PETER VAN LITH

DEEPTURTLE TOOLKIT

ROBOCUP MSL GAME ANALYSIS SOFTWARE

PUBLISHED 2020 BY: TECH UNITED (TU/e)

PUBLISHED BY: TECH UNITED TU/e www.TechUnited.nl

First printing, August 2020

Contents

Dedicated to the brave Robotic-Soccer enthusiasts,
who appreciate AI and Robotics and value
the power of GOFAI, Python and TensorFlow.

Change History

This software is part of a project that started in 2016 as a system to recognize the shirt labels of robots, competing in the RoboCup MSL competitions. This system used Deep Learning AI technology to recognize the ID and shirt color in order to identify the opponents. It was the start of a project, aimed at tracking the opponent's actions and is slowly developing into a system to analyze and understand team behavior.

It uses a combination of Knowledge Engineering principles, combined with Deep Learning in order to analyze, understand and explain the behavior of our opponents.

The following changes have been made to the system:

Change History

Version	Date	Description
V0.3.0	31-10-17	First Deep learning version
V0.3.1	13-11-17	Included pretrained neural net
V0.4.0	17-11-17	First training with simulator
V0.4.1	23-09-18	Converted to newest version of TF
V0.4.2	28-11-18	Included first version of Alpha-Zero
V0.5.0	11-12-18	First Replay version from MatLab files
V0.6.0	02-01-19	Moved to Eclipse 2018-12
V0.6.1	27-02-19	Added Monitor and Step function
V0.7.0	27-09-19	Include Game Situation Classification
V0.7.1	21-10-19	Included 3-level tracing
V0.7.2	28-10-19	New full search function
V0.7.3	03-11-19	Included GameSit search facility
V0.8.0	14-11-19	Included TensorFlow 2.0
V0.8.1	11-01-20	Included Formation Analysis
V0.8.2	11-03-20	First Action and Remark Analysis
V0.9.0	28-04-20	Restructure reading in data and storage
V1.0.0	04-07-20	Included error checking + install
V1.1.0	01-08-20	Included debug option and manual

This is the first alpha-release. Many issues are still unresolved and is intended for initial user tests. The error checking facility is almost complete. The additional Tabs in the application are under development and their use may result in crashes. Please provide feedback if you find any issues in the facilities of the ⟨*Load Tab*⟩.

The current version has the following known issues:

Known Issues

ID	Description
V1.1.0-001	Reappearing robots do not get the proper ID
V1.1.0-002	Some newer files crash during load
V1.1.0-003	New format must use meta-data
V1.1.0.004	STP files use new codes for DefCon

Part I

Converting and Checking the File

Introduction

This manual explains the use of the DeepTurtle Toolkit.

The toolkit has been developed to enable game analyses, using the ⟨*.mat*⟩ files, generated by the Tech United TURTLE robots. Analysis is performed in a number of steps, explained in the chapters following. The analyses may be performed on a single ⟨*.mat*⟩ file, but also on batches of files, so an overview of all ⟨*.mat*⟩ files of a certain year can be grouped together. Output of each step is written to a logfile and additionally statistics are written in a ⟨*.csv*⟩ file for further analysis.

The DeepTurtle Toolkit consists of the following parts:

1. File Conversion and Checks
2. The Analist and the Critic
3. Making Set-Pieces
4. Behavior Patterns
5. Learning Behaviors
6. Opponent Analysis

Currently only the first part, *File Conversion and Checks* has been completed. The next phase follows soon, but the entire system is part of an ongoing process that will take some years to complete. Each phase first describes what that part does and how the software is used. In addition a more detailed technical description is included, explaining how the software works and providing more background and implementation details. The intention is to distill several technical articles from this document as well.

Installing and Testing the Software

Unpack the ⟨*DeepTurle.tar.xz*⟩ file into your applications directory. From inside this directory ⟨*dist*⟩ start the program ⟨*.dist/DeepTurtle/DeepTurtle*⟩. The distribution also contains the testfiles that were used to create the examples in this manual and it starts up automatically with these files selected. The current distribution has been created for use with Linux. Versions for Mac and Windows may also be created when there is a need for them.

Before starting the program with your own files, copy the ⟨*MatFiles*⟩ directory to your desktop into which you place the ⟨*.mat*⟩ files that must be checked. This directory must contain two subdirectories ⟨*MAT and STP*⟩ in which subdirectories are placed. So the directory structure, where DeepTurtle looks for the ⟨*.mat*⟩ files is given below. The testfiles are copied as well and are located in the directory ⟨*oTestsets*⟩. Instead of a directory you may if course, also use a symbolic link.

```
Desktop MatFiles
  STP
    ...  Files with new STP format
  MAT
    ...  Directory with some .mat files
      file1.mat
      file2.mat
```

Loading the file

Loading a ⟨*.mat*⟩ file involves several steps to prepare the data for analysis. First of all the data is converted to a new World Model format, more suited for analysis. The position data of all robots in the field is recorded by each TURTLE individually, along with that TURTLE's interpretation of the game situation. There are many situations, where information is lost or in conflict with each other for brief moments, and these instances need to be corrected or removed.

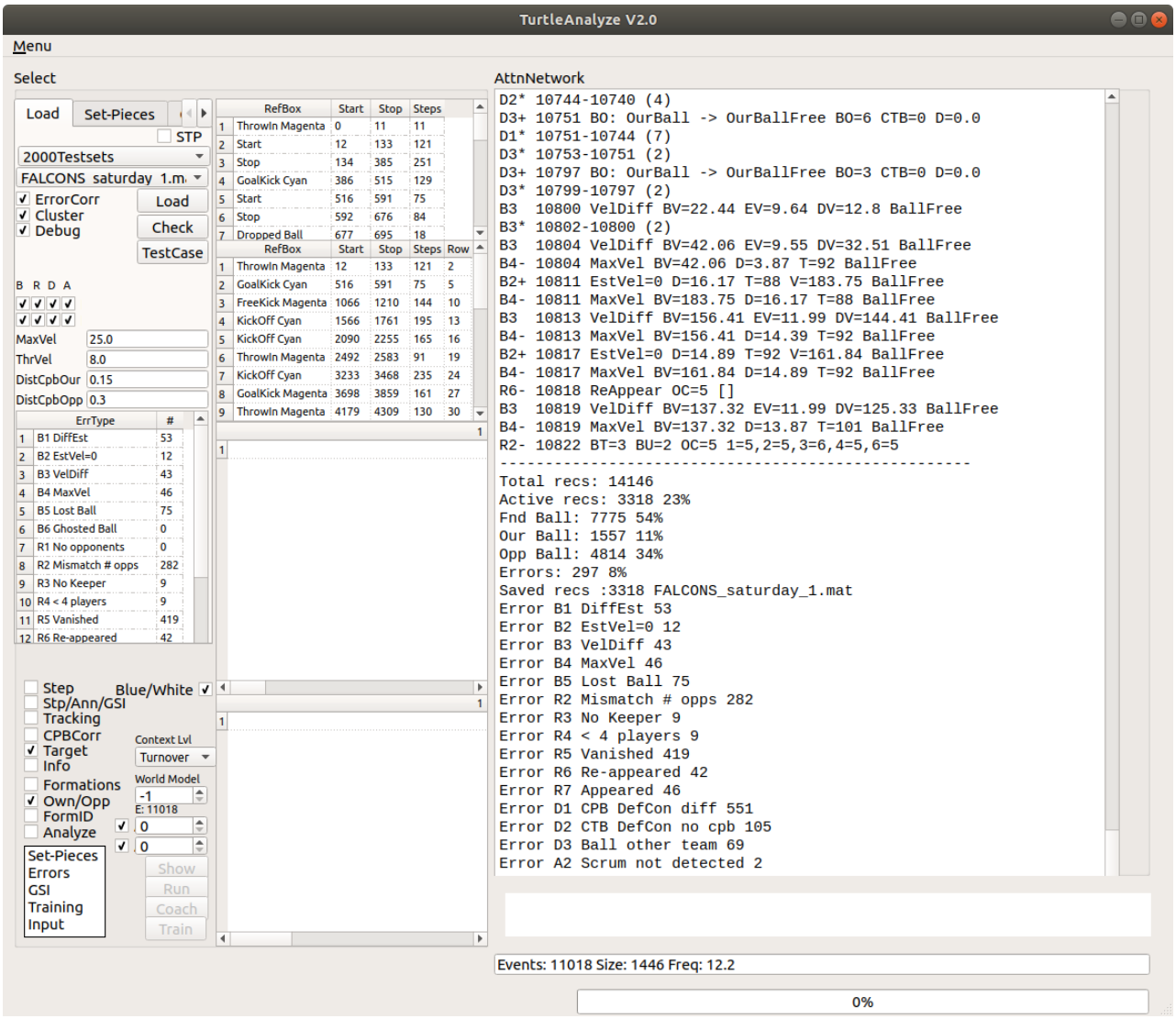
This first step therefore includes a number of checks to find out if the data is consistent and plausible. For instance, conflicts in assessments of the game situation or the position and speed of the ball are checked with the data from all TURTLEs and the best information is used. Inconsistencies and anomalies are removed to allow further analysis.

