ER-Force
Team Description Paper for RoboCup 2011

Florian Bauer, Peter Blank, Michael Bleier, Hannes Dohrn, Michael Eischer, Stefan Friedrich, Adrian Hauck, Jan Kallwies, Patrick Kugler, Dominik Lahmann, Philipp Nordhus, Benjamin Reck, Christian Riess

Robotic Activities Erlangen e.V.
Pattern Recognition Lab, Department of Computer Science
University of Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
info@robotics-erlangen.de
http://www.er-force.de/

Abstract. This paper presents an overview description of ER-Force, the RoboCup Small Size League team from Erlangen, Germany. The current hard- and software design of the robots and the motion control system are described. An insight in the software framework and strategy architecture is provided. Furthermore, upcoming changes and improvements are outlined.

Fig. 1. ER-Force robot design from 2010.

1 Introduction

This paper describes the RoboCup Small Size team ER-Force from the University of Erlangen-Nuremberg. The team was founded in fall 2006 by students from various engineering disciplines, such as computer science, mechatronics and electrical engineering. In 2007 we founded a non-profit association called "Robotic Activities Erlangen e.V.". This association is engaged in many robot-related activities including the support of two Robotics groups at local high schools. The team participates at the international competitions of RoboCup since 2009.

The following sections provide an overview of our current Small Size League team. In Section 2 the hardware and firmware architecture of the robots is described. We discuss the hardware of our 2010 system and outline the new developments and improvements. In Section 3 we examine the software architecture and revisit the techniques used for the filtering of the position data provided by SSL-Vision. Moreover, we give some details on the motion control of our robots. The strategy module and decision algorithms implemented in our system are described in Section 4.

2 Hardware

The design of our 2010 robots is shown in Fig. 1. The team consists of six robots that are identical in construction. The chassis consists of laser-milled aluminium plates connected with custom-built milled components. The lower part of the chassis contains the motors with wheels, the kicker unit with capacitor and the dribbler, while the upper part is completely reserved for the electronics and battery. In order to be able to reach the inner part of the robots, e.g. to change the battery, the cover plate is mounted with permanent magnets. The robot design is fully rule compliant and has a maximum diameter of 175 mm and a maximum height of roughly 120 mm. The robot covers less than 20 % of the ball along its z-axis projection at all times.

2.1 Drive

To allow an optimal mobility the ER-Force robots use an omni-directional drive-system. It is similar to other RoboCup Small Size teams, but currently uses only three wheels. The three wheels were custom built to provide optimal grip in the rotation direction and minimal resistance in any other direction. The actual speed of the wheels is monitored using quadrature encoders attached to the motor shafts. This information is used to adjust the motor PWM-signal to achieve the desired wheel speed using a cascaded controller which is running on a microcontroller at a control loop speed of 400 Hz.

This year we are upgrading our driving system to four brushless DC motors (Maxon EC 45 flat) with four omni-directional wheels. By switching to brushless DC motors we hope to benefit from two things. The new motors have more power and a higher efficiency than the old brushed motors, which will result in a higher maximum speed.

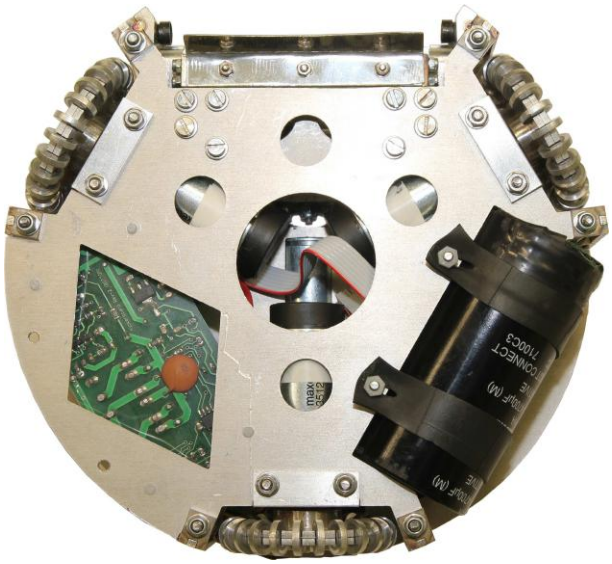

Fig. 2. Drive system.

