# RoboTeam Twente
# Extended Team Description Paper 2020

Lukas Bos, Yuhanun Citgez, Haico Dorenbos, Aaron Eichler, Rolf van der Hulst, Luuk Klein Nagelvoort, Timo van der Kuil, Jordi Luong, Lars Poort, Finn Prinsenberg, Casper de Regt, Luc Schoot Uiterkamp, Rik Schrijver, Emiel Steerneman, Paul Vacariu, Jesse van Werven, Sabine van der Werff, and Selina Zwerver

RoboTeam Twente
University of Twente (UT), Enschede, the Netherlands
De Hems 10, Bastille 304, 7522NL, Enschede, the Netherlands
Website: `https://roboteamtwente.nl`
Contact: `info@roboteamtwente.nl`

**Abstract.** Roboteam Twente is an interdisciplinary student team from the University of Twente, competing in the Small Size League (SSL) for Robot Soccer. This paper presents the teams plans to be realized before the Robocup of 2020 in Bordeaux. The paper focuses on mechanical changes to improve position control on short distances, maintainability and durability in both software and electronics, as well as a team-wide effort to incorporate robots with different physical specifications into the team in an efficient manner.

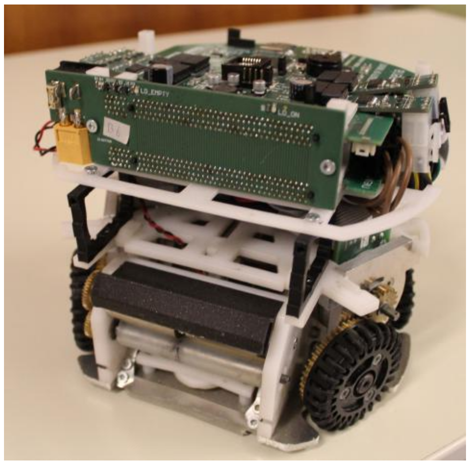

Fig. 1: One of the robots that went to the Robocup of 2019 in Sydney.

## 1 Introduction

Roboteam Twente is a multi-disciplinary team with students from the University of Twente and the University of Applied Sciences Saxion. The team was founded back in 2016 by a group of students striving to challenge themselves in the fields of robotics and AI. Now, three teams later, it is up to us to improve upon their design and to further innovate the SSL robots they built. Our teamgoal is to innovate and to inspire in the fields of robotics and AI.
Our technical team is subdivided into 4 subteams; Mechanics, Electronics, Control and AI.

Mechanics is concerned with the physical build and model of the robot. They design and build every part of the robot that is not a circuit board and are in charge of the mechanical maintenance of the current robots.

The electronics team designs and assembles our electronic circuit boards. They are also in charge of repairing any possible damage to the circuit boards.

The control team designs the robot control systems, which make sure that the robot can execute low-level behaviour. These low-level behaviours include for example following a path without straying out of the desired zone.

Finally, the AI team works on the high-level decision making of the system. They design our AI, creating strategies and conditions for their execution.

## 2 Robot Specifications

Table 1: Robot Specifications.

| Robot 4.0 | |
|---|---|
| Dimension | ⌀179 $x$ 149 mm |
| Weight | 2750 gram |
| Ball coverage | 20% |
| Driving motor | Maxon EC-45 flat 30 Watt<br>Maxon EC-45 flat 50 Watt |
| Dribbling motor | Maxon DCX 19s |
| Wheel diameter | 55 mm |
| Wheel gear ratio | 2:5 |
| Encoder driving motors | MILE 1024 CPT |
| Dribbling bar diameter | 10 mm |
| Dribbling bar length | 73 mm |
| Microcontroller | STM32F417VGTx |
| Ball sensor | zForce AIR Touch |
| Motor controller | ROHM BD63002AMUV |
| Inertial measurement unit | Xsens MTi-3-8a7g6t |
| Battery | 3S3P 11.1V LiPo<br>6S1P 22.2V LiPo |

## 3 Electronics

### 3.1 Circuit boards

Our electronics design consists of four different circuit boards, which we have named according to their position. Their names and functions are as follows:

- **The topboard**. The topboard consists of the processor, a communications chip, an accelerometer/gyroscope and a memory circuit. It handles all logic that has to be performed on the robot itself and communicates with the computer.
- **The bottomboard**. The bottomboard handles everything related to kicking and chipping, it contains a capacitor charging circuit and a large capacitor that supplies power to the respective solenoids.
- **The backboard**. The backboard supplies the power from the battery to the topboard and bottomboard, it also checks whether the battery is empty. The backboard also functions as a bridge for signals between the topboard and bottomboard.
- **The motordrivers**. The motordrivers are connected to the topboard and are four identical circuits that each respectively control a single motor.

### 3.2 Changes

The electrical design from last was largely performing sufficiently. There was only one relatively major issue that was experienced while using the old circuit boards. Namely that the motordrivers would burn out too often and were not reliable enough. More specifically, the MOSFET's that supply power to the motor would heat up significantly after extended amounts of playtime, with temperatures reaching 50 degrees Celsius. Due to this lack of reliability and durability the motordrivers had to be redesigned. There were also some changes resulting from this years adoption of motors that operate at a voltage of 24V and the consequential usage of 22.2V batteries.

**Motordrivers** To fix the motordriver issue, they were just redesigned completely. The PCB dimensions were retained, however, as changing those would result in more unnecessary changes. One adaptation that will be made specifically is that a different motor pre-driver Integrated Circuit (IC) will be used, which will be able to withstand a higher voltage (up to 26.4V). This will mean that the motordrivers will be able to deal with the higher voltages and will also have a higher longevity. The IC concerned is the ROHM BD63002AMUV. This chip also has a lower external component count and is therefore easier to interface on a PCB.

The old MOSFET's have been upgraded to be able to supply more power. They can now handle a higher current, partly due to a lower $R_{DS,on}$, the $R_{DS,on}$ being the internal resistance of the MOSFET and thus also the factor that causes the MOSFET heating up. The new $R_{DS,on}$ is as low as $1.9\Omega$, which means that the new MOSFET's should barely heat up with the currents the motordriver is dealing with. Also, the old circuit board contained three packages of two MOSFET's, whereas now it uses six MOSFET's with their own respective packages. This does increase the footprint of the MOSFET's on the PCB, but should dramatically remedy the thermal problems we experienced. In practice there were immediate results, the old motordrivers always felt hot to the touch, but the new motordrivers do not seem to heat up at all, even after an hour of play.

