Studienarbeit im Studiengang Informationstechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

On-Board Computer Vision for Autonomous Ball Interception

Implementing the Vision-Blackout Technical Challenge in the Robocup Small Size League

by Fabio Seel & Sabolc Jut

Abgabe: 21.06.2019

Bearbeitungszeitraum:	12.10.2018 bis 21.06.2019
Matrikelnummer, Kurs:	9255332, TINF16ITIN
Betreuer:	Prof. Dr. Jochem Poller
Unterschrift Betreuer:	

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: *On-Board Computer Vision for Autonomous Ball Interception- Implementing the Vision-Blackout Technical Challenge in the Robocup Small Size League* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mannheim, den 21. Juni 2019

List of Figures

Contents

1 Introduction

1.1 Motivation

The main goal of this work is the successful participation in the "Vision-Blackout"-Technical Challenge of the RoboCup 2019, an annual international soccer-robotics competition held in Sydney, Australia in 2019. While the main tournament of the Small Size League is of considerable significance, the technical challenge aims to push the technological boundaries in the league. The "Vision-Blackout" Technical Challenge in 2019 consists of the successful approach and intercept of a moving ball without reliance on the global vision feed and using only on-board sensors on the robot.

1.2 The RoboCup

The idea of the RoboCup is to "promote robotics and AI research, by offering a publicly appealing, but formidable challenge" [1]. While it is obvious that the direct social or economic output of the RoboCup is rather small, it empowers research that shows applicability for real world problems. Over decades, in artificial intelligence research chess has served similar purposes: Like chess, the RoboCup is considered to work on a "standard problem", where various approaches to a multiplicity of problems can be evaluated. In Table 1.1 a comparison between domain characteristics of computer chess and RoboCup is shown. It as well shows how RoboCup shifts the research focus to dynamic situations, while still providing limits and constraints, that allow for a bearable research cost.

	Chess	RoboCup
Environment	Static	Dynamic
State Change	Turn Taking	Real time
Info. accessibility	Complete	Incomplete
Sensor reading	Symbolic	Non-symbolic
Control	Central	Distributed

Table 1.1: Difference of domain characteristics between computer chess and RoboCup [1]

The RoboCup consists of four different leagues:

- RoboCupSoccer
- RoboCupRescue
- RoboCup@Home
- RoboCupIndustrial

Since this work is based in the RoboCupSoccer area, only the RoboCupSoccer league will be described further in the following.

1.2.1 The RoboCupSoccer

The goal of the RoboCupSoccer is to form a team of robots, capable to win a game against the human soccer world champion by 2050 [1]. Concerning the overall complexity of robotic soccer, the RoboCupSoccer was split into different leagues, each with an emphasis on different topics:

- **Humanoid** - the complete challenge: development of the robots, dynamic walking, visual perception of ball and players, self-localization ... It is also worth noting that the league refuses to use non-human like range sensors.
- **Standard Platform** - similar to the Humanoid league, but using a standardized robot (the NAO robot from Softbank Robotics)
- **Middle Size** - non-humanoid robots driving on wheels, but acting autonomously. Not using humanoid robots allows to put the research focus on the perception of the environment and multi-agent cooperation.

- **Small Size** - a hybrid centralized/distributed system. Unlike in the other leagues, localization and tactics are determined by systems outside of the robots. Hence, it is the fastest of the leagues and the research is mainly committed to tactical decision making and multi-agent cooperation.
- **Simulation** - virtual league. Without the challenges in perception and control, it is possible to fully focus on artificial intelligence and team strategy.

Different goals and restrictions lead to differing tasks to be solved in each league, including control and coordination of the robots, vision systems and more. The rules of the RoboCupSoccer leagues are changed every year, with the intent to assure research push. Mostly those changes make the league somewhat closer to real football. For example, by now the Small Size League aims to increase the number of robots to eleven, while in other leagues color coded balls have been replaced by arbitrary balls.

1.2.2 The Small Size League (SSL)

The Small Size League is one of the oldest leagues at the RoboCup. In the Small Size League, the competition is held with wheeled robots, hence the controlling is easier and the research focus can be put on multi-agent coordination and tactical decision making.

Setup Currently, there are eight vs. eight robots playing against each other on a field of 12m x 9m. The small size league has - unlike any other league - a vision system shared between all teams. Above the field, one or more cameras are mounted (cf. Figure 1.1). On top of each robot is a pattern containing circles in different colors and thus identifying the robot (and its orientation) uniquely while the ball used is an orange golf ball. The vision software on the shared computer calculates the positions and rotations of the robots using the unique patterns as well as the ball. After the calculation is done, global positions of the robots and ball(s) are broadcasted into the network and received by the computers of both teams. After this point, the teams are responsible on their own for everything that happens: They need an artificial intelligence that coordinates their robots, a way to communicate to the robots and, of course, the hard- and software of the robots themselves.

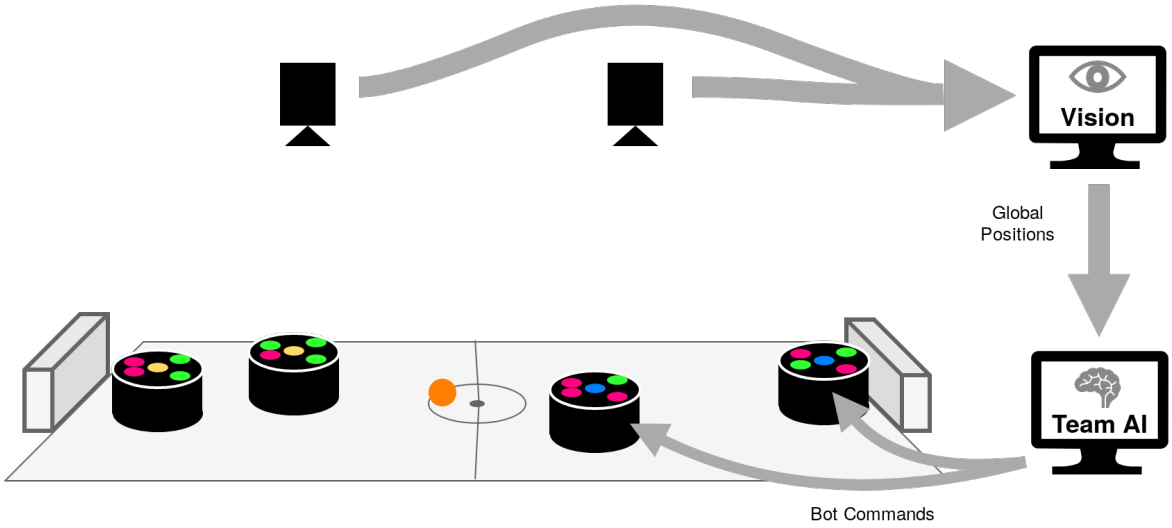


Figure 1.1: Basic Setup of the Small Size League

The robots The teams are responsible for their hardware, but several techniques have become standard (Figure 2.1). The use of omnidirectonal wheels allows the robots independently control their rotation and position, which makes them holonomic robots. A special device is used to kick the ball or even chip-kick it, the allowed speed of the ball is 6.5m/s. The "kicking unit" of many teams also contains a sensor for detecting contact to the ball. Additionally, a dribble device is integrated that allows to give the ball a backspin, so that the robot can control the ball.

1.2.3 TIGERs Mannheim

The project TIGERs (**T**eam **I**nteracting and **G**ame **E**volving **R**obot**s**) Mannheim is a robot soccer team participating in the Small Size Leauge of the Robocup. The project is maintained by students of the Cooperative State University Mannheim (DHBW Mannheim) and exists since 2009. The team's first appearance at the RoboCup was in 2011 and by 2014, for the first time a top eight rank was reached. After winning the European Championship in 2016, the team celebrated its biggest success in last year's RoboCup 2018. The third place was reached in the competition and the Excellence Award for the overall performance and representation of, as well as identification with the RoboCup goals was won.

1.3 The Vision-Blackout Technical Challenge

In the Small Size League, each year additional challenges are held. This is done to push development in specific research fields and prepare the teams for (possible) future changes of the rules. In 2019, one of those Technical Challenges is the so-called "Vision-Blackout" [2]. In this challenge, the teams have no access to the global position estimates usually offered by the SSL-Vision software. Therefore, onboard sensing and control is needed. The Technical Challenge consists of two stages where a single robot needs to accomplish a task. In both cases, the robot starts at least 2 meters away from all of the fields edges.

1.3.1 Stage One: Approach Stationary Ball

In the first stage, the robot has to find and grab a stationary ball that is placed at a random location within a 1m x 1m box around the robot. This requires the robot to find the ball, no matter if it is in front or behind it.

