.2009.5206848

21. Nathaniel Enos, Patrick Fenelon, Skyler Goodell, and Nicholas Phillips. 2012. Football operator and optical soccer engine (FOOSE). *University of Central Florida, Department of Electrical and Computer Engineering.*

22. Christopher J Fisher. 2010. Using an accelerometer for inclination sensing. *AN-1057, Application note, Analog Devices*: 1–8.

23. Luciano Floridi. 2021. Digital time: Latency, real-time, and the onlife experience of everyday time. *Philosophy & Technology* 34, 3: 407–412. https://doi.org/10.1007/s13347-021-00472-5

24. Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* 36, 4: 193–202.

25. Adriatik Gashi, Elke Hergenröther, and Gunter Grieser. 2023. Efficient training of foosball agents using multi-agent competition. In *Intelligent computing. SAI 2023*, 472–492. https://doi.org/10.1007/978-3-031-37717-4_30

26. Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. 2021. YOLOX: Exceeding YOLO series in 2021. Retrieved from https://arxiv.org/abs/2107.08430

27. Juan David Gutierrez-Franco, John Inlow, Jesse Graham, and Larry Huang. 2013. Automated foosball table. *California Polytechnic State University, Mech. Eng. Dep.*

28. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. Retrieved from https://arxiv.org/abs/1512.03385

29. Nicolás Hernández, Javier Rebolledo, Javier Torres, Gonzalo Carvajal, and Francisco Vargas. 2019. Design of an experimental platform for the automation of the goalkeeper of a foosball table. In *2019 IEEE CHILECON*, 1–7. https://doi.org/10.1109/CHILECON47746.2019.8988113
30. Dominik Honegger, Helen Oleynikova, and Marc Pollefeys. 2014. Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, 4930–4935. https://doi.org/10.1109/IROS.2014.6943263
31. Ronny Horst, David Hagens, Elke Hergenröther, and Andreas Weinmann. 2024. Real time state detection of a foosball game using CNN-based computer vision. *SAI Computing Conference, London (accepted).*
32. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. Retrieved from https://arxiv.org/abs/1704.04861
33. Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. Retrieved from https://arxiv.org/abs/1905.02244
34. Rob Janssen, Jeroen de Best, and René van de Molengraft. 2010. Real-time ball tracking in a semi-automated foosball table. In *RoboCup 2009: Robot soccer world cup XIII 13*, 128–139. https://doi.org/10.1007/978-3-642-11876-0/_12
35. Rob Janssen, Mark Verrijt, Jeroen de Best, and René van de Molengraft. 2012. Ball localization and tracking in a highly dynamic table soccer environment. *Mechatronics* 22, 4: 503–514. https://doi.org/10.1016/j.mechatronics.2012.02.009
36. Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. Retrieved from https://arxiv.org/abs/1412.6980
37. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. 1997. RoboCup: A challenge problem for AI. *AI Magazine* 18, 1. https://doi.org/10.1609/aimag.v18i1.1276
38. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25.
39. Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. 2020. Google research football: A novel reinforcement learning environment. Retrieved from https://arxiv.org/abs/1907.11180
40. Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).*

41. Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. 1989. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems* 2.

42. Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. 2022. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems* 33, 12: 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

43. Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Proceedings ECCV 2014*, 740–755.

44. Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. 2009. *OPC unified architecture.* Springer Science & Business Media.

45. Dani Mohebi. 2022. The study of semi-automated foosball table. *Tampere University of Applied Sciences, Degree Program in Mechanical Engineering.*

46. OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with large scale deep reinforcement learning. Retrieved from https://arxiv.org/abs/1912.06680

47. Nobuyuki Otsu. 1979. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics* 9, 1: 62–66. https://doi.org/10.1109/TSMC.1979.4310076

48. Nathan Otterness, Ming Yang, Sarah Rust, Eunbyung Park, James H. Anderson, F. Donelson Smith, Alex Berg, and Shige Wang. 2017. An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads. In *2017 IEEE real-time and embedded technology and applications symposium (RTAS)*, 353–364. https://doi.org/10.1109/RTAS.2017.3

49. M. Piccardi. 2004. Background subtraction techniques: A review. In *2004 IEEE international conference on systems, man and cybernetics*, 3099–3104 vol.4. https://doi.org/10.1109/ICSMC.2004.1400815

50. Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. 2012. Real-time computer vision with OpenCV. *Commun. ACM* 55, 6: 61–69. https://doi.org/10.1145/2184319.2184337

51. Tobias Rohrer, Ludwig Samuel, Adriatik Gashi, Gunter Grieser, and Elke Hergenröther. 2021. Foosball table goalkeeper automation using reinforcement learning. *LWDA*: 173–182.

52. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).*

53. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529, 7587: 484–489. https://doi.org/10.1038/nature16961

54. Mel Slater, Anthony Steed, and Yiorgos Chrysanthou. 2002. *Computer graphics and virtual environments: From realism to real-time*. Pearson Education.

55. Jim R Stefani, Alex J Herpy, Brett Gordon Jaeger, Kevin S Haydon, and Derek Alan Hamel. 2014. Automated foosball table. *California Polytechnic State University, Mech. Eng. Dep.*

56. Eli Stevens, Luca Antiga, and Thomas Viehmann. 2020. *Deep learning with PyTorch*. Manning Publications.

57. Satoshi Suzuki and Keiichi Abe. 1985. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing* 30, 1: 32–46. https://doi.org/10.1016/0734-189X(85)90016-7

58. Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking model scaling for convolutional neural networks. Retrieved from https://arxiv.org/abs/1905.11946

59. Mingxing Tan and Quoc V. Le. 2021. EfficientNetV2: Smaller models and faster training. Retrieved from https://arxiv.org/abs/2104.00298

60. Kresimir Vdovjak, Petar Maric, Josip Balen, Ratko Grbic, Davor Damjanovic, and Matej Arlovic. 2021. Modern CNNs comparison for fire detection in RGB images. In *Proceedings of the 17th international conference on machine learning and data mining MLDM 2022.*

61. Thilo Weigel. 2005. KiRo - a table soccer robot ready for the market. *Proceedings IEEE Int. Conf. Robotics and Automation* 2005: 4266–4271. https://doi.org/10.1109/ROBOT.2005.1570776

62. Thilo Weigel and Bernhard Nebel. 2003. KiRo – an autonomous table soccer player. In *RoboCup 2002: Robot soccer world cup VI*, 384–392. https://doi.org/10.1007/978-3-540-45135-8/_34

63. Thilo Weigel, Klaus Rechert, and Bernhard Nebel. 2005. Behavior recognition and opponent modeling for adaptive table soccer playing. In *KI 2005: Advances in AI*, 335–350. https://doi.org/10.1007/11551263/_27

64. Thilo Weigel, Dapeng Zhang, Klaus Rechert, and Bernhard Nebel. 2004. Adaptive vision for playing table soccer. In *KI 2004: Advances in AI*, 424–438. https://doi.org/10.1007/978-3-540-30221-6/_32

65. Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the european conference on computer vision (ECCV)*.

66. Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Computer vision – ECCV 2014*, 818–833.

67. Dapeng Zhang and Bernhard Nebel. 2007. Learning a table soccer robot a new action sequence by observing and imitating. In *Proceedings of the third AAAI conference on artificial intelligence and interactive digital entertainment*, 61–66. https://doi.org/10.1609/aiide.v3i1.18784