2.2 Kicker and Dribbler

We currently employ a solenoid kicker, which consists of a high voltage capacitor with a capacity of 4900 µF and a solenoid with a resistance of 1.5 Ω. The capacitor is charged by a step-up charging circuit to a voltage of up to 200 V. To activate the kicker a Power MOS-FET is used to drive the high current and voltage load. The current system is capable of shooting the ball at a speed of up to 8 m/s. A chip-kicking device using the same capacitor but a second solenoid was developed in 2009 and is still in use. We are currently redesigning the the kicker mechanics with a larger solenoid to make the mechanism even more robust and to increase the maximum ball speed.

The dribbler system in our current robots is placed above the kicking device. It consists of a rubber coated bar driven by a small DC motor (Maxon A-max 19). This bar was designed to exert backspin on the ball and keeping it in position. The current dribbler design proved to be insufficient, as it is almost impossible to receive passes with a stationary mounted dribbler. Therefore, we are constructing a passively damped dribbler bar which slows the ball down when it hits the robot. This should facilitate passing the ball at high speed.

2.3 Microcontrollers and communication

The 2010 robot generation is using three microcontrollers. An ARM7 receives commands from the radio module, runs the controller loop, and generates the PWM signal. The encoder signals are evaluated by an ATmega48, which is connected to the ARM7 via an SPI bus. Our solenoid kicker is actuated by a ARM

Cortex-M3 microcontroller located on a different board. In order to provide a clean and consistent interface to the different controllers in use, we wrote a library that encapsulates hardware specific features such as PWM-signal generation or bus communication. In our new electronic design we are using solely ARM Cortex-M3 microcontrollers.

For communication we use different types of radio transceivers, such as the NRF24L01 (2.4 GHz ISM band) or the RFM12 (434 MHz and 868 MHz ISM band). In our 2010 implementation the communication was only one-way from the computer to the robots. This will be improved in our new design by integrating status packages sent from the robots to the strategy computer.

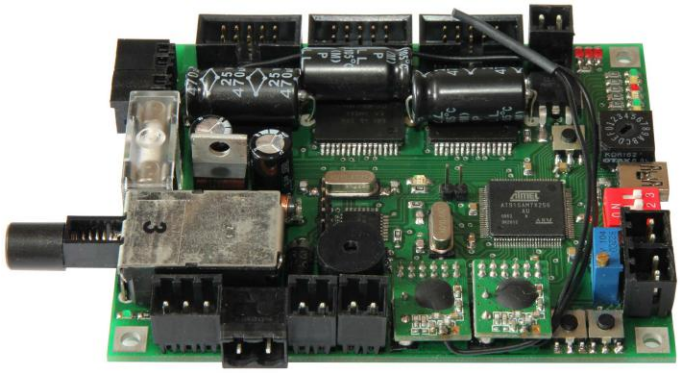

Fig. 3. Main control board.

3 Software Architecture

Our current software framework uses only a single application which takes care of receiving the SSL-Vision data, tracking, strategic decisions, path planning, and radio communications. Additionally it features an integrated simulator, a referee input module to quickly test standard situations, and is able to run two strategies simultaneously allowing us to play test games. It uses the Open Dynamics Engine and supports all features our hardware is capable of including dribbler and chip-kick. The application is written in C++ using the Qt Toolkit and runs on GNU/Linux and Mac OS X.

3.1 Tracking

For our strategy to work properly we need to have valid positions and velocities for each object. SSL-Vision already provides position information for all the robots on the field an the ball. However, the ball might be occluded by a robot due to the perspective projection of the camera or the object detection can fail

because of noise or other processing problems. These problems can be solved by a tracking algorithm, which is able to filter out noise, such as invalid objects, and can track an occluded ball.