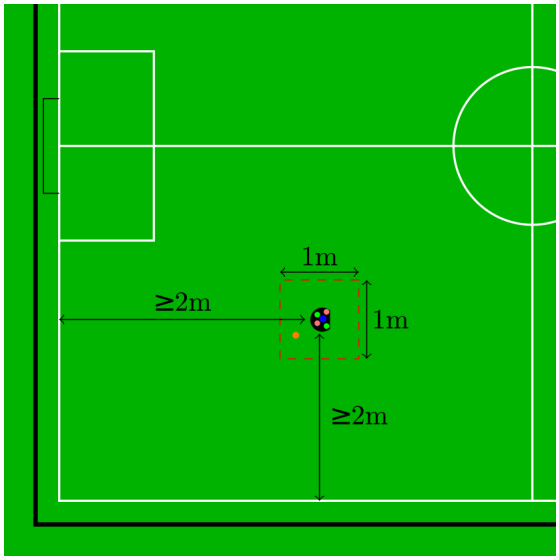


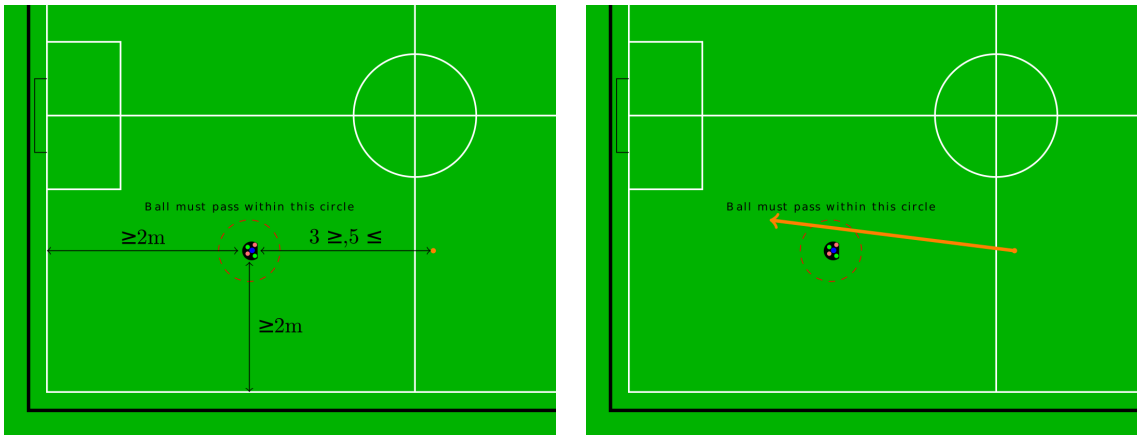
Figure 1.2: Constraints for stage one of the technical challenge [2]

One can achieve points for:

- **+1** Touching the ball with any part of the robot
- **+1** Touching the ball with the dribbler
- **+1** robot stopped with ball touching the dribbler at the end

1.3.2 Stage Two: Intercept a Moving Ball

In the second stage, a moving ball needs to be intercepted. The ball will start its trajectory between 3 to 5 meters with a speed of less than 6.5 m/s and will pass within 0.5m of the robots starting position. The starting position of the robot is the same as in the first stage and the robot faces the ball right from the beginning.



(a) Starting positions of ball and robot for stage two of the circle

(b) The ball needs to pass within a circle with a 0.5m radius around the robot

Figure 1.3: Constraints for stage two of the technical challenge [2]

Points can be achieved for the exact same criteria as in the first stage of the challenge.

1.4 Problem Definition

To successfully participate in the "Vision-Blackout"-Technical Challenge in the Small Size League as part of the Robocup 2019 competition, several tasks need to be done in hardware and software. In hardware, a camera and the corresponding data feed need to be integrated into the existing robots. In software, a computer vision ball-detection algorithm needs to be designed which not only detects a ball but also reports the position relative to the robot to the robot movement controller. Additionally, an algorithm for the robot movement based on the incoming ball positions from the ball detector is needed for both parts of the challenge.

2 Preliminaries

This chapter will focus on the preliminary information surrounding the technologies used in the implementation of the technical challenge.

2.1 Hardware

2.1.1 The TIGERs Robot

The robots of the TIGERs have been developed by the team itself and are highly specialized on the requirements for robotic soccer in the Small Size League. Like the Robots of the other teams, a TIGERs robot has omnidirectional wheels and a combined kicking, chip-kicking and dribbling unit.

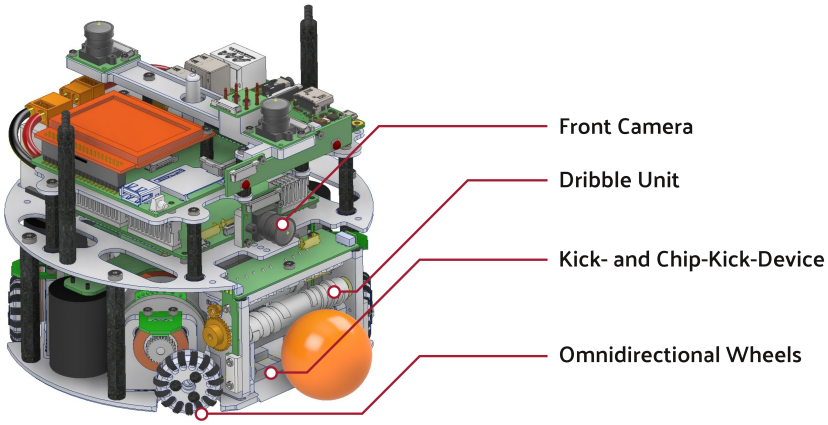


Figure 2.1: The 2019 generation of the TIGERs robot

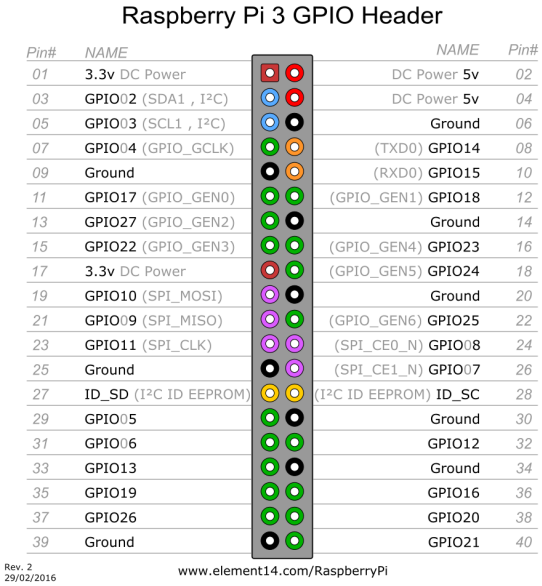
A variety of sensor inputs such as motor feedback, the infrared barrier of the dribbling unit or a gyroscope are connected to the mainboard, where the controlling of the robot takes place. Custom input and expendability are possible with a UART interface on the robot. Figure 2.1 shows that the 2019 generation of the TIGERs robots comes with a new feature: Onboard cameras that can be used as additional

sensor input, providing local vision information. The vision processing is supposed to take place on a Raspberry Pi that is integrated into the robot and communicates with the main robot computer via an UART interface.

2.1.2 Raspberry Pi

The Raspberry Pi is a Singele-Board-Computer (SBC) that is often used in education, Internet-of-Things-Applications or private projects (Figure 2.2a). Because it was developed to offer people to start learning about computers and programming, it is sold for a low price. The "Pi" contains a single chip system with an ARM-Microprocessor and interfaces for USB, LAN, a camera module and general purpose input/output pins (GPIO) (cf. Figure 2.2b). Two of those pins offer UART functionality (Pin 8 and Pin 10).

(a) The Raspberry Pi 3b+



(b) The Raspberry Pi 3b+ Pinout

2.1.3 Pi Camera Module

We use the the first generation of the Raspberry Camera Module as the camera connected to the Pi. The Raspberry Pi Camera Module is a specialized camera intended for use on the Raspberry Pi Single Board Computers. Unlike most cameras for general Linux computers, it is not connected via an USB interface but rather

with a flat cable connected directly to the PCB of the Raspberry Pi. The 5 Megapixel sensor itself has a native resolution of 2592x1944 pixels.

Figure 2.3: Illustrated sensor crops of the modes of the Pi Camera Module V1. [3]

Mode	Resolution	Aspect Ratio	Framerates	FoV	Binning
1	1920x1080	16:9	1-30fps	Partial	None
2	2592x1944	4:3	1-15fps	Full	None
3	2592x1944	4:3	0.1666-1fps	Full	None
4	1296x972	4:3	1-42fps	Full	2x2
5	1296x730	16:9	1-49fps	Full	2x2
6	640x480	4:3	42.1-60fps	Full	4x4
7	640x480	4:3	60.1-90fps	Full	4x4

Figure 2.4: Different modes of the Pi Camera Module and their characteristics [3]

The camera module can operate in a variety of different sensor modes, where the modes are distinguished by different resolutions, sensor crops and maximum frame rates. The different fields of view are shown in Figure 2.3. As seen in the table in 2.4, only the modes 2, 4, and 7 support both the full sensor frame and high frame rates and are thus relevant for our task. Mode 2 is the high resolution and low

framerate option with 5 Megapixels, but only at 15 fps. Mode 7 has the exact opposite characteristics with a resolution of only 0.3 Megapixels but a high framerate of up to 90 fps. The compromise between these two modes is the mode 4. At a resolution of 1.3 Megapixels and a maximum framerate of 42 frames per second it can both offer decent resolution and framerate.

2.2 Basic Computer and Robotic Vision Principles

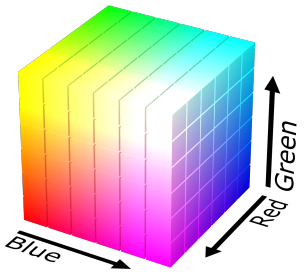
In the field of autonomous robots one of the key challenges is to give robots the possiblity to perceive their environment. A vision system, like the human one, appears to be a powerful instrument for tasks like orientation, distance estimation or self-positioning. That is the reason why a lot of robotic applications make use of computer vision, even when such vision systems tend to be very sensitive and error-prone. In computer vision, most methods to identify objects rely on the concepts of color segmentation and edge detection. To understand those concepts and how they can be applied, it is worth having a look at how an image is represented in a computer first. Here, a basic understanding of color spaces is crucial.