**Higher voltage** As stated above, we will be using 22.2V batteries for our system this year. Since our previous circuits are only compatible with 11.1V battery input some of the circuits will have to be changed in order to work with a higher voltage input. The majority of the changes will occur on the bottomboard; the capacitor charge controller IC will be swapped out with a LT3751 chip, which is a more flexible controller. On the backboard we will then be changing the battery empty detection circuit and on the topboard there will only be a couple of component swap outs. This is because the microcontroller and peripherals all work at either 3.3V or 5V, so only a few components in the voltage conversion will have to be changed.

## 4 Mechanical Design

Regarding the mechanical design of the robots, the focus was mainly on improving the driving capabilities and ball handling. The first part will be accomplished by implementing a different wheel configuration and higher power motors. The latter entails a redesign of the front end of the robot, including the chipping and dribbling mechanisms.

### 4.1 Wheel Configuration

The first design of the robot, from 2017 until 2019, saw its wheels placed in a 30-30 angle configuration with respect to the robot's driving direction. Figure 2 shows the bottom plate with a 30-30 wheel configuration and how those angles are defined.

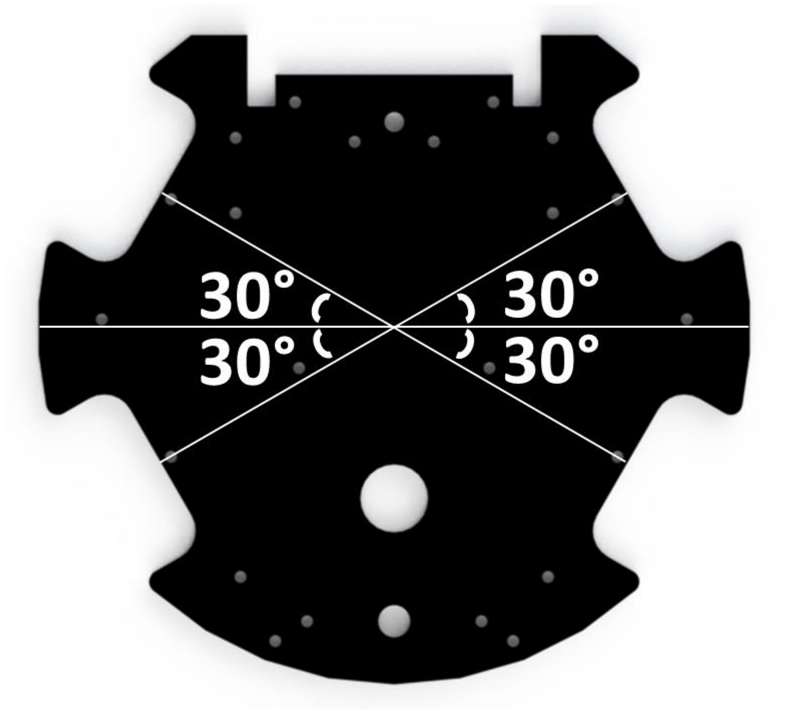

Fig. 2: Definition of wheel configuration angles on the bottom plate. Shown above is the 30-30 wheel configuration.

The main advantage of this set-up was that it allowed us to drive forwards and backwards at very high speeds. When driving sideways, however, the configuration proved to work insufficiently. This was a problem especially whenever accurate positioning was needed for ball receival. For this reason, the decision was made to alter the wheel configuration.

Four prototypes with different wheel configurations were laser cut and tested. The new configurations up for testing were 45-45, 30-60 and 30-45. The 30-30 configuration was used for comparison and was also laser cut to compensate for the material difference in our original robots. The criteria used for the tests were short distance sideways driving, long distance sideways driving and forward and backwards driving. The most important criteria were accuracy of positioning and velocity. We also took into account the receiving area that each configuration would allow.

Hypothetically, in that case, the 45-45 wheel configuration would be optimal for driving, since friction forces would cancel each other out making the movement of the robot more predictable [1]. The tests showed, however, that there was a negligible difference in driving performance between the 30-60 configuration and 45-45 configuration. The criterion on which to compare the two would then have to be the receiving area, which would be much larger in the 30-60 configuration. It was therefore decided to change the wheel configuration from 30-30 to 30-60.

### 4.2 50 Watt Motors

Since the field size may increase this year and we will also need to improve our positioning accuracy when moving across small distances, we will be implementing higher power motors in our robots this year. Up until now, the Maxon EC-45 flat 30 Watt motors were used. At the RoboCup 2020, a hybrid team will be used, where some of the robots will be equipped with Maxon EC-45 flat 50 Watt motors.

This means that the code will have to be compatible with different robots, which is a feature that will be increasingly useful in the coming years. It also provides us with the ability to test and analyze the performance of new designs during matches. In this way a clear conclusion can be obtained, which shows whether the new design is a significant improvement.

In the new design, the changing positions of the wheels in the back interfere with the position of the Geneva mechanism. Additionally, the 50 Watt motors are 5 millimeters thicker than the 30 Watt motors, which means that there is even less space for the Geneva mechanism. Since both of these adjustments in the design are thought to be a significant improvement on the robot functionality and since the Geneva mechanism did not give us a major advantage at the RoboCup 2019, it was chosen to remove the Geneva mechanism from the robot this year.

### 4.3 Chipping Mechanism

With the introduction of the larger 50 Watt motors, there was less space for the chipper. To compensate, the chipper was redesigned to be 12 millimeters less wide. This does not really influence the functionality, since the chipper will still make full contact with the ball independent of the ball's position on the dribbler.

Additionally, the mechanism used for the chipper at the RoboCup 2019 was not performing adequately, as it did not chip that far. This was due to a small range of motion as well as the fact that part of this motion was blocked by the dribbler. Changes were made by changing the location of the rotation point of the chipper (Figure 3 and Figure 4), which allowed the chipper a larger range of motion and longer contact with the ball. This also meant that the dribbler would no longer be in the way.

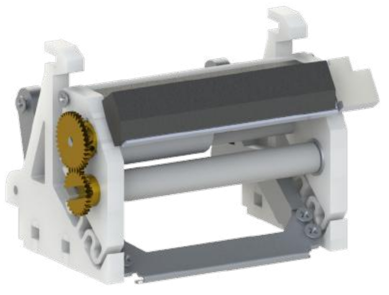
Fig. 3: Old Chipper with original rotation point.