A report is generated with warnings and errors, showing which steps contain problems and where these were corrected.All information between stops and restarts is removed, so we only use the real game moments.

Then the data is split into episodes from start to stop moments. A further subdivision is made in so-called Ball Possession Turnovers (BPT). Each sub-episode in which a team has possession of the ball is analyzed separately and serves as the basis for the game analysis in the steps following.

The User Interface

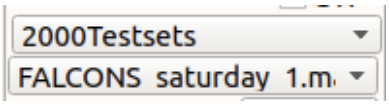
The User Interface consists of a number or ⟨*tabs*⟩, one for each phase. The first one is the ⟨*Load*⟩ tab, shown below.



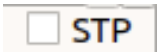
The user interface with the error log and statistics.

Figure 1: *

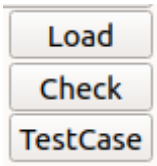
A FILE is selected by first selecting a directory and within that directory a single file. Multiple files are selected by just selecting a directory and using the Check button.



WHEN THE FILE IS OF A NEWER FORMAT, as a result of the introduction of STP, first check the STP box, before selecting a file. We are currently implementing the metaData facility. When that is done, this checkbox will disappear and the separate STP directory will no longer be needed.



AFTER THE FILE IS SELECTED, use the ⟨*Load*⟩ button to load the file. When you want to check an entire directory, select the ⟨*Check*⟩ button. The ⟨*TestCase*⟩ button is used to generate a new testCase for inclusion in the unit tests. It may also be used to inspect the contents of a single step of the World Model. Please note that when the ⟨*Debug*⟩ switch is on, Debug information is also included in the TestCase.



THE GROUP OF CHECKBOXES is used to select what is done with errors, found in the file. When ⟨*ErrorCorr*⟩ is turned off, no corrections are made. This is useful when you want to inspect certain errors and want to see the original data in the file, using the browsers, that will be explained in the Browsers chapter.

The ⟨*Cluster*⟩ option is used to group errors in successive steps into clusters, so that not every step is listed separately as an error. Clustering does not affect the error counts.

The ⟨*Debug*⟩ option is selected to show the original information for debugging in the SetPiece browser.

The system checks for errors of the following types: (See Table 1)

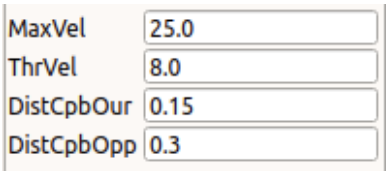
1. B - Ball info
2. R - Robot info
3. D - DefCon info
4. A - Action info

By selecting the boxes, the error checking may be disabled for certain types. The second row is to suppress warnings as well.

THE DEFAULTS are used to influence how critical the checking procedures are with respect to the speed and distances, used in the checking rules. The following values are used:

1. **MaxVel** - Used as maximum velocity of the ball. If the ball velocity is higher than this value, the movement is considered irrational and is ignored.
2. **ThrVel** - Is the threshold value to compare the estimated ball velocity against the apparent velocity, calculated from the distance travelled between two time steps.
3. **DistCpbOur** - Distance in meters to consider a TURTLE to be owning the ball. It is used to check the validity of the CPB switch in the original data.
4. **DistCpbOpp** - Distance in meters to consider an opponent to be owning the ball. Because we can only judge if an opponent has the ball by the distance, this is set less critical than our own distance. In many cases however, the TURTLE software considers distances of around 60 cm to be owning the ball.

   The checks can be made more or less critical by changing these default values.



During the analysis all errors and warnings are counted and the total counts are shown in the ⟨*ErrType*⟩ table. Once the file has been loaded, this table can be used to select an error type. All steps in which the error occurs, will then be loaded into the ⟨*DefCon Table*⟩, as explained in the Browsers section.

	ErrType	#
1	B1 DiffEst	53
2	B2 EstVel=0	12
3	B3 VelDiff	43
4	B4 MaxVel	46
5	B5 Lost Ball	75
6	B6 Ghosted Ball	0
7	R1 No opponents	0
8	R2 Mismatch # opps	282
9	R3 No Keeper	9
10	R4 < 4 players	9
11	R5 Vanished	419
12	R6 Re-appeared	42

An Error Log is created and shown in the large window on the right side of the User Interface. It lists all errors, using a 2-position code. The third position is a *, + or - sign or a blanc.
A ⟨-⟩ sign means that this is a warning only and has no direct impact on the results. It usually signals a conflict that could be resolved, but may be worth further investigation.
A ⟨+⟩ sign means that this is an error that resulted in a change in the output. It also lists the data that was used to determine that there was an error, along with the corrected value. If the option ⟨*ErrorCorr*⟩ was disabled, no + signs will be shown
A ⟨ ⟩ means that this was an error, but no correction was made, either because there was insufficient information or the error is corrected elsewhere.
A ⟨*⟩ means that there was a cluster of errors and the beginning - and end step is shown along with the number of steps.

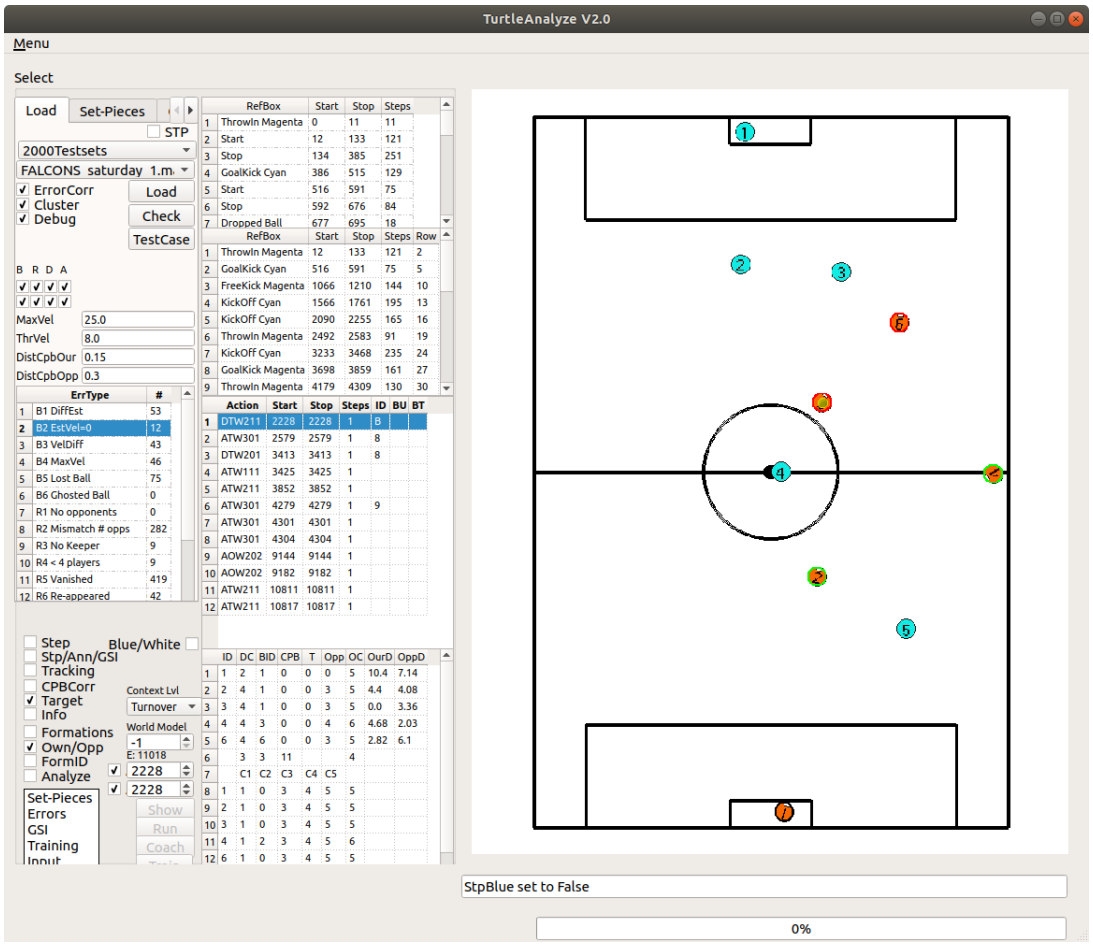
Error Logs appear in the large textbox on the right. This text is scrollable and may be copied and pasted to other programs.