2.2.1 Discretization of a Scene

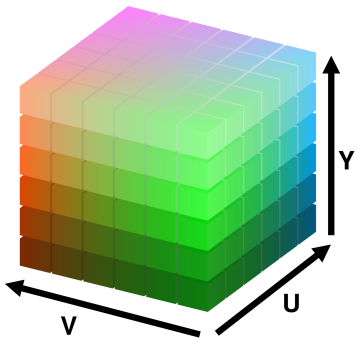
A camera discretizes both the space and the color. Conceptually, this discretization is done by the sensor of the camera and results into pixels (local discretization) and corresponding color values. A single pixel hence represents the light detected by the camera's sensor at a specified position in space, and evaluates the color of the percepted light.

2.2.2 Color Spaces

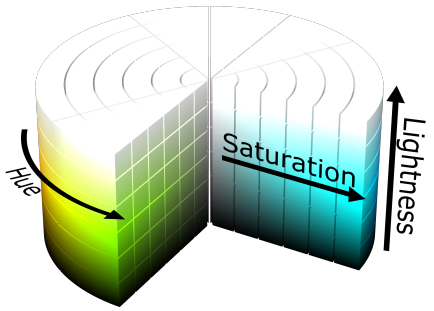
The color usually consists of three different values - "channels" - that must be offset against each other to result in the actual color. However, the color can be represented by different models - so called color spaces. For different applications, different color spaces are appropriate.



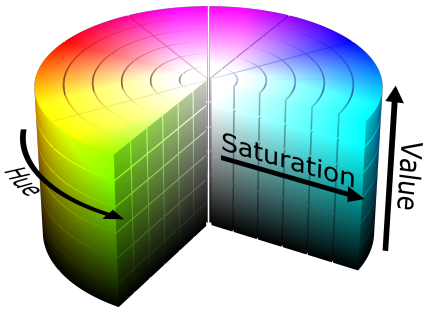
(a) RGB Color Space: a cube [4]



(b) YUV Color Space: a cube



(c) HSL Color Space: a cylinder [5]



(d) HSV Color Space: a cylinder [6]

Figure 2.5: Color Spaces visualized: RGB, YUV, HSL and HSV

RGB

The three primary colors red, green and blue (RGB) are additively combined to a color. The RGB space is a color space closely coupled to the human perception of color. This is one of the major advantages of this color space and the reason why it is used widely.

YUV

YUV is a color space where the luminance is in an isolated channel (Y). The human vision system is more sensitive to luminance than to chrominance (the U and V channel). An image in the YUV space can be compressed by downscaling the chrominance channels [7], which has only little effect on a humans perception. For that reason, YUV is often used for fast transmission of images. The "downscaling" of the chrominance channels is called subsampling. Different subsampling systems and ratios are conceivable. They are identified by a sequence of numbers:

- horizontal sampling reference (usually 4)
- number of chrominance samples per row
- number of changes of chrominance samples between two rows.

Figure 2.6 shows how this subsampling works. For example in a 4:2:0 subsampling, four pixels of the first row in the luminance channel are presented by two pixels in the U and V channels. In the second row, zero of these pixel change, which means, that the color values for the second row is the same as for the first row. Hence, the U and V channel in a YUV 4:2:0 image have only half the width and height (and thus only a quarter of the amount of pixels) as the Y channel.

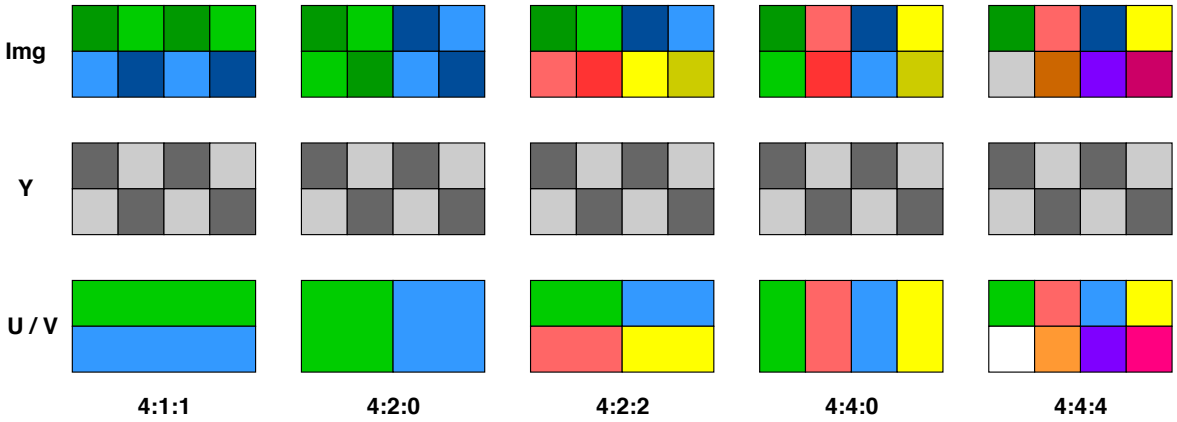


Figure 2.6: Different Mappings for Chroma Subsampling

HSL and HSV

Hue, saturation and lightness (HSL) or value (HSV) are the three channels of this color spaces. HSL and HSV are similar to each other, but not identical. What distinguishes them from other color spaces is that they are both cylindrical geometries and not cubes. The reason is that "hue" is represented as an angular dimension starting with red (0°) passing the other primary colors green and blue at 120° and 240° and returning to red (360°). Their major advantage is that the actual color can be retrieved from one single channel (hue). This simplifies detection of color coded objects.

2.2.3 Color Segmentation

Color segmentation describes the method of identifying an object based on its color. If possible, often even a colored marker is used to be able to track an object. Different methods for color segmentation exist.

Methods The most obvious and simplest one is to define constant thresholds for each color channel . While this is easy to implement and to use, it results in rectangular blocks in the color space, which might not give the desired accuracy for the color detection [8]. Alternatively, a Look-Up-Table can be defined, that contains for every possible value in the color space the corresponding detection color [9]. Conceptually different is the use of nearest neighbor classifications. Only some of the pixels are classified as one of the colors. The classification of the other pixels is done by determining what is the dominant color of the k nearest neighbors of already classified [10].

Masking Masking an image describes the process of calculating a binary image based on certain criteria. The mask allows to analyze only the pixels that fulfill the criteria. Color Segmentation can be used for defining a mask, all pixels that match the wanted color are masked with "1" (true), the rest is masked with "0" (false).

2.2.4 Edge detection

If the color of an object can be or is one of two important features of an object, the shape of it is the other one. The outline of an object can be retrieved by edge detection. Edge detection methods operate on the neighborhood of a pixel and examines the relative contrast between them. There are several first order (maximum slopes) and second order (zero crossings) approaches and different operators have been proposed, such as Canny [11], Prewitt [12] or Marr-Hildredth [13]. It is possible to examine each color channel individually and thereby acquire more specific information.

2.2.5 Reconstruction of the 3D Position - Mapping

For robotic vision it is usually not enough to identify an object in an image, but the information where exactly it is often is of the same or even higher value. If

a position in three-dimensional space is needed, a single camera can usually not deliver sufficient information. In such a case, two cameras can be used and stereoscopic vision can be applied. Often additional information or reduced information requirements allow to use only one camera. For example, if only the distance to an object is searched, the size of the object in the image will satisfy the request. If it is known that the object moves along a plane in the three dimensional space - like a ball on a field - the position in the image can as well be mapped directly to a world position. Both ideas combined offer the possibility to locate an object accurately in three dimensional space with only a single camera. However, this is not possible if the original size of the object is not known beforehand. Moreover, information about the camera - such as lense distortion, position and orientation - is needed to correctly identify a position.

2.3 UART Communication

The *Universal Asynchronous Receiver Transmitter* (UART) is a hardware device for asynchronous serial communication. It consists of a receiver and a transmitter, that need to be configured to the same same baud rate, character length, parity, and stop bits.

Data Frame In every frame one character is transmitted. The character can have five to nine bits, but usually one byte (eight bits) is used. To signal the start of a message delivery, a start bit reverting the idle state (high-voltage) is used. Next, the data bits are placed, followed optionally by a parity bit. The stop bit is high-voltage again, returning the voltage to the idle state. This way, between the start bit (low-voltage) and the end of the message there are at least two signal edges between two characters. This allows re-synchronization of receiver and transmitter.

The messaging modes are often described by the data/parity/stop shorthand notation which the number of data bits per frame, the type of parity using the letters E for even parity, O for odd parity and N for no parity bits and the number of stop bits in the frame. This results in strings like 7E1 for a frame with seven data bits, an even parity bit, and one stop bit. Most commonly used is the 8N1 mode using 8 data bits, no parity and a stop bit in each frame.