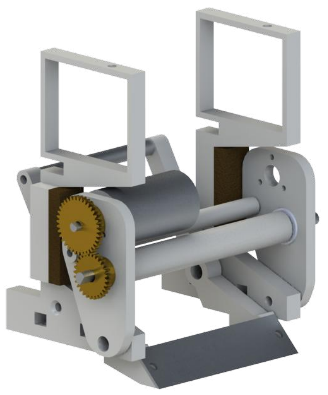
Fig. 4: New chipper with adapted rotation point.

### 4.4 Dribbling Mechanism

Looking at last years competition, there were a lot of areas in which it was evident that we could improve the front end significantly. The main focus points were the redesign of the damping system and a different shape of the dribbler bar.

**Damping** The hinges of the previous dribbler assembly each had two S-shaped cut-outs in the front, shown in Figure 5, that were to act as a spring and enable the dribbler bar to move backwards in order to dampen the speed of an incoming ball. In practice, it appeared that the construction was too rigid to move backwards at lower ball speeds. At high ball speeds, the dribbler assembly *was* able to move backwards, but even then there was not enough damping and the mechanism basically acted like a spring, re-transferring the energy right back into the ball. Both issues caused the ball to bounce of the dribbler as soon as the robot received it, which meant that receiving was problematic. Another drawback of the cut-outs was that the hinges were not able to withstand any large impact, such as collisions with other robots, causing the hinges to break at the S-shapes.

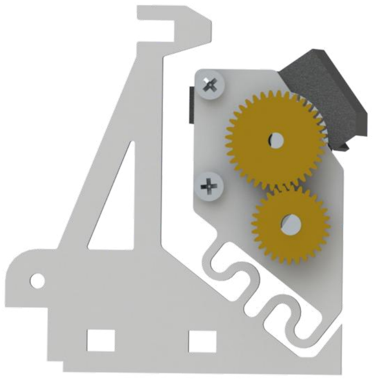
Fig. 5: Previous design of the dribbler hinges.

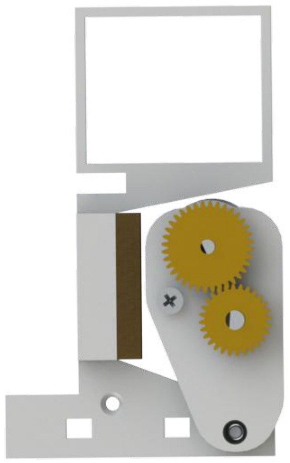
Fig. 6: New design of the dribbler hinges.

To overcome the aforementioned difficulties, the hinges were redesigned and damping was added, as shown in Figure 6. The new hinges have a rotational joint at the bottom and damping will be added at the back them. During backward motion, the hinges will run into the foam padding, such that the major part of the kinetic energy of the ball can be dissipated. Since the hinges are able to rotate freely, this system is also applicable to lower ball speeds. The support for the redesigned hinges also attaches to the battery holders. This is done to allow for easier assembly of the front and the battery holders, but also to increase the durability of the battery holders. Maintaining and regaining of the equilibrium position of the hinges will be done by limiting the forward motion in the front and creating a spring-damper system behind the hinge.
To validate the new set-up of the dribbler hinges, tests need to be carried out.

**Dribbler bar** Currently, a straight dribbler bar is used, which consists of a metal shaft as core and a silicon tube as outer layer. The latter provides grip on the ball and a little additional damping, and is hard and durable enough to still enable good ball control. The problem, however, is that the dribbler has no way to actively center the ball. This means that the ball would always be in a different position when shooting, resulting in inconsistent straight kicks and chip kicks.

To improve the consistency of both kicks, several different dribbler bars are being tested. The tested shapes are an hourglass shape, a straight bar with a thread and a crowned pulley, all with slight variations in, amongst others, thread pitch or taper angle. The designs are shown in Figure 7, 8 and 9, respectively.

The hourglass-shaped dribbler bar proved to be ineffective in centering the ball both in practice and in theory. It was found that the ball was driven to the sides, instead of taking position in the centre of the dribbler bar. This can be explained as follows: the dribbler bar has a variable radius, $r$, introducing an increasing circumferential velocity along the dribbler bar from the center to either side, given by

$$v = r\omega. \tag{1}$$

This increasing velocity profile along the dribbler bar imposes an outward spin on the ball, causing it to move away from the center, which was undesirable. One could consider this as a cascade reaction, forcing the ball fully outward eventually.

The dribbler bar designs of a straight bar with a thread and the crowned pulley both need extended testing. An improved design of the dribbler bar will be implemented at the RoboCup 2020.

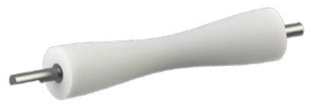
Fig. 7: An hourglass shaped dribbler bar.

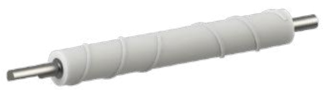
Fig. 8: A straight dribbler bar with thread.

Fig. 9: A crowned pulley dribbler bar.

## 5 Control

### 5.1 Wheels control

In order to control the robot, it is necessary to transform its desired velocity into the required speed of each of the wheels and vice versa. The situation of the new asymmetrical wheel configuration is shown in Figure 10.

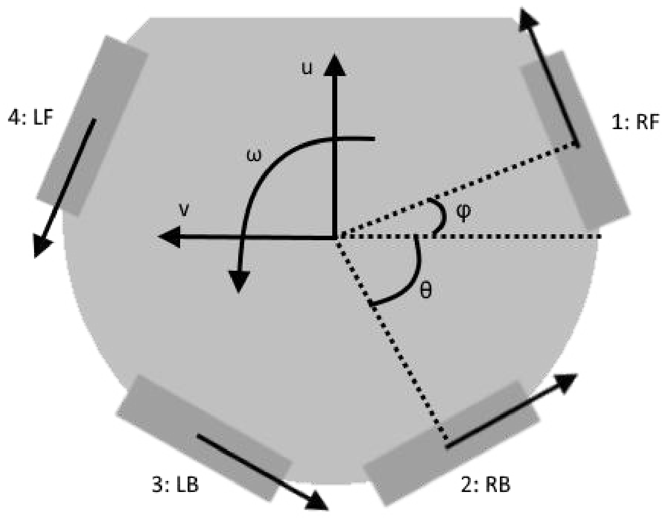

Fig. 10: Definitions of asymmetrical wheel configuration

The transformation [1] from robot velocity to wheel speeds is given by

$$\begin{bmatrix} \omega_{RF} \\ \omega_{RB} \\ \omega_{LB} \\ \omega_{LF} \end{bmatrix} = [bodyToWheels] \begin{bmatrix} u \\ v \\ \omega \end{bmatrix} \tag{2}$$

$$[bodyToWheels] = \frac{1}{r} \begin{bmatrix} cos(\phi) & sin(\phi) & R \\ cos(\theta) & -sin(\theta) & R \\ -cos(\theta) & -sin(\theta) & R \\ -cos(\phi) & sin(\phi) & R \end{bmatrix}$$

in which $r$ denotes the radius of the wheels and $R$ the radius of the robot. On the robots the last column, angular velocity, is not included. The reason for this is that the robot control is separated into translational velocity control and angle control. Therefore, the reference angular velocity does not have to be accounted for by the wheel speed references.