Error Log

```
EPS 12 ---------------------------------------------
R3- 12 NoGK
D2+ 83 CTB: OurBallFree -> Unknown CTB=4 D=0.29
D1+ 87 CPB: OurBallFree -> OurBall CPB=4 CTB=4 D=0.2
D2* 87-83 (4)
D1* 90-87 (3)
D1+ 97 CPB: OppBall -> OurBall CPB=1 CTB=3 D=0.2
D3+ 99 BO: OppBallFree -> OurBallFree BO=1 CTB=0 D=0.0
D3* 101-99 (2)
D2+ 103 CTB: OppBall -> Unknown CTB=4 D=0.25
R2- 105 BT=6 BU=4 OC=4 1=3,2=3,3=3,4=3,6=4
D2* 105-103 (2)
R5+ 106 Vanish OC=3 [9]
D3+ 107 BO: OurBall -> OurBallFree BO=4 CTB=0 D=0.0
D3* 109-107 (2)
B3  113 VelDiff BV=20.73 EV=6.79 DV=13.94 BallFree
D2+ 113 CTB: OurBallFree -> OurBall CTB=4 D=0.14
B4- 115 MaxVel BV=154.55 D=12.21 T=79 BallFree
B3* 116-113 (3)
R6- 118 ReAppear OC=4 []
R7- 121 Appear OC=5 7
R2- 122 BT=2 BU=1 OC=5 1=4,2=5,3=4,4=5,6=4
R5+ 123 Vanish OC=4 [8]
EPS 516 ---------------------------------------------
```

The Browsers

Once the file is loaded and the ⟨*errType*⟩ and ⟨*RefBox*⟩ tables have been filled, selections may be made from these tables. Selections are always for one or more steps and the positions of all players are shown in the field. The ⟨*Blue/White*⟩ option may be selected to change the background color of the field.



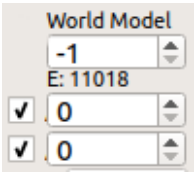
The Field view of a selected error type

Figure 2: *

When selecting an error type from the ⟨*ErrType*⟩ table, the ⟨*Action Table*⟩ is filled with all occurrences of this error. When one of these errors is selected, the Field image is shown with all robots in position. When the error is part of a cluster of errors, the first and last occurrence or the error is shown, indicating the start- and end positions of all TURTLEs and the ball.

	Action	Start	Stop	Steps	ID	BU	BT
1	ATW202	83	86	4	8	1	
2	DTW202	103	104	2	8	1	
3	AOW202	113	113	1	8	1	
4	ATW211	1110	1110	1	8	1	
5	ATW211	1121	1122	2	8	1	
6	ATW211	1135	1140	6	8	1	
7	ATW202	1142	1146	5	8	1	
8	ATW202	1148	1154	7	8	1	
9	ATW202	1156	1161	6	8	1	
10	ATW202	1164	1165	2	8	1	
11	ATW202	1168	1175	8	8	1	

When a step is selected, either by clicking the Action Table, or by selecting a new step number and the Debug switch is ON, then the Debug information is shown. This consists of a table with the following fields:



1. ID - The TURTLE ID
2. DC - DefCon code as used by this TURTLE
3. BID - ID of the TURTLE, whose Ball info is used
4. CPB - Status of the TURTLE's CPB switch
5. T - Team CPB ID
6. Opp - ID of Opponent that has the Ball
7. OC - Opponent Count of this TURTLE
8. OurD - Distance to the Ball for our TURTLES
9. OppD - Distance to the Ball for each Opponent

	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
1	1	2	1	0	0	0	5	10.4	0
2	2	4	1	0	0	3	5	4.4	4.73
3	3	4	1	0	0	3	5	0.0	4.68
4	4	4	3	0	0	4	6	4.68	2.03
5	6	4	6	0	0	3	5	2.82	3.17
6		4	3	11		4			
7		C1	C2	C3	C4	C5			
8	1	4		1	2	3	5		
9	2	4		3	1	2	5		
10	3	4		2	1	3	5		
11	4	4	5	2	3	1	6		
12	6	4		1	2	3	5		

The last line contains the IDs of the best data, used in the new World Model. The Technical Background of this part (chapter) offers a more detailed description of these fields, along with some examples.

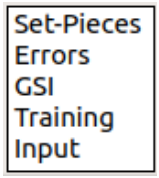
The bottom part contains a table, showing for each TURTLE, which opponents it sees. Especially with 'missing' or 'disappearing' robots, this can be used to check which opponents cannot be seen. It may be used to determine the amount of overlap or the reason why these opponents disappear from view. In this example all TURTLES, except 4, cannot see opponent 2.

The ⟨*RefBox Table*⟩ shows all episodes of the game. This part is only included here for completeness, but is explained in more detail in Part III (Set-Pieces).

	RefBox	Start	Stop	Steps	Row
1	ThrowIn Magenta	12	133	121	2
2	GoalKick Cyan	516	591	75	5
3	FreeKick Magenta	1066	1210	144	10
4	KickOff Cyan	1566	1761	195	13
5	KickOff Cyan	2090	2255	165	16
6	ThrowIn Magenta	2492	2583	91	19
7	KickOff Cyan	3233	3468	235	24
8	GoalKick Magenta	3698	3859	161	27
9	ThrowIn Magenta	4179	4309	130	30
10	ThrowIn Magenta	4584	4724	140	33

The three-state checkbox ⟨*Stp/Ann/GSI*⟩ is used to select one of the three views of the game state (will be replaced by the view selection box, currently at the bottom of the Tab window.):

1. **Stp** - SetPiece view in which the positions and trajectories of all TURTLEs and the Ball are shown along with the positions of the opponents.
2. **Ann** - Annotations view. During load it shows the error log. In other stages, background information is shown in the text window.
3. **GSI** - Shows the Game Situation Images, used for the learning part of the system. These are images that represent the game state and individual images for each agent, representing the Agent Situation Image (ASI). These images are input to the Neural Network, that learns the behavior patterns. More details can be found in Part V.



By selecting the check box, you can switch between the three views. During Load only the SetPiece and Annotation views are used.

Technical Background

Introduction

As part of a project to analyze and understand team behaviors of our opponents, we first analyze the behavior of our own team. Because the logfiles contain a lot of detail about the goals and actions of our robots, this allows analyses that are not possible on the data of our opponents, for which we only have location data.

The internal data from our robots is used to create a model that serves as ground truth, using all available information about actions, targets and combined game states. This will be used to learn to recognize behaviors from the position data only, first for our own team, so we can verify if the model learns properly. Then the same model will be used to make predictions about our opponents. In order to do this, the ground truth must be consistent, which is the main aim of this phase.

During game play, speed and accuracy of the robot positions are most important, however during analysis consistency of the data between robots and successive steps are more important. There are inconsistencies in the information of individual robots, but also differences in judgment of the game situation between robots.

This chapter explains the anomalies that were found and describes how the conflicts are resolved. They may partly be caused by a lack of understanding how the data should be interpreted. In other cases they may have been introduced by small errors, that were possibly resolved in later versions of the TURTLE software.

But in order to use all available logfiles for analysis, these conflicts in older files need to be detected and resolved, before they become useful. There may also be some underlying problems still remaining in the software, that need to be fixed. We checked the most recent files, and they also contain inconsistencies. Hopefully this information is of use to the team, responsible for the development on the TURTLE software. Otherwise it will serve to clarify the way that the data is organized.

To perform the analysis, we first studied the GreenField source code and used the logic as much as possible in our Python code. In the GreenField code, the information of a single robot is used as a basis to show the game conditions, so initially the same procedure was adopted. However in some instances there were inconsistencies, so the data from all active robots was combined to create a more complete world model. This showed that in some cases robots disagreed about the game situation and a facility was created to find these inconsistencies and fix them, using all the available data. The $\langle .mat \rangle$ files are converted on-the-fly by our software as part of the analysis. A new internal file format is generated, that is better suited for the intended analyses and contains the integrated World Model data with most known inconsistencies resolved.