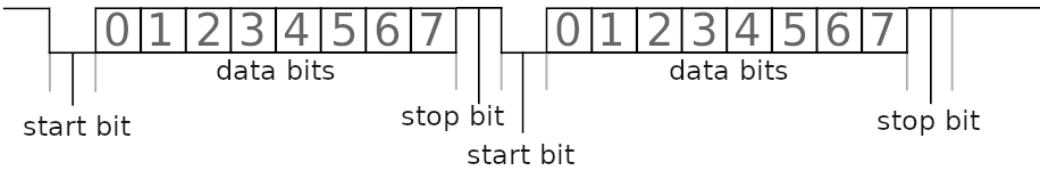


Figure 2.7: Two 8N1 UART frames with the annotated start and stop bits

3 Approaches for Object/Ball Detection

One of the most important challenges faced in this work is the detection of the ball. A lot of research has already been done in this topic, and in all RoboCupSoccer Leagues some kind of ball detection is needed. There are several common approaches for ball detection we want to discuss in the following section.

3.1 Hough Transform

The use of the Hough Transform is one of the most common approaches for object detection. It requires an edge image and allows to identify geometric shapes in it.

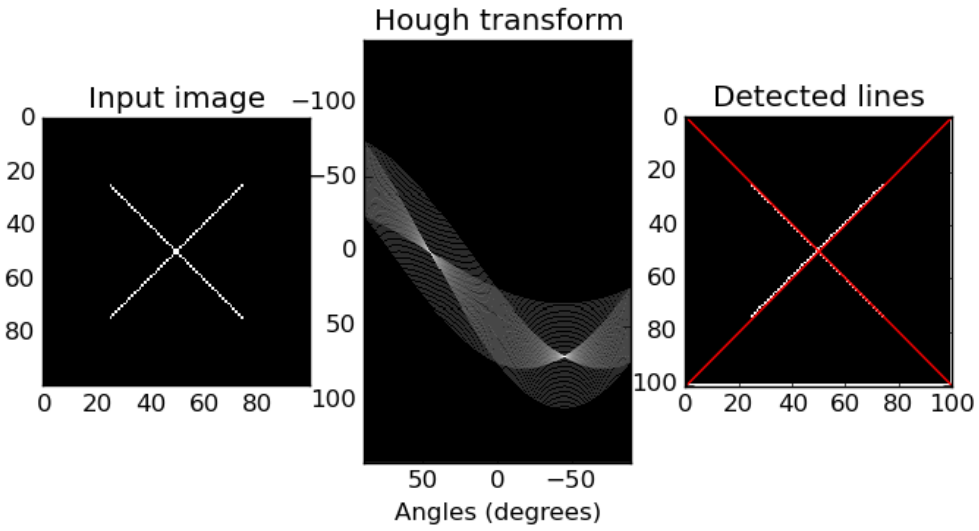


Figure 3.1: Hough Line transform: The source image, the transformed image and the detected lines

3.1.1 Basic Principle - Line Detection

Consider an edge image containing a cross, as the one in Figure 3.1. A line can be described by the angle, at which they appear in the image and the distance of this line to a fixed point, lets say the center of the image or the bottom left corner. Now each white pixel is allowed to "vote" which lines pass through it. This results into the transformed image, where each possible combination of angle and distance contains the sum of votes. If a lot of pixels voted for the same angle and distance, it is very likely that there is a line with this parameters.

3.1.2 Modification for Circle Detection

The modification of the Hough transform to be able to detect circles needs an additional dimension in the transformed space. This dimension represents the radius, and hence restricting possible radii reduces computation cost. The other two dimensions are the x and y coordinate of the circle's center. Each radius has its own "layer". In this layer, a canditate to be part of a circle votes for all pixels that could be centerpoints of a circle of this radius. This is identical to drawing a circle of that radius around the candidate. If enough points vote for a combination of center position and radius, the circle is detected. A comparison of different implementations for circle detection using the Hough Transform can be found in Yuen et al. [14].

The major drawbacks of Hough-based approaches in ball detection is a relatively high computational cost. However, if the image contains sufficiently strong edges, the use of a Hough Transform delivers stable and accurate results.

This approach has been used in various research projects in robotics, e.g. for catching a flying ball [15].

3.2 Blob Detection

In the Small Size League, an orange ball is used. The leagues shared vision system makes use of color segmentation and, as a second step, blob detection to identify the ball. A blob is a connected region of pixels labeled as the same color. For the grouping of identically labeled pixels, the neighborhood of the pixels is examined. Horizontal grouping is achieved by run-length-encoding of each line. Vertically, the

resulting runs are merged using "tree-based union with path compression". A more specific description of this method can be found in [9].

Another approach to identify blobs is the use of summed-area-tables, also know as integral images, as Pluhatsch [16] suggests. In integral images, for each pixel the sum of the pixels in the rectangle stretching to the top left pixel of the image is computed. This can be done very efficiently. For any rectangle in the image, the "inner sum" can now easily be determined by the values of the rectangle's corner pixels. If we now take a binary image, obtained by color classification (all pixels labeled with the color 1, rest 0), the summed-area table can be used to efficiently search for minimal regions containing a maximum of classified pixels.

3.3 Background Substraction

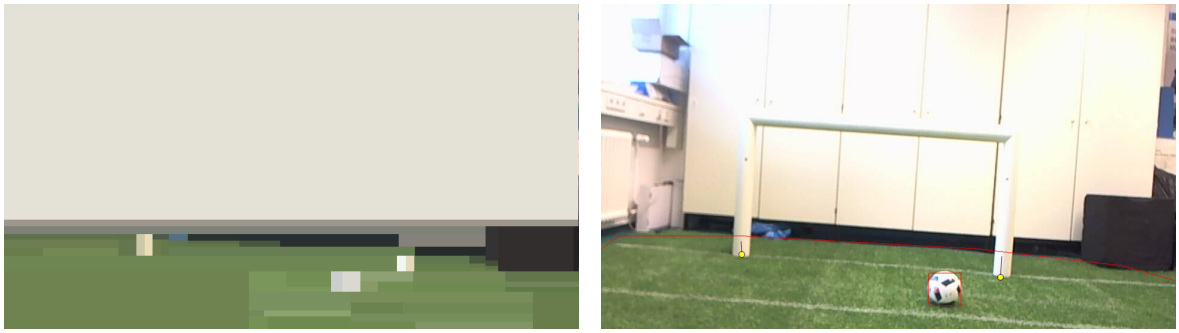
In several applications, it is as well possible to perform a background substraction in order to identify the ball [17]. Those methods rely on the assumption, that the background of a scene does not change, and hence everything that appears after substracting the background image is an object of interest. Substracting the background can be seen as a special form of masking an image. However, this is only applicable for use with static cameras.

3.4 Convolutional Neural Networks (CNNs)

Currently state of the art and highly examined by researchers is the use of Deep Convolutional Neural Networks (CNNs) for object detection in computer vision. Providing a large training dataset with positive and negative sets, a CNN achieves robust results, independent of lighting conditions [18]. The use of neural networks in this domain would be impossible without the use of Graphical Processing Units (GPUs), which allow major speed-ups (x10 to x30) compared to CPU only processing. This is unfortunately one of the major drawbacks of this approach, and a reason why in onboard systems it often can not yet be used.

3.5 Using Superpixels for Object Detection

One of the problems in computer vision is, that by the abstraction to pixels, only few information about the objects themselves can be retrieved. The human vision however perceives objects or areas at once. The approach of superpixels is, that regions of the same (or similar) color are grouped, with each group resulting called "superpixel". That means superpixels are sensitive to edges and are representing colors - the reason they seem perfect for object detection applications. Moreover, when operating with superpixels, the calculation cost is reduced, because "irrelevant" information is "hidden" inside the superpixel. The question is how to determine those superpixels efficiently. Draegert [19], member of a team participating in the Humanoid League, describes a way how to iteratively build a binary space partitioning tree (called PLANT), where the leaf nodes can be seen as superpixels. A merge of leaf nodes to one single superpixel can be performed to increase the adaption to diagonal structures. The splitting of the image is always axis aligned and aimed to increase the dissimilarity between the two resulting nodes and thus increasing the homogeneity of the nodes themselves. The use of superpixels in RoboCup appears to be very promising: Extracting the field contour or detection of blobs (e.g. for ball detection) have been done [19], as it is shown in Figure 3.2.



(a) Image reduced to superpixels using a PLANT

(b) Ball and field contours retrieved from the superpixel image

Figure 3.2: Application of superpixels for ball and field contour detection in the Humanoid League of the RoboCup

4 Implementation

In this chapter we present our approach for the technical challenge, as well as how it was integrated into the existing systems. Additionally we validate the implementation and present the results.

4.1 Setup

In our setup, we separate the controlling of the robot and the processing of images strictly. Hence, a second microcomputer - we use a Raspberry Pi - is added to our robot. A camera is connected to the Raspberry Pi, which scans each frame delivered by the camera for balls. The detected balls are send to the robot, to whom this information is like any other sensor input. This setup allows to have less critical demands to the image processing and reduces the chance fatal failures in the robots controlling unit. At maximum, a sensor information is lost. Moreover, it increases the replaceability of the image processing hard- and software, since only the interface for sending detected balls to the robot needs to be met.

Figure 4.1: The basic flow of data in our setup

4.2 Reference Methods for Ball Detection

As explained in chapter 3, a broad set of methods exists to detect balls. OpenCV (Open Computer Vision, [20]) is a standard libray for vision systems and provides support for Blob Detection and Circular Hough Transform. Another implementation tested for reference is the Small Size League's Vision software, which has as well a method for Blob Detection implemented and already shown to be applicable at RoboCup [21].