The transformation from wheel speeds to robot velocity can then be determined by taking the pseudo-inverse of $[bodyToWheels]$ [1]. The pseudo-inverse is used because the matrix $[bodyToWheels]$ is non-square and gives a generalization of the inverse matrix. This results in

$$\begin{bmatrix} u \\ v \\ \omega \end{bmatrix} = [bodyToWheels]^{+} \begin{bmatrix} \omega_{RF} \\ \omega_{RB} \\ \omega_{LB} \\ \omega_{LF} \end{bmatrix} \tag{3}$$

$$[bodyToWheels]^{+} =$$
$$r \begin{bmatrix} \frac{cos(\phi)}{2(cos^2(\phi)+cos^2(\theta))} & \frac{cos(\theta)}{2(cos^2(\phi)+cos^2(\theta))} & \frac{-cos(\theta)}{2(cos^2(\phi)+cos^2(\theta))} & \frac{-cos(\phi)}{2(cos^2(\phi)+cos^2(\theta))} \\ \frac{1}{2(sin(\phi)+sin(\theta))} & \frac{-1}{2(sin(\phi)+sin(\theta))} & \frac{-1}{2(sin(\phi)+sin(\theta))} & \frac{1}{2(sin(\phi)+sin(\theta))} \\ \frac{sin(\theta)}{2R(sin(\phi)+sin(\theta))} & \frac{sin(\phi)}{2R(sin(\phi)+sin(\theta))} & \frac{sin(\phi)}{2R(sin(\phi)+sin(\theta))} & \frac{sin(\theta)}{2R(sin(\phi)+sin(\theta))} \end{bmatrix}$$

Also the forces that the wheels exert on the ground can likewise be transformed to forces on the body of the robot [1].

$$\begin{bmatrix} F_x \\ F_y \\ F_\theta \end{bmatrix} = [forceCoupling] \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_3 \end{bmatrix} \tag{4}$$

$$[forceCoupling] = \begin{bmatrix} cos(\phi) & cos(\theta) & -cos(\theta) & -cos(\phi) \\ sin(\phi) & -sin(\theta) & -sin(\theta) & sin(\phi) \\ R & R & R & R \end{bmatrix} \tag{5}$$

### 5.2 State estimation

It is important to know the state of the game in order to decide what to do and to control the robots to execute what is expected of them. For this the states have to be estimated, which is done using Kalman filters. On the computer the state of all robots and the ball are estimated. Additionally, there is a state estimation on each robot. The current Kalman filters are relatively simple, however, and the estimations are not always accurate enough. By improving and extending the filters, the performance of the robots could be improved. This section explains several of the additions to make the state estimations more accurate and more detailed. Some generally known methods will also be taken into consideration, but these will not be further explained. The relative methods include for example implementing a two phase ball model[2, 3] and control input in the Kalman filters.

**Feedback** Feedback from the robot to the computer has been introduced in the TDP of 2018[4]. In Table 2 the feedback packet that will be used this year is shown. To improve the state estimation, the acquired information could be helpful. By taking into account the velocity of the robot as an observation in the Kalman filter, the estimated state no longer solely relies on vision data. This will result in more accurate state prediction in case the robot data from vision is either not optimal or unavailable.

Table 2: Feedback packet sent from robots to computer

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| robot ID | | | | | | | |
| A | B | BS | HB | ball position | | | |
| $\rho$ | | | | | | | |
| $\rho$ | | | angle | | | | |
| angle | | | | | $\theta$ | | |
| $\theta$ | | | | | | | |
| WB | signal strength | | | | | | |

| | |
|---|---|
| **A**ccelerometer calibrated | Accelerometer on the robot is calibrated |
| **B**attery low | Battery on the robot is low |
| **B**all **S**ensor working | Ball sensor on the robot is working |
| **H**as **B**all | Robot has the ball |
| **W**heels **B**raking | Wheels are in braking mode |
| $\boldsymbol{\rho}$ and $\boldsymbol{\theta}$ | Velocity of the robot in polar coordinates |

**Robot model** For improvement of the control of the robot and making design choices, it is useful to have a mathematical model of the robots. The robots can be seen as a body with four independent motors.

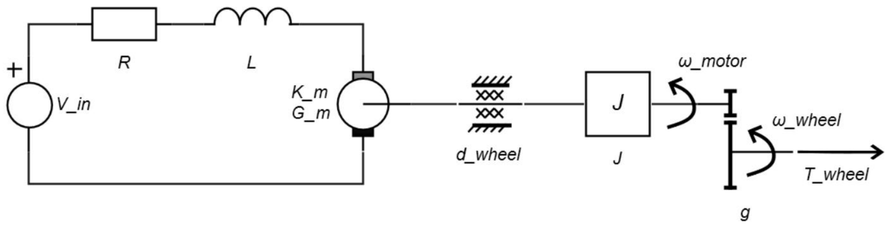

Fig. 11: Schematic of the model

The electrical part of the motor model consists of a voltage input $V_{in}$ (applied as a PWM signal) and a coil. The motor behavior itself is modeled using its speed and torque constants which are defined by $G_m = \frac{\omega}{e}$ and $K_m = \frac{T}{i}$ respectively where $\omega$ is the output speed of the motor, $e$ the voltage drop across the motor, $T$ the torque delivered by the motor and $i$ the current through the motor. The coil can be modeled as an inductance $L$ and a resistor $R$ in series (since the coil is essentially a long wire, its resistivity will play a role). The mechanical part can be seen as a shaft with an inertia $J$ (the inertias of the gears and wheels are included in this inertia), a damper $d_{wheel}$ to simulate friction of the motor, gears and wheels and gears ($g = \frac{\omega_{motor}}{\omega_{wheel}}$) to translate speeds and torques from the motor to the wheels.

