An Open Source Vision Pipeline Approach for RoboCup Humanoid Soccer

Niklas Fiedler, Hendrik Brandt, Jan Gutsche,
Florian Vahl, Jonas Hagge and Marc Bestmann

Hamburg Bit-Bots, Department of Informatics, University of Hamburg,
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany
{5fiedler, 8hbrandt, 7gutsche, 7vahl, 5hagge, bestmann}
@informatik.uni-hamburg.de
http://robocup.informatik.uni-hamburg.de

Abstract. We are proposing an Open Source ROS vision pipeline for the RoboCup Soccer context. It is written in Python and offers sufficient precision while running with an adequate frame rate on the hardware of kid-sized humanoid robots to allow a fluent course of the game. Fully Convolutional Neural Networks (FCNNs) are used to detect balls while conventional methods are applied to detect robots, obstacles, goalposts, the field boundary, and field markings. The system is evaluated using an integrated evaluator and debug framework. Due to the usage of standardized ROS messages, it can be easily integrated into other teams' code bases.

Keywords: RoboCup; Open Source; Computer Vision

1 Introduction

In RoboCup Humanoid Soccer, a reliable object recognition is the foundation of successful gameplay. To keep up with the rule changes and competing teams, continuous development and improvement of the detection approaches is necessary [2]. The vision pipeline we used previously was neither easily adaptable to these changes nor to our new middleware, the Robot Operating System (ROS) [15]. Additionally, it was hard for new team members to contribute.

Thus, we developed a completely new vision pipeline using Python. In the development, we focused on a general approach with high usability and adaptability. Furthermore, our team offers courses for students as part of their studies, in which they are able to work with our existing code base. Therefore, the code has to be optimized for collaboration within the team. It needs to be easily understandable, especially because the team members change regularly while the code base has to be actively maintained. In addition to that, it allows new participants to work productively with a short training period and thus providing a sense of achievement.

Fulfilling the requirements of our domain is made possible by versatile modules (see 3.1), which are designed to solve specific tasks (e.g. color or object

detection), but are also applicable in more generalized use cases. Since we want to promote further development and encourage collaboration, we publish our code under an Open-Source license. Due to this and the usage of the ROS middleware as well as standardized messages [5], the vision pipeline can be integrated into code bases used by other teams.

2   Related Work

RoboCup Soccer encourages every participating team to continuously enhance their systems by adapting the laws of the game [2]. Thus, the number of teams using neural networks as a classifier for batches of ball candidates and even the amount of those employing Fully Convolutional Neural Networks (FCNNs) in their vision pipeline is increasing. This section is based on the team description papers submitted by all qualified teams for the RoboCup Humanoid Soccer League[1].

The most frequently used methods for object detection are single shot detectors (SSDs), based on existing models such as You Only Look Once (YOLO) [16]. In addition to the Hamburg Bit-Bots [4] (our team), the Bold Hearts [18] and Sweaty [7] are using fully convolutional neural networks (FCNNs) in their vision pipeline. With their FCNN [6], the Bold Hearts are detecting the ball and goalposts. Team Sweaty is additionally able to detect several field markings and robots with their approach [19]. In the Humanoid KidSize League, the processing hardware is restricted due to the size and weight limits. While the basic structure of a Convolutional Neural Network for classification is similar between teams (e.g. [1]), the candidate acquisition varies. The EagleBots.MX team uses a cascade classifier with Haar-like features [12]. Rhoban detects regions of interest by using information about the robot's state and "a kernel convolution on an Integral Image filter" [1]. The team Electric Sheep uses a color based candidate detection and relies solely on "number of pixels, ratio of the pixels in the candidate area and the size of the candidate area" [3] to classify them. In contrast to this, [11] proposes a method of candidate acquisition and classification for the Standard Platform League (SPL) based on high-contrast regions in grayscale images without the need of color.

