



**Frankfurt University of Applied Sciences**  
Faculty of Computer Science and Engineering

# A Continuous Integration system with diversified opponents and dynamic team configurations for RoboCup 2D

---

Ein Continuous Integration System mit diversifizierten Kontrahenten und dynamischen Teamkonfigurationen für RoboCup 2D

Master Thesis by

ALEXANDER JULIAN VIETH

Date of submission: June 16, 2022

1. Review: Prof. Dr. Eicke Godehardt
2. Review: Prof. Dr. Thomas Gabel

## DECLARATION

---

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig 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.

*Frankfurt am Main, June 15, 2022*

A handwritten signature in black ink that reads "Alexander Vieth". The signature is written in a cursive, slightly slanted style.

---

Alexander Julian Vieth

## ABSTRACT

---

The team for the Robotic World Cup Initiative (RoboCup) of the Frankfurt University of Applied Sciences (FRA-UNited) is utilizing a Continuous Integration Environment (CI) to automatically run 1000 games of 2D Simulated soccer against a previous world cup winner every night. This process allows for a continuous assessment of FRA-UNited's performance against a high-level competitor but runs the risk of overfitting against a specific opponent. A potential risk of this approach is that even if the performance against the training team improves, some behavioral faults which might exist against other teams will remain undetected, thus degrading the overall team performance. In this Thesis the existing CI system is reworked to minimize the risk of overfitting and to allow for a more robust analysis of FRA-UNited's general performance by allowing for diversification of opponents and defining arbitrary team configuration values. This system is then used to compare the previous assumed performance against a new broader performance, as well as analyzing the consequences of changing certain team configuration values.

## ZUSAMMENFASSUNG

---

Das Team für die Robotic World Cup Initiative (RoboCup) der Frankfurt University of Applied Sciences (FRA-UNited) verwendet ein 'Continuous Integration Environment' (CI) um jede Nacht automatisch 1000 2D-Fußballspiele gegen einen früheren Weltmeister zu spielen. Dieser Prozess ermöglicht einen kontinuierlichen Einblick in die Leistungsfähigkeit von FRA-UNited gegen eine hochrangige Mannschaft, ist aber aufgrund des gleichbleibenden Gegners anfällig für 'overfitting'. So besteht das Risiko, dass auch wenn sich die Leistungsfähigkeit gegen das Trainingsteam verbessert, einige Schwachpunkte im Teamverhalten unentdeckt bleiben, was die Gesamtleistung mindern könnte. Im Rahmen dieser Thesis wurde das bestehende CI System mit dem Ziel erneuert 'overfitting' entgegenzuwirken und durch diversifizierte Gegner sowie das Definieren von beliebigen Teamkonfigurationsparametern einen besseren Einblick in die allgemeine Leistungsfähigkeit von FRA-UNited zu ermöglichen. Dieses System wurde anschließend verwendet die bisherige angenommene Leistungsfähigkeit mit einer neuen allgemeineren zu vergleichen. Zusätzlich wurden die Auswirkungen von Änderungen der Teamkonfigurationsparametern analysiert.

## CONTENTS

---

<b>I</b>	<b>THESIS</b>	<b>1</b>
<b>1</b>	<b>INTRODUCTION AND MOTIVATION</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Motivation . . . . .	3
<b>2</b>	<b>RELATED WORK</b>	<b>5</b>
2.1	Helios . . . . .	5
<b>3</b>	<b>BACKGROUND</b>	<b>6</b>
3.1	Introduction . . . . .	6
3.2	Current Architecture . . . . .	6
3.2.1	Introduction . . . . .	6
3.2.2	Jenkins Server . . . . .	7
3.2.3	Grails Webserver . . . . .	8
3.2.4	React Web Application . . . . .	8
3.2.5	Data Model . . . . .	9
3.2.6	Database . . . . .	9
3.2.7	Test Computers . . . . .	10
3.2.8	RoboCup Tournament Software . . . . .	11
3.3	Adjacent Work . . . . .	11
3.4	Limitations . . . . .	12
<b>4</b>	<b>IMPLEMENTATION</b>	<b>13</b>
4.1	Introduction . . . . .	13
4.2	General Concept . . . . .	13
4.3	Project Setup . . . . .	14
4.4	Arbitrary Matchups . . . . .	14
4.5	Uploading And Managing Arbitrary Team Binaries . . . . .	21
4.6	Arbitrary Config Parameter For FRA-UNited . . . . .	24
4.7	Matches And Match Statistics . . . . .	26
4.8	Web Application . . . . .	29
4.9	Additional Work . . . . .	29
4.10	CI Workflow . . . . .	32
4.11	Database Model . . . . .	33
<b>5</b>	<b>RESULTS</b>	<b>34</b>
5.1	Introduction . . . . .	34
5.2	Measured Datapoints . . . . .	34
5.3	Control Experiment . . . . .	35
5.3.1	Setup . . . . .	35
5.3.2	Results . . . . .	36
5.4	Comparing Multiple Matchups . . . . .	36
5.4.1	Setup . . . . .	37
5.4.2	Expectation . . . . .	37
5.4.3	Results . . . . .	37
5.5	Testing Config Value Significance . . . . .	40
5.5.1	Setup . . . . .	40
5.5.2	Control Experiment . . . . .	40
5.5.3	Results . . . . .	41

6	FUTURE WORK	44
6.1	Forking HLM . . . . .	44
6.2	Login and Security . . . . .	44
6.3	Config values for individual teams . . . . .	44
6.4	Log file analyzer . . . . .	44
6.5	Containerization of the python analysis server . . . . .	44
6.6	Log file player . . . . .	44
7	CONCLUSION	45
II	APPENDIX	46
	BIBLIOGRAPHY	47

## LIST OF FIGURES

Figure 1	HELIOS2021 Training Environment[20]	5
Figure 2	Current CI workflow	7
Figure 3	Current Database Model	10
Figure 4	Routes for the current protocol	14
Figure 5	Matchup Creator - Fixed Mode User Interface	15
Figure 6	Matchup Creator - Commit Mode User Interface	15
Figure 7	Route for protocol creation	16
Figure 8	Example Matchup data in protocol JSON	16
Figure 9	HLM Config Template	17
Figure 10	CI Step - Fetching teams	18
Figure 11	HLM match result file - Match.yml	19
Figure 12	Grails Matchup Classes	20
Figure 13	Matchup Database Tables	20
Figure 14	Team Zip File Contents	21
Figure 15	Team Upload User Interface	22
Figure 16	Grails Team Class	22
Figure 17	Team Database Table	23
Figure 18	Team Inspector Page	23
Figure 19	Routes For Team Management	23
Figure 20	Config Parameter User Interface	24
Figure 21	Grails Config Classes	25
Figure 22	Config Database Tables	25
Figure 23	Routes For Protocol Results	26
Figure 24	Protocol Results - Menu with 3 matchups selected	28
Figure 25	Protocol Results - Plots	28
Figure 26	Local Python Server Routes	30
Figure 27	Interesting Matchups - Menu	31
Figure 28	Interesting Matchups - Tables	31
Figure 29	New CI Workflow	32
Figure 30	New Database Model	33
Figure 31	Win rate VS <i>CYRUS2019</i>	35
Figure 32	Fouls VS <i>CYRUS2019</i>	35
Figure 33	Team Behavior vs <i>CYRUS2019</i>	36
Figure 34	Goal Timestamp Histogram vs <i>CYRUS2019</i>	36
Figure 35	Offsides VS <i>HfutEngine</i>	38
Figure 36	Offsides VS <i>CYRUS</i>	38
Figure 37	Offsides VS <i>Alice</i>	38
Figure 38	Offsides VS <i>YuShan</i>	38
Figure 39	Team Behavior vs All	39
Figure 40	Team Behavior vs <i>CYRUS</i>	39
Figure 41	Team Behavior vs <i>YuShan</i>	39
Figure 42	Team Behavior vs <i>Alice</i>	39
Figure 43	Team Behavior vs <i>HfutEngine</i>	39
Figure 44	Age 3 - Tackles	41
Figure 45	Age 3 - Passes	41
Figure 46	Goal Timestamp Histogram - Age 0	42

Figure 47      Goal Timestamp Histogram - Age 3   . . . . . 43

Figure 48      Goal Timestamp Histogram - Age 4   . . . . . 43



## LIST OF TABLES

---

Table 1	Match Datapoints . . . . .	9
Table 2	New Match Datapoints . . . . .	27
Table 3	Results for <i>foresee_opponents_max_age</i> for value 0 to 5 . . . . .	41

## ACRONYMS

---

CI	Continuous Integration
API	Application Programming Interface
GORM	Grails Object Relational Mapping
ML	Machine Learning
VM	Virtual Machine
OS	Operating System

Part I

THESIS

## INTRODUCTION AND MOTIVATION

---

### 1.1 INTRODUCTION

The Robotic World Cup Initiative (RoboCup) is a long running international landmark project with the goal of surpassing the human world cup soccer team by the middle of the 21st Century[12]. Such an endeavor spans over a list of research fields including robotics and artificial intelligence (AI) which contribute continuously to the landmark goal. RoboCup consists of various disciplines, such as RoboCupRescue, RoboCupSoccer and RoboCupIndustrial, which all host a league of their own. For this Thesis only the RoboCup Soccer discipline is of relevance, which is itself divided into different leagues, based on the underlying physical properties of the players. Examples are the Humanoid League, the Standard Platform League, and the Simulation League. While significant progress has been made across all disciplines and subsequent leagues[16][7][4] since RoboCups inception in 1997[11], this Thesis will only focus on the 2D Simulation League to which the Frankfurt University of Applied Sciences contributes to with the development of its own team *FRA-UNited*.

As the successor of the Brainstormers team, which was established in 1998 by Martin Riedmiller and has been discontinued since 2010, FRA-UNited has fueled various research topics in the fields of reinforced machine learning, multi-agent systems as well as artificial intelligence and has been participating in RoboCup's 2D simulation league since 2016[8], placing second in 2017[22].

The 2D simulation league requires each team to provide a coach and 11 player agents, all of which will operate autonomously. A match is usually divided into 6000 update cycles (ticks) where each tick every player receives noisy perception data for each of the simulated optical, auditory, and haptic sensors to act upon. To facilitate these matches, the simulation league utilizes a *RoboCup Soccer Server* (rcsserver), which - as of this writing - is maintained and developed by members of Japan's top team *HELIOS*, in an open source project[1]. The server saves every performed action, as well as the connection status for each player in log files, which can be used to replay a given match and/or create statistics on a match's progression. Depending on the configuration of the soccer server, the number of update cycles can vary, either due to custom playtime settings or due to overtime and penalty shootouts. In the context of this thesis, a 6000-cycle match is assumed.

This Thesis aims to improve the capabilities of an existing continuous integration system which performs analysis, processing and displaying of large quantities of match log files to enable a more accurate assessment on the quality of FRA-UNited's performance. Accurate assessment in this case refers to a representative and unbiased view on a team's performance, measured by win rate, goal distribution and other statistics, on the basis of a large set of matches.

To validate, test and analyze FRA-UNited's progression over the course of development, a continuous integration (CI) system has been implemented, along with a web-based user interface. The CI system automatically plays 1000 matches with the newest version of FRA-UNited against the current world cup champion *CYRUS*[13], summarizes each match into a JSON file and aggregates the resulting match statistics on a team version basis. A web-based interface allows the user to inspect various datapoints regarding individual games and all matches played under a selected version of FRA-UNited using a set of plots. Work has also been done to implement an outlier finder system, which should be used by the CI to automatically find anomalous matches. Allani[3] also demonstrated such a system for finding outlier in a large set of match statistics, which will be discussed in later sections. For this thesis, the goal was to expand the analytic capabilities provided by the CI to gauge the performance of FRA-UNited more accurately. Performance - in this case - is not necessarily defined by the win rate alone but can also describe the quality of datapoints like pass chains or ball possession, depending on the target of the analysis.

## 1.2 MOTIVATION

While the current CI Environment certainly is suitable for assessing FRA-UNited's performance to some degree of confidence (since successfully competing against the world champion could indicate good overall performance), it is limited in describing team characteristics in a general scope. The confidence in the current assessment depends in large parts on the underlying assumption that team performance has transitive properties, such that if team *A* beats team *B* and team *C* beats *A*, the implication is that *C* also beats *B*. This property, in general, holds true for real soccer, since the top ranking teams usually beat the teams in the lower brackets, indicated by the observable trend in point distribution in (for example) the German Bundesliga [6]. The consistency in performance for teams with comparable rankings however is more volatile in real soccer, as well as in RoboCup. The 2021/2022 Bundesliga results show some weakness in the transitivity: the eventual frontrunner *FC Bayern München* tied to and lost against *Borussia M'gladbach*, which ended up in 10th place and lost to 7th place *1. FC Köln* twice[21]. *1. FC Köln* in turn, has lost to *FC Bayern München* twice. This is evidence that individual team performance is dependent on the given matchup, along with numerous other outside variables (player health, motivation, strategy etc.), so one of the goals of this Thesis is to enable and analyse multiple matchups and team configurations with the ultimate goal of building a better understanding of FRA-UNited's general performance.