The inductance can be described as:

$$L\frac{di}{dt} = e_L \tag{6}$$

Using Kirchhoff's voltage law:

$$L\frac{di}{dt} = V_{in} - e_R - e_{motor} \tag{7}$$

Using the motor equations and Ohm's law:

$$L\frac{di}{dt} = V_{in} - iR - \frac{\omega_{motor}}{G_m} \tag{8}$$

The angular velocity of the motor can be transformed to the angular velocity of the wheels using:

$$\omega_{motor} = (-)g\omega_{wheel} \tag{9}$$

The minus sign in this expression will be omitted from this point on since the rotation of the motor can also be defined to turn the other way around without consequence. This yields a set of 4 equations, describing the state of the current through the motors:

$$L\{\frac{di_n}{dt}\} = \{V_{in,n}\} - R\{i_n\} - \frac{g}{G_m}\{\omega_{wheel,n}\} \tag{10}$$

The mechanical side of the motor can be described as:

$$J\frac{d\omega_{motor}}{dt} = T_{motor} - T_{d,wheel} - T_{out} \tag{11}$$

Where $T_{out}$ is the torque that the motor delivers for driving the robot, as seen from the motor side of the gears. Using the motor equations and shifting the loads to the wheel side of the gears yield a set of four equations describing the angular velocity of the wheels:

$$Jg\{\frac{d\omega_{wheel,n}}{dt}\} = K_m\{i_n\} - gd_{wheel}\{\omega_{wheel,n}\} - \frac{1}{g}\{T_{wheel,n}\} \tag{12}$$

Using the steady state solution of the above sets of equations and with no load applied (i.e. wheels are off the ground) yields a solution for the friction coefficient of the motor model:

$$d_{wheel} = \frac{K_m}{gR\omega_{wheel}}(V_{in} - \frac{g\omega_{wheel}}{G_m}) \tag{13}$$

The dynamic behavior of the body of the robot can be described with the equations:

$$m\frac{dv_x}{dt} = \sum F_{wheels,x} - d_x v_x$$

$$m\frac{dv_y}{dt} = \sum F_{wheels,y} - d_y v_y$$

$$I\frac{d\Omega}{dt} = \sum F_{wheels,\theta} - d_\theta \Omega \tag{14}$$

Where $d_n v_n$ is the friction friction force experienced in its respective driving direction due to for example field roughness. These equations can be combined in matrix format:

$$[M]\{\frac{dv_n}{dt}\} = \{\sum F_n\} - [D]\{v_n\} \tag{15}$$

The forces that the motor delivers to the body are coupled via:

$$\{\sum F_n\} = [forceCoupling]\{f_{wheel,n}\} \tag{16}$$

Where $f_{wheel,n}$ is the force that the respective wheels apply to the ground in the direction of the wheel. These forces are converted to torques delivered by the wheels using:

$$r\{f_{wheel,n}\} = \{T_{wheel,n}\} \tag{17}$$

Where $r$ is the radius of the wheels. Using this result in the body dynamics:

$$[M]\{\frac{dv_n}{dt}\} = \frac{1}{r}[forceCoupling]\{T_{wheel,n}\} - [D]\{v_n\} \tag{18}$$

Equation 12 can be rewritten as:

$$\{T_{wheel,n}\} = gK_m\{i_n\} - g^2 d_{wheel}\{\omega_{wheel,n}\} - g^2 J\{\frac{d\omega_{wheel,n}}{dt}\} \tag{19}$$

Putting this result in equation 18

$$\begin{aligned}[M]\{\frac{dv_n}{dt}\} =& \frac{1}{r}[forceCoupling](gK_m\{i_n\} - g^2 d_{wheel}\{\omega_{wheel,n}\} \\ &- g^2 J\{\frac{d\omega_{wheel,n}}{dt}\}) - [D]\{v_n\}\end{aligned} \tag{20}$$

Converting wheel speeds to velocities of the body of the robot yields:

$$\begin{aligned}[M]\{\frac{dv_n}{dt}\} =& \frac{1}{r}[forceCoupling](gK_m\{i_n\} - g^2 d_{wheel}[bodyToWheels]\{v_n\} \\ &- g^2 J[bodyToWheels]\{\frac{dv_n}{dt}\}) - [D]\{v_n\}\end{aligned} \tag{21}$$

Rearranging the terms:

$$\begin{aligned}([M]+&\frac{g^2 J}{r}[forceCoupling][bodyToWheels])\{\frac{dv_n}{dt}\} = \frac{gK_m}{r}[forceCoupling]\{i_n\} \\ &- (\frac{g^2 d_{wheel}}{r}[bodyToWheels] + [D])\{v_n\}\end{aligned} \tag{22}$$

The accelerations of the body of the robot can be found by taking the inverse of the matrix on the left-hand side of the equation:

$$\begin{aligned}\{\frac{dv_n}{dt}\} &= ([M] + \frac{g^2 J}{r}[forceCoupling][bodyToWheels])^{-1} \\ (\frac{gK_m}{r}&[forceCoupling]\{i_n\} - (\frac{g^2 d_{wheel}}{r}[bodyToWheels] + [D])\{v_n\})\end{aligned} \tag{23}$$

With the help of equation 8, which can be rewritten to:

$$\begin{aligned}\{\frac{di_n}{dt}\} =& \frac{1}{L}(\{V_{in,n}\} - R\{i_n\} \\ & - \frac{g}{G_m}[bodyToWheels\{v_n\}) \end{aligned} \tag{24}$$

With equations 23 and 24 the state of the robot (the four currents through the motor and the three velocities of the body of the robot) is described and can be predicted using an integration scheme such as the forward Euler method. Furthermore, the friction coefficients of the body can be determined by combining the steady state solutions of equations 23 and 24:

$$\begin{aligned}[D]\{v_n\} =& \frac{gK_m}{Rr}[forceCoupling]\{V_{in}\} - (\frac{g^2K_m}{RrG_m}[forceCoupling] \\ & + \frac{g^2 d_{wheel}}{r})[bodyToWheels]\{v_n\}\end{aligned} \tag{25}$$

These can be solved easily for the elements of matrix $D$ since the input voltage is known and the velocities are measurable states (note that wheels speeds and velocities can be converted forwards and backwards using $[bodyToWheels]$ and $[bodyToWheels]^{+}$) and $D$ is a diagonal matrix.