The field boundary is usually detected by computing the convex hull around the largest accumulation of green pixels detected by a field color mask. The team MRL is working on a neural network based approach for field boundary detection [13]. For localization purposes, some teams detect field markings by color [4] while others are relying on the detection of landmarks, such as line crossings or the penalty marks [19].

---

[1] `https://www.robocuphumanoid.org/hl-2019/teams/` last accessed: 2019-06-14

3   Overview and Features

The vision pipeline is implemented as an adaption of the *pipe-and-filter* pattern [14]. Every module implements one or multiple filters. The data flow is defined in the main vision module. To accommodate the nonlinear data flow and to optimize the runtime performance, the output of filters is stored in the modules. This includes intermediate results of related tasks (e.g. when red robots are detected, general obstacle candidates are stored in the module, too). The stored measurements are invalidated as soon as a new image is received. Using launch scripts, the whole vision pipeline, as well as provided tools, can be started with a single command in a terminal. Additional parameters allow launching the vision pipeline with additional debug output, without the FCNN, or to be used in a simulation environment (rosbags or the Gazebo simulator).
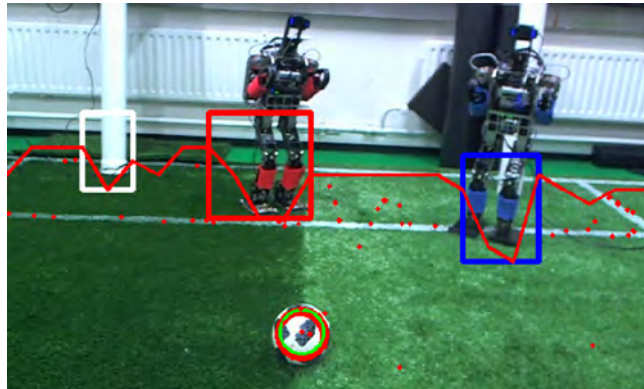

Fig. 1: Exemplary debug image. The detected field boundary is marked by a **red line**, field markings are represented as a set of **red dots**. Around detected obstacles, boxes are drawn (**red**: red robot, **blue**: blue robot, **white**: white obstacles i.e. goalposts). The best rated ball candidate and discarded candidates are indicated by a **green circle** or **red circles** respectively. [4]

3.1   Modules

The following subsections present modules currently used in our vision pipeline by giving a general overview of their function. Some modules contain multiple implementations of the same task as more efficient ones were developed and required evaluation. Modules which are no longer in use because they were replaced (e.g. a Hough-circle ball candidate detection and a classifier) are left out of this paper but kept in the repository.

Color Detector As many of the following modules rely on the color classification of pixels to generate their output, the color detector module matches their color to a given color space. These color spaces are configured for the colors of
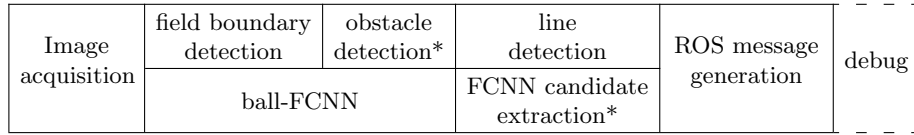
Image acquisition	field boundary detection	obstacle detection*	line detection	ROS message generation	debug
	ball-FCNN		FCNN candidate extraction*		

Fig. 2: Schematic representation of the vision pipeline. After the image acquisition, the pipeline is split into two threads. One handles conventional detection methods and the other handles the FCNN detecting the ball. Afterward, ROS messages and debug output are generated. The debug step is optional. Modules marked with an * implement the *Candidate Finder* class.

the field or objects like goalposts and team markers. Two types of color space definitions are implemented: Either it is defined by minimum and maximum values of all three HSV channels or by predefined lookup tables (provided as a Pickle file or YAML, a data-serialization language). The HSV color model is used for the representation of white and the robot marker colors, red and blue. The values of the HSV channels can be easily adjusted by a human before a competition to match the white of the lines and goal or the team colors of the enemy team respectively. This is necessary as teams may have different tones of red or blue as their marker color. On the other hand, the color of the field is provided as a YAML file to include more various and nuanced tones of green. These YAML files can be generated using a tool by selecting the field in a video stream (in the form of a live image or a ROS bag). Additional tools visualize included colors or increase the density of color spaces by interpolation between points in the RGB space.