Training and validating against the same team might not prepare well against other teams of similar ranking. This problem is called *overfitting* and is generally not desirable, as over-specialization can make a team vulnerable to small changes in enemy team behaviour. To combat *overfitting* and to allow for more control over the training process, this Thesis expands upon the current CI system with arbitrary matchups, adding matchup-based filtering when viewing the results and enabling more team configurability, all while preserving the CI characteristics of automatically fetching the newest FRA-UNited version every night. This work also compares the performance of FRA-UNited by measuring win rate, goal distribution and other datapoints for different configurations and matchups in the new system against the results from the current system.

## RELATED WORK

Other 2D RoboCup teams have presented their approach to solve the training and performance assessment problem. Generally a large amount of matches is required to reduce the influence of chance on the result, as every game contains some amount of randomness. In RoboCup 2D, a player agent has a limited time frame in which a decision on the next move has to be made. This has the side effect of placing an exact time requirement for a match to play out - encouraging a distributed approach to playing large amounts of matches in parallel.

### 2.1 HELIOS

Team HELIOS is a joint team of Japan's Okayama and Osaka Prefecture University and has been a top ranking member of the RoboCup 2D ladder[2][23][24]. In their 2021 team description paper, they describe a performance evaluation system used for RoboCup2021[20] utilizing SlackBot, Amazon S3 and Google Sheets. A user can create a job with branch and opponent information as well as the number of games to a slackbot. A server then assigns client PCs with the jobs depending on CPU load. Client PCs with high load are considered *busy* and are not assigned jobs. Client PCs run the assigned games, analyze the resulting log files into a CSV file and push them to a shared storage (Dropbox and Amazon S3). The resulting CSV files are aggregated into a Google Sheet, which the user can use to evaluate the performance.

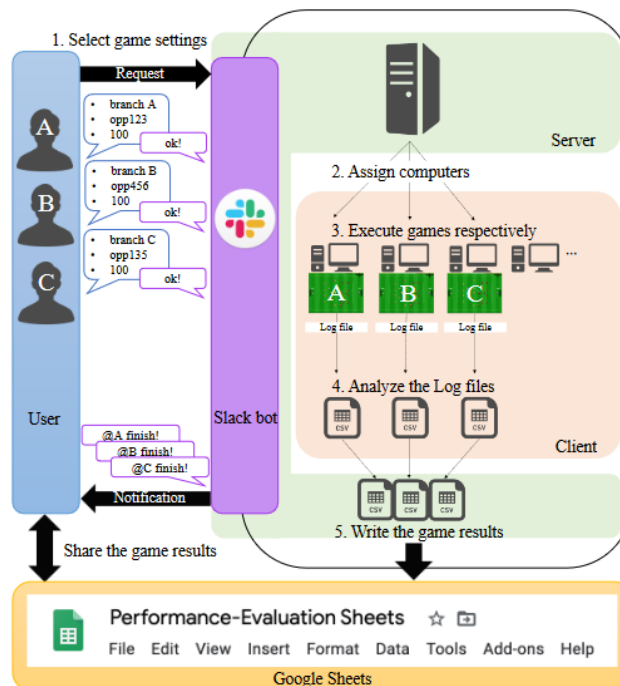


Figure 1: HELIOS2021 Training Environment[20]

## BACKGROUND

---

### 3.1 INTRODUCTION

In this chapter the current CI is described in terms of architecture, workflow, key components, adjacent work, and limitations. The current CI system utilizes 20 test computers, which fetch the newest version of FRA-UNited each night and play 50 matches respectively against the previous world cup winner - CYRUS. Playing a match is followed by a python script, which analyzes the resulting log files and generates a JSON file containing a summary of the match with datapoints such as number of goals per side, pass chains and fouls. These match JSON files are then sent to a Web Server. A Web Application gives the user the ability to inspect individual matches using plots and provides a commit view, where all matches under a selected version of FRA-UNited are used to calculate various statistics. The exact contents of the match JSON file in the current system will be discussed in data model section 3.2.5.

In this chapter, the current architecture is described in detail, along with its limitations. This will function as the basis upon which all changes were made in the context of this Thesis. A brief overview on the configurability of FRA-UNited in the current system is also given, as these interfaces will be used in later sections. In section 3.2 the current architecture is discussed, along with the relevant technologies. Section 3.4 contains the limitations and potential risks of the current system in the context of game and performance analysis as well as architectural limitations. There has also been work done to further improve analytic capabilities, which are discussed in section 3.3.

### 3.2 CURRENT ARCHITECTURE

#### 3.2.1 *Introduction*

The current CI system consists of 5 components:

- A Jenkins server providing team builds and version info's
- A Grails Web Server to collect and service match data and statistics
- A React Web Application
- 20 test computer running Shell and Python scripts
- RoboCup tournament software

Each of these components has been altered for this thesis to various degrees, so it is helpful to examine the role that each component fulfills in the current system. The only interaction a user has with the current system is by pushing a new commit to the Git repository, registered in the Jenkins server and by looking at the visualized results on the React Web Application. As the name



suggest, the CI system functions mostly autonomously, with the process of fetching the newest FRA-UNited build, playing/analyzing games and pushing summaries to the webserver is initiated by the test computers, and not on demand compared to team HELIOS’s training setup[20]. This has the advantage of the collecting server only being loosely coupled to the test computers, as the server can be oblivious to the source of incoming matches. Scaling this system up or down requires only minimum effort. To enable the over-night running of matches, the test computers are configured with a cronjob, which fetches the newest version of the CI scripts and executes a specific entry point script, resulting in the following workflow:

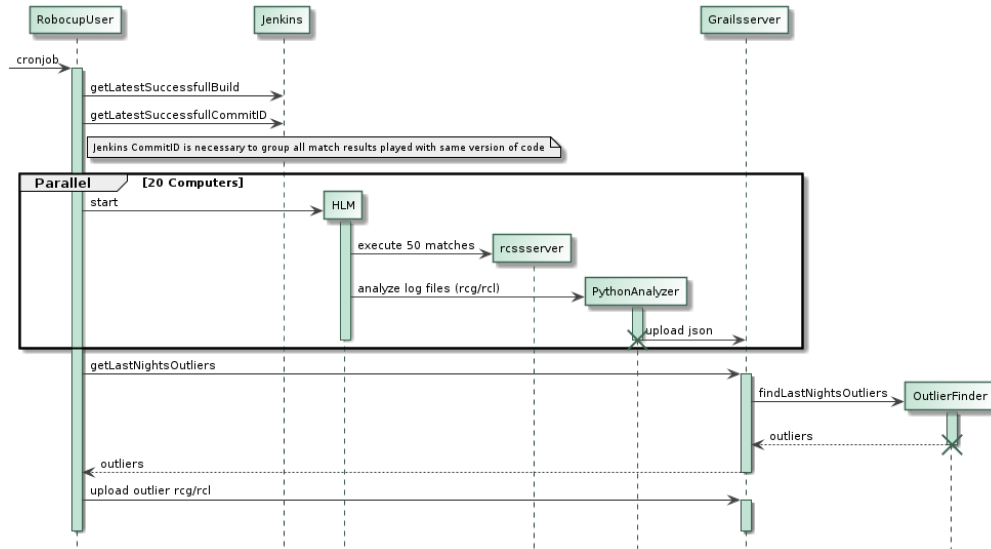


Figure 2: Current CI workflow

### 3.2.2 Jenkins Server

Jenkins is a web-based system for Continuous Integration which provides on-demand building and referencing of projects. In the case of the current CI system, Jenkins is used to fetch the newest build of FRA-UNited’s code base as well as the corresponding commit ID, which is used to link a given match to a version of the team. In terms of software architecture, Jenkins provides an additional layer of abstraction, as the end user doesn’t need special knowledge on a given version control system to receive a build, in addition to being invisible to the developer using said version control system.

The CI system relies on 3 Jenkins routes to exist:

- fetch newest FRA-UNited build (used by the test computers)
- fetch newest commit ID (used by the test computers)
- fetch meta data for a given commit ID (used by the web application)

### 3.2.3 *Grails Webserver*

To collect the analyzed match data and provide the Web Application with aggregated matches on a commit basis, a *Grails* Web Server is used. *Grails* is a framework for the creation of Web Applications and REST APIs[5], the latter of which has been done for the current CI system. The REST API provides routes to fetch specific matches, commit statistics, lists of all matches and commits and a route which calculates, and formats commit data for a histogram plot. Additionally, a route exists to trigger a rudimentary outlier finder, which searches through all new matches, for the highest and lowest value for certain datapoints. This route, however, is not used by the Web Application, not the test computers, as of this writing. The API also exposes a route to save files to the filesystem of the server, which is used to upload log files produced by the test computers. *Grails* implements a model, view controller pattern[14], where controller classes are the organizing entry points for HTTP-requests, domain classes define the data which is being operated on (Model) and view classes, which can contain Markup to be rendered in the HTTP-response. *Grails* also offers various tools for abstraction, such as *Grails Object Relation Mapping*(GORM), which uses Hibernate to create database tables based on *Grails* domain classes and automatic URL mapping based on *Grails* controller classes which by default contain *Create, Read, Update, Delete* (CRUD) functions, aimed at REST API design.

### 3.2.4 *React Web Application*

*React* is a component based framework for the development of web interfaces, created and maintained by Meta (previously Facebook)[18]. In this Thesis, *React* will refer to ReactJs. *React*, along with ReactBootstrap (a front-end framework for ReactJs) allows development using regular JavaScript and can be compiled to an optimized set of plain JavaScript files. This functionality makes the project highly compatible with almost every browser and require no additional software on the client side. The current CI system also utilizes the open source graphics library *Plotly*[17] for the creation and rendering of plots. The current *React* project has 3 main pages: a specific match page, where a match ID can be selected from a dropdown to view the match JSON file in the form of a scatter plot and stacked bar chart. The second main page allows the user to select a specific commit and view various plots containing all matches played under the commit, alongside some meta data, such as the commit message. The last main page aggregates all commits and displays the result in the same plots as the specific commit page.

### 3.2.5 Data Model

The core of the current and new version of the CI system are the before mentioned match JSON files - produced by a log file analysis script written in Python. These match files contain a variety of the datapoints such as goals, ball control and tackles in an attempt to summarize a given match in numerical form. Most of the datapoints have an *\_l* or *\_r* suffix indicating the left or right team. Each match file contains the following datapoints:

Name		Type
id		Number
date_created		String
commit_id		String
team_r	team_l	String
possession_r	possession_l	Percentage Number
ball_on_side_r	ball_on_side_l	Percentage Number
passes_r	passes_l	Count Number
total_shots_r	total_shots_l	Count Number
shots_on_target_r	shots_on_target_l	Count Number
fouls_r	fouls_l	Timestamp Array
corners_r	corners_l	Timestamp Array
free_kicks_r	free_kicks_l	Timestamp Array
offsides_r	offsides_l	Timestamp Array
yellow_cards_r	yellow_cards_l	Timestamp Array
red_cards_r	red_cards_l	Timestamp Array
goals_r	goals_l	Timestamp Array
pass_chains_r	pass_chains_l	Timestamp Array
tackles_r	tackles_l	Timestamp Array

Table 1: Match Datapoints

### 3.2.6 Database

As *GORM* creates and updates database tables based on the given domain classes, there exists multiple tables in the database which are not used. This is due to some classes functioning as transitional data holders (storing data in objects to communicate between method calls), classes no longer being used and not deleted, or have been altered to no longer contain certain member variables. Such tables will not be included in the database models presented in this Thesis, as they are more utility than core functionality and would pollute the diagrams visibility.