The robots are tracked by a multiple target tracker which is using robot stacks, which can contain multiple possible position estimates. At first a stack created from the previous iteration is searched for each new robot candidate. If a robot with the same identification as the new candidate is found and is closer than 20 cm, the candidate is added to this stack, otherwise it is added to a newly created stack. Afterwards each stack with more than one robot is reduced to a single robot by averaging the position over all robots on this stack. The speed is approximated from the difference to the last position. If a stack does not contain any robots, e.g. if the robot is currently not visible, the position of the robot is estimated as $p_i = p_{i-1} + v_{i-1} \times \Delta t$. Here p_{i-1} is the position in the previous time-step and v_{i-1} is the last known speed, which is assumed to be constant. If no robot was assigned to a stack for more than one second, it is no longer tracked as the robot has most likely been removed from the field.

Tracking the ball is a more difficult task, as it moves up to 20 cm between two frames and can be occluded by a robot due to the perspective projection of the camera. Since a simple Kalman Filter can not model the non-linear dynamics introduced by ball occlusions, we decided to use a Particle Filter [1] to track the ball. Particle Filters or Sequential Monte Carlo Methods (SMC) are a different approach on tracking. The idea is to get a number of randomly distributed samples and to calculate a weight for each of them. Then the samples are updated, i.e. they are moved according to the system dynamics. The common Particle Filter, also known as Sampling Importance Resampling (SIR), Sequential Monte Carlo Filter, Bootstrap Filter, or Condensation Filter, is a direct implementation of Bayesian Filtering.

In detail, we implemented a Sampling Importance Resampling filter which uses a model with different state transition functions depending on the situation. If the ball is visible and not in the neighborhood of a robot, a simple linear state transition model is used. If the ball is close to a robot, modelling the motion is more complex, as the ball may occluded and dribbled or shot by the robot. If it is occluded and the robot is not moving, no measurements are available and the model is entirely probabilistic and influenced only by the noise model. However, if the robot is moving, the ball is assumed to be still near the same robot and the ball position estimate is updated using the motion of the robot. As the robot may shoot the ball, an instant acceleration must also be included into the model. The only assumption that can be made in this case is that the ball can only be shot away from the robot in a single direction. However, the current implementation does not perform any reasoning on which side of a robot the ball is positioned and does not use this information to improve the estimate. But we are still planning to integrate this and other extension into our state transitioning model until the competition.

3.2 Strategy

The strategy itself is implemented in the scripting language Lua [2]. The script gets all available information such as positions and current referee command and generates appropriate commands for each robot. To allow for easier testing the script is automatically reloaded when the files have changed resulting in an instant behavior change upon saving of the script.

3.3 Motion control

To allow smooth motion of the robots the positions generated by the strategy need to be processed by a motion controller. The position controller (see Fig. 4) gets the current position of the robot p_{measured} from the vision system, compares it to the desired position p_{setpoint} and calculates the postion error e_{position}:

$$e_{\text{position}} = p_{\text{setpoint}} - p_{\text{measured}} \quad . \tag{1}$$

This is used to calculates a setpoint for the velocity of each robot:

$$v_{\text{setpoint}} = \begin{bmatrix} v_{x,\text{setpoint}} \\ v_{y,\text{setpoint}} \\ \omega_{\text{setpoint}} \end{bmatrix} = K_p e_{\text{position}}(t) + K_i \int_0^t e_{\text{position}}(\tau)\, d\tau + K_d \frac{d}{dt} e_{\text{position}}(t) \quad . \tag{2}$$

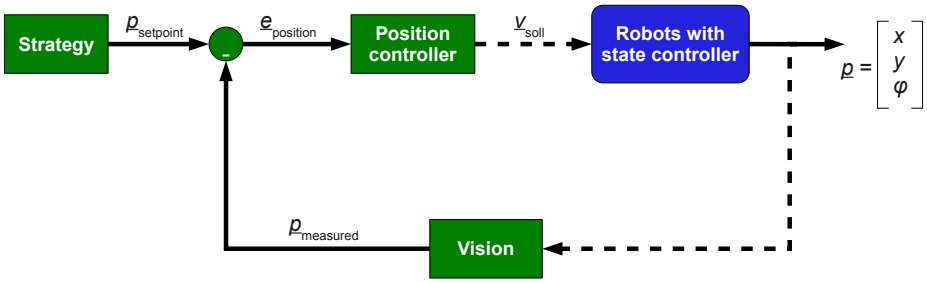

Fig. 4. Complete controller structure.

