Berlin United - FUmanoids Team Description Paper for RoboCup 2015

Daniel Seifert, Lutz Freitag, Jan Draegert, Simon Gene Gottlieb, Roman Schulte-Sasse, Gregor Barth, Malte Detlefsen, Jan Bernoth, Niklas Rughöft, Michael Pluhatsch, Martin Wichner, and Raúl Rojas

Institut für Informatik, AG Intelligente Systeme und Robotik,
Freie Universität Berlin, Arnimallee 7, 14195 Berlin, Germany
http://www.fumanoids.de

Abstract. This Team Description Paper describes the humanoid robot team *Berlin United - FUmanoids* and presents robots for participation in RoboCup 2015. A general overview of the team and its history will be given as well as insight into research interests and particular areas of the robots' software and hardware.

1 Introduction

Berlin United - FUmanoids is a humanoid robot team participating in the Humanoid KidSize League at RoboCup. The team was founded in 2006 as the successor of the Mid- and SmallSize team *FU-Fighters*.

During the time of participation at RoboCup, the team had significant successes in competitions, achieving 2nd place in 2009 and 2010, 3rd place in 2007 (its debut competition), 4th place in 2011 and reaching the quarter finals in 2008, 2012 and 2013. In 2014 the team scored 1st place in both the RoboCup Iran Open and the RoboCup German Open competitions.

This paper presents the team's research interests, contributions to the RoboCup community as well as the hardware and software of the FUmanoid robots.

2 Research and Contribution

The main *research interests* are:

- dezentralized *control architecture* for humanoid robots
- *image processing* to reliably classify YUV-triples as logic colors even when light conditions change
- *vision* and *detection* algorithms handling the real-time requirements of soccer games
- *localization* algorithms calculating the robot's position along all 6 degrees of freedom

– *path planning* to avoid obstacles and approach the ball quickly
– fast and stable *walking*

The team is committed to further the Humanoid League and the research exchange between teams. For this reason the team is one of few Humanoid League teams that releases their source code and schematics for hardware components[1]. The software release ([3]) consists of *FUmanoid*, the main program running on the robot, and *FUremote*, the control and debugging program based on Eclipse/RCP. Additionally, the developed framework is also released ([2]). The custom built hardware is part of the release [4] as well as the software running on the microcontrollers. Theses and papers on the robots are available on our website[2].

3 Hardware

3.1 Mechanical Structure

The current robot model was designed and constructed with respect to simplicity and both human-like proportions and capabilities. It features a parallel kinematic leg design, with an additional servo motor added to allow the torso to move laterally, imitating the human spine movement that keeps the torso upright. The total height is 65 cm.

For actuation, we use Dynamixel servo motors from Robotis Inc., namely RX-28 and RX-64 servos. They provide 20 degrees of freedom — 5 per leg, 2 for upper-body movement, 3 per arm and 2 in the head (figure 1). To remove jitter caused by worn-out potentiometers, we modified the servos to use a hall sensor (magnetic encoder) for reliable measurements of the current joint angle.

3.2 Sensors

We equipped the robot with the following sensors:

Actuators: The feedback of the actuators includes the current joint angle, motor speed and load.

IMU: The sensor board includes an integrated 9 axis IMU, featuring gyros, accelerometers and magnetometers. A Kalman filter provides filtered output that is used by the robot for stabilization as well as calculation of the camera perspective in order to obtain localization data.

Camera: The robot is equipped with a commercially available webcam (Logitech HD Pro Webcam C910). Using a resolution of 640x480 (VGA) it delivers 30 frames per second.