The current CI system is mainly based on two database tables: a match table, where each row contains the data from an analyzed match log file and a commit table, where each row is mapped to a unique commit ID. The com-

mit table holds statistics of match info's, aggregated by the commit ID field. As matches are run each night, the commit statistics must be updates if new matches are introduced under the same commit ID. Based on this, the following database UML diagram is constructed:

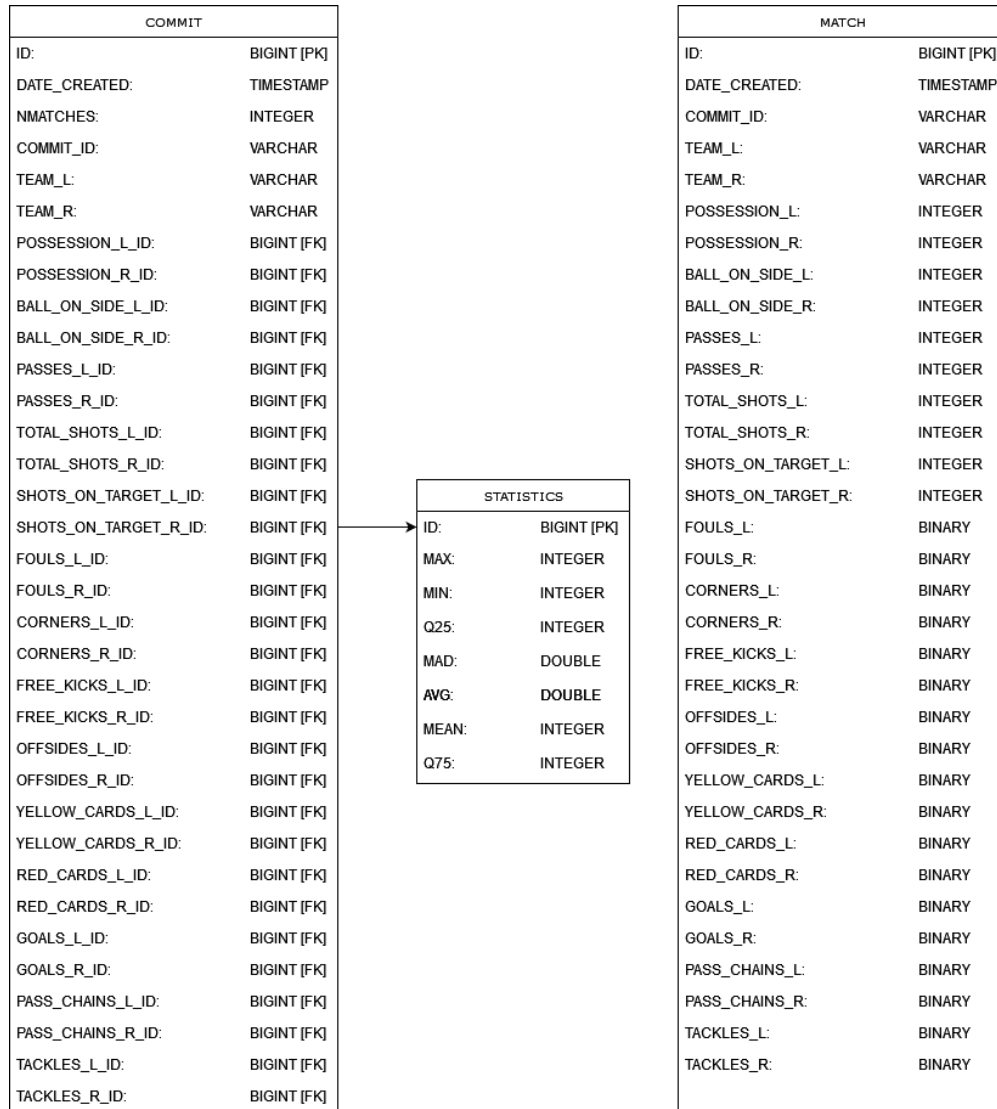


Figure 3: Current Database Model

### 3.2.7 Test Computers

The test computers are 20 physical devices, as opposed to virtual machines (VM), which run on the Linux based Open Suse operating system. Every device has a cronjob set up, which executes a predefined script at 8pm local time, which in turn initiates the previously mention CI workflow. Crucially this initial script fetches the newest version of the CI scripts from git at the start, allowing convenient control over an arbitrary amount of test computers, without the need for a direct connection. These scripts are written for the Linux bash shell command line interpreter. To analyze the log files and upload the result to the webserver, the test computers run a set of python scripts.

### 3.2.8 RoboCup Tournament Software

The RoboCup Soccer Server (rcsserver) is the standard platform to compute RoboCup matches and is also used for the world cup tournaments. On top of the rcsserver, the CI system uses the *Hech League Manager*(HLM), created by Hechenblaickner in 2004[10], which is a ruby program for starting and managing RoboCup tournaments. HLM offers different types of tournaments, such as round robin, one-vs-all, or a predefined list of matches. The current system runs HLM on a one-vs-all repetition mode, which plays a given number of matches with the first defined team against all other specified teams. Since the only opponent team is *CYRUS*, this effectively results in N matches against *CYRUS*, with N being the number specified in the number of repetitions variable. This value is set to 50, which results in 1000 matches with the 20 test computers. As HLM handles the team binaries, it expects a specific folder structure where the teams can be found, which can be configured using a YAML configuration file. When specifying teams in the HLM config, the team's name has to match up with the folder name, the respective team is located in. Since the current system only deals with *CYRUS* and *FRA-UNited*, this folder structure is hard baked into the CI shell scripts.

## 3.3 ADJECENT WORK

In a previous Thesis[3], Allani demonstrated a machine learning system, written in Python, to find outlier in a set of 1000 RoboCup matches by using 2 different algorithms:

- Local Outlier Factor (LOF)
- Isolation Forest (IF)

While both algorithms can be used to find outlier in a given sample, they achieve this goal in different ways and may not land on the same outlier. LOF measures the distance to each point in a sample to find regions of high and low density. Regions of high density, i.e., with a comparatively short average distances between points in the sample, are called cluster. Finding outlier means finding a point with the greatest distance to the nearest cluster. IF segregates a given sample into so called partitions, the shape of which is generated randomly but within boundaries, defined by the minimum and maximum values in the sample. IF is fast and has comparatively low memory requirements.

While Allani showed this system to be functioning, it has not yet been included into the CI system. As the Thesis only aimed to show that outlier can be found in the context of RoboCup matches, it lacks functionality that would make it accessible to the CI system in a production setting, such as an interface to calculate outlier on demand, a device where the python script is executed and some limitations when feeding the system less than 1000 matches. As will be discussed in the following limitations section 3.4, the current system does not handle log file logistics, thus including Allani's outlier finder requires work to be done in expanding the grails webserver, the database and CI scripts.

### 3.4 LIMITATIONS

There certainly is always room for improvement in a project such as the FRA-UNited's CI environment, however as of this writing the current system can be improved by 3 major changes:

- Allowing arbitrary opponents
- Allowing multiple matchups at once
- Allowing configurability of FRA-UNited

The current system always plays a fixed number of matches against a fixed opponent with the configuration of the newest FRA-UNited version. Moreover, the only way to change either of these factors, is by having a user manually copy a new team binary to the version control system under the right folder and edit the *HLM* tournament configuration file to fit the new team binary folder. To try out different FRA-UNited configurations a user must create and push a new team commit with the given values, thereby polluting the commit history. From a software development point of view, the quality of such a CI system, besides producing correct results, lies in the practicality and useability, which must reach a certain level, otherwise the system won't be used. Not having a streamlined process for such a setup increases the likelihood of mistakes being made, which is one of the motivating factors behind this Thesis. By allowing multiple different matchups and defining configurations remotely, the hope is to gain a more sophisticated understanding of FRA-UNited's performance, as well as being able to experiment with different configurations relatively quickly and gauge the resulting impact on team performance.

Additionally, the existing adjacent work, mentioned in 3.3 has not yet been included into the current CI system. While not at the core of this Thesis, having a working outlier detection with the results being accessible via the web application, would contribute to the overall goal of this work - expanding the analytic capabilities of the CI system.

In the following chapter, the implementation of an extension to the current CI system is discussed. The focus of this implementation is to tackle the limitations mentioned in this chapter, along with additional features and reworks of existing components.

## IMPLEMENTATION

---

### 4.1 INTRODUCTION

In this chapter, the details of the necessary changes and additions to the current CI system to realize the goals of this Thesis are discussed. These goals include solutions to the limitations mentioned in 3.4, as well as the reasoning behind the software design and architecture decisions made during this Thesis. The following sections are separated into discussing the high-level ideas, listing the concrete places that need changes to facilitate these ideas and the actual implementation of the proposed changes. As large parts of this work are centered around software engineering, a design pattern or paradigm is mentioned at places where it was consciously applied.

The overarching goal of this thesis is to facilitate the following abilities to the user of the CI system:

- Specify certain parameters regarding the CI's general behavior
- Specify arbitrary matchups
- Upload and manage arbitrary team binaries
- Set arbitrary config parameters in arbitrary config files for FRA-UNited
- Inspect the resulting matches and statistics
- Extra: Download matches found by the outlier finder

To implement these features, almost every part of the current system had to be altered to various magnitudes, as it was not designed to deviate from its fixed settings. The following sections deal with each feature separately, although there occasionally exist some overlap.

### 4.2 GENERAL CONCEPT

To allow the features mentioned above, the first step was to move away from having the commit ID as a central key to connect matches. The main reason for this decision was that a commit is not a great candidate to group matchups or configurability under, in a software architecture sense. A commit ID is a representation of a version of FRA-UNited and so is disjointed from the defined matchups or configuration. This point becomes clear when a commit only contains minor changes or refactoring's. In such a case the matchups and configurations remain the same but would now have to be associated with a new commit ID.

The new system functions around so called *protocols*, which encapsulate a training run. A training run is what is executed by 20 test computers every night, and which was previously static - besides always fetching the newest

version of FRA-UNited. The new system allows the user control various aspects of the training run, such as the number of games played, which teams are to play against each other and how the teams are to be configured before starting the matches. These values are grouped together under a *protocol*, which functions as the central organizing key from a user and software architecture perspective. A User can create and view protocols using the Web Application, which will be discussed in its own section 4.8. To summarize: the new system centers around user defined protocols, which connect all information relevant for the nightly training runs, such as matchups, configuration values and setup scripts.

### 4.3 PROJECT SETUP

The new system builds on top of the currently existing one and therefore shares the technologies mentioned in the background chapter 3. The 4 big components in this system are the *Grails* Web Server, CI shell scripts, *Jenkins* server and the *React* Web Application.

The *Grails* Web Server has been extended to expose a set of new routes, which will be mentioned for each component where they are used in. For the general concept *Grails* now provides routes to fetch various aspects of the current protocol.

```
[GET] /protocol/current      # All protocol data
[GET] /protocol/current/id   # Protocol ID
[GET] /protocol/current/yaml # HLM configuration file
[GET] /protocol/current/teams # Distinct team names
```

Figure 4: Routes for the current protocol

These routes are used in the core CI shell script *runHLM.sh*, setting up all the required files and folders to realize a protocol. The HLM config is a YAML file and is built from a template on the *Grails* server, when a protocol is being created. HLM offers multiple modes for the tournament to be played, alongside parameters such as the path where team binaries are located and where the *rcserver* executable can be found. The detailed description of how the HLM config file is constructed will be covered in the Arbitrary Matchups section 4.4. The *Jenkins* server remains unchanged and continues to provide the current build of FRA-UNited, as well as the corresponding commit id. The *React* Web Application overhaul will be discussed in section 4.8. Lastly the workflow of the new CI system will be visualized and summarized in section 4.10.

### 4.4 ARBITRARY MATCHUPS

For the implementation of arbitrary matchups, a top-down approach was used. Moreover, the design and capabilities of the user interface via the Web Application was the first step taken after conceptualizing the protocol system. Starting off, a divide and conquer approach was used to assume that certain features already exist, prior to development, such as the ability for a user to



upload and manage teams. This approach forced a loosely coupled implementation since no specific features, to closely couple to, existed yet. To allow arbitrary teams and keep the CI characteristic of automatically fetching the newest FRA-UNited version the user can choose between a fixed or commit protocol mode.

**Protocol Creator**

**Protocol Settings**

Mode:

Name:

Number of matches per Instance:

**Match Ups**

Team Left	Team Right	<input type="button" value="Delete"/>
Team Left	Team Right	<input type="button" value="Delete"/>
Team Left	Team Right	<input type="button" value="Delete"/>

Figure 5: Matchup Creator - Fixed Mode User Interface

**Protocol Creator**

**Protocol Settings**

Mode:

Name:

Number of matches per Instance:

**Match Ups**

FRA-UNited-NEWEST	Team Right	<input type="button" value="Delete"/>
FRA-UNited-NEWEST	Team Right	<input type="button" value="Delete"/>
FRA-UNited-NEWEST	Team Right	<input type="button" value="Delete"/>

Figure 6: Matchup Creator - Commit Mode User Interface

In fixed mode, both teams in a matchup can freely be selected, while the commit mode sets the left team to the reserved team name *FRA-UNited-NEWEST*.

This name orders the test computers to fetch the newest version from Jenkins instead of requesting a uploaded team binary - closely resembling the current CI system. The selected matchups are added to the protocol object in JavaScript along with the selected mode. When creating a protocol, the protocol object is stringified into a JSON string and sent to the *Grails* Web Server using a new route:

```
[POST] /robocup/protocol/new          # Create new protocol
```

Figure 7: Route for protocol creation

```
{
  "mode": "commit",
  "matchups": [
    {
      "team_l": {
        "id": 1,
        "name": "FRA-UNITed-NEWEST"
      },
      "team_r": {
        "id": 5,
        "name": "HfutEngine2021"
      }
    },
    {
      "team_l": {
        "id": 1,
        "name": "FRA-UNITed-NEWEST"
      },
      "team_r": {
        "id": 3,
        "name": "Alice2021"
      }
    }
  ]
}
```

Figure 8: Example Matchup data in protocol JSON

This data is used by the *Grails* Web Server to build the HLM YAML config file, which is later pulled by the test computers. *Grails* uses a static template to build the HLM config, which is parsed and edited using the *snakeyaml* java library.

```

mode: matchlist
title: FraUNited CI
server_conf: ./config/rcssserver/server_official.conf
player_conf: ./config/rcssserver/player_official.conf
teams_dir: /tmp/robocup/hlm/teams/
agent_range: 1..12
game_log_extension: .rcg
text_log_extension: .rcl
match_sleep: 10
stylesheet_url: results.xsl
server: localhost
rcssserver_bin: /home/projekt/robocupci/bin/rcssserver
statistics: true
statistics_bin: /tmp/robocup/robocup_log_analyzer/RoboCup_main.py
statistics_dir: /tmp/robocup/logs
hosts: [localhost, localhost]
matches: []

```

Figure 9: HLM Config Template

The HLM tournament mode *match list* allows the user to specify a list of tuples  $[team_l, team_r]$  under the *matches* key in the YAML config, where the entries of the tuple are the folder names of the respective teams. Previously a repetition mode was used in the HLM file, along with a *number\_of\_repetitions* value set to 50, to play 50 games. This value is not recognized in *match list* mode, where the length of the match list value determines the number of games played. When defining a protocol, the user can choose the number of games to be played, which can be useful to run different protocols more quickly with less games each. This value is set to 50 by default to preserve the previous configuration. When defining more than one matchup, the desired number of games must be distributed evenly, as every test computer plays the same matches. The number of matches per matchup  $N$  is determined by  $N = \lfloor \frac{n}{l} \rfloor$  where  $n$  is the desired number of matches and  $l$  the number of defined matchups. This will undercut the desired number of matches in some cases, which can either be accounted for by increasing the number of matches, letting the protocol run another day or be tolerated, as a small number of matches should not impact the results to a meaningful degree on average.