**Accelerometer filter** The accelerometer on the robots is introduced in the TDP of 2018 [4]. This sensor, of type Xsens MTi-3-8a7g6t, is mounted on the robot and measures, amongst others, accelerations and rotations. It has been primarily used to estimate the yaw of the robot very accurately, making it less reliant on computer and vision data. The yaw measured by the sensor has a standard deviation of 0.0011 $rad/s$ in rest and 0.037 $rad/s$ while driving. The accelerations had a standard deviation of 0.025 $m/s^2$ in rest and 2.5 $m/s^2$ while driving. The accuracy of the accelerations while driving is not enough to be reliable in the state estimation. This is most likely due to vibrations of the robot during movement. The other measurements used for state estimation are the wheel encoders, which give the velocity of the robot transformed from the speed of each individual wheel. However, when a wheel slips, the speed that the encoders measure will be inaccurate. This then also results in an inaccurate estimated velocity. Using the accelerations measured with the accelerometer, this situation will not occur and therefore including this data in the state estimation is beneficial.

The issues with the accelerometer of previous years were the offset of the measurements, low frequency noise (drift) and high frequency noise. The offset was eliminated by reading out the free acceleration instead of the actual acceleration. This eliminates the z-component of the acceleration, the gravity, if the robot is not standing fully horizontal. For the high frequency noise, a low-pass filter was considered, but appeared to be an insufficient solution. When driving, the robot behaves at unpredictable frequencies, introducing the possibility of removing useful data, when eliminating fixed frequencies. Therefore, the high frequency noise is suppressed by post processing the data with a moving average. This kind of filter takes the average of the last few data points on every time step. This results in a smoother signal, making the acceleration pattern clear and usable. The moving average does, however, introduce a delay to the measurements since data of the past is used to calculate the acceleration. As a result, the number of data points used in the average should be chosen optimally. On the one hand, there is a risk of using too many data points, which introduces a larger delay on the signal, makes the computations slower and smoothes out the signal too much. On the other hand, too few data points might not be sufficient to remove the high frequency noise.

By implementing a moving average the variance of the acceleration data was decreased. The standard deviation of the data while driving is reduced from 2.5 $m/s^2$ to a much better preliminary value of 0.4 $m/s^2$. The data can now be used in the Kalman filter more accurately. To verify the performance of the applied moving average, several tests were executed. This was done with robots with the old wheel configuration, driving in straight lines in both x-direction and y-direction, with reference velocities of 0.5 $m/s$ and 1.0 $m/s$. The output of the Kalman filter without moving average is shown in Figure 12, the output of the Kalman filter with moving average is shown in Figure 13.

In order to quantify the performance of both Kalman filters, the average standard deviation of the estimated velocities in driving direction has been calculated. The results can be found in Table 3. It shows that the standard deviation for the Kalman filter with the applied moving average on the IMU data is lower in all cases, which is also visualised in the graphs. This indicates an overall improved performance of the Kalman filter if a moving average is applied.

Drifting of the measurements might be calibrated when it will become a problem.

Table 3: The average standard deviation of the estimated velocity of the robot with and without a moving average applied to velocity output of the IMU.

| **Driving direction** | **x-direction** | **x-direction** | **y-direction** | **y-direction** |
|---|---|---|---|---|
| **Reference velocity [m/s]** | 0.5 | 1.0 | 0.5 | 1.0 |
| **Without moving average** | 0.034 | 0.050 | 0.061 | 0.080 |
| **With moving average** | 0.025 | 0.037 | 0.053 | 0.059 |

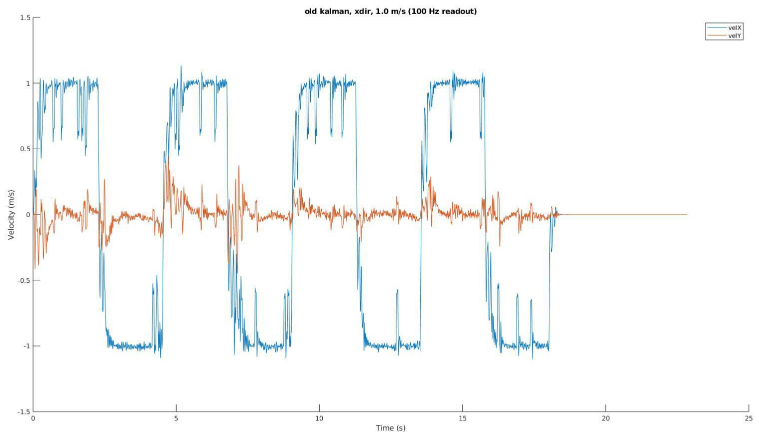

Fig. 12: Estimation of the velocity in x- and y-direction without moving average applied.

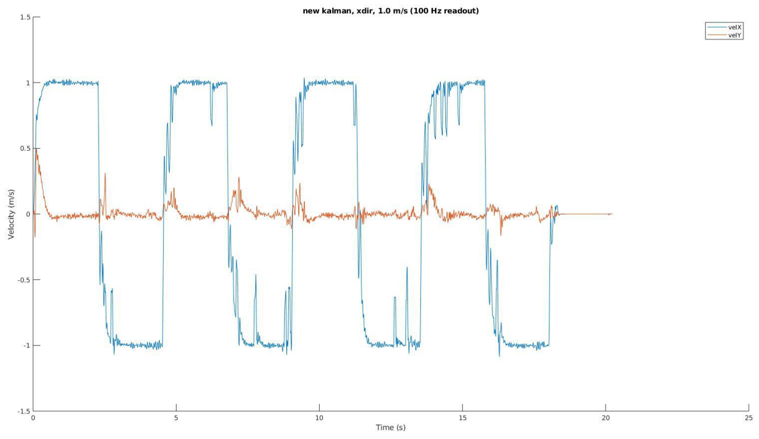

Fig. 13: Estimation of the velocity in x- and y-direction with moving average applied.

# 6 AI

## 6.1 Knowledge Transfer

At RoboTeam Twente, each year the team is filled with an almost completely new team of full-time AI developers. Most of these AI developers are in the later stages of their studies and will most likely not be present for more than 3 years after they finish their fulltime year at the Roboteam. This means they will not always be around to explain their code to new generations, as they will be studying or working. For this reason, a higher emphasis will be put on the maintainability and readability of the system: though it may still be helpful, well written code does not need someone to explain it, it explains itself through documentation and clear design.