The World Model

Integrating the data of each individual robot into a World Model involves several different procedures, depending on the particular items. This results in a new data structure that contains the following elements:

1. 0 - Ball Info
   Pose (X Y)
   ID of robot closest To Ball (CTB)
   ID of robot with CPB (Our or Opp)
2. 1 - 7 Turtle Info (7 elements)
   Pose (X Y O), ID
   Role, Role in Formation
   Agent Formation, BallID, Action
   Target (X Y)
3. 8 - 12 Opponent Info (5 elements)
   Pose (X Y), ID
   Role in Formation
   Agent Formation
4. 13 - 22 General information for each time step
   Seq #, Time Stamp
   Error, Vanishes, Certainty
   RefBox, DefCon
   Our Formation, Opp Formation
   Ball Owner state Us &Them (BO,BU,BT)

This information is only collected for real game actions, which for each RefBox situation happen between Start and Stop conditions. The remaining time steps only contain a single item, the Start and Stop code, to signal preparation activities that are not used for analysis.

The generated set-pieces are based on moments, defined by Ball Possession Turnovers (BPT). Information of Ball Ownership (BO) is closely related to the DefCon situation (See Figure 3). Sometimes the data contains incorrect assessments of the DefCon situation, sometimes also related to erroneous judgments of Ball Ownership. Table 1 describes the various situations.

Conflict Resolution

The anomalies that will be described in the sections to follow, are happening temporarily and may have a limited influence during a competition. When using the data for analysis however, conflicts may create a game condition that leads to erroneous conclusions. So we want to remove these conflicts as much as possible. There is another side benefit of performing error checking; they may reveal new or existing software problems during development or prior to a competition, that may be hard to spot with other methods. Fortunately there is enough redundancy in the information, which allows us to detect errors and additionally to make corrections in the following situations:

1. RefBox and DefCon
2. Ball Position
3. CPB
4. OppBall
5. Turtle locations
6. Opponent locations
7. Opponent IDs

Ball, Robots, DefCon & Action Conditions			
#	**Ball**	Cond	Data
B1-	DiffEst	BallEst != pose	ballDiff action
B2+	EstVel=0	Ball Vel > 0	D=ballDist T=time V=ballVel actionStr
B3	VelDiff	Ball Vel != appVel	BV=ballVel EV=estVel DV=velDiff action
B4-	MaxVel	Unlikely movement	BV=ballVel D=ballDist T=time action
B5+	LostBall	Ball lost	
B6	Ghosted	Ghosted ball	
	Robots	Cond	Data
R1	No Opps	Warning	BT=bestOpp BU=bestOur OC=oppCount ids
R2-	Mismatch	Turtles disagree	BT=bestOpp BU=bestOur OC=oppCount ids
R3-	NoGK	Warn No Keeper	
R4-	Less<5 opps	Warn missing Opps	actCount
R5+	Vanished	Opp disappeared	OC=oppCount [vanish]
R6-	Re-Appear	Opp re-appeared	OC=oppCount [vanish]
R7-	Appear	Missing opp back	OC=oppCount
	DefCon	Cond	Data
D1+	CPB Diff	DefCon CPB error	defCon->newDefCon CPB=cpb CTB=ctb D=ctbDist
D2+	CTB Diff	CTB=CPB < 0.6m	defCon->newDefCon CPB=cpb CTB=ctb D=ctbDist
D3+	BO Diff	Ball other team	defCon->newDefCon BO=BO CTB=ctb D=ctbDist
D4+	Invalid	Invalid DefCon	defCon->newDefCon BO=BO CTB=ctb D=ctbDist
	Action	Cond	Data
A1	NoBall	Shoot/Dribble no CPB	ID=id action
A2+	Scrum	Scrum undetected	CPB=ourCPB CTB=oppCTB defCon
A3+	NoScrum	Scrum not true	CPB=ourCPB CTB=oppCTB

Table 1: Consistency checks in Ball, Robots, DefCon and Action situations. Code with a - sign are warnings only.

DefCon

The RefBox situation is communicated by RefBox commands and is the same for all robots. This is sometimes not the case for the DefCon situation. Because robots have different beliefs about their position and those of the opponents, their judgment about the Game Situation differs.

This is primarily caused by differences about who currently owns the ball. Only the robot that currently has the ball in its ball-handler is certain about ball ownership. Often situations are found, where several of the robots think that a certain robot owns the ball, while that robot knows this is not the case. Therefore we take the CPB as the most reliable source of information and check this against the estimated position of the ball. To make analysis possible, the DefCon situation is crucial and should adhere to the state diagram in Figure 3.

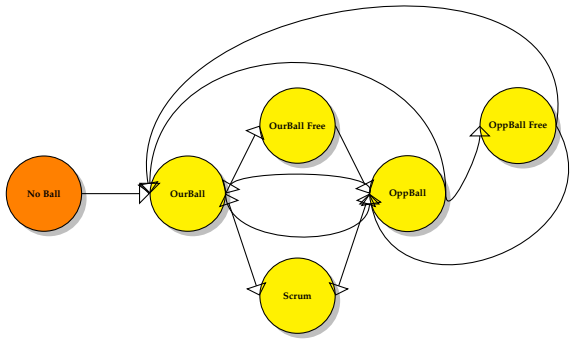


Figure 3: Game Turnover State Transition Diagram.

Ball Position and Velocity

If the current robot has its CPB flag turned on, the ball position is calculated from the robot's position and orientation. Otherwise, the most frequently used ball information is taken. The system also records which opponent currently owns the ball. To check this we find the robot, closest to the ball (CTB) and compare them. If another robot is closer to the ball, this is taken as the opponent CPB. However when the distance to the ball of an opponent is less than the default ⟨*CpbDist*⟩, this is not considered as owning the ball and an error is raised. In case the ball is lost, the last known position of the ball is used. In addition we also check the estimated ball velocity against its apparent velocity, which we calculate from the distance traveled between successive steps. If the ball has apparently moved, but the ball velocity was zero, we discard this movement. Also if the apparent velocity is higher than the default maximum ⟨*maxVel*⟩, this movement is ignored as well.
Occasionally, a situation arises when a 'vanished' or 'missing' opponent is moving with the ball. In such a case we signal a 'Ghosted Ball' situation (see section *Ghosted Ball movements*)
A number of things became apparent, while performing these checks. Many of the reported inaccuracies or inconsistencies occur in clusters. For instance, during a dribble or a shoot, the difference between the turtle's ball position and the estimate is usually around 17 cm, which could be the diameter of the ball. After a shot there is often a relatively large difference between the estimated velocity and the apparent velocity. This may be caused by a difference in frequency of measurement between the ball position and the robot positions of each TURTLE. When a shot is detected, there is a relatively long delay, before the ball movement is seen, sometimes about 8 time steps (400 - 800 ms)

CPB

One of the most important items is the Check Ball Possession (CPB) switch. This is a physical switch in the TURTLE's ball handler, that is pressed by the ball. In some cases we see that other robots 'think' that a certain robot owns the ball, while that particular robot 'knows' that this is not the case and vice versa. Each robot keeps an item that indicates which robot's ball info was used. In order to find the 'best' ball position data, we count which robot's ball info is used most. This is done for each cycle. We then check if this robot has its CPB switch turned on. This robot's information is then used to determine the position of the ball. When the CPB is off, we take the most frequently used ballID as basis for the ball position.

OppBall

For ball possession by the opponents, we follow a different procedure. Each TURTLE keeps a list of the positions of all opponents and a list with ID's associated with each of these positions. Not every robot can see all opponents, so we count how many opponents each robot can see. The robot that sees most robots is selected as the one, whose opponent positions are used. If several robots see the same number of robots, then the robot, whose ball info is used most is selected.
This however creates a problem with the identification, because each TURTLE assigns its own ID to each opponent. To overcome this issue we re-establish each opponent's identity in each step of an episode. See section *Opponent IDs* for more details.
We also count how many robots believe the opponent has the ball. This is an indication of how reliable the information is. Sometimes one of our robots has the ball (CPB), while the other robots believe, the opponent has the ball. This should usually be an indication of a Scrum situation. When multiple robots believe that the opponent has the ball, we also check the distance to the ball that each of these robots has calculated. We only do this however when the distance to the ball is less than the default ⟨*CpbDist*⟩.

Turtle locations

In the TURTLE software a complex procedure is used to determine the most probable location. For the analysis of team behavior, we use relative positions in an 'apparent' team formation, so the exact location is of less importance than during a game.

Opponent locations

Each robot keeps a list of robot locations and a separate list of IDs associated with these robots. In some cases we noticed that there seemed to be more than 5 robots on the field. In that case it is hard to determine which information is correct. Especially in the case when an additional robot seems to own the ball, there is a need to remove one of the other robots, since we only keep track of 5 robots.