Each velocity setpoint is then transmitted to the according robot. The velocity needs to be controlled by the control software running on the microcontrollers of robots.

The simplest way to control an omni-directional drive-system is to control each wheel individually with a simple PID controller. Since 2010 our new approach for the motion control is to use a state controller, which takes the rotational speed of all wheels into account. It then controls the velocity of the robot as outlined in Fig. 6. The three state variables are the velocities of the

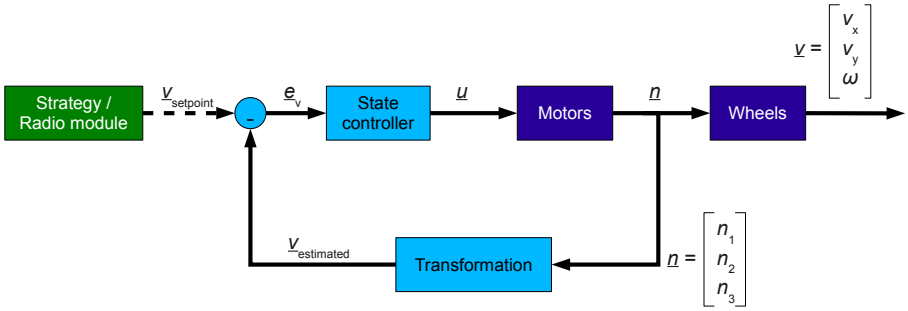

Fig. 5. The state controller implemented on the robots.

robot in x and y direction and the angular speed. The used coordinate system is shown in Fig. 6. The actual controller implemented in our system is a PI state controller. Currently we cannot measure the velocity of the robot directly but we measure the rotational speed of each wheel and transform it into velocities. The problem with this approach is that the wheels can slip and the velocity is measured wrongly. A new idea to improve the velocity estimate is to use laser motion sensors which are capable to measure the velocity of the robot directly.

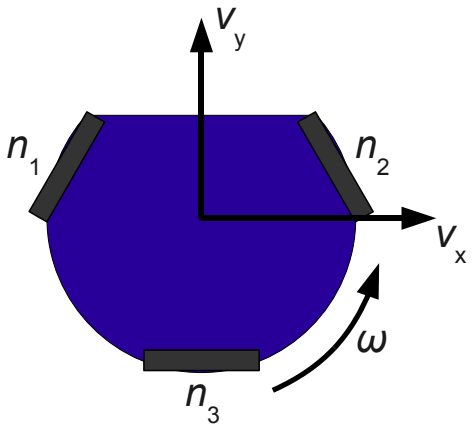

Fig. 6. Velocities used for motion control.

3.4 Radio communication

We are using a two way communication to our robots. During each iteration our strategy module (described in chapter 4) updates the destination positions for the robots. The relative movement speed (in robot-local coordinates) is calculated, and sent via USB to a radio sender module. To simplify the communication

we have implemented a small C library which can be used in the C++ strategy application as well as on the radio transceiver itself. It focuses on three aspects:

- Same source code on strategy computer and robots
- Small packet sizes to keep latency low
- Forward compatibility to be able to use robots without reprogramming them after a protocol extension

The feedback channel is used to report battery level, light barrier state, and other system information from the robots to the control computer for visualization.

4 Strategy Module

The strategy module uses the output from the tracking algorithm to create an abstract representation of the world. Based on the current position of the robots and the ball and the actions taken in the previous frames, the strategy then produces motion and action vectors (dribbling, kicking or chipping) for each robot.