[1] http://www.fumanoids.de/opensource
[2] http://www.fumanoids.de/publications

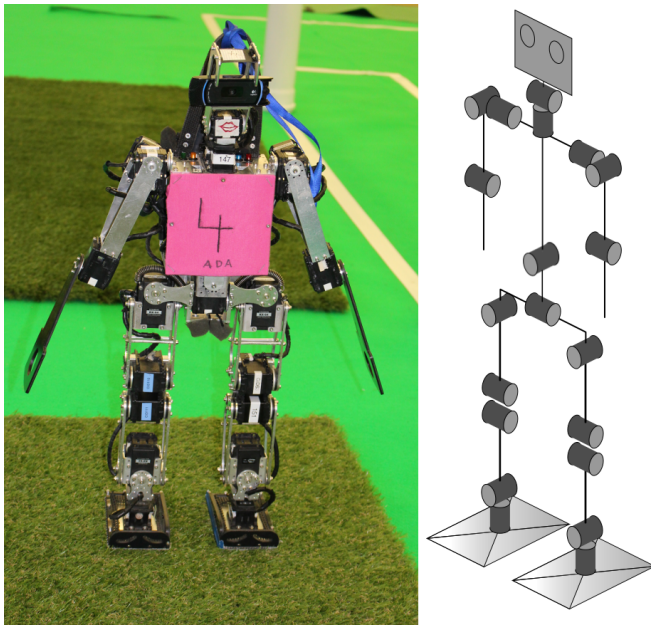


Fig. 1: Current model (left) and its kinematics (right)

3.3 Main Computing Unit

In order to satisfy increased performance requirements, we are using an ODROID-X2 board. This board features an Exynos4412 Quad-core ARM Cortex-A9 CPU clocked at 1.7 GHz. It provides all necessary extension interfaces, such as multiple USB ports (for the camera and the WiFi module), Ethernet and an UART connection.

We utilize Linux as a operating system, based on a custom-compiled kernel and the Linaro distribution.

3.4 Sensorboard

In order to improve communication with actuators and sensors, we developed our own sensor board. It supports connecting the actuators of each leg and the upper body separately in order to set and get servo positions in parallel. Two ARM Cortex M4 processors, clocked at 168 MHz each, handle both the Kalman filter, which is used by the IMU, and the efficient servo communication. Data and actions, e.g. movements of the robot, that are triggered via a dedicated serial connection, can be requested by the main unit.

3.4.1 Hardware Features To prevent the servo motors from damage we installed several safety mechanisms on the robots. A typical condition resulting

in damaged electronics was the deisolation of the servo wires. This condition led to an overvoltage on the data lines towards and away from the servos destroying the ICs connected to the data wires. In order to prevent damage a TVS-diode is attached to each bus to drain overcurrent to ground in case of an overvoltage. Furthermore a comparator is attached to each data wire acting as an off-switch when the voltage climbs above a threshold.

4 Software

4.1 Inverse Kinematics

Robots are actuated with servomotors which are either configured with target angles and velocities or torque. Setting the desired parameters in a "raw" fashion is very cumbersome when dynamic movements shall be performed. Inverse kinematics creates a different way of describing motions [1]: Instead of manually calculating the target angles for a given pose we create *tasks* which represent some target configuration. An example: We want the robot's right hand - this is the *task*'s endeffector - to move to a specific point in the torso's reference coordinate frame - this is the *task*'s base coordinate frame. The inverse kinematics calculates the angle changes in the motors necessary to accomplish the *task*. A *task* calculates the following:

Jacobian A matrix which represents the movement of the endeffector for infinitesimal angle changes from the perspective of the *task*'s base coordinate frame.

error vector A vector in the *task*'s base coordinate frame representing the current error. This is usually how much and in which direction the endeffector has to change.

The inverse kinematics implements the ability to formulate multiple levels of *tasks*. This is useful when some *tasks* are more important than others and the less important *tasks* should not change the error of the more important ones. Those *tasks* are solved in the remaining degrees of freedom (redundant servos) of the more important *tasks*. Furthermore it is possible to define multiple *tasks* for the same level of importance including a weighting for each *task*.

4.2 Locomotion and Stability Control

The walking system of the *FUmanoid* team is based on dynamic trajectories that are designed to realize *Zero Moment Point* (ZMP) optimization. These motion patterns allow the robots to walk in every direction with speeds up to 25 cm/s. The walker is implemented with usage of the *inverse kinematics* solver shown in 4.1.

Walking is mostly defined from the perspective of the feet and involves the whole body dynamics. The tasks (in terms of inverse kinematics tasks) involve the movement of the feet with respect to each other and the movement of the

center of mass (COM) with respect to the supporting foot. The target positions of the COM-tasks are based on an inverted pendulum model of the whole body. The lateral velocity of the COM is matched to either keep the linear momentum inside the support polygon (first step phase, swing foot is behind support foot) or to accelerate towards the target position of the swing foot. As long as the tasks can be fulfilled this suffices the criteria of a ZMP based walker. Due to the utilization of the whole body dynamics even bodyparts which are not intuitively associated with the process of walking (e. g. arms) are used to manipulate the COM and its dynamics.