The CI script on the test computers first pulls and stores the created YAML file from the Web Server to the HLM tournament folder, as is required by HLM. HLM now expects folders with name of the teams specified in the match list to exist in the folder specified in the config file. The CI script pulls the required team names from *Grails* and sets up each team individually in a loop. This process can be found in pseudo code form below.

```

# Fetch and store HLM config
FETCH /robocup/protocol/current/yaml INTO hlm/tournament/config.yml
# Fetch teamnames
FETCH /robocup/protocol/current/teams INTO $teamnames

FOR $name IN $teamnames:
    processTeam($name)

processTeam($teamname):
    teamFolderPath = hlm/teams/$teamname
    teamZipPath = $teamsFolderPath.zip

    # Check if the teamname is the reserved FRA-UNITed CI name
    IF $teamname IS 'FRA-UNITed-NEWEST':
        # Fetch latest build from Jenkins
        FETCH /job/RoboCup/ws/*zip*/HelloWorld.zip INTO $teamZipPath

        # Fetch latest commit id and overwrite the current commit id file
        FETCH /job/RoboCup/lastBuild/api/xml?xpath=//lastBuiltRevision/
            SHA1 INTO $commitIdFile

        # Unpack and build FRA-UNITed
        UNZIP $teamZipPath INTO $teamsFolder
        GOTO $teamsFolder/robocup
        MAKE
        RENAME $teamsFolder/robocup $teamFolderPath
    ELSE
        IF $commitIdFile EXISTS:
            # If a commit ID file exists nothing has to be done.
            # Overwriting is only done for FRA-UNITed-NEWEST
        ELSE:
            # If no commit id file exist, a default is created
            WRITE ">_1<" INTO $commitIdFile
        fi;

        # Unpack team binary to
        FETCH /robocup/teams/zip/$teamname INTO $teamZipPath
        UNZIP $teamZipPath INTO $teamFolderPath

        DELETE $teamZipPath
    fi;

```

Figure 10: CI Step - Fetching teams

It is important to note, that because the rcserver is unable to play matches if both teams announce themselves with the same name, the user has to make sure that uploaded team binaries do not specify the same team name. Two versions of FRA-UNITed, for example, must not both announce themselves as 'FRA-UNITed', rather 'FRA-UNITed-A' and 'FRA-UNITed-B'. As the way to store the announced team name is not regulated with a common configuration file, each team is free to include its name in any way it wants. Therefore, this is an aspect which could not directly be accounted for by the new CI system, without having major requisites be pushed onto the user. An approach to combat this limitation has been implemented and is going to be discussed in the team upload chapter 4.5. As the user should be free to choose the name

of the uploaded team, match JSON files produced after each game need to also contain the user-given name for the respective teams. Otherwise, it will be impossible to map a given match file to a matchup, because the logfiles produced by the rcserver only contain the announced name, which is now allowed to be separate from the user-given name. To work around this, the script parses a match result file, generated by HLM named *match.yml* to figure out the respective team names and sides.

```
---
team_l:
  country: Germany
  team_dir: "/tmp/robocup/hlm/teams/FRA-UNITed-NEWEST"
  exception: false
team_r:
  country: China
  team_dir: "/tmp/robocup/hlm/teams/YuShan2021"
  exception: false
statistics: true
scoreboard: true
robocup2flash: false
```

Figure 11: HLM match result file - Match.yml

Here the team side is mapped to the user-given name via the *team\_dir* value. Since this is used to parse the user-given team name, the Web Application does not allow the slash character `'/'` in the name when uploading a new team, as the script searches for the last index if the `'/'` character in the *team\_dir* value. From a software development point of view, this is a weak solution, as it relies on a component which has no direct relation to what it is used for. Eventual changes in the future to the format of this file will have undesired consequences and will break the CI script, as well as the match filtering when viewing the protocol results. An alternative solution to this issue will be discussed in the future works chapter 6.

As *GORM* handles the creation of database tables, the design of the database is mostly auto-generated. However *GORM* allows the user to influence the database table relations by using various keywords, such as *belongsTo* and *hasMany* among other directives. *belongsTo* accepts a class definition and orders *GORM* to append a foreign key pointing at the defined class to the database table of the current class. *hasMany* is *GORM*'s approach to persist lists of class instances as a member of another class. To keep the classes organized, a protocol *hasMany* matchups which in turn contain team info objects with the name of the selected team.

```
class MatchUp {
    TeamInfo team_l
    TeamInfo team_r

    static belongsTo = [trainingProtocolData: TrainingProtocolData]
}

class TeamInfo {
    String name
}
```

Figure 12: Grails Matchup Classes

This structure provides a clear representation of what is being stored, remains readable and can be extended with future datapoints with relatively low effort. This remains true for the database table design generated by *GORM* using the relations specified:

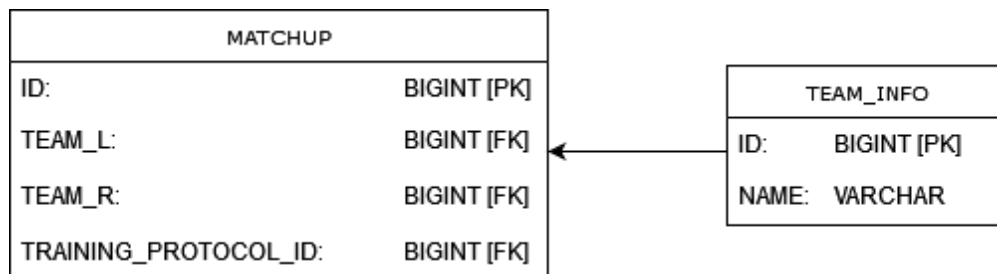


Figure 13: Matchup Database Tables

## 4.5 UPLOADING AND MANAGING ARBITRARY TEAM BINARIES

The teams that a user can select when defining a protocol have to be uploaded via a newly added *Upload Teams* page in the Web Interface. To allow for personalized organization, the user can choose an arbitrary name under which the team will be referenced later - in the match results as well as when filtering matches in the backend. To upload a team, the user must select a zip file containing the binary of the desired team and provide a team name. While the name of the zip file is not important, the format its contents are. A requirement of the new system is that the team must be in a .zip format and unpacking the file must result in a folder which contains the start and kill scripts for the rcserver. Below an example with the using a FRA-UNITed binary. The arrow represents the unzipping process: *unzip file.zip -d target\_folder* on a Linux system.

FRA-UNITed.zip	→	agent/ coach/ start kill start_team_fra-united.sh start_team.sh team.yml
----------------	---	--

Figure 14: Team Zip File Contents

Since teams are free to choose their own structure on how they manage their required files, some problematic cases arise and require special knowledge on the given team. For example the Romanian team *Oxsy*, which has placed 6th at the last world cup, requires a specific configuration file to contain a list of absolute paths to the teams binary folder in order for the team to work properly. For organizational puposes every manual change to the test computer or Web Server is avoided when possible. To compensate for teams needing special configuration, an additional setup script can be uploaded alongside the team's binary. The script is executed after a team has been set up in the process described in 10, and allows the user to perform a specialized setup of the given team. To support this, the script is called with 2 command line arguments: \$1 = absolute path to the unpacked team folder, \$2 = the user-given name for the team. In the case of *Oxsy* a script is used to set the config paths manually to the provided team folder path. As the new system attempts to allow arbitrary teams, a requirement was to avoid including functions to accomandate a specific team, so the choice was made to let the user be in charge of everything that needs to be done to make a team work - only providing an interface the user can use to streamline the process. This makes the new system more independent in regard to the specific implementation of a given team. This requirement was inspired by the *Dependency Inversion Principle* of the SOLID design principles of object oriented programming[15], which states that entities must depend on abstractions, not on concretions. In this case, the concrete steps needed to enable a team are abstracted away, using the script structure. From the systems point of view, it can be assumed that, if a setup script was uploaded, running the script will fulfill all the necessary requirements of the given team.

The Web Application has been extended to enable the team and script upload, along with additional checks regarding the requirements directed at the team's name and hints regarding the format of the zip file contents. Since a setup script is not always needed, its upload is optional.

Figure 15: Team Upload User Interface

A team is uploaded via a HTTP multipart form, containing 1 or 2 form entries. The Web Server expects a multipart request form with a file entry named *'teamzip'* and optionally searches for another entry named *'setupScript'*. The corresponding byte arrays are stored under the given team's name. Team binary sizes can also vary drastically in size, with the binary of team *CYRUS* in 2021 being 48.5 MB, whereas team *Yushan's* 2021 binary only measures 2.8 MB. The setup scripts files are expected to be small with around 2-15 KB. The maximum blob size in the database was set accordingly, with an added safety margin, as team binary sizes are especially difficult to predict in the future. The maximum size for team binaries is set to 100 MB. Setup scripts can be up to 250 KB large.

```
class Team {
    String name
    byte[] file
    byte[] setupScript

    static constraints = {
        file(maxSize:100000000)
        setupScript(maxSize: 250000)
    }
}
```

Figure 16: Grails Team Class



TEAM	
ID:	BIGINT [PK]
NAME:	VARCHAR
FILE:	BINARY
SETUP_SCRIPT:	BINARY

Figure 17: Team Database Table

To enable the user to manage the uploaded teams, a page in the Web Application has been created, which allows the user to see all uploaded teams, as well as the option to download the team binary and setup script. The user can also remove teams, using this page.
















Available Teams			
Name	Download Binary	Download Setup Script	Remove Team
Cyrus			
YuShan2021			
Alice2021			
HfutEngine2021			
Oxxy			
Cyrus2019			
TestOxxy			

Figure 18: Team Inspector Page

To facilitate the upload, download and deletion of teams, additional routes have been added to the *Grails* Web Server. These routes are used by the Web Application, as well as the CI script when fetching the teams specified in the protocol.

```
[GET] /teams/remove/$teamname    # Remove the team
[POST] /teams/new/$teamname      # Add a new team
[GET] /teams/all                 # Fetch all teamnames
[GET] /teams/zip/$teamname       # Download the binary
[GET] /teams/setupScript/$teamname # Download the setup script
```

Figure 19: Routes For Team Management

## 4.6 ARBITRARY CONFIG PARAMETER FOR FRA-UNITED

Large aspects of FRA-UNITed are either directly or indirectly influenced by multiple config files, each containing configuration values regarding certain team components, such as the coach or the general player. A use case often found in development, is trying out various config values to assess the impact on the team's performance, which is a great fit for the changes made in this Thesis. Using top-down development again, the user interface was build first, with the requirement of being able to set arbitrary config values for multiple config files. As of this writing, the config file format used by FRA-UNITed is a dialect of the INI format[ini] and therefore is split up into sections, containing key value pairs '*key = value*'. FRA-UNITed's config files do not a INI-typical *default* section, instead, global config entries are places section less at the start of the file. Based on this, the following Web Interface was implemented:

Figure 20: Config Parameter User Interface

The config filename is a relative path, rooted in the FRA-UNITed team folder. This way the exact location of a given config file is not relied upon by the new CI system - giving the user more freedom to organize the files himself. Using the 'Add Config' button, creates another form in the same style as seen in the graphic above 20. Adding a config entry extends the config form with an additional section-key-value input. When creating a protocol, the desired config values are attached to the request JSON body, which is send to the *Grails* Web Server. The *GORM* relation directives are again utilized to define relation between the protocol and config class. This impacts the way *GORM* creates the database tables.

```

class Config {
    String fileName

    static belongsTo = [trainingProtocolData: TrainingProtocolData]
    static hasMany = [entries: ConfigValue]
}

class ConfigValue {
    String sectionName
    String key
    String value

    static belongsTo = [config: Config]
}

```

Figure 21: Grails Config Classes

As done for the matchups, the *belongsTo* and *hasMany* relations are used, which causes *GORM* to add a foreign key in the *Config* table to the *TrainingProtocolData* table. In this case a *Config* has many config entries, which are linked to a config row using the *config\_id* column. The resulting database model has the following structure:

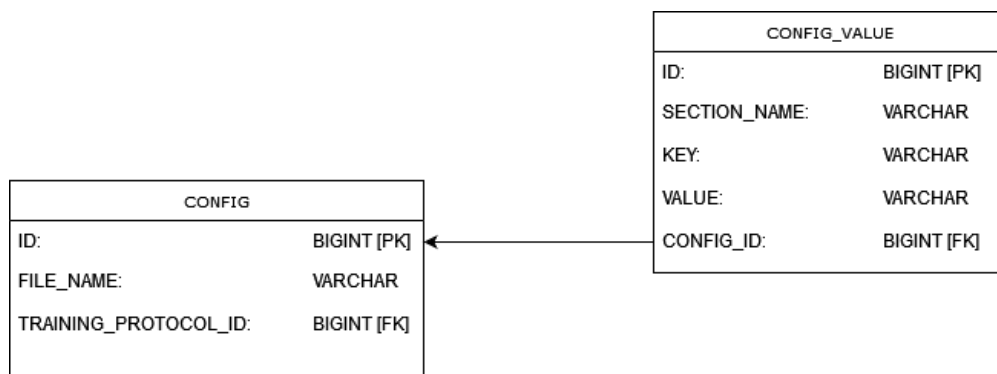


Figure 22: Config Database Tables

## 4.7 MATCHES AND MATCH STATISTICS

In the current system functions around the commit ID, attached to the match JSON files created by the test computers. The new system is structured around protocol IDs which in turn can contain matches with different commit IDs. The user now should be able to view the current plots with different sources, as the underlying datapoints have not changed. Currently, these plots expect a commit class mentioned in 3, which is calculated based on all matches under the given commit. This class has been rebranded to a *MatchBatch*, which represents the statistics of an arbitrary list of matches. The new system then allows the creation of *MatchBatches* based on the user selection:

1. All matches under a protocol
2. All matches in a matchup under a protocol
3. All matches with a certain commit ID under a protocol
4. A combination of 2 and 3

This allows efficient reuse of the same classes and only requires small changes to the plot classes in the Web Application, which works towards the software engineering principle *Don't Repeat Yourself* (DRY). The principle encourages the reuse of code, which in this case meant streamlining the plot data source to the same *MatchBatch* class and then reuse said class for every plot instance. The new system exposes various routes to fetch different sets of *MatchBatches* for a given protocol:

```
[GET] /protocol/$protocolId/results/matches
[GET] /protocol/$protocolId/results/matches/batch

[GET] /protocol/$protocolId/results/matches/$matchUpId
[GET] /protocol/$protocolId/results/matches/$matchUpId/batch

[GET] /protocol/$protocolId/results/matches/$matchUpId/$commitId
[GET] /protocol/$protocolId/results/matches/$matchUpId/$commitId/batch

[POST] /protocol/$protocolId/results/matches/paramBatch
[POST] /protocol/$protocolId/results/histogram
```

Figure 23: Routes For Protocol Results

Using the paramBatch route requires matchups and commit ID to be included as JSON objects in the request body.

As discussed in the arbitrary matchup section 4.4, the CI system uses the user-given name of the uploaded teams, as opposed to the name the teams use to announce themselves to the rcserver. The announced name can be found in the logfile produces by the rcserver after a match, whereas the user-given name is parsed from a file created by HLM. The match JSON, which is sent to the Web Server, has been extended to also include the parsed team name, protocol ID and the log filename which will be discussed in the additional work section 4.9. This leaves the match data in the following state under the new system:

Name		Type
id		Number
date_created		String
protocol_id		Number
log_file_name		String
commit_id		String
matchup_team_name_r	matchup_team_name_l	String
team_r	team_l	String
possession_r	possession_l	Percentage Number
ball_on_side_r	ball_on_side_l	Percentage Number
passes_r	passes_l	Count Number
total_shots_r	total_shots_l	Count Number
shots_on_target_r	shots_on_target_l	Count Number
fouls_r	fouls_l	Timestamp Array
corners_r	corners_l	Timestamp Array
free_kicks_r	free_kicks_l	Timestamp Array
offsides_r	offsides_l	Timestamp Array
yellow_cards_r	yellow_cards_l	Timestamp Array
red_cards_r	red_cards_l	Timestamp Array
goals_r	goals_l	Timestamp Array
pass_chains_r	pass_chains_l	Timestamp Array
tackles_r	tackles_l	Timestamp Array

Table 2: New Match Datapoints

Visualizing the match results has previously been done using an array of different plots, such as pie charts, box plots and histograms. By changing the data source object these plots expect to the before mentioned *MatchBatch*, the new system can reuse large parts of the established dashboard. The plot data sources are now reloaded based on the user's selection of protocol, matchups and commit ID. Sticking with the top-down design approach, the user interface was constructed first. To allow the user to compare the performances of one or more matchups, a list of buttons can be toggled on or off - one for each matchup. Selected matchups are colored black. The protocol and commit ID can be selected using dropdowns.

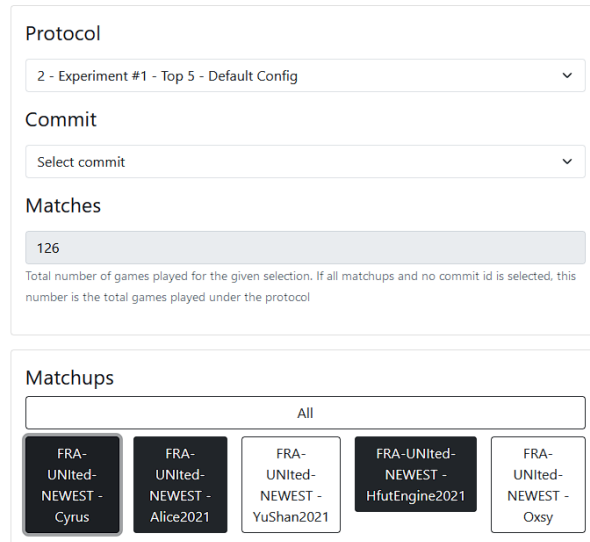


Figure 24: Protocol Results - Menu with 3 matchups selected

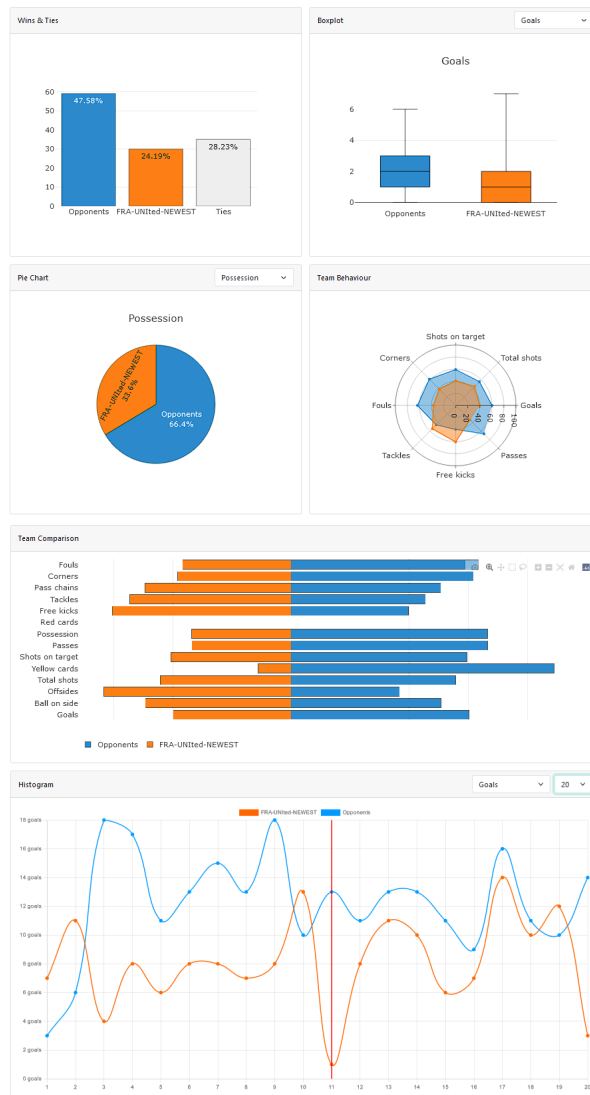


Figure 25: Protocol Results - Plots

This user interface and the named routes will be used when discussing the results later and should be familiar to users of the current CI system, as the plots on the surface remain the same.

#### 4.8 WEB APPLICATION

The current user interface is implemented using the *React* framework[18], which encapsulates each component into a single JavaScript function that is called once per render cycle. Maintaining a state during cycles is done by using *React*'s *useState* functions, which accepts a name for the state/variable, as well as a function to change said state. This pattern is encouraged by the creators of the *React*[19] and is used in all of the implemented components inside the Web Application. *React* components can be used as DOM elements in other components and can be given input hooks for inter-component communication. These hooks are often stateful variables created in a *useState* function. For this Thesis an attempt was made to break down each component in subcomponents, such that every sub component fulfills a sub task, stays compact and remains readable. This strategy is inspired by the *Separation Of Concerns* (SOC) design principle, which aims to prevent monolithic components, that handle all functionality at once. Essentially, a subcomponent is created if a given component can be logically divided into sub tasks. The design of the protocol creator can be used to demonstrate this approach. The protocol creation mask, which the user sees when defining a protocol, is located inside a protocol creator container, which contains a protocol creator. Creating a protocol can be divided into the general data (protocol name, number of games), defining matchups, and defining config values. Each of these sub tasks is represented in a component of their own. The matchups creation subcomponent can itself be divided into components which create individual matchups. While not necessarily always the best approach to code design, dividing components this way makes each function smaller and therefore increases the likelihood of being more readable, which is especially important in projects such as these, where development is distributed over different students and course-projects.

In total, 12 new components got added and almost all other files got changed to allow the user to upload and manage teams, inspect, and download interesting matches, define protocols with arbitrary matchups and config values as well as see and interact with the results of the protocol.

#### 4.9 ADDITIONAL WORK

Since the goal of this Thesis was to expand the analytic capabilities of FRA-UNite's CI system, some adjacent topics emerged during development. Such as including the outlier finder implemented by Allani[3] into the CI workflow and making its results accessible to the user. Allani's work, as discussed in the background chapter 3.3, was written in python and finds anomalous match JSON files in a pool of 1000 matches, utilizing *Pandas* and *Scikit-Learn*. To include Allani's outlier finder into the CI workflow, required some parts of his project to be altered. Since the CI script plans to start the outlier finding process 30 minutes after all matches have been concluded, a way to call Al-

lanis work on demand was needed. As his project was written in Python, a language barrier also had to be overcome between the *Grails* Web Server and the outlier finder. Python is widely used in scientific circles and has many libraries for statistical analysis and machine learning - all of which might be interesting in the future for analyzing matches. The implemented solution was to run a small *Flask* Web Server on localhost on the machine the *Grails* Web Server is run on. *Grails* can now send an HTTP post request containing the match JSONs to the python server, which performs the outlier detection on demand. This server can also be used in the future to enable additional analysis, by specifying additional routes. The *Flask* Web Server is run in the background, using the Linux *screen* program.

For this Thesis, 3 routes on the python Web Server were created:

```
[POST] /calculate-outlier/all      # Results of both LOF and IF
[POST] /calculate-outlier/lof     # Outlier from LOF
[POST] /calculate-outlier/if      # Outlier from IF
```

Figure 26: Local Python Server Routes

An issue in the preprocessing phase was also fixed, where the removal of highly correlated datapoints in the match JSON file could lead to the match ID field being removed, rendering the output useless. The outlier finder previously calculated a score for each match and returned the given match JSONs sorted by that anomaly score. As each match log file has a file size above 20 MB, it's not feasible to store a large quantity every night. To manage this, the outlier finder only returns the top 5 most anomalous matches per outlier algorithm (LOF, IF). A less sophisticated outlier finder had also been previously implemented on the *Grails* Web Server, which finds the match with the most goals received and the match where FRA-UNited had the lowest ball possession, the highest number of timestamps where the ball was on FRA-UNited's side and the most enemy shots, all at the same time. The results of both outlier finders are combined for a list of interesting matches. Interesting matches are linked to the protocol they were played under and are deleted after 5 days. This will result in a maximum storage load of 1.2 GB at any given time if 5 nights in a row 12 different interesting matches are found (5 IF matches + 5 LOF matches + most goals received match + most dominant match).

The user can find and download the list of uploaded outlier for each protocol under the *Interesting Matches* tab.





Interesting Matches					
74 - VS Alice2021- foresee 4					
Interesting Matches (match logfiles are deleted after 5 days)					
Match	Protocol	Commit	Logfilename	Reason	Zip
1196	74	445bc69cf3e7f3d5b256b739ec58fb8938b6df98	FRA-UNITed_0-vs-Alice2021_16	MOST_GOALS_RECEIVED_AROUND_HALFTIME	
1196	74	445bc69cf3e7f3d5b256b739ec58fb8938b6df98	FRA-UNITed_0-vs-Alice2021_16	MAXIMUM_ENEMY_SHOTS	

Figure 27: Interesting Matchups - Menu

INTERESTING_MATCH_INFO		INTERESTING_MATCH	
ID:	BIGINT [PK]	ID:	BIGINT [PK]
TRAINING_PROTOCOL_ID:	BIGINT [FK]	NAME:	VARCHAR
MATCH_ID:	BIGINT [FK]	ZIPPED_LOGS:	VARCHAR
CREATED:	DATE		
COMMIT_ID:	VARCHAR		
REASON:	VARCHAR		
LOG_FILE_NAME:	VARCHAR		

Figure 28: Interesting Matchups - Tables

Work has also been done to correct parsing issues in the log file analyzer. In some cases, missing data would halt or crash the analysis, due to deprecated function arguments or infinite loops. Since the CI scripts were overhauled for this Thesis, the main *runHLM.sh* script got refactored by removing unnecessary path operations and unused or commented-out code, optimizing path and file declarations and allowing to start the script with a different target server for debugging at a local machine (*./runHLM <serverURL>*). Default behavior remains the same.