Opponent IDs

To analyze individual robot behavior we need a consistent ID across all time steps of an episode. First of all we need to keep track of robots that are disappearing or re-appearing. We also determine if there is a robot in the opponent's goal area.
To track the opponents, we first identify the goal keeper. Then for the remaining robots, we take the most recent robots from the TURTLE's opponent ID list and track their movements between steps, using the hungarian algorithm (a.k.a. Kuhn-Munkres). See Table 2 for an example.

Tracking Opponents

To track our opponents we combine the opponent position information from all TURTLEs. To combine this data with the previous time step, we consider 3 different conditions. At the start of an episode we determine how many opponents are in the field and assign an ID to each of them, based on their distance to the opponent goal. The Keeper always gets ID 1. If there is no keeper, its slot is empty. If more opponents are missing, their slots are also empty and are labeled as Inactive and considered 'missing'. When during the episode robots disappear, they are considered 'vanished' and we take the last known position for that robot. In all other cases the robots are considered Active and are labeled as Opponent. When using the Hungarian, we calculate the distance between the current and previous position

for each opponent, so we can assign the ID that was generated at the start of the episode. For 'missing' robots we generate a position outside the field that is unique for each opponent. A similar, but farther away position is generated for 'vanished' robots. The Hungarian then calculates the assignment of the ID's by comparing the distances between the current and previous positions.

Disappearing and Reappearing IDs

TURTLEs keep a list of 10 obstacles and decide, based on their diameter if this is an opponent. Occasionally, a TURTLE may lose track of one or more opponents. In such a case, there are for a brief moment less than 5 robots visible (see step 583 in Table 2) .When the robot re-appears, a new ID is assigned (ID 5 in step 584), after which the system stabilizes and the oldest ID is removed (step 586). After that the system once again sees 5 robots. To deal with this situation, step 583 will keep the pose information of the robot with ID 4. In the next steps it will take the last 5 IDs from the list, dropping the old ID of 3. Robots are only physically removed from the field during a stoppage. So only if an episode starts out with less than 5 robots, the episode will assume that the opponent plays with 1 or more 'missing' robots. .

```
581 [3.  7.  2.  1.  4.  0.  0.  0.  0.  0.]  5 Opps
582 [3.  7.  2.  1.  4.  0.  0.  0.  0.  0.]
583 [3.  7.  2.  1.  0.  0.  0.  0.  0.  0.]  4 Opps
584 [3.  7.  2.  1.  4.  5.  0.  0.  0.  0.]  6 Opps
585 [3.  7.  2.  1.  4.  5.  0.  0.  0.  0.]
586 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]  5 Opps
587 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]
588 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]
589 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]
590 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]
591 [7.  2.  1.  4.  5.  0.  0.  0.  0.  0.]
```

Table 2: Example IDs of robots in steps 581-591

Another way of looking at the disappearance and re-appearance of robots is checking the positions of the opponents table (see Table **??**). When the data comes from different TURTLEs, their data needs to be correlated and be given a new ID. This is done in the first step of an episode. ID 1 is reserved for the keeper. The remaining poses are sorted on distance from the opponent's goal, to make sure that when a keeper was outside it's goal, it may later occupy this slot.
In all successive steps, the identities are assigned based on proximity, using a cost matrix. Notice that in step 106 (Table **??**), the pose of robot 2 is lost. We then take it's previous position and use this instead (not shown here). When a robot re-appears later on, it's empty slot is taken by the newly appearing robot in step 118. Also notice the warnings given in step 104 and 118, signaling disagreement between the Turtles about the number of observed opponents.

```
+ 104 [[0.0, 0.0, 0.0, 8.0, 0.0],
       [0.28, 6.11, 0.33, 9.0, 7.0],
       [3.51, 4.17, 0.33, 10.0, 7.0],
       [0.06, -0.16, 0.33, 11.0, 7.0],
       [-0.54, -5.16, 0.33, 12.0, 7.0]]
WM  2-Robot: 6 4 105 4 1=3,2=3,3=3,4=3,6=4
+ 105 [[0.0, 0.0, 0.0, 8.0, 0.0],
       [0.31, 6.1, 0.33, 9.0, 7.0],
       [3.51, 4.18, 0.33, 10.0, 7.0],
       [0.09, -0.17, 0.33, 11.0, 7.0],
       [-0.59, -5.17, 0.33, 12.0, 7.0]]
+ 106 [[0.0, 0.0, 0.0, 8.0, 0.0],
       [0.0, 0.0, 0.0, 9.0, 0.0],
       [3.46, 4.13, 0.33, 10.0, 7.0],
       [0.17, -0.25, 0.33, 11.0, 7.0],
       [-0.82, -5.22, 0.33, 12.0, 7.0]]
+ 107 [[0.0, 0.0, 0.0, 8.0, 0.0],
        . . . . . .
+ 116 [[0.0, 0.0, 0.0, 8.0, 0.0],
+ 117 [[0.0, 0.0, 0.0, 8.0, 0.0],
       [0.0, 0.0, 0.0, 9.0, 0.0],
       [3.47, 3.84, 0.33, 10.0, 7.0],
       [0.79, -0.85, 0.33, 11.0, 7.0],
       [-1.81, -4.97, 0.33, 12.0, 7.0]]
WM  2-Robot: 1 1 118 4 1=4,2=3,3=4,4=3,6=4
+ 118 [[1.15, 5.51, 0.33, 8.0, 7.0],
       [0.0, 0.0, 0.0, 9.0, 0.0],
       [3.53, 3.83, 0.33, 10.0, 7.0],
       [0.81, -0.95, 0.33, 11.0, 7.0],
       [-1.94, -4.97, 0.33, 12.0, 7.0]]
+ 119 [[1.15, 5.51, 0.33, 8.0, 7.0],
       [0.0, 0.0, 0.0, 9.0, 0.0],
       [3.53, 3.83, 0.33, 10.0, 7.0],
       [0.81, -0.95, 0.33, 11.0, 7.0],
       [-1.94, -4.97, 0.33, 12.0, 7.0]]
```

Table 3: Example of disappearing and re-appearing opponents

Ghosted Ball Movements

When an episode starts with one or ore missing robots, this may mean that they are not on the field, but they could also be temporarily invisible. Especially in the case, where a 'missing' or 'vanished' robot has the ball, the system will connect the ball with the wrong robot. When that ball is moved by actions of a missing robot, we see 'ghosted' ball movements. Such a situation appears in step 4991, where opponent 2 is missing and appears in step 5014.
There is another example of a ball moving, where the opponent is missing. (Find this example and describe it.)

Checking Who Sees Who

Robots that go 'missing' and 'vanish' occur frequently. When the other TURTLEs can see the missing robots this is compensated for, but sometimes there is no overlap between what the TURTLEs see, and the position of an opponent becomes unknown. In these cases, we keep the previous position of the opponent, until it re-appears. If one or more opponents are not visible at the start of an episode, it may mean that they are damaged and will not

be present during the episode ('missing'), or they are temporarily invisible ('vanished'). They may appear or re-appear later in the episode.
In Table 13 the situation of step 586 is shown. TURTLE 3 sees 6 opponents, of which the most recent 5 ones are used. TURTLEs 2, 4 and 6 see four opponents, while TURTLE 1 sees all of them. The numbers are the indices of the ID list from which the data is retrieved. The opponents that are missing are C5 (1, 4) and C2 (6). Now check the field image. TURTLEs 2 and 4 are missing C5, while TURTLE 6 is missing C2. C5 and C2 are close together, so this may cause the problem in this case. Using this facility we may check if proximity also causes other missing opponent occurrences.
Notice that the two 'vanished' robots C2 and C5 are at the far end of the field. The TURTLEs show their start and end position in this episode. Only our Keeper sees both opponents, that some other TURTLEs cannot see. It seems that robots that are far away and close together are mostly the robots that go 'missing'. That seems true for most of the 'vanishing' robots, which mostly are the opponent GoalKeeper and a nearby robot. Other cases are rare.

Missing Opps Step 586

Stp	ID	C1	C2	C3	C4	C5	OC
586	1	4	2	3	1	5	5
586	2	1	3	4	2		4
586	3	4	2	3	1	5	6
586	4	1	3	4	2		4
586	6	1		4	2	3	4

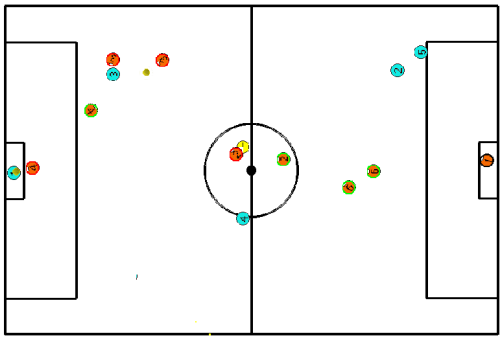


Table 4: Example of view of all opponents by each TURTLE.