4.1 Overview

The artificial intelligence in the RoboCup Small Size team ER-Force runs after the tracking and before the motion control system. Our approach in the 2011 system consists of three main components. First, an *observer* collects game statistics that are used by other components.

Second, *roles* influence the behavior of a robot. Roles are dynamically assigned, depending on the game situation. The transition between roles depends on the game situation, and is modelled using finite state machines (FSMs). These FSMs feature a cascade of smaller state machines with partially randomized transitions. A special case are referee decisions. Referee decisions change the state immediately to so-called referee states. Such states handle the particular referee situation, for instance to keep the minimum distance to the center spot. Finally, *skills* model a single, separate action. Skills of different complexity are used to perform particular roles.

We describe these components bottom-up, starting with the skills of the robots.

4.2 Skills

Skills are implemented as elementary functions of varying complexity. Particular skills can be used within other skills. A skill can be a very straightforward task, such as *move to point* or *grab ball*. Skills can also be used to model more complex medium level and high level functions like *pass-and-shoot*.

The implementation of higher level skills can become rather complex. It may use sub-skills, and require additional reasoning. For instance, it may require

objective evaluation functions and geometric considerations, similar to the ones described in [3]. To perform these tasks, skills use information from the observer (described below).

4.3 Roles

Each role encapsulates a single robot behavior. The execution of a role is typically implemented as a FSM consisting of single skills. Available roles are

- Goal defenders: robots that directly protect the goal.
- Field defenders: robots that try to intercept passes.
- Ball grabber: robot that runs for the ball.
- Attacker: robot that shoots at the goal (when in ball possession).
- Assistant: robot that breaks clear, i.e. it aims to get in a position that is not covered by defending opponents.
- Pass player: robot that plays a pass (when in ball possession).
- Runner: robot that assists offensive moves, e.g. to receive a passed ball.

4.4 Role Assignment

A role distribution constitutes a mapping of roles to robots. For instance, one of the robots obtains the *goal keeper* role, while three others obtain defending roles. One such distribution is referred to as tactic. A team is assumed to follow either an offensive or defensive tactic. Depending on game statistics from the observer, an offensive or defensive tactic is chosen.

A tactic contains a set of roles for the team, e.g. how many robots should stay on defense. Furthermore, it states whether the robot that controls the ball should dribble, play a pass or shoot at the goal. Thus, the current tactic completely describes the immediate next actions of the team.

Within a tactic, the roles are greedily distributed among the robots, according to their current positions. The offensive player aims to obtain the ball, and to pass and shoot at the opponents goal. The goal defender aims to cover the own goal.

The selection of a tactic is based on the game state. For instance, within an offensive situation a robot might be able to safely play a pass next to the opponent's goal. This tactic is typically preferred over a direct shot on the goal. A tactic can at any time abort itself if it assumes that it can not successfully finish the objective.

Subject to recent development was the refinement of roles that involve multiple players. This is most relevant during pass play. The current implementation considers pass play as an outcome of "reasonable moves". A more advanced approach is to consider pass play as a part of the solution itself. Thus, we are currently modelling joint roles that involve multiple robots. One example for a joint role is "get next to the goal", which implies for instance one ball grabber, two assistants and a planning component in order to raise the success probability to pass to one of the assistants.