## 4.10 CI WORKFLOW

Accounting for all the changes mentioned in the previous sections, a new workflow chart can be created, in a similar way to the current workflow, shown in figure 2. This workflow does not include the user interaction necessary to create the protocol and to provide the required team binaries.

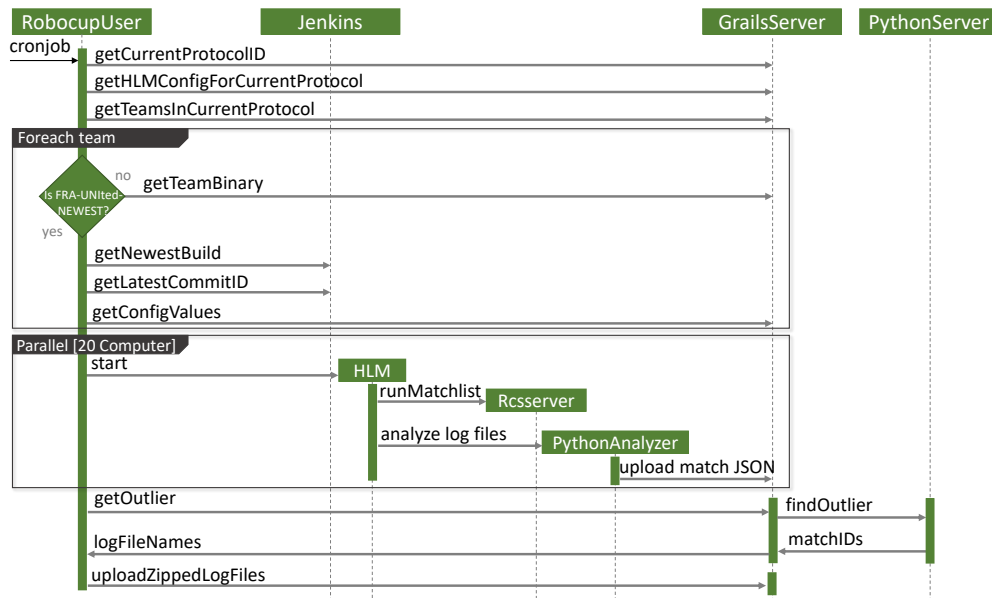


Figure 29: New CI Workflow

## 4.11 DATABASE MODEL

The full database model of the new system can be found below:

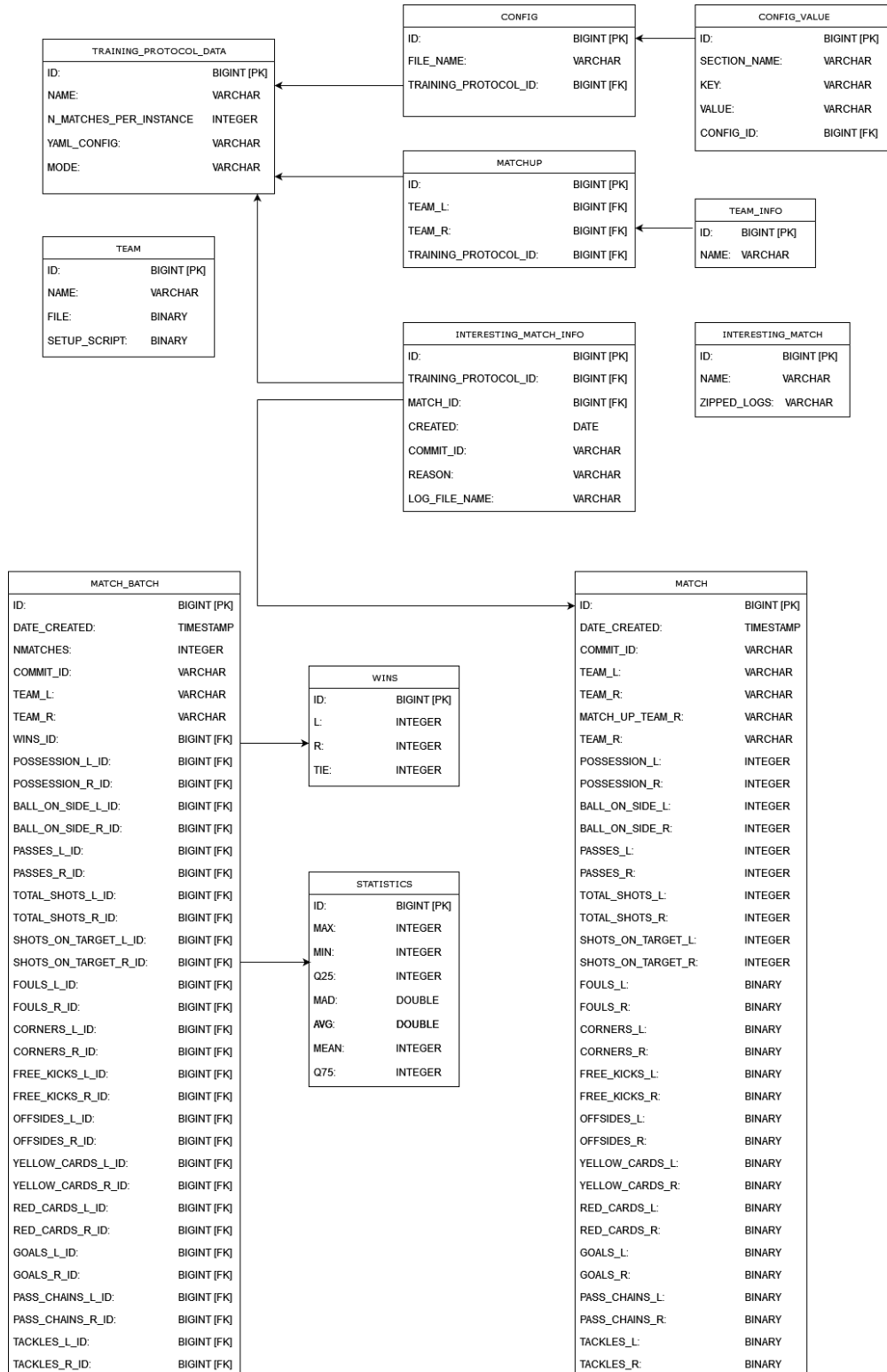


Figure 30: New Database Model

## RESULTS

---

### 5.1 INTRODUCTION

Due to an unresolved error in the nightly execution of matches, the 20 computers were unavailable to perform experiments on. In an attempt to compensate for this situation, 12 Docker container on 2 private computers were used to run protocol experiments. Though this approach worked to some extent, the sample size per run was less than half of the expected amount. As established in this Thesis, large quantities of matches are required, to confidently make statements on the observations. Since this could not be established, the results should be interpreted with a pinch of salt. None the less, this chapter demonstrates how the new CI system can be used to produce differently configured FRA-UNited teams, as well as a head-to-head comparison of the results of the current CI system against a multi matchup result in the new system.

### 5.2 MEASURED DATAPOINTS

To evaluate the results of the different experiments, various datapoints are used, including:

- Win rate
- Goal ratio
- Passes
- Possession
- Tackles

In a competition setting, the win rate trumps all other datapoints. However, judging the quality of a team based on win rate alone might overlook crucial characteristics a team possesses. For example, losing most matches by a close decision while performing intelligent strategies. The average goal ratio will be used to determine if there exists a general strength imbalance. Passes and possession will be used to measure how FRA-UNited reacts to changes in configuration. This will be discussed in more detail in the config value significance section 5.5. In the current system, a clear trend was observed, where *CYRUS* would score significantly more goals against FRA-UNited around the half time point, due to differences in stamina management. To observe this datapoint across different matchups, the goal timestamp histogram will be used.

### 5.3 CONTROL EXPERIMENT

#### 5.3.1 Setup

This experiment mimics the current CI system, where the newest FRA-UNITed build is matched against *CYRUS2019*. These matches are used later to see if a difference in team performance can be measured in the new CI system with multiple matchups. As this matchup is well tested, we will not include any assumptions here since they would be heavily influenced by familiarity bias. We assume that the results of these matches are the baseline, against the new CI system is compared against. As for all other experiments, 200 matches have been played between FRA-UNITed and *CYRUS2019*.

*CYRUS2019* has proven to be a formidable opponent over the last few years, as indicated by its placement in the top brackets of the simulation league. In the current CI system, *CYRUS2019* won consistently over half of the matches. The matches played for this Thesis however were more often won by FRA-UNITed. Besides the continuous improvement of FRA-UNITed and the small sample size, another factor might be the cause of the improvement. As will be described in the following sections in the context of the multiple matchups experiment, the new CI system uses the newest version of the rcserver, which in its latest release removed the ability to perform a dash with negative velocity. Since the version of FRA-UNITed used for these matches has already received patches to compensate for this change, the 2019 version of *CYRUS* might be influenced in a negative way by this change.

FRA-UNITed won 44.26% of the 200 matches played, with a tie-rate of 20.90%. As will be true for the newer version of *CYRUS*, used in later experiments, the amount of aggressive maneuvers is clearly reflected in the match statistics. With an average of 27.07 tackles and 9.35 fouls per game, compared to FRA-UNITed's 22.30 tackles and 1.69 fouls. This lends FRA-UNITed around 8 freekicks per game on average. In the following graphs, FRA-UNITed will be colored orange.

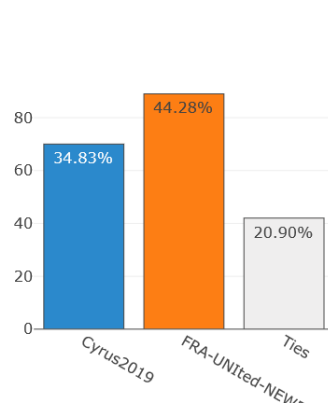


Figure 31: Win rate VS *CYRUS2019*

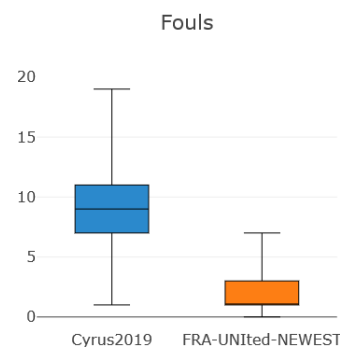


Figure 32: Fouls VS *CYRUS2019*

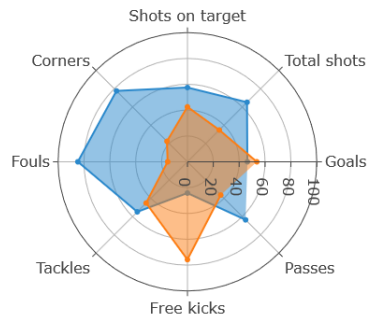


Figure 33: Team Behavior vs *CYRUS2019*

A spike in received goals can also be observed when approaching the halfway mark. This is most likely due to the difference in stamina management, as discussed before, where FRA-UNITed players spend most of their stamina before reaching the halfway break, rendering them too slow to properly guard against fast offenses.

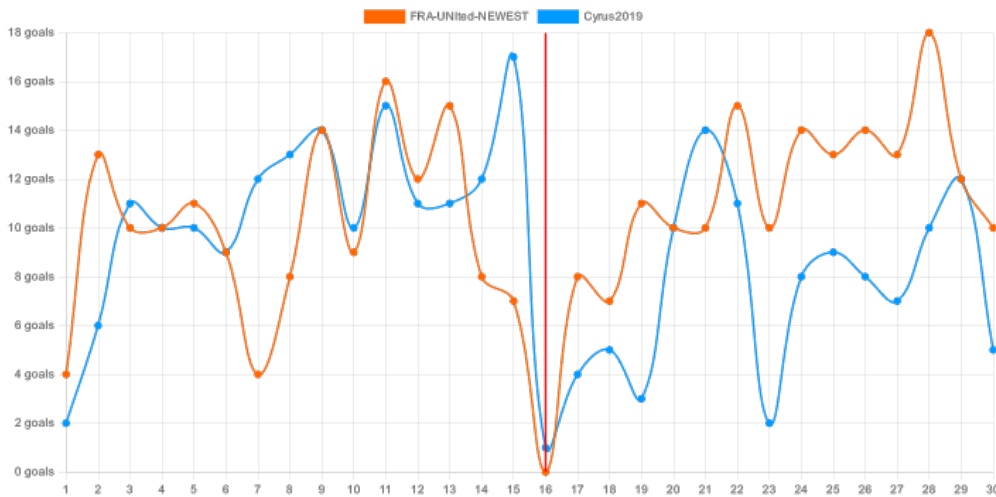


Figure 34: Goal Timestamp Histogram vs *CYRUS2019*

### 5.3.2 Results

## 5.4 COMPARING MULTIPLE MATCHUPS

One of the main changes to the CI system is the introduction of arbitrary matchups and the potential benefit of diversifying the opponent teams to get a broader view on FRA-UNITed's performance. In this section, the new system will be used to play 200 matches against the top 5 of the 2021 World Cup. The overall statistics will be discussed, as well as how the aggregated view of all matchups combined, compares to individual matchups. The goal is to highlight certain assumption regarding FRA-UNITed's behavior which could be made when only viewing individual matchups, and then compare them

to a broader view, using the performances across all matchups. Further, potential drawbacks and pitfalls of using multiple matchups will be discussed.