Actions

Finally we check if the robot's action is consistent with the other known data. If a robot is performing a Shoot action, but is not in possession of the ball, this is an error situation. The same applies for a Dribble operation. In a Scrum situation, two robots must be very close to the ball.

The Log-file

The log-file is a large data structure that contains the data used by each of the TURTLEs for every time step. Examples given in this chapter are mostly from the file:

`Falcons_first_half.mat from 3 Dec 2018.`

In addition we also checked some more files both from 2018 and 2019, to make sure these were not incidental problems. We also checked the most recent file:

`07Jul2019_06_45_Water_first_half.mat.`

As a result of the introduction of STP the file format has been changed, so our software was adapted to allow te use of different file formats, so we can compare. In addition meta-Data was included in the file, which is much more flexible, but also introduced new codes that made comparison with older codes incompatible, so we must provide a facility to convert the codes to the newer values. Some more work is still needed to make a full evaluation of the situation of the latest version of the log-files.

Checking Criteria

The different error categories as shown in Table 1 are based on a number of criteria, that can be altered by changing some variables in the user interface. The Checking Rules are based on the following considerations:

Ball, Robots, DefCon & Action Conditions		
#	**Ball**	Cond/Data
B1-	DiffEst	The logfile contains the estimated Ball Position (BallEst). This is checked against the actual Ball Position. If the difference is larger than the $\langle DistCpbOur \rangle$ or $\langle DistCpbOpp \rangle$, this error is raised. $\langle DistCpb \rangle$ is the distance to the ball when a robot is considered to own the ball. The difference is given and the CPB owner is shown. If there is no ball owner, the action of the current robot is shown. D=ballDist T=time V=ballVel actionStr
B2+	EstVel=0	If the estimated velocity (EstVel) is zero, but the ball has moved between two time steps, this error is raised. The velocity, time and distance are shown as well as the robot action as in Ball1. D=ballDist T=time V=ballVel actionStr
B3	VelDiff	When the difference between the estimated velocity and the apparent velocity is larger than half the maximum ball velocity, this error is raised. The Ball velocitu is shown along with the estimated velocity, the distance and the time. Also the action is shown as in Ball1. BV=ballVel EV=estVel DV=velDiff action
B4+	MaxVel	If the estimated velocity is nonzero and the apparent velocity is larger than the $\langle MaxVel \rangle$ this is an unlikely movement. It is ignored and an error is raised. The same data is shown as in Ball3 BV=ballVel D=ballDist T=time action
B5+	LostBall	When we have a lostBall situation, where the ball cannot be found, the last known position of the ball is used.
B6+	GhostBall	When we have a ball moving without a known ball owner BV=ballVel D=ballDist T=time action
	DefCon	Cond/Data
D1+	CPB Diff	When the current ball owner (bestCpb) DefCon does not match the calculated DefCon an error is raised. The old and new DefCon are shown along with the CPB and the CTB and distance to the ball. defCon->newDefCon CPB=cpb CTB=ctb D=ctbDist
D2+	CTB Diff	When the calculated oppBall DefCon does not match the DefCon in a ballFree situation, the old and new DefCon are shown along with the robot closest to the ball (CTB) and the distance to the ball. defCon->newDefCon CPB=cpb CTB=ctb D=ctbDist
D3+	BO Diff	When the calculated DefCon does not match the DefCon in a ballFree situation, the old and new DefCon are shown along with the most recent ball owner (BO) and CTB with the distance to the ball. defCon->newDefCon BO=BO CTB=ctb D=ctbDist
D4+	Invalid	When the DefCon code is invalid, a warning is issued. A new DefCon is calculated and used instead. defCon->newDefCon BO=BO CTB=ctb D=ctbDist

Ball, Robots, DefCon & Action Conditions		
	Robots	Cond/Data
R1	No Opps	If during an episode all opponents are missing or vanished, a warning is issued. The bestOpp and bestOur are shown along with the number of oponents and the ID list. BT=bestOpp BU=bestOur OC=oppCount ids
R2-	Mismatch	If the turtle whose opponent data is used detects a disagreement between the number of opponents, a warning is given and a list of all seen opponents is given for each active turtle. Data shown is the same as Robots1. BT=bestOpp BU=bestOur OC=oppCount ids
R3-	NoGK	When an episode starts without a keeper in the opponent's goal area, a warning is issued. Missing opponents are considered missing during the entire episode. When they suddenly appear, they are no longer missing. Missing robots are treated differently than vanishing robots.
R4-	Less<5 opps	When an episode starts with less than 5 opponents, a warning for missing robots is issued. The number of active robots is shown. actCount
R5+	Vanished	If one of the opponents vanishes, a warning is issued. The number of opponents is shown along with the vanished list. OC=oppCount [vanish]
R6-	Re-Appear	If one of the vanished robots re-appears, a warning is issued. info shown is the same as Robots5. OC=oppCount [vanish]
R7-	Appear	If one of the missing robots appears, a warning is issued. info shown is the same as Robots5. OC=oppCount [missing]
	Action	Cond/Data
A1	NoBall	If a turtle action is to shoot the ball or to dribble, but that turtle does not own the ball, an error is raised. The ID and action is shown. ID=id action
A2+	Scrum	When we have an ourBall or oppBall situation, but the other party is also at CPB distance, a Scrum was not detected. The CPB and CTB are shown along with the DefCon. The new DefCon is Scrum. CPB=ourCPB CTB=oppCTB defCon
A3+	NoScrum	When the DefCon is Scrum but both robots are not in CPB distance, there is no Scrum and an error is raised. The CPB and CTB are shown. CPB=ourCPB CTB=oppCTB

Table 5: Consistency checks in Ball, Robots, DefCon and Action situations. Code with a - sign are warnings only.

Anomalies in the input

In a number of tables, we show examples of the kind of problems, found in the files. In each table we list the data from all TURTLEs in a number of fields:

1. **Stp** - Step number
2. **ID** - RobotID
3. **DC** - DefCon 1=OurBall, 2=OurBallFree, 3=Scrum, 4=OppBall, 5=OppBallFree
4. **BID** - BallID, ID of the TURTLE whose ball data was used
5. **CPB** - CPB for this TURTLE
6. **T** - CPBTeam, CPB code used by the team
7. **Opp** - BallOpp indicator which opponent has the ball
8. **OC** - Opponent count indicates how many opponents the TURTLE sees
9. **Dist** - Distance between the ball and the closest robot

We integrate this data by finding the best data for Our robots, the CPB and the Opp in three fields; ⟨*bestOur*⟩, ⟨*bestCpb*⟩, ⟨*bestOpp*⟩.

- **bestOur** is the robot whose ball info is used most frequently. If the robot owns the ball this info is used instead.
- **bestCpb** is the robot whose CPB switch is on.
- **bestOpp** is the robot that sees most opponents. If ⟨*bestOur*⟩ sees the same number of opponents, then ⟨*bestOpp*⟩ and ⟨*bestOur*⟩ are the same.

The tables shown in the following sections show the best in the last line.

DefCon does not match CPB

In step 89 robot 4 has the ball (CPB). All robots also think 4 has the ball. 4 is the most used for the Ball. All counted 4 opponents, so 4 is taken as bestOpp. Get Defcon of bestOur, which is 2 (OurBallFree), while the other robots think it is 1 (OurBall). Unfortunately state 2 for robot 4 is wrong, since it has the ball and should be OurBall. Strangely enough the other robots have the correct DefCon.

OurBall Step 89

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
89	1	1	1	0	4	0	4	11.92	0
89	2	1	4	0	4	0	4	6.78	0
89	3	1	3	0	4	0	4	2.37	0
89	4	2	4	1	4	0	4	0.20	0
89	6	1	4	0	4	0	4	7.84	0
+		2	4	4		4			

Table 6: Summary of step 89. Turtle 4 has the ball, but its DefCon is OurBallFree. Strangely enough the other robots correctly believe OurBall.