The latest change of rules [17] also allowed natural lighting conditions. Therefore, pixels in the image mainly surrounded by pixels whose color is already in the color space are added.

However, only colors without any occurrences above the field boundary are chosen to ensure a stable adaptation to changes in shade and lighting. In Figure 3, the images in the left column show an erroneous detection of the field boundary while the dynamic color space module has been deactivated. The color lookup table was deliberately chosen to be a bad representation of the green of this field which can be seen in the resulting color mask on the left. A few seconds after the activation of the dynamic color space using the dynamic reconfiguration feature, the field mask marks most of the field, which leads to an increased detection precision of the field boundary and therefore of obstacles. The dynamic color space feature is implemented in a separate ROS node and runs parallel to the main vision to increase runtime performance. It then publishes the updated color space for the vision.

Field Boundary Detector Detecting the boundary of the field is important as it is used for obstacle and field marking detection and to adapt the field color space.

In order to determine an approximation of the field edge, the module searches for the topmost green pixels in columns of the image. To increase runtime performance not every pixel in every column is checked. Because of this, the resulting field boundary is not perfectly accurate which can be seen by the smaller dents in Figure 1 in situations in which field markings are close to the edge of the field. These dents are too small to be classified as obstacles and therefore insignificant compared to the improvement of runtime performance.

Depending on the situation different algorithms are used to find the topmost green pixel. The first iterates over each column from top to bottom until green is found. As the robot will be looking down on the field most of the time, the field will be a large part of the image. Searching for it from the top is therefore very reliable and fast in most cases. However, this poses the problem that the robot might see green pixels in the background when looking up to search for objects which are further away (e.g. goalposts). Then these would be detected resulting in an erroneous measurement of the topmost point of the field.

To address this problem, a different algorithm iterates from the bottom of a column to the top until a non-green pixel is found. As the white lines inside of the field would then be falsely detected as the outer boundary, a kernel is applied onto the input image to take surrounding pixels into consideration.

Due to the use of a kernel, the second method is significantly slower and should only be used when necessary. It is therefore only chosen when the robots head is tilted upwards by a certain degree, which makes it likely that the background will occupy a large portion of the image.

After finding the field boundary, its convex hull is calculated. This is necessary because the actual border of the field might not be visible by the robot since obstacles can partially obstruct it. As a convex hull eliminates the dents in the detected field boundary that were caused by obstacles, it resembles a more accurate representation.

Line Detector The Bit-Bots team uses field markings determined by the line detector to locate the robot in the field [4]. Unlike other teams (e.g. [8]), it does not output lines but points in the image which are located on a marking. This approach reduces the computation effort necessary to gather the information needed for the localization method. Pixels are randomly chosen in the image below the highest point of the field boundary. Afterward, the color of the pixel is matched to the color space representing the color of field markings via the *Color Detector*.

To improve the performance of this method, the point density is adapted accordingly to their height in the image. Additionally, the point density is increased in areas with line detections in the previous image.

Candidate Finder As multiple modules (FCNN Handlers and Obstacle Detectors) detect or handle candidates which can be described in multiple ways (e.g.

coordinates of corner points, one corner point and dimensions or center point and radius), a generalized representation is necessary. To approach this issue, the *Candidate* and *Candidate Finder* classes are used.

The *Candidate* class offers multiple representations of the same candidate as properties. Additionally, a function checking whether a point is part of the candidate is provided.

The *Candidate Finder* is an abstract class which is implemented by modules detecting candidates. It ensures a unified interface for retrieving candidates and thereby increases adaptability.