4.2.1 Blob Detection based Methods

The following two reference implementations make use of blob detection for ball detection. However, they differ in the input needed and how they are implemented.

OpenCV SimpleBlob

OpenCV's SimpleBlob Detector takes a grayscale image and searches for blobs of non-zero pixels. This means that before being able to apply this filter, it is a good idea to mask the input image. This can be done with other OpenCV methods, such as "inRange". However, the best color space to filter for specific colors is HSL or HSV, which is not available natively from the camera. Hence, a conversation into HSL or HSV color space is appropriate. 4.1 shows how the detection can be implemented.

```
1  void searchBalls(cv::Mat frame)
2  {
3      // Create a Simple Blob Detector
4      cv::SimpleBlobDetector::Params params;
5      params.minThreshold = 254; // use only 100% white pixels
6      params.maxThreshold = 255;
7      //... more params ...//
8      cv::Ptr<cv::SimpleBlobDetector> detector;
9      detector = cv::SimpleBlobDetector::create(params);
10
11     // Define color thresholds
12     cv::Scalar hsv_min(0, 117, 142, 0);
13     cv::Scalar hsv_max(24, 255, 255, 0);
14
15     // Apply a Color Threshold for masking the image
16     cv::cvtColor(frame, frame, cv::COLOR_BGR2HSV, 0);
17     cv::inRange(frame, hsv_min, hsv_max, frame);
```

```
18
19      // Detect Blobs ("Keypoints" -> center position and area)
20      std::vector<cv::KeyPoint> keypoints;
21      detector->detect(frame, keypoints);
22  }
```

Code 4.1: Reduced example for applying OpenCVs SimpleBlobDetection

Applying the SSL Vision

The vision system used in the league makes use of the method described in section 3.2. Hence, color thresholds can be defined and if blobs of sufficient size of the defined color appear, they can be returned as balls. The SSL Vision has been developed for use with the natively available YUV color space. This method is easy to use but highly sensitive for lighting changes, although in YUV Color Space the luminosity is separated from color channels. Moreover, the processing is already done in parallel. Because of the algorithm used offers nearly constant processing time, the algorithm can be seen as real-time compatible. However, when tested on the Raspberry Pi, the processing always took more than 20ms, which did not fulfill our requirements. This is not a big surprise, since it is intended for use with multiple colors to be detected. A lot of optimization work could be done to reduce the overhead and possibly achieve the speed wanted.

4.2.2 Edge Detection based Methods

Both of the previously defined methods have problems detecting the actual shape of an object. If - for instance because of shadows - parts of the ball appear not precisely enough in the defined color, it might be missed. With Edge Detection, it is not required to define the color. On the other hand, edge detection is not trivial if an image is blurry. Edge detection was first described in [13].

OpenCV Hough

The Circular Hough Transform implemented in OpenCV takes a grascaled image and internally applies and edge detection (Canny), and finally applies the actual Circular Hough Transform. To avoid false detections, it is recommended to blur the image beforehand. Several parameters can be passed to the function in order to get the wanted result [22], as it is shown in Code 4.2:

- **dp** - the inverse ratio of resolution used for detection. This parameter can be used to downsize the image before applying the detection.
- **minDist** - the minimum distance between to circles to be detected
- **cannyMax** - the upper threshold used in the Canny Edge Detection algorithm. The lower threshold will be half of cannyMax
- **minVotes** - the minimum votes a circle needs to be detected
- **minRadius** and **maxRadius** - range of acceptable radii. Larger or smaller circles will be rejected.

```
1  void searchBalls(cv::Mat frame,
2      int kernelSizeBlur, int sigmaGauss, int dp, int minDist,
3      int cannyMax, int minVotes, int minRadius, int maxRadius)
4  {
5      // Grayscale image
6      cv::Mat gray;
7      cv::cvtColor(frame, gray, CV_BGR2GRAY);
8
9      // Blur image to reduce false edges
10     cv::GaussianBlur(gray, gray, cv::Size(kernelSizeBlur,
           kernelSizeBlur), sigmaGauss);
11
12     // Get the circles
13     std::vector<cv::Vec3f> circles;
14     cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, dp, minDist,
           cannyMax, minVotes, minRadius, maxRadius);
15 }
```

Code 4.2: Reduced example for applying OpenCVs Hough Circle Transform

While this implementation is very robust, it has two major drawbacks: First, the processing speed is slow. One of the reasons for that is the use of the Canny Edge Detector, which unfortunately can not be replaced since the OpenCV interface does not allow to apply only the Hough Circle Detection step and skip the edge detection. Second, a moving ball might appear as an elliptoid rather than a circle, but in this implementation, only circles are detected.

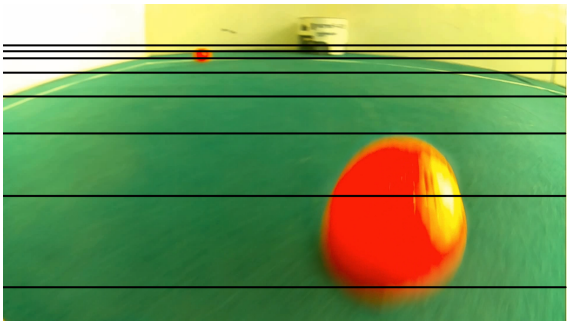
4.2.3 Concept: Grid Algorithm

The idea for the *Grid Algorithm* is to separate the image into smaller parts, on which only parts of the ball are supposed to be detected (a half or an edge). This is done in order to reduce the number of pixels to analyze by maximizing the information offered. First of all, a grid is defined with respect to the camera parameters and the angle of the camera to the field, so that the size of a gridcell always corresponds to the expected ball sizes. Since the declaration of the grid and the calculation of the lines can be done in advance, it is not consuming any time in the actual analysis of the image. For each cell an identification algorithm is performed: One possibility to implement it is to perform an analysis along the diagonals, checking whether it contains pixels that might be part of a ball and if they are at the beginning or end of the line. Depending on which diagonal(s) contain "ballpixels" or not, the cell can be labeled respectively. As well, a lot of "false-alarms" can be rejected, when irrational combinations appear.

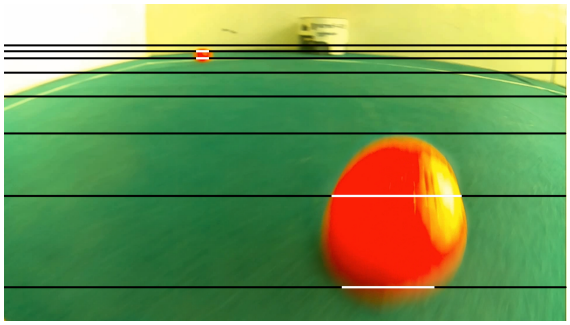
However, this method requires very exact configuration of the grid and a lot of a priori knowledge, but does not seem to be sufficiently better than other approaches. Hence, only the concept was developed but never implemented.

4.3 Scan Lines

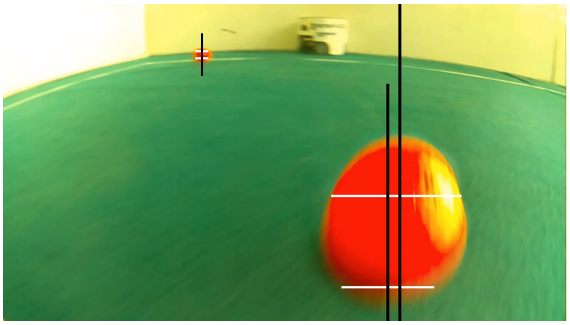
After implementation, neither of the approaches ran fast enough to satisfy the requirements - although possibly they could be tweaked to get there. In the following, we will present two different approaches that focus on reducing the operations needed for detection. Mainly inspired by the work of Lu et al. and their ball recognition algorithm using rotary and radial scan lines, we implemented another approach. As in subsection 4.2.3, we first of all aimed to reduce the number of pixels to process. The basic idea here is simple: Along few horizontal lines ball candidates shall be detected. For those candidates, in the center of the line, a vertical scan is performed. If it is succesful, after a plausability check a ball is detected. The basic process is shown in Figure 4.2 and will be further explained in the following.



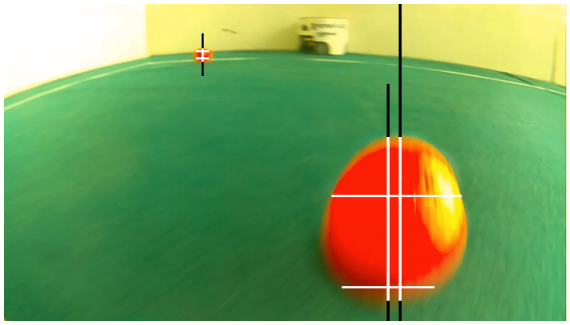
(a) Initial Scan Lines with logarithmic distribution



(b) Detected horizontal Ball candidates marked white



(c) Vertical Scan Lines defined based on the horizontal candidates



(d) Detected vertical ball candidates marked white

Figure 4.2: Scan Lines process to detect balls in the image

4.3.1 Choosing the Color Space

As seen before, it is important to choose the appropriate color space for the detection. However, switching between color spaces should be omitted if possible. Since for reducing the amount of data to transfer from the camera to a computer most cameras already use YUV, it seems a good idea to check if a detection can be done in YUV space. If we take a look at Figure 4.3, we can see that orange - the color we want to detect - is defined as a high positive "V" value and a low, negative "U" value. Moreover, as shown in section 2.2.2, the U and V channel are independent of color. Hence, we can ignore the Y channel. Next, we aim to grayscale the image in order to have one single channel to operate on. The way we do this is $V - U$. This results in an image where previously orange pixels have the highest possible value. This is optimal for the edge detection and line analysis procedure explained later.