Conclusion: Robot 4 has the ball, but sets DefCon to ourBallFree (2), which should be OurBall (1)
One would expect that such a change would be picked up by the other robots in a few cycles, however only in step 95, the situation changes, because robot 4 detects that the opponent has the ball and sets DefCon to 4 (OppBall). It takes 2 steps, before the other robots notice, with the exception of the keeper, who beliefs OppBallFree (5) (Sequence not shown). Because the majority takes the information from robot 4, we can still use the correct DefCon.
In step 95 (see table 11), robot 4 detects oppBall with a distance of 90 cm between the opponent and the ball. However this robot is the only one that sees this opponent as having the ball, while 90 cm is a bit much to qualify as owning the ball. Actually TURTLE 3 is closer to the ball at a distance of 0.36, although it does not have the CPB on. The TURTLE software however, takes this as true. None of the others seem to agree. bestOur is 3, so DefCon is 1 (OurBall), which is correct after all.

OurBallFree Step 95

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
95	1	5	1	0	0	0	4	13.03	0
95	2	1	3	0	0	0	4	7.81	0
95	3	1	3	0	0	0	4	0.36	0
95	4	4	4	0	0	3	4	2.14	0.90
95	6	2	3	0	0	0	4	8.28	0
+		1	3	0		3			

Table 7: TURTLE 4 believes Opp 3 has the ball. but the distance to the ball is 90 cm, while TURTLE 3 is closer to the ball.

Multiple robots seeing an oppBall situation

In step 96 three TURTLEs see that opponent C3 has the ball. This time the opponent is still not closer to the ball. TURTLE 3 and 4 see the opponent as closest to the ball, but TURTLE 3 is still closer. The majority thinks that DefCon is ⟨*oppBall*⟩ (4), while the actual situation should be ⟨*ourBallFree*⟩.

oppBall 3x Step 96

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
96	1	5	1	0	0	0	4	13.07	0
96	2	4	3	0	0	3	4	7.83	0.85
96	3	4	3	0	0	3	4	0.31	0.84
96	4	4	4	0	0	3	4	2.19	0.84
96	6	2	3	0	0	0	4	8.28	0
+		4	3	10		3			

Table 8: Three TURTLEs agree that the opponent has the ball. However the distance to the ball is > 80 cm for all of them.

Missing a Scrum situation

In step 97 the same opponents are seen with the closest opponent for robot 3 at 74 cm. At the same time robot 3 also registered its CPB, which means that both robots own the ball, which signals a ⟨*Scrum*⟩ situation, but it reports ⟨*OppBall*⟩. The other robots report ⟨*OppBallFree*⟩. TURTLE 3, who owns the ball reports ⟨*oppBall*⟩, which should be ⟨*ourBall*⟩. This is very hard to explain and does not seem to make sense. In this case, TURTLE 3 is ⟨*bestOur, bestOpp and bestCpb*⟩ at the same time and so its DefCon is taken, which is ⟨*OppBall*⟩ and is wrong, since it's CPB is turned on. This should be ⟨*ourBall or Scrum*⟩.

Missing Scrum Step 97

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
97	1	5	1	0	0	0	4	13.12	0
97	2	4	3	0	0	3	4	7.87	0.75
97	3	4	3	1	0	3	4	0.2	0.74
97	4	4	4	0	0	3	4	2.28	0.75
97	6	4	4	0	0	3	4	8.28	0.75
+		4	3	3		3			

Table 9: Situation where both parties are reported to own the ball, but does not match the DefCon. Opponent 3 is too far from the ball to own it, but the report should either be OurBall or Scrum.

OppBall too far away

In step 99 robot 6 is still the only one seeing the opponent close to the ball, but at a distance of 1.40m and reports ⟨*OppBall*⟩. Most TURTLEs seem to agree, even though the ball is too far away and TURTLE 3 is closer to the ball.

OppBall far away Step 99

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
99	1	5	1	0	0	0	4	12.83	0
99	2	4	3	0	0	0	4	7.47	0
99	3	5	3	0	0	0	4	0.79	0
99	4	4	4	0	0	0	4	1.54	0
99	6	4	3	0	0	3	4	8.09	1.40
+		5	3	0		3			

Table 10: TURTLE 6 believes the opponent has the ball, however the ball is at a distance of 140 cm. This report is not used, since TURTLE 3 has the most BallID's and also sees 4 opponents.

DefCon does not match CPB

This situation happens when there is a discrepancy between a robot's assessment of ball ownership and actual ownership. TURTLE 4 has it's CPB turned on. There may be a delay before the other robots have communicated this change, but in some cases the situation is not registered at all. It means that the DefCon is ⟨*Our/OppBall*⟩ while it should be ⟨*Our/OppBallFree*⟩. This does generally not result in erroneous decisions, except in cases where a shot or dribble is involved.
For the opponents this is different. There we can only judge if an opponent owns the ball, if it is close to the robot. In practice we see situations, where there is a ⟨*oppBall*⟩ situation, where the distance is 1m or larger. In many cases there is a 1:4 difference in ball possession. This cannot be correct and it seems that the ⟨*oppBall*⟩ condition is not strict enough.
In this example in step 106, which we also have seen in the disappearing opponents in Table **??**, TURTLE 4 has it's CPB set, so it is the ⟨*bestCpb*⟩ and also the ⟨*bestOur*⟩. It has a DefCon of ⟨*OurBall (1)*⟩, so all seems well but in step 107 the CPB is off. ⟨*OurBall*⟩ remains set until step 118 (not shown). One would expect that within a few steps this would have been picked up by the other TURTLEs. However, they continue to believe that the DefCon state is ⟨*OppBallFree (5)*⟩.

OurBallFree Step 106

Stp	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
106	1	5	1	0	4	0	3	12.24	0
106	2	5	4	0	4	0	3	6.8	0
106	3	5	3	0	4	0	3	2.18	0
106	4	1	4	1	4	0	3	0.2	0
106	6	5	5	0	4	7	3	7.83	0
+		1	4	4	4	4			

Table 11: In step 106 CPB is set for TURTLE 4. Step 107 has no CPB, but OurBall remains set until step 118.

Vanishing and Re-Appearing Opponents

In step 5699 all TURTLEs see only 4 opponents. But they do not all see the same opponents. TURTLE 1, 3 and 6 are missing C2, while Turtle 2 and 4 are missing C5. With this information, the positions of all opponents should be known. At this time, we are not yet using such information, but a following version will use this information.

	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
1	1	1	4	0	4	3	4	8.79	0.39
2	2	1	4	0	4	3	4	4.56	0.38
3	3	1	4	0	4	3	4	1.53	0.37
4	4	3	4	1	4	3	4	0.2	0.39
5	6	1	4	0	4	3	4	1.83	0.36
6		3	4	4		4			
7		C1	C2	C3	C4	C5			
8	1	3		1	2	4	4		
9	2	4	2	1	3		4		
10	3	4		1	3	2	4		
11	4	4	2	1	3		4		
12	6	3		1	2	4	4		

Table 12: Example of vanishing opps step 5699.

In step 4637, TURTLE 3 sees all opponents, where before C1 and C3 were missing. So in this step they are re-appearing and their positions restored.

	ID	DC	BID	CPB	T	Opp	OC	OurD	OppD
1	1	2	3	0	0	0	3	11.43	0
2	2	5	3	0	0	0	4	7.71	0
3	3	5	3	0	0	0	5	0.4	0
4	4	5	3	0	0	0	4	6.41	0
5	6	2	6	0	0	0	3	2.25	0
6		5	3	0		3			
7		C1	C2	C3	C4	C5			
8	1		3		1	2	3		
9	2		4	3	1	2	4		
10	3	5	4	3	1	2	5		
11	4		4	3	1	2	4		
12	6		3		1	2	3		

Table 13: Example of re-appearing opponent step 4637.

Part II

The Analist and Critic

This part is currently under development and checks each episode for undesirable game situations. It generate remarks about the game situation in general and details for each of the field players. It is based on heuristics, provided by an expert Futsal coach and delivers advice about situations that are undesirable.

In the next phase of the system this part will be implemented and results in a similar logfile as during load, but then targeted at situations that should be avoided.

The following checks will be performed:

1. **Action** - What is an agent doing?

2. **Intent** - Why is the agent perfoming this action?

3. **Remarks** - Critical remarks about agent actions.