Obstacle Detector Most obstacles obstruct the actual field boundary because they are inside the field and have a similar height as the robot. Therefore, the area between the convex hull of the field boundary and the detected field boundary itself is considered an obstacle, if its size is greater than a threshold. Possible objects in the field are robots of both teams, goalposts and other obstacles, like the referee. The classification of an obstacle is based on its mean color, as a goalpost will be predominantly white while robots are marked with their team color (blue or red).

FCNN Handler The FCNN Handler module is used to preprocess images for a given FCNN-model and to extract candidates from its output. The preprocessing consists of a resize of the input image to fit the input size for the given model. As FCNNs return a pixel-precise activation (two-dimensional output of continuous values between 0 and 1), a candidate extraction is necessary. To optimize the runtime performance, the candidate extraction is implemented in C++ and accessed as a Python module.

For debug-purposes and to continue the processing of the FCNN-output in another ROS node, a generation of a ROS message containing the output in the form of a heatmap is included.

Currently, we are using an FCNN to locate the soccer ball in the image. It is based on the model proposed in [20] and trained on images and labels from the *Bit-Bots ImageTagger*[2] [9].

Debugging and Evaluation To improve the results of the vision pipeline, it is essential to analyze and evaluate the results of algorithms and parameters in the pipeline. This is achieved by using ROS bags, the debug module, the *Evaluator* node and by analyzing messages between nodes.

To ease debugging, the *debug* module allows the user to create a debug image (the current input image with all detected features drawn into) via a simple interface, resulting in live insights into the vision pipeline (see Figure 1).

To measure the runtime of specific components, external profilers and internal time measurements are used. The neural networks are evaluated separately. To analyze the performance of the whole vision pipeline, we developed an external *Evaluator* node, which is presented in Section 4.

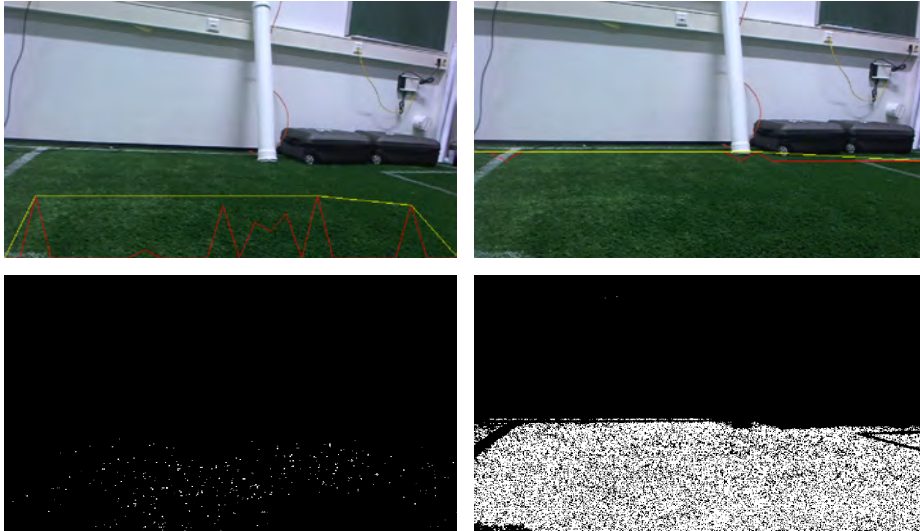[2] `https://imagetagger.bit-bots.de` last accessed: 2019-06-14

Fig. 3: Demonstration of the dynamic color detector. A deliberately badly fitting color space was used on the top left image. The resulting color mask is shown in the bottom left. White pixels represent field-color detections. The dynamic color space adapted the original color space which resulted in the improved field-color detection depicted in the bottom right color mask which is based on the input image in the top right.

3.2 Configuration

Currently, the vision pipeline configuration consists of 80 parameters. Parameters are used to define variables which can have different optimal values depending on the environment (e. g. color definitions or stepsizes in scanline approaches). All parameters are defined in a single file in the YAML format. The ROS parameter server manages these parameters and parses the configuration file. Parameters are stored in the vision module to improve performance. Via its dynamic re-configuration server, ROS allows the adaption of parameters in a graphical user interface while the module is running. A change of the value of a parameter triggers a callback in the vision pipeline which propagates the change to the modules. The dynamic configuration, which is applicable to all parameters, allows users to inspect the effect of the changed parameters with a minimal delay. The results are drawn into the debug image. Thereby, the user is able to adapt parameters intuitively based on runtime and detection performance.