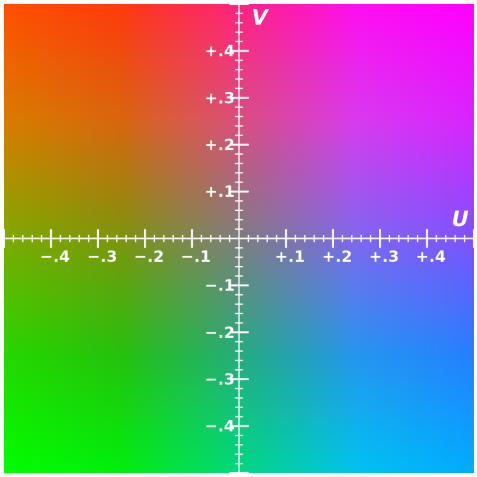


Figure 4.3: UV Plane of the YUV color space. Orange is located in the upper left corner, at positive values for the V channel and negative values for the U channel.

4.3.2 Definition of Horizontal Lines

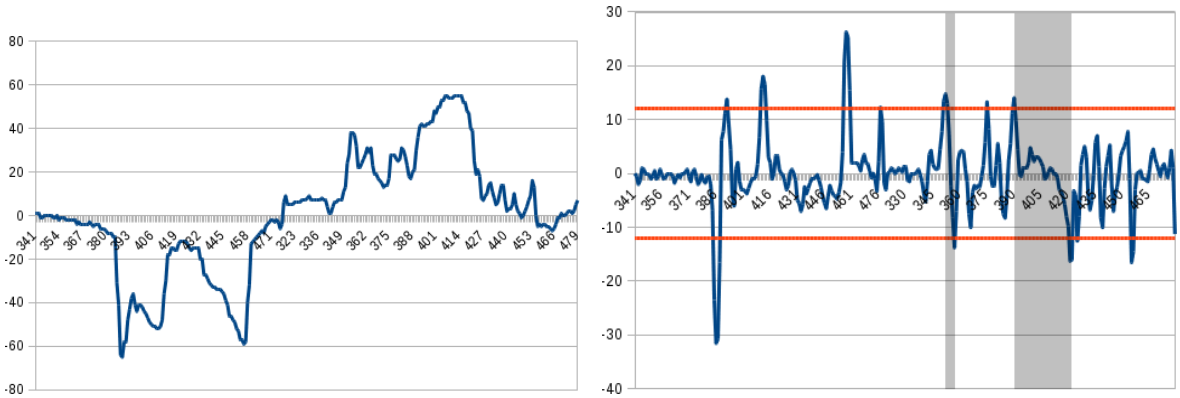
Instead of a whole grid, only horizontal lines are defined. The position of those lines is chosen in a way, that in regions, where only large balls can appear, the distance between them is high, while in regions, where small balls are expected, only a small gap is between the lines (as it can be seen in Figure 4.2a). This is done to make sure that every ball in the image is hit by at least one of those lines. We implemented different methods to define the position of the horizontal Scan Lines, so that for configuration only the position of the first line and the last line, as well as the way the lines are distributed in between (linear, logarithmic or quadratic) needs to be set.

4.3.3 Line Analysis

Along the lines a simple edge detection is performed. The effect of the process is shown in Figure 4.4. Important at this point is that the direction of the edge must be known. We will use that information for detecting starts and stops of ball candidates: A start is marked by a positive edge, while the stop of an ball candidate is marked by a negative edge. Since we need the direction of the edge, we use first order edge detection. The kernel we use is highly asymmetric:

$$\begin{bmatrix} -0.75 & -0.25 & 0.75 & 0.25 \end{bmatrix} \tag{4.1}$$

Evaluations showed that this kernel results in strong responses for sharp edges while still providing sufficient response for slightly blurred edges.



(a) Intensity along a line in the combined UV-Chanel. The elevated region in the right part is a ball.

(b) Line after applying edge detection. Additionally, thresholds are shown in orange and resulting ball candidates in grey.

Figure 4.4: Extracted Line: UV-Intensity and Edge Detection. Edges higher than the threshold are rejected as long as they are not followed by a negative edge.

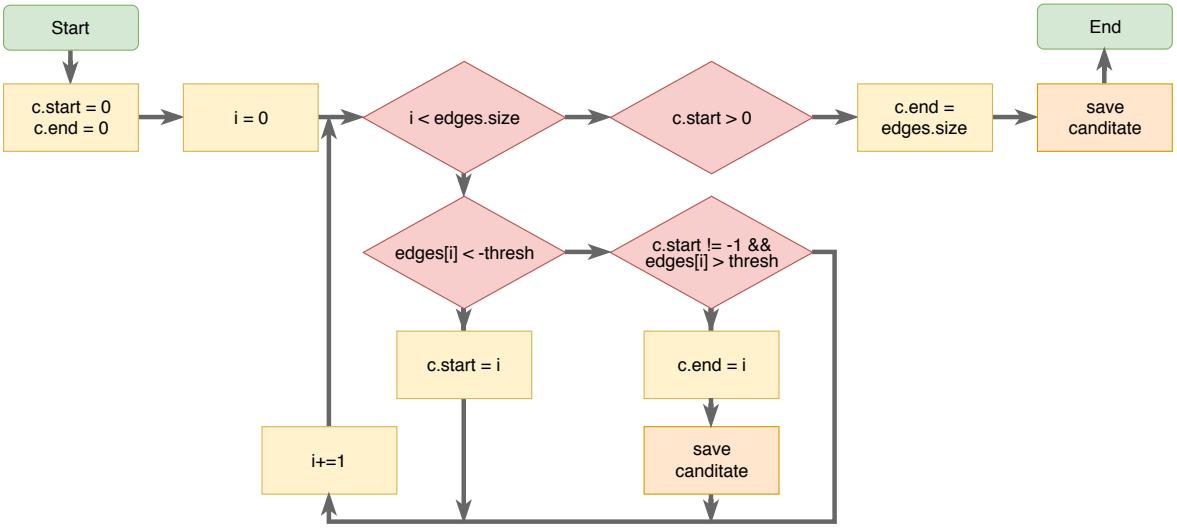


Figure 4.5: Algorithm for detecting ball candidates along a single line

This information will be used in the following step, where the resulting edges of the line are analyzed and searched for ball candidates. The procedure is shown in Figure 4.5. Beginning on one side of the image, first an edge to orange is searched ("start of ball candidate"). When found, the position is updated. If there is another start that is found before the inverse edge has been found, the former position is discarded and the starting position is updated. This allows to identify balls in front of objects of similar colors. If a "positive edge" is followed by a "negative edge", the candidate is added to a list of candidates and the search is started again. One major advantage when scanning lines is, that the pixels of one line in the image are held continuously in the storage, and thus access is very quick.

4.3.4 Vertical Scan

For all found canditates, the same procedure is used to perform a vertical scan along a line through the midpoint of the candidate. Another optimization can be used at this point, since it is not necessary to scan the line at full height of the image but only a region around the vertical position of the scan line. The region is defined by the width of the horizontal ball candidate (of course some extra area is searched). All ball candidates where vertically nothing is found are discarded. The vertical scan is performed in exactly the same way as the horizontal scan before.

4.3.5 Plausability Check

When both horizontal and vertical scan have been successful, finally a plausability check is performed. Does the vertical candidate intersect the horizontal line from which it originates? Is the bounding box close to quadratic? Of course, on the sides of the image the checks are adjusted accordingly. Those checks allow not to detect only perfect circles, but as well ellipses and similar shapes. However, a rectangle matching those checks will be detected as a ball. This is not a problem since it is highly unlikely that an orange rectangle will appear on the field.

4.3.6 Estimating Radius and Confidence Values

The horizontal scan lines are likely to intersect the ball out of center. That is why the width of the horizontal candidates is a bad guess for the diameter (and hence the radius). On the other hand, the vertical scan line position is defined in a way that it

should intersect the ball in the middle. For that reason, the radius is calculated by taking half the length of the vertical candidates and the horizontal width is ignored completely. By now, circles of different colors can be detected, as long as they appear in front of a background that is darker on the channel we use. For example, a green ball in front of a blue area could be detected. In order to choose the right, orange ball in such a situation a confidence value is defined: The average "orangeness" of five pixels inside the ball is calculated. The pixels are the center of the detected ball, and half the radius in the four axis-aligned directions.

4.3.7 Combining Ball candidates

A ball can and should be intersected by plural lines. This results into plural candidates for the same ball. The goal is to identify every ball only once, and in order to achieve that, the candidates need to be combined to only one single ball. This is done by defining a minimum distance between the center of balls. All candidates that are closer to each other are sorted into groups of ball candidates. Finally, for each group a single ball is calculated for representing the group. This is done by electing the median values for the center position (x, y) and the radius. For the certainty the highest value of all balls in the group is picked.