The walker is designed to be usable on other robot platforms as well since the static properties (e. g. where is the COM in the idle position; what is the geometry of the robot) are once calculated with the forward kinematics and used as parameters for trajectory generation. This leads to a highly reusable walking process which is highly customizable as well.

Though the actual (joint-)movements are not defined within the walker the gait is very similar to natural bipedal beings.

4.3 Computer Vision

The vision module consists of independent layers:

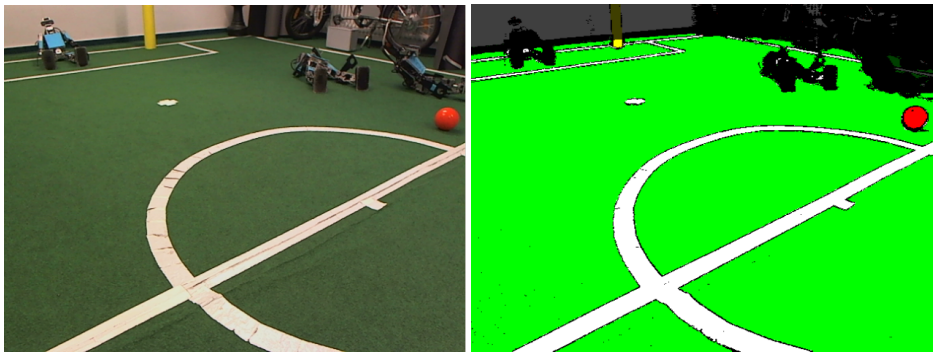
Color Analysis The color analysis is done on a high dimensional cluster classification base. Each cluster is built around sample pixels which represent a *logical color* (green/field, white/goal, white/field lines, ...). A further description on the algorithm is given in 4.3.1

Field Contour Analysis The field contour divides the image in two parts where the part above the field contour contains only information about objects outside the soccer pitch and can therefore be discarded. The other part contains the visible areas of the pitch and is used for further image processing steps. The calculation of the field contour is based on vertical scan lines of green classified pixels.

Object Extraction In the object extraction step all the relevant objects on the pitch are constructed out of the features. The currently provided objects are *ball*, *goals*, *field line points* and *obstacles*. The positions of goal poles in the image are extracted by sampling along the field contour. For extracting the ball, field line points and obstacles a scan grid is used. The grid's density is dynamically generated for each frame based on the expected size of the objects. The scan grid will search for possible location of objects based on the game color. Given a positive game color evaluation a shape based check is performed. This approach in combination with the color analysis allows to detect multiple-colored balls.

4.3.1 Color Classfication Every *logical color* is modeled as a *Gaussian Mixture Model* (GMM), meaning a weighted sum of Gaussian (Normal) distributions. This method has been known to describe the typical shades and variations of the colors quite well. The model also accounts for the case of multicolored items like

the new ball, since it will just fit separate Gaussian components to the different colors. The training of the parameters is done right before the game so the individual lighting and location conditions are considered. The samples needed for the training are created by identifying the objects to be classified in the camera image. To fit the model to the samples the *Expectation-Maximization algorithm* is used, which can be thought of as a *kMeans* clustering algorithm with variable covariance matrices. The exact number of Gaussian components is calculated automatically and individually for each *logical color* according to its complexity.



(a) Typical camera output

(b) The classificator differentiates between the logical colors *ball*, *field*, *field line*, *goal* and *unknown*

Fig. 2: Raw camera picture and the associated color classified image

4.3.2 Object Extractors As described in 4.3.1 there are *logical colors* encoding objects in the game environment. The robot transforms each raw (YUV) camera image to a color classified image whose pixels encode the *logical colors* of each raw pixel. Thereby each pixel can represent multiple *logical colors* at the same time using bit masks. Further we calculate an integral image on the color classified image. The integral image's pixels contain an integer for each *logical color* with the amount of pixels of this color in the rectangle from the top left to the pixel's coordinates.

$$p_{integral}(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} p_{color}(i, j)$$

With $p_{integral}(x, y)$ as the pixel at x, y in the integral image and $p_{color}(i, j)$ as the pixel at i, j in the color classified image.