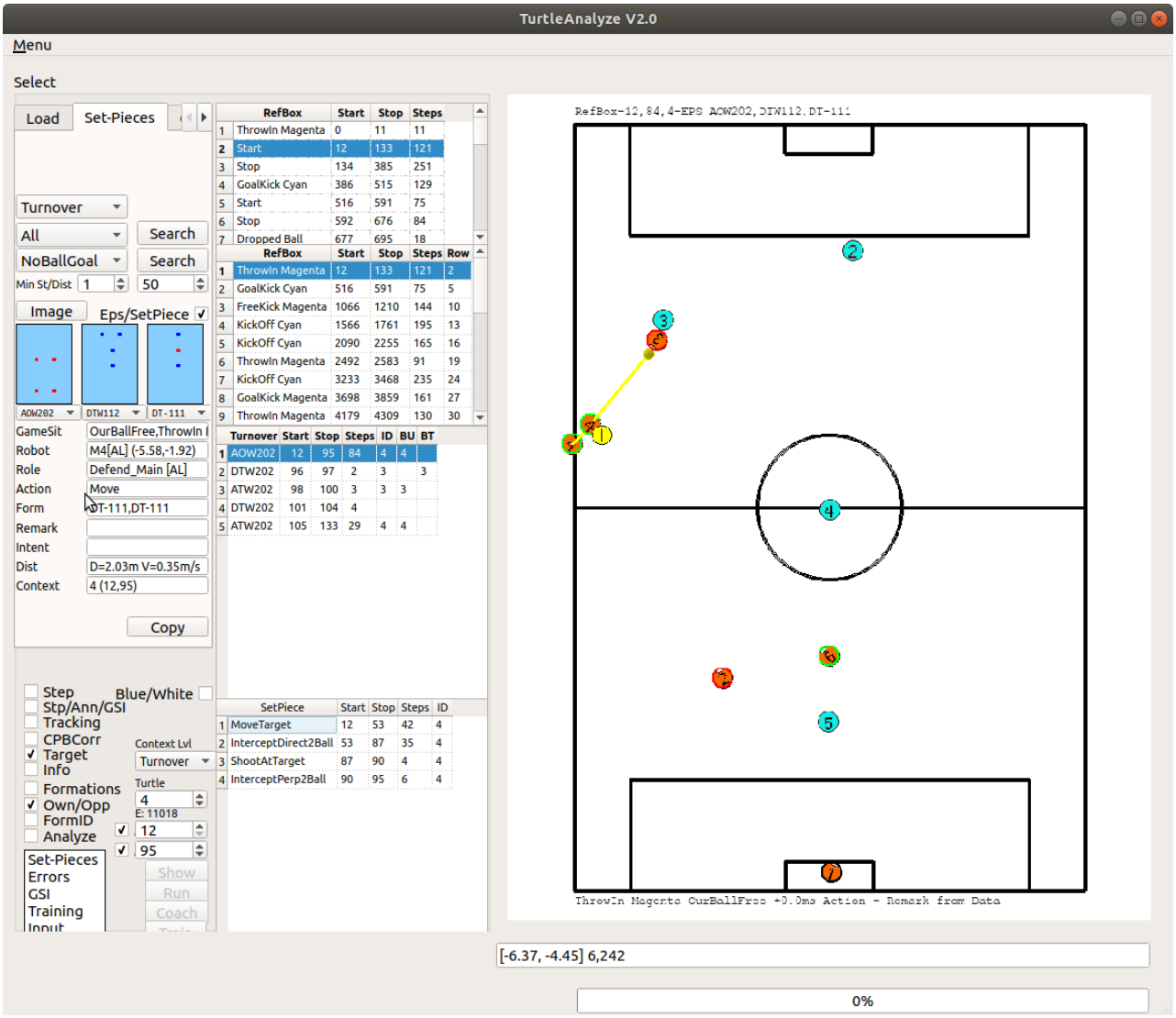
The Set-Piece view of the system

Figure 4: *

Part III

Making Set-Pieces

Set-Pieces are used by Soccer and Futsal coaches to describe how certain situations can best be handled. In coaching manuals, these are depicted as situation sketches, showing the positions of players and the ball, along with descriptions of the actions players should perform. The richness of information of the ⟨*.mat*⟩ files allows us to generate such images and form the basis for the learning phase of the system, where the set-pieces are used to analyze the behavior of opponent teams.



The Set-Piece view of the system

Figure 5: *

Part IV

Behavior Patterns

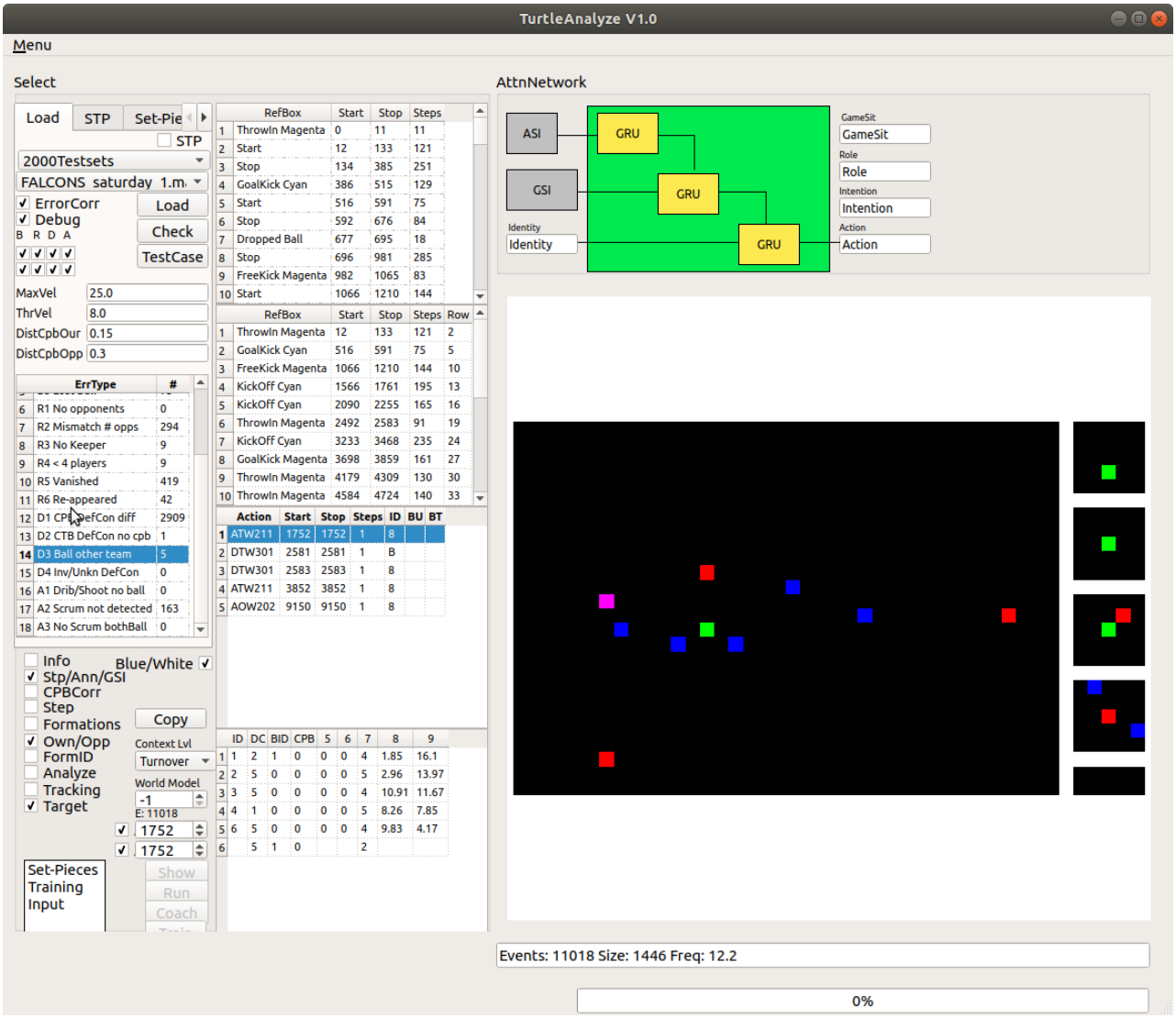
During this phase, the system will find frequent patterns that are used during one or more games. It detects situations like passing, attacking or defensive strategies for different game situations and player positions.

This part generates set-pieces for the entire team and detailed tactics, used by agents.

Part V

Learning Behaviors

This is the actual Deep Learning part of the system. Using the information, gathered by the preceding stages, a symbolic model has already been built, using Knowledge Engineering principles, leading to explainable models of agent behaviors. This GOFAI approach teaches the learning component of the system, what certain behaviors like dribbling, rotating, passing and marking look like. With that symbolic information the system can now learn to understand the behavior of the opponents, for which we only have position data. The system learns to recognize opponent behaviors and classifies these into a library of often observed behaviors of each of our opponents, by analyzing the game-logs, captures in the ⟨*.mat*⟩ files.



The GSI view of the steps in an episode, that forms input to the Neural Network.

Figure 6: *

Part VI

Opponent Analysis

When the system has learned behavioral patterns, these are used to analyze the behavior of our opponents and strategies and tactics, followed by competing teams are detected and collected.