#### 5.4.1 *Setup*

The top 5 teams in the World Cup tournament in 2021 where:

1. CYRUS
2. HELIOS
3. YuShan
4. HfutEngine
5. Alice

Due to a technical issue, Helios is not currently available in the new CI system. For the matches, the default configuration of FRA-UNited will be used. For the top 4, the binaries from Dayo of the the World Cup will be used, which were taken from the RoboCup archive [9].

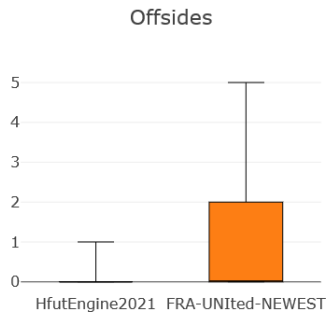
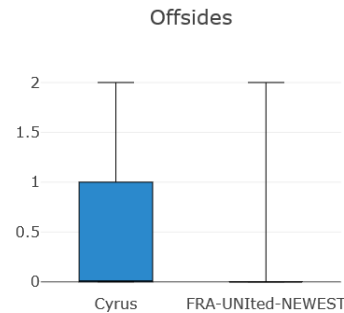
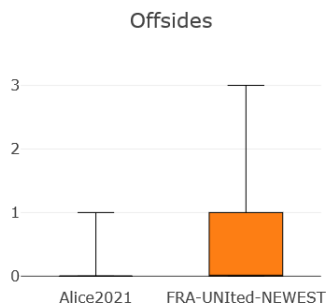
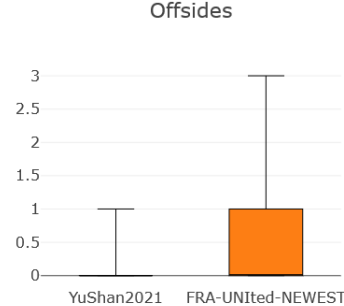
#### 5.4.2 *Expectation*

When comparing the results of FRA-UNited versus the 2021 top 4, it is expected that the average win rate will exceed the measurements from the current CI system. This is in part expected due to a change in the rcserver where dashes with negative velocity were disabled, which was a widely used strategy in the movement system of many teams. This will degrade the performance of some teams. As the newest FRA-UNited has already made steps to fix this issue, it has an advantage over the teams from 2021. Since this a real use case, as team binaries are mainly accessible after a RoboCup event such as the World Cup, and these experiments aim to utilize the new CI system, this situation is tolerated. The comparison should still be valid, as this change impacts every team. However, some teams might rely on the negative dashes more than others, which could inflate FRA-UNited's win rate. The circumstances will be considered when analyzing the results. Additionally, as 250 matches are played, only 50 matches per matchup are used for this evaluation. The new CI system, running on the 20 test computers, would produce 200 matches per matchup with the same configuration, significantly increasing the validity of the produces results. The main goal is to compare the relative differences between *CYRUS2019*, which was the previous benchmark team, used by the current CI system and the top 4 of the 2021 World Cup, used by the new CI system.

#### 5.4.3 *Results*

The new system allows for the selection of one or more matchups, using the new Protocol Results page. The data used in this experiment was obtained using this Web Interface. Under the new system, a clear difference in performance is measured when comparing different matchups, even when accounting for fluctuations due to randomness. Of the 200 matches played

against 4 different opponents, FRA-UNited won 44.44% of matches, while 17.17% of the matches ended in a tie. Which is very similar to the results against *CYRUS2019* in the current CI system. While certainly needing further tests, this supports the effectiveness of previous benchmark, as in terms of win rate, a reasonably correct assesment could be made. On average FRA-UNited scored 1.85 goals per match, compared to 1.99 goals of the opponents. As FRA-UNited won around 6% more games than its opponents, the difference in average goals is most likely due to a great imbalance in goals in some matchups, where unusually many goals were received in comparatively small number of matches. Evidence for this assumption can be found in the matchup against *HfutEngine*, where 63.26% of the 50 matches played where losses, with 18.37% ties. Over the course of these matches, FRA-UNited received almost double the amount of goals per match (3.02) on average, when compared to the 50 matches against *YuShan* (on average 1.59 goals per match). As mentioned in the introduction, such high deviations might be the result of underfitting, due to the low sample size. China's *HfutEngine2021* dominated FRA-UNited with, it seems, clever positioning and movement, without relying on too aggressive maneuvers, which would increase the possibility of receiving a yellow card. FRA-UNited received unusually many offside calls in this matchups, which could be attributed to the positioning and coordination of *HfutEngine2021*. Since these values are based on only 50 matches, the results should be viewed with caution. In a relative comparison however, these datapoints stand out, as FRA-UNited does not receive nearly as much offsides against *YuShan2021*.

Figure 35: Offsides VS *HfutEngine*Figure 36: Offsides VS *CYRUS*Figure 37: Offsides VS *Alice*Figure 38: Offsides VS *YuShan*



As shown in the control experiment 5.3, team *CYRUS* has a noticeably more aggressive play style, as indicated by the amount of tackles and that it receives a yellow card on average every match. This characteristics also is visible with the relatively low sample size of 50 matches against *CYRUS2021*. Comparing FRA-UNited's team behavior chart to *CYRUS* and the top 5, shows a less pronounced overall performance, while varying significantly from individual matchup to matchup. FRA-UNited is colored orange in the following charts.

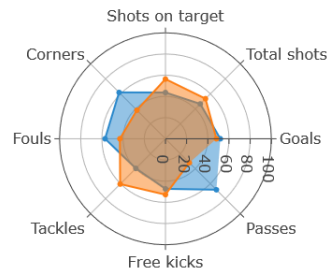
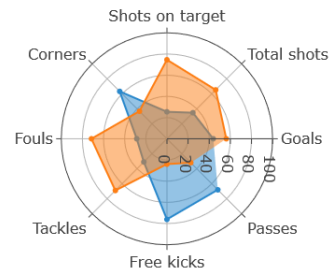
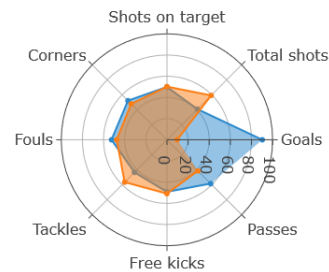


Figure 39: Team Behavior vs All

Figure 40: Team Behavior vs *CYRUS*    Figure 41: Team Behavior vs *YuShan*Figure 42: Team Behavior vs *Alice*Figure 43: Team Behavior vs *HfutEngine*

## 5.5 TESTING CONFIG VALUE SIGNIFICANCE

To demonstrate how the new configuration capabilities can be utilized, this experiment is done by changing the *foresee\_opponents\_max\_age* parameter of the player agent config and comparing the resulting matches. The parameter is an integer used by the agents to predict how far the linear trajectory of an opponent is predicted into the future, based on the observed movement. This experiment will test values ranging from 0 to 5, where 0 results in no prediction by the agent. The assumption for this experiment is, that predicting the position too far into the future will result in the agent misjudging its own movements when intercepting an opponent player. Predicting nothing or very little might result in a similar outcome, where this time the opponent is followed too directly - thereby the agent is getting outplayed by sudden velocity changes. If this assumption is correct, there should be a measurable difference in performance between these tested values due to better or worse trajectory prediction and agent positioning. This should also have at least a small impact on possession and the goal ratio, since intercepting opponents more often increases the probability of a change in ball possession, while also pressuring the opponent to pass the ball to avoid the agent. However, significant improvements in goal ratio would need further testing with a larger sample size. A pass is detected if a kick results in an agent of the same team, to be in the possession of the ball next. If the same agent that performed the kick has gained the possession of the ball again, then the kick is not considered a pass. Ultimately, the experiment seeks to find out if a certain value between 0 and 5 can outperform the default setting of 3.

### 5.5.1 Setup

For each tested config value, 200 matches against China's *YuShan2021* team were played. *YuShan* has placed third in 2021[13], and fourth in 2019[23], making it a top contender in the recent years. While a sample size of 200 matches can still be vulnerable to underfitting, the result should still be reasonably hardened against drastic outlier. A conscious effort has been applied to compensate for expectation bias in cases where a large deviation from the baseline is measured. While a possibility exists, that such measurements represent the true impact of the applied changes, results of this kind are not advertised as hard evidence by this Thesis. Due to the sample size, such results should rather be interpreted as anecdotal evidence. Further testing is needed.

### 5.5.2 Control Experiment

*foresee\_opponents\_max\_age*, by default, is set to the value 3, so we evaluate this value first and will reference the here presented results as *default results* in later comparisons. The following values will be compared against these results to get a relative estimate on the impact of the applied changes.

The *default results* for FRA-UNited versus *YuShan2021* yields a win rate of 48.5% for FRA-UNited with 18.5% ties. On average, FRA-UNited scored 1-3 goal per match while receiving 1-2. The highest recorded number of goals, however, is the same for both teams at 7. FRA-UNited performs significantly

more tackles than *YuShan2021* with a median of 37 and a maximum of 64, whereas *YuShan2021* tackles on average around 19 times per match, with a recorded maximum of 35. Perhaps fittingly, *YuShan2021* is more active in terms of passes, with a median of 131 and a maximum of 177. FRA-UNITed averages 65 passes a match, with a maximum of 109.

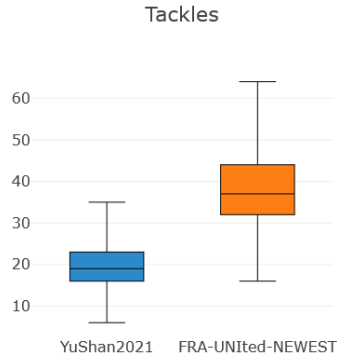


Figure 44: Age 3 - Tackles

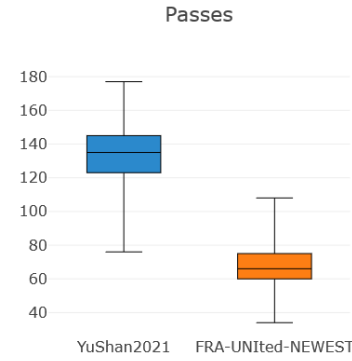


Figure 45: Age 3 - Passes

Running 200 matches per configuration yields the following table:

### 5.5.3 Results

			Avg Goals		Avg	
	Wins	Ties	Scored	Received	Tackles	Possession
Age: 0	45.00%	24.00%	1.73	1.54	38.02	34.35%
Age: 1	37.00%	23.00%	1.92	1.99	37.46	36.57%
Age: 2	47.50%	19.50%	1.88	1.69	39.81	35.05%
Age: 3	48.50%	18.50%	1.95	1.63	37.33	36.32%
Age: 4	38.00%	19.50%	1.64	1.78	38.98	33.95%
Age: 5	43.00%	20.50%	1.86	1.77	39.98	34.98%

Table 3: Results for *foresee\_opponents\_max\_age* for value 0 to 5

These results suggest, that only changing *foresee\_opponents\_max\_age* has only a small impact on the measurable statistics. Due to the sample size of 200 matches, the fluctuations in win rate might be in part due to the inherent randomness of each match. A pattern is observable however, where the number of stalemates increases for low age values, with disabling opponent prediction (age = 0) would result in 7.5% more ties than using the default config. An explanation of this difference could be, that the lack of predictions causes FRA-UNITed to choose a more direct path in defensive play when guarding or pressuring an opponent. This change in behavior could throw off *YuShan2021* such that its offensive strategy is less effective. If this were true, we would expect a lower number of goals received on average, which with a differ-

ence of close to 0.1 goals on average cannot be backed by evidence. Alternatively, not predicting the opponent's position in an offensive scenario could cause a FRA-UNited player to underestimate an interception movement by *YuShan2021*. We would expect that a larger number of offensive attempts by FRA-UNited would be prevented by *YuShan2021*, which could be measurable using the *shots\_on\_target* datapoint as well as less goals being scored on average. The *shots\_on\_target* average effectively doesn't decrease between age = 0 and other age settings. However, the maximum number of *shots\_on\_target* decreased by 18 compared to the default config. As the maximum values only indicate the upper boundary, they can't be used for making a statement on a general pattern. Indeed, the matches played with age = 0 have the lowest average scored goals with a difference of 0.22 goals less than the best score by age = 3. This difference can be viewed as negligible, so further testing, using a larger sample size is required to be able to come to a more definitive conclusion.

Comparing the goal-timestamp distribution of age = 0 and age = 3 also shows that under the default config, a significant amount of the received goals are happening close to halftime. A pattern which was also observed in the current CI system against *CYRUS2019*. This seems to change however when disabling the opponent path prediction to be more evenly distributed. Other values, such as age = 4 also contain the half time pattern, as well as a more pronounced spike near the end of the matches.