With the integral image we can efficiently search for bounding boxes of blobs

with a specific *logical color*. The search (*shrink operation*) is performed as a binary search for each parameter of the boundary box (left bound, right bound, top bound, bottom bound). Subsequently, the color coverage of the box is calculated. If it undercuts a threshold the *shrink operation* is repeated in each of the four quadrants of the shrunk bounding box. This is implemented recursively.

This procedure is used to detect the ball, goals, field lines and obstacles. Since this year's rule changes, the ball has mostly the same color as the field lines. We take this into account by testing each ball candidate — each bounding box containig white — for specific "*ball-like*" features we have trained. In advance of each match the ball is shown to the robot. It detects the area in which the ball is located and creates both a feature descriptor of certain areas of the ball and a matrix which transforms the feature vector to a rotation invariant feature space. During the game we calculate the feature descriptor for each ball candidate and match it against the trained feature vector.

4.4 Modelling

4.4.1 Object Modelling In order to track the ball, obstacles and goals, we employ extended Kalman filters. The ball model involves the position and the velocity of the ball. The goals are modelled by four independent poles. This approach allows us to distinguish between the opposite and the own goal. Obstacles are modelled through dynamic amount of Kalman filters. They also keep track of the team colors.

4.4.2 Self Localization The current self localization is done by using an extended Kalman Filter which estimates 6-DOF. This approach allows us to track the position of the robot.

The extended Kalman Filter uses a series of measurements observed over time to produce estimates of unknown variables. It uses a two-step process. The prediction step predicts the current state and its uncertainties. This step alternates with the update step. This step corrects the current state with incoming measurements. Because this filter works recursively, it can run by using only the present input measurements and the previously calculated state. This allows real time application.

In our implementation we capture 6-DOF containing rotation and translation to estimate the position of the camera. The rotation is represented as a quaternion whereas the translation is represented as a vector. In our prediction step we use a gyroscope, kinematics and the odometry to estimate the new position of our camera. As measurements we use different visual features extracted from the camera image, as well as the accelerometer and the computed robot height provided by the kinematics.

Since this approach only allows us to track the position, we use a different strategy to guess the initial position. For this we use a list with possible starting points. We compare observed features in our images to expected features. Out

of this we compute a confidence value. At the end of our initialization process we pick the highest confidence value as our starting position.

4.5 Path Planning

In order to navigate on the field, avoid obstacles and approach the ball properly the robot needs to plan paths. In the previous years our robot's navigation relied entirely on potential fields [5]. The advantage of potential fields is the ease of describing the environment with respect to its traversability. Yet potential fields perform poorly if the robot is at a local optimum, e. g. all influencing elements of the potential field cancel each other out at a given position.

Rapidly Exploring Random Trees (RRTs) are robust against local optima but perform poorly with regards to execution time. We combine the advantages of RRTs with the advantages of potential fields to create a robust path planner with great computational complexity. For each cycle of execution we let the RRT explore either up to a maximum amount of nodes or a maximum amount of exploration steps. This caps the size of the tree. After exploration, we optimize the tree's nodes along the potential field. During optimization the distances of a node to its child nodes are capped to a maximum distance to keep the structure of the RRT. If the distance between two nodes shrinks below a certain threshold, the nodes' subtrees are merged into the node closer to the RRT's root. This thins out the amount of nodes of the RRT which can explore again in the following execution cycle.

5 Conclusion

With the outlined improvements to the software of the robots we are looking forward to participate in the RoboCup 2015 competition.

References

1. Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.
2. FUmanoids. Berlin United Framework 2014, 2014. Available online at `http://www.fumanoids.de/code/framework`.
3. FUmanoids. FUmanoids Code Release 2013, 2014. Available online at `http://www.fumanoids.de/code/coderelease`.
4. FUmanoids. FUmanoids Hardware Release 2014, 2014. Available online at `http://www.fumanoids.de/code/hardware`.
5. Stuart Russell, Peter Norvig, and Artificial Intelligence. Artifical intelligence a modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.