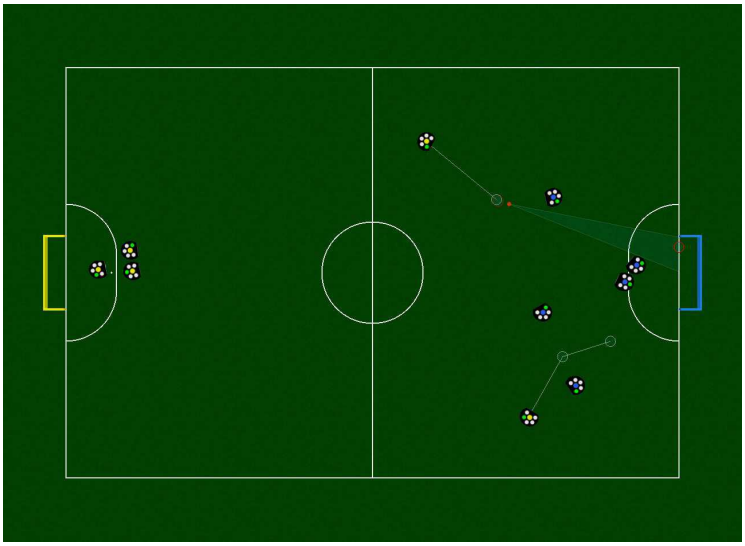

Fig. 7. Visualization of the reasoning for determining a tactic. One option is to shoot directly. Meanwhile, the second robot changes to the assistant role. If it succeeds in obtaining a better shooting position, the robot at the ball will not shoot himself, but pass.

4.5 Observer

Parallel to the role assignment and execution, an observer analyzes the match by gathering game play statistics. These include general referee information, ball specific information and player specific information:

- Referee information
  - Number of goals per team
  - Number of direct/indirect free kicks per team
  - Number of penalty kicks per team
- Ball specific information
  - Ball's average position and variance
  - Ball's average speed
- Player specific information
  - Each player's average position and its variance
  - Average position of all robots of a team
  - Number of shots
  - Number of shots towards a goal
  - Number of successful ball interceptions and ball passes

For this purpose, it is necessary to interpret the opponent players' behavior. Therefore, the observer recognizes events such as *kicking the ball*, *grabbing the ball*, *passing the ball* or *intercepting the ball*. For all of these events some conditions are specified to recognize them. For example the following conditions have to be fulfilled for the event *kicking the ball (player P)* in a time interval $[t-1, t]$:

- The ball has been in a specified orbital region around P (the radius is depending on the ball's velocity).
- The ball's moving direction has been changed significantly.
- The ball's speed has been increased.

4.6 Enhancements

The observer provides statistical information on the game flow. Using this information in medium level functions and high level functions (for defining the robots' behavior) leads to remarkable improvements. The analysis of the game refers to both the opponents activity and the overall situation. The role assignment and skills can take these aspects into account for planning tactical moves of the own robots. Besides this, the system is able to react to damages of our own robots, e.g. by swapping the roles of a robot having a broken chip-kick device with another robot.

After improving the skills we are currently working on an agent-oriented system in order to make planning more flexible and decentralized.

5 Conclusion

The current hardware design we presented in this paper is very similar to the one we used last year at RoboCup 2010, as most components proved to be very reliable. However, we are also planning to improve some parts as the kicker and the drive and we want to test a lot of new concepts like a two-way communication link. Major improvements, however, are present in the software of our system.

The strategy software was greatly improved, as we are integrating roles that involve multiple players. This was done to allow better robot interaction. Moreover, we did a lot of work on the role assignment algorithms. By analyzing the robots behavior and collecting statistical game data it is possible to react in a much more precise way to the opponent. We also implemented an improved tracking algorithm for robots and ball positions based on Particle filtering.

As we presented a lot of improvements we are eager to test them at RoboCup 2011 in Istanbul.

References

1. Arulampalam, M.S., Maskell, S., Gordon, N.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on Signal Processing **50** (2002) 174–188
2. Ierusalimschy, R.: Programming in Lua. Lua.org (2006)
3. James Bruce, Stefan Zickler, M.L., Veloso, M.: CMDragons: Dynamic Passing and Strategy on a Champion Robot Soccer Team. In: 2008 IEEE International Conference on Robotics and Automation. (2008) 4074–4079
4. Brooks, R.A.: How to Build Complete Creatures Rather than Isolated Cognitive Simulators. In VanLehn, K., ed.: Architectures for Intelligence. Lawrence Erlbaum Associates Inc. (1992) 225–239
5. Mitchell, T.: Machine Learning. McGraw-Hill (1997)