4 Evaluation

To evaluate the vision pipeline, the *Evaluator* node from the *bitbots_vision_tools* package is used. The *Evaluator* reads a YAML file containing image names and corresponding object annotations from a whole image set. A corresponding export format and test data is provided in the *Bit-Bots ImageTagger*[9]. Afterward,

the annotations are verified and preprocessed. Sequentially, it feeds the images into the vision pipeline (as ROS messages) and waits for the vision output.

On a Jetson TX2, the system processes 8.1 images per second without debug output generation. Our vision pipeline is not separated into several ROS nodes (except the dynamic color space node which is actively adapting the parameters of the pipeline) because of the delay in the message generation and passing process and because multiple modules require common information (e. g. the color module). ROS nodelets [22] cannot be used as they are currently not supported in rospy, the ROS interface for Python.

To determine the precision of a detection, the Jaccard-Index (Intersection over Union, see [21] is used. Two masks are generated for each annotation type. One mask is based on the label created or at least verified by a human and the other one on the output of the vision pipeline. A Jaccard-Index of 1 signifies a perfect detection, while 0 represents a completely wrong detection.

The results of our evaluation are presented in Table 1. The values are the mean of the Jaccard-Indices of all labels for each detection class. In this evaluation, a fully labeled and publicly available set[3] of 707 images was used.

Table 1: The mean Jaccard-Indices of the detections separated by detection class.

class	ball	field boundary	line	goalpost	robot red	robot blue
mean Jaccard-Index	0.677	0.925	0.021	0.183	0.149	0.380

The context of RoboCup Humanoid Soccer required compromising detection precision for runtime performance. For example, while the height of an obstacle is not relevant to avoid collisions, it has a significant impact on the used metric. The low rating for line detection is owed to our point based detection method compared to line based labels. In our case, we assume that the detected line points are sufficient for their purpose in the robot self-localization. The indices of the goalpost, robot and especially the obstacle detections are positively biased due to high numbers of negative samples included in the data set.

Despite the drawbacks of the metric, it allows a quantitative analysis of changes in parameters and algorithms. We propose this method of evaluation as it allows different vision pipelines to be automatically tested under the same conditions. Therefore whole vision pipelines are comparable to each other within the same metric with one conveniently usable tool, given they are using ROS [15] and standardized messages [5].

---

[3] `https://imagetagger.bit-bots.de/images/imageset/261/` last accessed: 2019-06-14

5   Conclusion and Further Work

We presented a state of the art vision pipeline for the RoboCup Soccer context.

Through its implementation in Python, the system trades runtime performance for high adaptability and expandability while maintaining a usable frame rate and offering a state of the art neural network based ball-detection method. It is a starting point for new teams and teams transitioning to Python or ROS. Additionally, the modules can be integrated into existing vision systems. The *Evaluator* node offers a novel approach to analyze the complete vision pipeline while existing approaches solely evaluate isolated parts of the system. Thus, changes in a single module and their effect on other modules and overall performance can be evaluated.

In the future, we are planning to improve our FCNN to detect multiple object classes efficiently. In particular, the performance of our Neural Networks can be improved as we believe that our Jetson TX2 is not used to its full capacity. To achieve this and detect very distant objects reliably we are considering implementing the method proposed in [10].

Additionally, we are going to investigate the applicability of using more external information (e. g. the robot pose or a world model) in the vision pipeline for further optimizations.

We invite other teams to use (and adapt) the presented vision pipeline or modules in their own software stacks. The project is publicly available on GitHub via `https://github.com/bit-bots/bitbots_vision`.