4.4 Communication

4.4.1 Setup

The detected balls are sent to the robot using an UART interface. The raspberry pi supports uart using pin 8 as the sending pin (TX) and pin 10 as the receiving pin (RX). Due to the realtime application of this project is important, an update to the robot needs to be sent 60 times per second. Since the ball detection algorithm can detect multiple balls, these need to be filtered. While a confidence value is calculated by the software on the Raspberry Pi, noise can lead to the wrong ball (one that does not actually exist) having a higher confidence value than the real ball. The robot will be responsible for the filtering, while the Raspberry Pi just sends all the balls that it detects if the number of balls is less than 10, or just the 10 balls with the best confidence if there are more balls detected. In any case the balls should be sorted by confidence before sending.

4.4.2 Definition of the Message

In our messages to the robot we want to send a number of detected balls since all the filtering will be done on the robots. In each message we send a list of balls with the size of 10 and the resolution of the image. The resolution of the image is a array with the size of 2 for the width and height of the image. Each value has a size of 16 bits since our resolution exceeds 256 pixels. For each ball we send the center position of the ball as two signed 16 bit integers. These values are signed because the center of the ball could be outside of the frame while only part of the ball is visible on the camera and thus have negative positions. A ball also includes its radius in pixels as an unsigned 8 bit integer and a confidence value which is also an unsigned 8 bit integer. The confidence is defined by the average value of the subtracted U-V channels at various positions within the ball. When sending 10 balls, the total message size is 64 Bytes. When sending data with 8N1 encoding and a baudrate of 115200 baud, we can transmit up to 180 messages per second to the robot using the UART interface.

```
1  #define BALLCOUNT 10
2  typedef struct _TransmitBallContainer
3  {
4    struct _ball
5    {
6      uint16_t pos[2]; // X positive right, Y positive down (in relative
           display pos. 0<=n<=10000)
7      uint8_t radius; // in realtive sensor area, 0=>no area, 10000=>area
           of entire screen
8      uint8_t confidence;
9    } balls[BALLCOUNT];
10   uint16_t res[2];
11 } TransmitBallContainer;
```

Figure 4.6: The definition of the message that will be sent to the robots.

4.5 Robot Programming

4.5.1 Sensor Input

The incoming date via the UART interface needs to be passed to the skills, specifically the "Get Ball" skill designed for the first part of the challenge, and the "Intercept Ball" skill designed for the second part of the challenge. Since the skills have access to the Sensor Input data, the messages should be written into that structure. Since the robot also has to do the filtering of the incoming ball position data, the UART receiver will pass the data to a ball filter first. This ball filter decides which of the received balls is the correct one and saves this into the sensor data structure. Since all sensor data will be logged, a small size is important to ensure that the storage device for the log is not filled too fast.

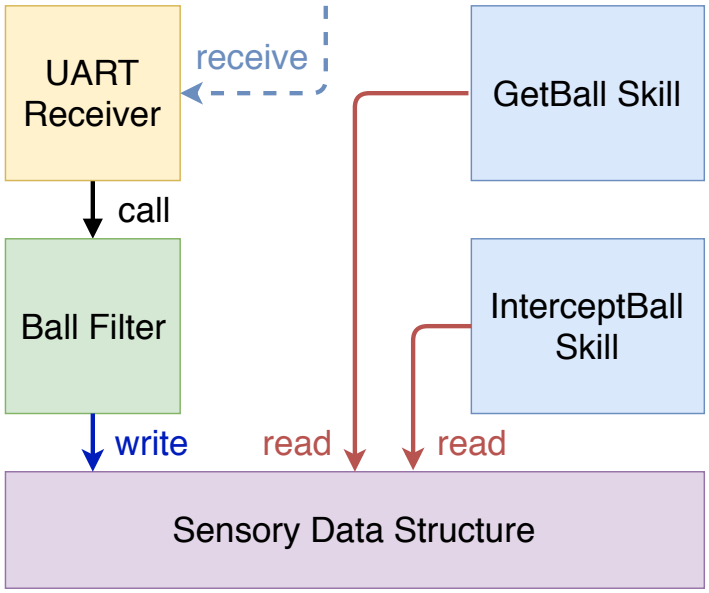


Figure 4.7: Data flow on the Robot

4.5.2 Procedure to Approach a Stationary Ball

The first stage of the technical challenge is to approach a stationary ball and stop with the ball touching the dribbler. To accomplish this, we use an internal state machine for the control of the robot. One of the major advantages of the approach presented here is, that it is not needed to map the ball to either world or relative

coordinates. The controlling makes use only of the position of the ball in the image. It consists of four states, as it can be seen in Figure 4.8. In the following, we will explain the states and which move constraints are set, as well as which commands are send to the robot and which sensors are used.

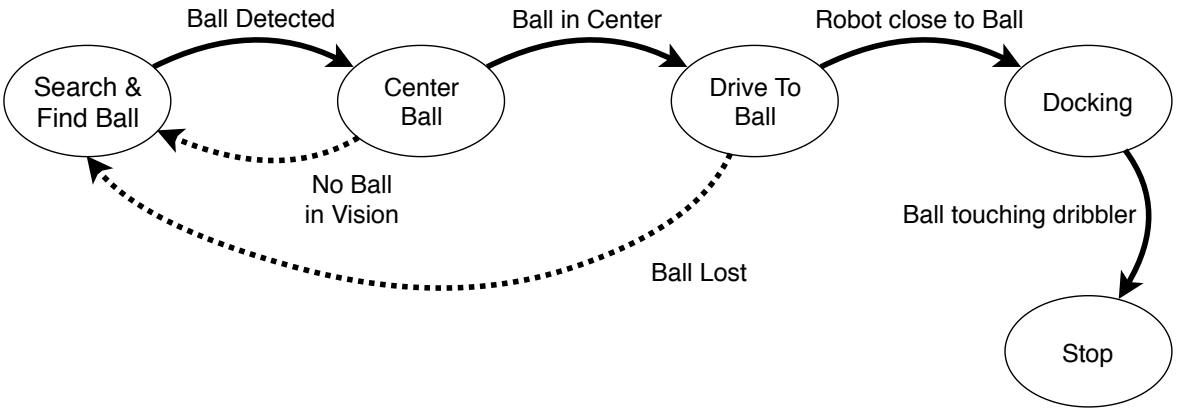
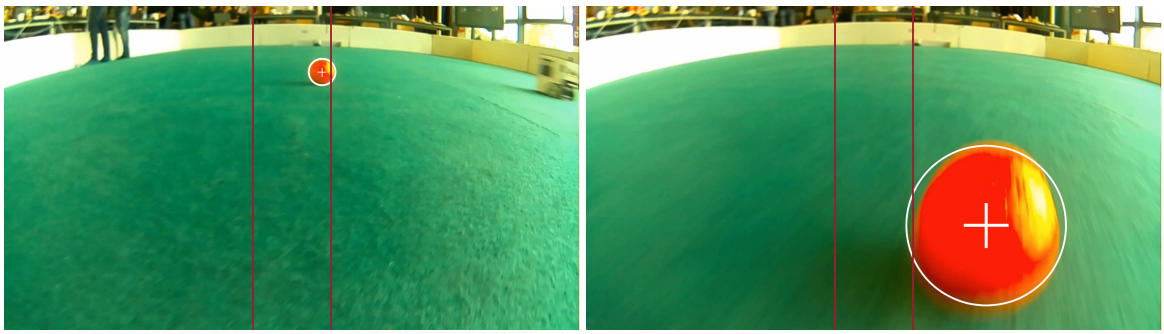


Figure 4.8: Stages for the robot control to collect a stationary Ball

State 1: Searching and Locating the Ball Since we use only one camera, we can only see what is in front of the robot. Hence, to find a ball that is located anywhere around the robot, the robot needs to turn. It turns until a ball is detected, then it switches to the next state.

State 2: Centering the Ball in the Robots Vision Field In this state, the goal is to correctly orientate the robot to the ball. To do so, we define a "deadzone" in which the center of the ball should be. The ball turns left and right (depending on where in the image the ball is located). For this, only the x coordinate is needed. As soon as the ball enters the "deadzone" (+/- center of the image), the rotation of the robot is stopped and the next state is entered. The purpose of the deadzone is, is to prevent the robot from overshooting because of hysteretic effects when braking. Moreover, since the deadzone is constant in the image domain, the needed rotational accuracy increases with the closeness of the ball to the robot (cf. Figure 4.9). If, at some point during this state, no ball is seen anymore, the robot goes back to the first state and restarts the search.



(a) Ball is far away, but in deadzone (b) Ball is close, so higher accuracy is needed

Figure 4.9: The deadzone concept for correctly orientating the robot

State 3: Drive to the Ball The next state is very simple, since it needs only a simple move forward of the robot in direction of the robot. If the ball goes out of center, the robot slightly turns until it is back inside the deadzone. The robot drives at a relatively high speed towards the ball. It must be avoided that the robot approaches the ball with a high velocity and risks to push it further away. Hence the "docking" state is entered when the robot is close to the ball.

State 4: "Docking" In the docking state, the robot drives at a slower speed to grab the ball without pushing it. As soon as the ball is closer than a certain distance, additional sensor information is needed to accurately grab the ball. The optimal case would be that the robot approaches the ball without pushing it at all. The infrared barrier at the dribbler unit is used to detect if the robot touches the ball. As soon as the infrared barrier is interrupted, the robot needs to stop and will enter the last state - stop.