One of the priorities this year is redesigning and documenting parts of our system that are unintuitive and unmaintainable. This will make it easier for new software developers to get familiar with the codebase, so they can start contributing as quickly as possible. A major part of this plan is the pipelining of the AI.

## 6.2 Pipelining our AI

The AI code is currently not centralized. Calculations are done in a lot of places, sometimes very low level in the process and sometimes very high level. Additionally, virtually all decisions are made in the Behaviour Tree of a strategy, which makes these trees complex, and makes it very difficult to communicate between robots as the communication is not possible between nodes on the same tree level. This problem was previously solved by introducing global coaches that a tree could ask for advice, but these global variables introduced a lot of maintenance problems as it was not clear when they were being used and where they were being used, making it very difficult to reproduce behaviour. To solve this, a pipeline will be built in our AI so there is a clear, reproducible sequence of decisions, and it is also clear where the decisions are made.

Additionally, the abstraction levels of the strategies are not always intuitive. For example, some skills we have are very low-level and make no decisions themselves, but there are also some skills that encompass a lot of behaviour and decision-making, making them very complex.
Our redesigned system is largely inspired by the skill-tactic-play design from CMDragons[5] and the adapted version from TU Eindhoven [6], with some modifications. We will give a short summary of the program design below:

There will be a set of Plays. The play is a sequence of tactics. In a tactic, multiple robots cooperate to achieve a common goal, for example 2 robots performing a pass and play. Each robot then also has a set of skills, which are low-level behaviours like driving to a location, or shooting.

**Play Checker** Each play has a set of invariants and preconditions. Invariants are conditions that must be true at all times. When a play is in action, and one of its invariant becomes false, the play becomes invalid for use. The Play Checker checks whether each play is feasible for the current situation, and returns all of the plays that are approved.

**Play Decider** These plays are then given to a Play Decider, which looks at all the plays and scores all of them based on how relevant they are to the current game state. In this module plays are evaluated based on the state of the world, but also a more reward based analysis is done here. For example, if the Roboteam is losing 1-0, then in the last 5 minutes of the game we might decide on a more aggressive strategy because we want to score an equalizing goal.

**Play Executor** The Play Executor is concerned with gathering all the necessary information for the play to execute. For example, if we are doing a pass play, then it needs to be decided which robot is the best robot to pass to, and whether it should be a deep, shallow, or normal pass.

Finally, all of this information will be given to the behaviour tree associated with the play and this tree will then be executed. This marks a major difference with the current system: rather than calculating a lot of information in the behaviour trees/skills, all of the strategic information is calculated before the strategy is executed. The specific implementation of which part of the opponent's goal a robot should shoot at is still calculated in the tree. Thus, the behaviour tree acts as the body and the AI as the brain.

The previous system architecture did not support specific decision making modules, rather, it became one large strategy that could handle almost any situation, but the strategy being played was almost always the same. With this architecture, the AI is more actively choosing which strategy they want to use each tick, hopefully resulting in more efficient behaviour as specific plays can be found that best match a certain situation.

Finally, by making this process very pipelined, it is easy to see where you should make adjustments if something goes wrong, because each module has a clear responsibility and has only one responsibility. This will greatly help new developers contribute to the system. It is expected that this new design will make the code easier to understand, modify and maintain, as well as keeping the code modular.

### 6.3 Hybrid team

This year the team will work on building specialized robots that will be responsible for specialized tasks. The process will commence by introducing several robots that have stronger motors, allowing them to accelerate faster. This makes these robots a valuable commodity and they must be allocated efficiently.

**Robot Assignment Problem** With a homogenous set of robots, assigning robots to tasks is dominated by the locations of the robots and how long it takes them to go somewhere. This gives rise to a linear program which is currently solved with the Hungarian algorithm[7].

However, when the team is no longer homogenous, we run into the issue that some robots are better at doing something. We therefore need to look at the cost of a robot going somewhere versus the reward for the robot performing this action, and with certain robots the reward will be higher based on their physical configuration. Additionally, we need to make a distinction about which task is the most important to perform, so we get the best robots performing the most important tasks. This will be simplified by using heuristics.

Currently, the assignment problem has been implemented as follows. Let $A$ be the set of robots that need assignments, and let $B$ be the set of possible roles for the robots. In our assignment problem, all robots are fit to do each task, so the resulting graph becomes a complete bipartite graph. The standard linear assignment problem looks as follows:

$$\min \sum_{i,j \in A,B} x_{i,j} w_{i,j} \tag{26}$$

The cost is currently based on the distance from the robot to its destination. However, it needs to be extended to take into account that there are different kinds of robots in the team.

Each edge in the graph will be evaluated based on the cost of robot i moving to the location role j indicates, and the cost of robot i executing role j multiplied by the proficiency $P$ of the robot at executing this task, and the importance $I$ of the task.

This cost metric will have this form:

$$w_{i,j} = C_{i,j}^{\text{move}} + C_{i,j}^{\text{do}} P_{i,j} * I^2 \tag{27}$$

In order to find a good cost function for this, trial and error can be used. However, the cost function can also be learned.

By creating several scenarios and corresponding correct and incorrect assignments, it is expected the cost functions can be scored. This provides a more formal manner of determining the cost function.

## 6.4 Simulator

Although we were reasonably satisfied with the usage of grSim [8], the team decided after the previous RoboCup that a new simulator was necessary. The reasoning for this was primarily that there were multiple issues with the physics library used by grSim, Open Dynamics Engine (ODE).

Although our design is distinctly different in some places, the internal simulator used by ER-Force [9] and grSim were big inspirations for us. For the team the purpose of a simulator is both to test out new strategies and tactics, but also to do tests.
There are three main reasons for using a new physics library. First and foremost, due to the way the physics engine works, grSim scales quite badly to a larger amount of robots. This is because ODE's accurate solution method solution time scales cubically with the amount of objects in the world. With the SSL's ambition to move to 11 robots, it was noted that on many of our developers' system the desirable framerate of at least 60 Hz was no longer obtainable. There are quick (linear) methods which sacrifice accuracy for speed, also in ODE. However, these suffer from many accuracy problems. Other libraries which are tuned to use these linear methods typically give much better performance.
It was then attempted to integrate the simulator into our existing software architecture, but the team ran into issues with this as grSim is not easily portable as a library. Another issue is that in ODE a lot of post-hoc damping and smoothing is necessary to produce 'physical-movement' due to the system-matrix formulation. In grSim very high parameters for constraint force mixing (CFM) are used to resolve this. There were also problems with accurately simulating the control on our robots, specifically the rotational (yaw) control using the Xsens IMU as detailed in our previous ETDP[4]. This was mostly due to the high forces and small timescales involved in robot control.
The decision fell on Bullet[1] as the library to be used for physics simulation. Bullet is a physics library that is under active development. It supports many useful features, and has an easy to use and Object Oriented C++ API. Below is a description of some features of our new simulator.