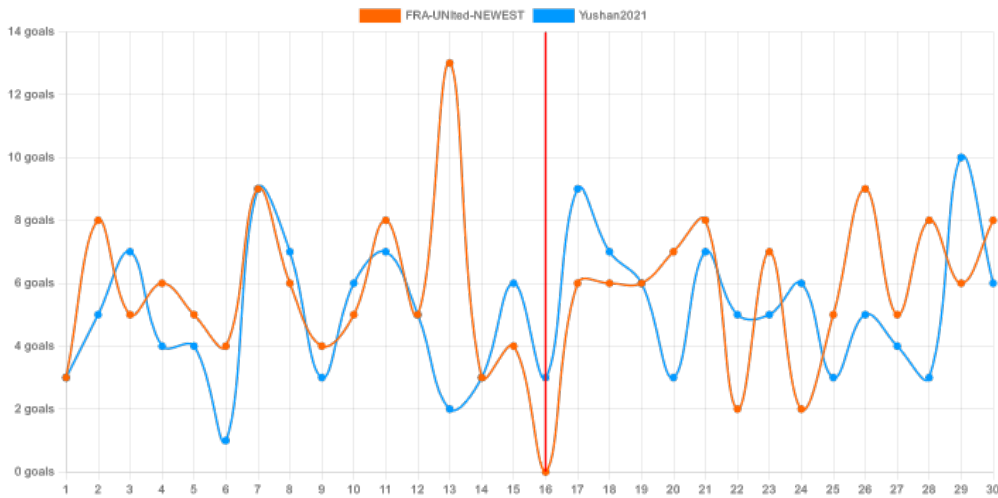


Figure 46: Goal Timestamp Histogram - Age 0

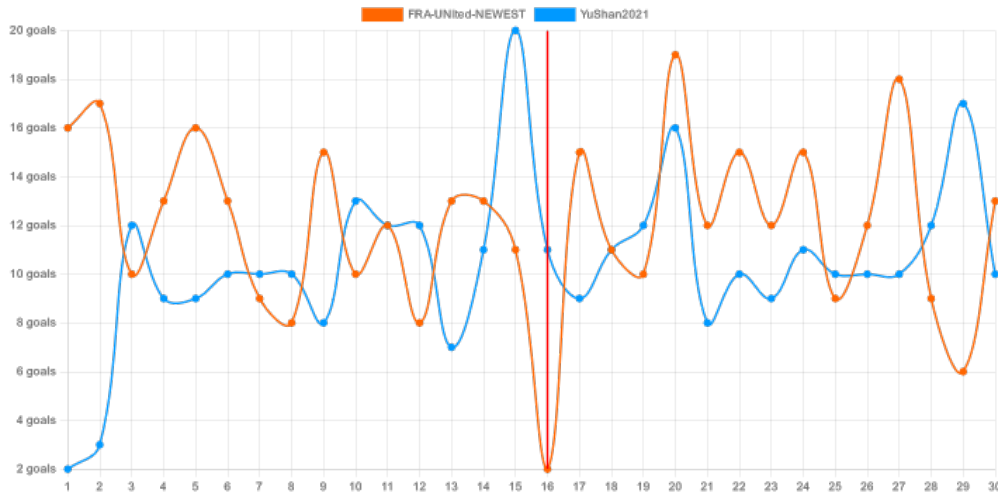


Figure 47: Goal Timestamp Histogram - Age 3

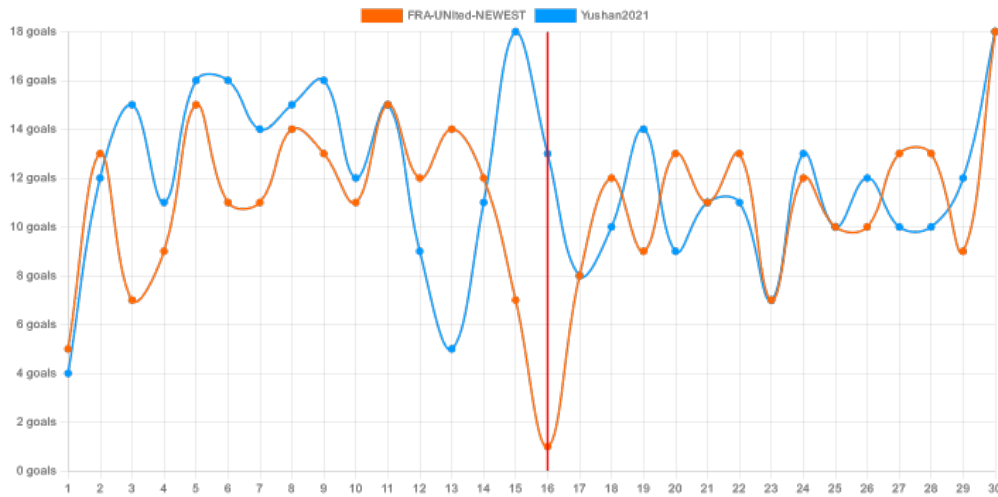


Figure 48: Goal Timestamp Histogram - Age 4

Generally, the here produced results indicate that, decreasing *foresee\_opponents\_max\_age* has no beneficial consequences, which could be measured using the CI system. There might exist a meaningful difference in the average running distance of FRA-UNITed's players or the relative distance to opponent players when guarding, however such datapoints are not yet obtained by the log analyzer scripts.

This experiment showed, that changing team configurations, using the new CI system, can result in measurable differences in the match statistics produced. As the evidence for wide ranging changes in performance for different *foresee\_opponents\_max\_age* values is inconclusive, a test with more match volume is needed. However, this experiment showed, how to use the new CI system and can be seen as a proof of concept for future experiments.

## FUTURE WORK

---

### 6.1 FORKING HLM

The new CI system can be further improved upon in future projects by forking the HLM project to have a clean way of extracting the user-given team names after running a match. To achieve this, the *statistics* function the *match.rb* script must be extended by adding the team's name as a command line argument to the *system\_statistics\_bin* function call. This way, no parsing would be necessary, as the user-given name is known at this point.

### 6.2 LOGIN AND SECURITY

The Web Application does not contain any security features, which was not a problem before, as it was a unidirectional system. But now, since a user can upload and download executable files, a login system should be implemented to shield the system from unauthorized access.

### 6.3 CONFIG VALUES FOR INDIVIDUAL TEAMS

Configurations are set on a protocol basis. A valid use case however would be testing multiple configurations in a single protocol to avoid having to wait multiple days. The existing config component could be made into a dialog component and added to a team info object, instead of the protocol object.

### 6.4 LOG FILE ANALYZER

The log file analyzer often struggles with parsing log files, which results in the match not being sent to the *Grails* Web Server. Some of these issues have been fixed in this Thesis but more testing is needed to find all parsing errors.

### 6.5 CONTAINERIZATION OF THE PYTHON ANALYSIS SERVER

In the new system, the *Flask* Web Server, which is used to find outlier on demand is run in the background using the *screen* command. A more transparent solution would be to use a program like *Docker* and start the server in a container. This would simplify testing, deployment, and monitoring.

### 6.6 LOG FILE PLAYER

In a previous Thesis, a web-based log file player was implemented, using *React* and *Electron*. Including this project into the existing Web Application would enable the user to directly view the interesting matches on demand, without the need to start the *rcsmonitor* locally. As the log file player requires many libraries the scope for such an inclusion is uncertain.

## CONCLUSION

---

The RoboCup 2D simulation league continues to evolve, with a steady increase in the capabilities on how the teams play soccer. As every match of 2D soccer is tainted to various degrees by randomness, a need to play a large number of matches arise, to limit the impact of chance on the overall results. For this reason, a CI system has been implemented. This Thesis discussed various drawbacks of the current CI system, which is used by the developers of Frankfurt University Of Applied Sciences' RoboCup team - FRA-UNITed, with the goal to assess its performance. The main points discussed where:

- Risk of overfitting, by only playing against the same team
- No interaction and influence on the CI system besides new commits
- Unused potential for configuration testing

In this work, a new CI system was demonstrated, which allows the managing and usage of arbitrary team binaries, along with the ability to freely specify config values for arbitrary config files. These new capabilities are organized under a protocol structure, which defines a nightly run of matches. To facilitate this functionality, wide reaching changes to the *Grails* Web Server, *React* Web Application and CI shell scripts had to be made. By using a top-down approach to software design, with the emphasis on applying the *divide and conquer* principle when possible, a new Web Interface was constructed, to provide the user of the CI system with more control over the nightly training process.

Moreover, a measurable difference in performance could be shown, which has the potential of being a more accurate assessment of FRA-UNITed's team quality. Additionally, the new configuration capabilities were put to the test, by using them to analyze the impact of changing a trajectory prediction parameter and comparing the resulting differences in outcomes.

This Thesis also proposed future topics, which could further improve the CI system, such as adding security features to the Web Server and allowing configuration of arbitrary teams on a matchup basis, instead of on a protocol basis. The CI system remains to be a powerful tool, to gauge FRA-UNITed's performance over the course of its development and has been extended to limit or eliminate the discussed drawbacks. Further research into the CI system and additional capabilities of an automated training process must be done in order to further benefit the development of FRA-UNITed.

## Part II

### APPENDIX



## BIBLIOGRAPHY

---

- [1] Hidehisa Akiyama. *Rcsserver*. <https://github.com/rcsoccersim/rcssserver>. 2022.
- [2] Hidehisa Akiyama, Tomoharu Nakashima, Takuya Fukushima, Jiarun Zhong, Yudai Suzuki, and An Ohori. "Helios2018: Robocup 2018 soccer simulation 2D league champion." In: *Robot World Cup*. Springer. 2018, pp. 450–461.
- [3] Mohamed Amine Allani. "Analyse und Darstellung von RoboCup Spielen mit Maschinellern Lernen." unpublished master thesis. 2020.
- [4] Grzegorz Ficht and Sven Behnke. "Bipedal humanoid hardware design: A technology review." In: *Current Robotics Reports* 2.2 (2021), pp. 201–210.
- [5] Grails Foundation. *A powerful Groovy-based web application framework for the JVM built on top of Spring Boot*. <https://grails.org/>. 2022.
- [6] Fussballdaten.de. *Abschlusstabellen der Bundesliga*. <https://www.fussballdaten.de/bundesliga/2022/tabelle/>. 2022.
- [7] Thomas Gabel, Egbert Falkenberg, and Eicke Godehardt. "Progress in RoboCup revisited: the state of soccer simulation 2D." In: *Robot World Cup*. Springer. 2016, pp. 144–156.
- [8] Thomas Gabel and Constantin Roser. "FRA-UNited-team description 2016." In: *RoboCup 2016 Symposium and Competitions: Team Description Papers*. Leipzig, Germany. 2016, pp. 10–18.
- [9] Stefan Glaser. *RoboCup.info Archive*. <http://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/2021/Day0/>. 2022.
- [10] Andreas Hechenblaickner. *Hech League Manager*. 2004–2010.
- [11] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Ei-ichi Osawa. "Robocup: The robot world cup initiative." In: *Proceedings of the first international conference on Autonomous agents*. 1997, pp. 340–347.
- [12] Hiroaki Kitano, Minoru Asada, Itsuki Noda, and Hitoshi Matsubara. "RoboCup: Robot world cup." In: *IEEE Robotics & Automation Magazine* 5.3 (1998), pp. 30–36.
- [13] RoboCup Soccer Simulation League. *Final rankings*. <https://ssim.robocup.org/soccer-simulation-2d/2d-competition/2021-2/final-rankings>. 2021.
- [14] Avraham Leff and James T Rayfield. "Web-application development using the model/view/controller design pattern." In: *Proceedings fifth ieee international enterprise distributed object computing conference*. IEEE. 2001, pp. 118–127.
- [15] Robert C Martin, James Newkirk, and Robert S Koss. *Agile software development: principles, patterns, and practices*. Vol. 2. Prentice Hall Upper Saddle River, NJ, 2003.

- [16] Mauricio Matamoros, Viktor Seib, Raphael Memmesheimer, and Dietrich Paulus. "RoboCup@ Home: Summarizing achievements in over eleven years of competition." In: *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE. 2018, pp. 186–191.
- [17] Plotly.com. *Plotly JavaScript Open Source Graphing Library*. <https://plotly.com/javascript/>. 2022.
- [18] Reactjs.org. *React - A JavaScript library for building user interfaces*. <https://reactjs.org>. 2022.
- [19] Reactjs.org. *Using Multiple State Variables*. <https://reactjs.org/docs/hooks-state.html>. 2022.
- [20] Masaki Yamaguchi, Ryota Kuga, Hiroki Omori, Takuya Fukushima, Tomoharu Nakashima, and Hidehisa Akiyama. "HELIOS2021: Team Description Paper." In: *RoboCup 2021 Symposium and Competitions, Worldwide*. 2021.
- [21] bundesliga.com. *Spielplan 2021-2022*. <https://www.bundesliga.com/de/bundesliga/spieltag>. 2022.
- [22] robocup2017.org. *RoboCup2017 Nagoya Japan Results*. [https://www.robocup2017.org/results\\_soccer\\_simulation2d.html](https://www.robocup2017.org/results_soccer_simulation2d.html). 2017.
- [23] ssim.robocup.org. *Final Ranking of Soccer Simulation 2D League in 2019*. <https://ssim.robocup.org/soccer-simulation-2d/2d-competition/2019-2/results/>. 2019.
- [24] ssim.robocup.org. *Final Rankings of Soccer Simulation 2D League 2021*. <https://ssim.robocup.org/soccer-simulation-2d/2d-competition/2021-2/final-rankings/>. 2021.