References

1. Allali, J., Gondry, L., Hofer, L., Laborde-Zubieta, P., Ly, O., et al.: Rhoban football club – team description paper. Tech. rep., CNRS, LaBRI, University of Bordeaux and Bordeaux INP (2019)
2. Baltes, J., Missoura, M., Seifert, D., Sadeghnejad, S.: Robocup soccer humanoid league. Tech. rep. (2013)
3. Barry, D., Curtis-Black, A., Keijsers, M., Munir, S., Young, M.: Electric sheep team description paper. Tech. rep., University of Canterbury, Christchurch, New Zealand (2019)
4. Bestmann, M., et al.: Hamburg bit-bots and wf wolves team description for robocup 2019 humanoid kidsize. Tech. rep., Universität Hamburg, Germany and Ostfalia University of Applied Sciences, Wolfenbüttel, Germany (2019)
5. Bestmann, M., Hendrich, N., Wasserfall, F.: ROS for humanoid soccer robots

6. van Dijk, S.G., Scheunemann, M.M.: Deep learning for semantic segmentation on minimal hardware. In: RoboCup 2018: Robot World Cup XXII. Springer (2018)
7. Dorer, K., Hochberg, U., Ülker, M.W.: The sweaty 2019 robocup humanoid adult-size team description. Tech. rep., Univ. Appl. Sci. Offenburg (2019)
8. Fan, W., Chen, X., Jiang, J., Li, C., Pan, Y., Zhou, C., Xiong, R.: Zjudancer team description paper. Tech. rep., State Key Lab. of Industrial Control Technology, Zhejiang University, Hangzhou, China (2019)
9. Fiedler, N., Bestmann, M., Hendrich, N.: ImageTagger: An open source online platform for collaborative image labeling. In: RoboCup 2018: Robot World Cup XXII. Springer (2018)
10. Houliston, T., Chalup, S.K.: Visual mesh: Real-time object detection using constant sample density. RoboCup 2018: Robot World Cup XXII (2018)
11. Leiva, F., Cruz, N., Bugueño, I., Ruiz-del Solar, J.: Playing soccer without colors in the spl: A convolutional neural network approach. RoboCup 2018: Robot World Cup XXII (2018)
12. Luna, J.P.V., Vázquez, S.G.R., Martinez, I.J.C.J., Ramírez, I.D.N.: Eaglebots.mx team description paper. Tech. rep., Highest Institute of Technology of Tepeaca (2019)
13. Mahmoudi, H., Fatehi, A., Gholami, A., et al.: Mrl team description paper for humanoid kidsize league of robocup 2019. Tech. rep., Mechatronics Research Lab, Dept. of Computer and Electrical Engineering, Qazvin Islamic Azad University, Qazvin, Iran (2019)
14. Philipps, J., Rumpe, B.: Refinement of pipe-and-filter architectures. In: International Symposium on Formal Methods. pp. 96–115. Springer (1999)
15. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., et al.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
16. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
17. RoboCup Technical Committee: Robocup soccer humanoid league laws of the game 2018/2019. Tech. rep. (2019)
18. Scheunemann, M.M., van Dijk, S.G., Miko, R., Barry, D., Evans, G.M., et al.: Bold hearts team description for robocup 2019 (humanoid kid size league). Tech. rep., School of Computer Science, University of Hertfordshire (2019)
19. Schnekenburger, F., Scharffenberg, M., Wülker, M., Hochberg, U., Dorer, K.: Detection and localization of features on a soccer field with feedforward fully convolutional neural networks (FCNN) for the adult-size humanoid robot Sweaty. In: Proceedings of the 12th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Birmingham (2017)
20. Speck, D., Bestmann, M., Barros, P.: Towards real-time ball localization using cnns. In: Robot World Cup XXII. Springer (2018)
21. Tan, P.N.: Introduction to data mining. Pearson Education India (2006)
22. Tully Foote, R.B.R.: Ros wiki: nodelet, `https://wiki.ros.org/nodelet`, accessed: 28.03.2019