- Situations were incorporated, that are a complete and easy to create description of the state of the game. These situations can then be saved as files and loaded into the simulator.

- The simulator is deterministic, so that given the same input, the simulation will return exactly the same results. This allows for rigid testing. The future goal is also to add an network-interface so that it's possible to easily create integration tests for software using these situations.

- Time scale can be adjusted to run faster/slower than real time or the library can be used in a machine learning project manually to simulate as fast as possible.

- The friction parameters conforming to the two-stage ball model as described by the FU-fighters [2] are also implemented. This turned out to be a lot

[1] Bullet physics library

more accurate than the model used by grSim, allowing for more realistic simulation.

- Just like grSim, robot and world settings can easily be loaded from files. Similarly, we also have options for simulating noise and delays.

- One new feature is that we simulate camera settings using parameters used at the RoboCup. This way the offset at the middle line is simulated, as well as the nonlinear trajectories the ball follows when in parabolic flight. Also rough pixel positions can be determined.

- One of the nicer features of Bullet is convex hull support. This makes ball-robot collisions a lot more realistic and prevents the ball ending up inside of the robots, as sometimes happened in grSim.

Note that at the moment of writing this simulator is still under active development. It is publicly available as an open-source project: roboteam_mimir

### 6.5 Replacing ROS with Google Protobuffers

During previous years the Robotic Operating System (ROS) was used as underlying framework for the software. ROS was used for interprocess communication between data processing, decision making and message transmission. Using the ROS parameter server settings were shared between the different processes. By using ROS and the accompanying $\text{ros}_{\text{msgs}}$ library it was easy to monitor values that where sent over the network. On the other hand, understanding the infrastructure of ROS takes some time for new developers. Another disadvantage of ROS was that installing it would take a long time and a lot of disk space. This would cost a considerable amount of compile time as well as a relatively large hit on runtime performance. Another disadvantage of ROS is the fact that it is only supported on Linux operating systems.

Because RoboTeam Twente has an annual cycle of switching team members, new developers have to be trained every year. This means that the steep learning curve of ROS, as well as the installation effort outweighs the advantages that ROS has to bring for RoboTeam Twente. Therefore it was decided to find a replacement.

ROS provided to following utilties:

- Interprocess communication using the $\text{ros}_{\text{msgs}}$ package
- Shared settings using the $\text{ros}_{\text{params}}$ package
- Plotting data packets using PlotJuggler
- Monitoring time in the system using ros::time

**Interprocess communication and global settings** For each of these utilities a replacement needed to be found. A replacement for the interprocess communication was found in Google Protobuffers, which are used by the SSL league as well. A library was created to mimic the messaging library $\text{ros}_{\text{msgs}}$ such that it could be easily replaced with protobuf messages. This uses the publisher-subscriber pattern to provide clear communication between the different applications, and communcation is easily shared over computers using TCP.
As the ROS params were not available anymore, a channel was created for the settings that would use the same publisher subscriber model. This heavily increased the speed of which settings could be applied and is more consistent with the other parts of the system. The roboteam_ai repository is responsible for publishing these settings, as roboteam_ai shows a complete visualization of the system and already deploys a configurable user interface.

**Replacing ros::time** ros::time could be easily replaced with the default std::chrono library. Using some self-written functionality this easily replaces the utilities provided by ROS.

**Plotting data** As PlotJuggler is not usable without ROS a replacement needed to be found. The control team does rely on charts from the data sent to the robots so a good plotting application is rather important. As no proper replacement was found, a new application is created just for this purpose, which we call roboteam_monitor.
roboteam_monitor takes the roboteam_proto library to get the message definitions. In the roboteam_proto library these definitions are given, auto-generated and there is also a specification as to which messages can be found on which TCP channel. Roboteam_monitor can then listen to these TCP channels as well and, using the Google Protobuf Reflection methods, decode the message sent over the channel.
Roboteam_monitor makes the user create a chart with one or more series. Every series can represent an input of data, which can be filtered. A user can therefore create a chart where simultaneously the velocity of robot 0 and 1 is plotted, while it also plots the ball velocity from another channel.
Plot configurations can be stored to a json file, so a user does not need to constantly configure the chart, but can just use preconfigured settings. For the future, the idea is to add more mathematical functions to roboteam_monitor, as well as a dashboard to support multiple chart configurations in one view as well.

## References

[1] Raul Rojas and Alexander Gloye Förster. “Omnidirectional Control”. In: *KI - Künstliche Intelligenz* (2005).

[2] R. Rojas and M. Simon. *Like a rolling ball.* retrieved at http://robocup.mi.fu-berlin.de/buch/rolling.pdf on the 7th of January 2020.

[3] ER-Force. *ER-Force Extended Team Description Paper Robocup 2016.* Tech. rep. 2016.

[4] RoboTeam Twente. *RoboTeam Twente 2018 Team Description Paper.* Tech. rep. 2018.

[5] Brett Browning et al. “STP: Skills, tactics, and plays for multi-robot control in adversarial environments”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 219.1 (2005), pp. 33–52.

[6] Lotte de Koning et al. “Skills, tactics and plays for decentralized multi-robot control in adversarial environments”. In: *Submitted to AMAAS* 172 (2017).

[7] Wikipedia. “The Hungarian Algorithm”. In: (). retrieved at https://en.wikipedia.org/wiki/Hungarian$_algorithm$.

[8] Valiallah Monajjemi, Ali Koochakzadeh, and Saeed Shiry Ghidary. “grSim – RoboCup Small Size Robot Soccer Simulator”. In: *RoboCup 2011: Robot Soccer World Cup XV.* Ed. by Thomas Röfer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 450–460. ISBN: 978-3-642-32060-6.

[9] Michael Bleier et al Florian Bauer Peter Blank. *ER-Force Extended Team Description Paper Robocup 2011.* Tech. rep. 2011.