State 5: Stop This state is used to make sure that the robot stops completely. At this point, the procedure has come to an end.

4.5.3 Procedure to Intercept a Moving Ball

The second stage of the challenge requires a different approach. The biggest difference is, that the ball is moving towards the robot and hence the robot should not drive towards the ball, because it might reflect the ball if it is too fast. The other difference is that the robot does not need to search the ball. Of course, the ball trajectory could be estimated and a general approach for both challenges could

be applied, but we decided to implement a procedure, where no further external information, such as the robots global position, is needed.

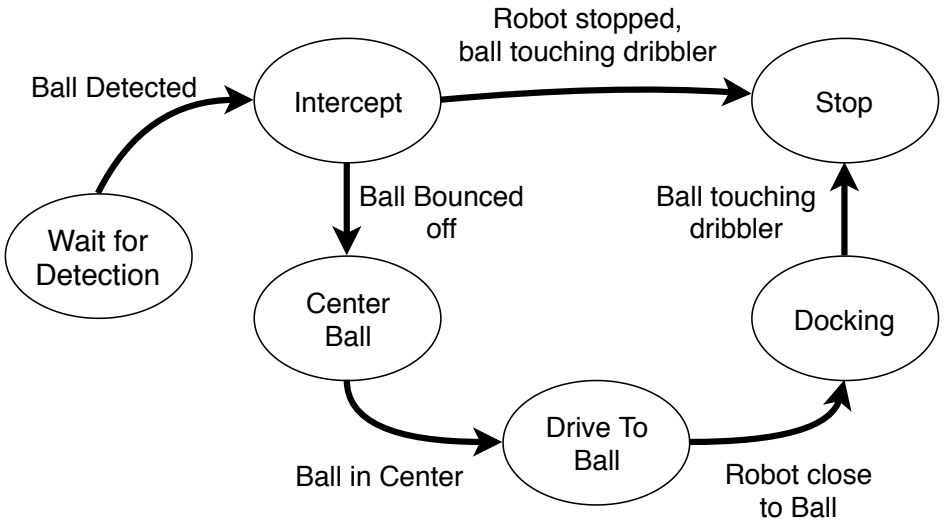


Figure 4.10: Stages for the robot control to collect a moving Ball

State 1: Wait Since the constraints of the challenge require the robot to face the ball at the beginning of the challenge, the robot waits for any detected ball in the first state.

State 2: Intercept As soon as the ball is detected, the robot tries to center the ball on the x axis of the image using sideways movements. Since the ball will be moving towards the robot, no forward movement is needed at this point. In addition to the sideways movement, the robot will be moving backwards to slow the relative speed of the ball to the robot to ease the receiving of the ball using the dribbler apparatus. This of course can be limited by a wall that is behind the robot. When to ball interrupts the light barrier and stays there, the challenge is completed and the robot will go into the stop state. If however the ball bounces off and the light barrier is briefly interrupted only to be clear again, the robot will try to watch the ball and detect weather the ball has left the frame to the left or right side of the screen. After that it goes into the center ball state.

State 3: Center Ball After the direction in which the ball has left the frame was saved by the Intercept state, the robot will rotate into that direction until the ball is

visible and then the robot will center the ball on the x axis of the frame and moves into the approach ball state.

State 4: Approach Ball This identical to the approach ball state in the stationary ball scenario. The robot moves towards the ball until the light barrier is interrupted and then goes into the stop state.

State 5: Docking The Docking state needs a slight modification compared to the one used in the first stage of the challenge. Additionally, the dribbler roll needs to be active, so that the ball will receive a backspin and stop its movement. Then, the robot needs to stop cautiously its own movement and the dribbler roll without losing the ball.

State 6: Stop This is the end state and ensures that the robot will take no further action.

4.6 Validation

The different parts of the project have been validated individually. For the ball detection, this has been done by applying it in different environments and examining the furthest distance where a ball is still detected. While in noisy surroundings false positives are retrieved, as soon as an actual ball is in the camera's field of view, the correct ball is chosen. With the setup, a ball up to five metres away from the robot still was detected. The UART communication has been tested first between two Raspberry Pis and was successful. When trying to connect the Raspberry Pi to the robot, however several errors occured. One of the problems we were able to identify is that sequences of bits with the value "0" longer than eight bit resulted into communication errors. The problem could be solved by sending only one ball and performing the filtering step on the Raspberry Pi. Finally, the Bot Skill was first executed in abbreviated forms, so that the robot simply stops completely as soon as a ball is detected. This allows to assume that the integration of the three components is successful. First complete runs of the task under technical challenge conditions however showed that fine-grained tuning still is needed. However, the tests showed that the approach can also be used during games in order to improve the reception of our own passes and the deflection and interception of the opposing

teams passes. This will provide a big advantage in the competition, since a better ball control allows for faster and more complex plays.

5 Conclusion and Future Work

During this project the fundamental techniques needed for the "Vision Blackout" technical challenge were developed. This includes a concept for an onboard vision system, featuring separate microprocessors for onboard vision and the robot control. For the ball detection, several approaches have been evaluated and a new approach, the "Scan Lines", has been implemented. This approach excels at high processing rates and thus can offer the robot information at high rates, which allows more precise reactions. Moreover, the communication between the microprocessors has been established and succesfully tested. For the two stages of the technical challenge, state machines have been designed and first tests have proven them to be correct.

While many challenges and hurdles were overcome during this project, there are additional challenges that need to be mastered in order to tune the implementation to perfection. One of these challenges is the usage of proper control theory when approaching and intercepting the ball to ensure smooth movement of the robot and thus a cleaner and less error prone capture of the ball at the front of the kicker. Additionally, a more robust movement algorithm that predicts the path of the ball using projection mapping and advanced predictive filtering can be implemented. The algorithm for ball detection is very dependant on hard edges on the ball to detect it properly. Fast moving balls can pose a challenge for this type of detection since motion blur is a big enemy for edge detection based algorithms. A future version of the algorithm can utilize a combination of different approaches like the Hough Transform and Blob detection based algorithms to assist the current implementation.

References

[1] *Robocup - Objective*. URL: https://www.robocup.org/objective (visited on 05/13/2019).

[2] *SSL-Vision Blackout Technical Challenge*. 2018.

[3] Dave Jones. *Pi Camera Documentation*. URL: https://picamera.readthedocs.io/.

[4] *RGB color solid cube*. URL: https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png.

[5] *HSL color solid cylinder*. URL: https://commons.wikimedia.org/wiki/File:HSL_color_solid_cylinder.png.

[6] *HSV color solid cylinder*. URL: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png.

[7] Douglas A. Kerr. "Chrominance Subsampling in Digital Images". In: (2012).

[8] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. *Machine vision*. Vol. 5. McGraw-Hill New York, 1995.

[9] James Bruce. "Realtime machine vision perception and prediction". Undergraduate thesis, School of Computer Science. Carnegie Mellon University, Pittsburgh, 2000.

[10] T Brown and Jack Koplowitz. "The weighted nearest neighbor rule for class dependent sample sizes (Corresp.)" In: *IEEE Transactions on Information Theory* 25.5 (1979), pp. 617–619.

[11] John Canny. "A computational approach to edge detection". In: *Readings in computer vision*. Elsevier, 1987, pp. 184–203.

[12] Judith MS Prewitt. "Object enhancement and extraction". In: *Picture processing and Psychopictorics* 10.1 (1970), pp. 15–19.

[13] David Marr and Ellen Hildreth. “Theory of edge detection”. In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207.1167 (1980), pp. 187–217.

[14] HK Yuen et al. “Comparative study of Hough transform methods for circle finding”. In: *Image and vision computing* 8.1 (1990), pp. 71–77.

[15] Oliver Birbach, Udo Frese, and Berthold Bäuml. “Realtime perception for catching a flying ball with a mobile humanoid”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5955–5962.

[16] Michael Pluhatsch. “Objekterkennung anhand von farbklassifizierten Integralbildern und der Kullback-Leibler Divergenz”. Bachelor Thesis. Freie Universität Berlin, July 2017.

[17] Jinzi Mao, David Mould, and Sriram Subramanian. “Background subtraction for realtime tracking of a tennis ball.” In: *VISAPP (2)*. Citeseer. 2007, pp. 427–434.

[18] Simon O’Keeffe and Rudi Villing. “A Benchmark Data Set and Evaluation of Deep Learning Architectures for Ball Detection in the RoboCup SPL”. In: *RoboCup 2017: Robot World Cup XXI*. Ed. by Hidehisa Akiyama et al. Cham: Springer International Publishing, 2018, pp. 398–409. ISBN: 978-3-030-00308-1.

[19] Jan Draegert. “Efficient Superpixel Creation in High-resolution Images by Applying a PLANT”. Master Thesis. Freie Universität Berlin, July 2017.

[20] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[21] Stefan Zickler et al. “Five Years of SSL-Vision–Impact and Development”. In: *Robot Soccer World Cup*. Springer. 2013, pp. 656–663.

[22] *OpenCV Documentation*. URL: `https://www.docs.opencv.org/2.4.13.7/`.

[23] Huimin Lu et al. “A robust omnidirectional vision sensor for soccer robots”. In: *Mechatronics* 21.2 (2011), pp. 373–389.