Bar-Ilan University, Faculty of Engineering
52900 Ramat-Gan, Israel

Team RoBIU

Team Description Paper for Humanoid KidSize League of RoboCup 2018

TAMAR VICLIZKI
SHAY YASHAR
EVYATAR NEZER
LIRAZ BENBENISHTI
DORON NAHARI
ALON MAR CHAIM

Academic Supervisor:
Dr. Kolberg Eli

Mentors Dr. Abramov Benjamin & Mr. Amsalem Rafi

December 9th, 2017

Contact Person: TAMAR VICLIZKI
E-mail: viclizkitamar@gmail.com

Abstract

Team RoBIU was founded in 2010. The team consists of undergraduate students from Bar-Ilan University, Faculty of Engineering. This paper presents an overview description of the hardware and software layer of the kidsize humanoid robots of RoBIU team. The following documents describe the robot's hardware specifications and a high level description of the various software algorithms, including real-time image processing, stabilization, sensors and camera based localization, debug features, robot agents inter-communication and high-level behaviours implementation.

1 Introduction

This paper describes the RoboCup Kid Size League team RoBIU from Bar-Ilan University. The team was founded in 2010 and this is the 6th year that the team is applying for participation in the KSL league. Each year the team is assembled with new undergraduate senior year computer engineering students, as a part of their final year project under the supervision of Dr. Eli Kolberg and the mentors Dr. Beni Abramov and Mr. Rafi Amsalem.
One purpose of the Robocup KSL project is to let the students experience a large-scale projects, which incorporates many challenges, such as strategic orientation, coping with deadlines, mediating between target groups and managing the development of software-intensive systems.
RoboCup 2018 would be a great setting to study our enhanced software and team performance.

Prior Performance In RoboCup Competitions

2014 - 2^{nd} Round Robin.
2013 - 1^{st} Round Robin.
2012 - Quarter finals.

Enhancements of the Robot's Compared to the Previous Year

In order to improve our robot's abilities, we will set a wider angle for head

tilt such that the ball can be seen even if it just touch the feet.
In addition, the new robot is higher, which gives it the ability to look farther in the field. It also has cleats for better grip when moving in the field.

The software was developed by the current team members, based on last year program. The program includes new algorithms with better performance in terms of runtime, accuracy and robustness. Moreover, we improved the robot's scanning and following algorithms.

We added features of white line, corners, and junctions identification in addition to distance to objects measurements.

These improvements do not use GPS or local- detection systems. They are based on gyroscope and camera that feed the input to localization algorithms.

In addition, the robot's code is a multi-threading code in order to enhance the operation of the different systems that work in parallel.

2 Hardware

The robot's hardware - Motors, Sensors and Specifications - is presented in the *"Robot Specifications"* document.

3 Software

Our robot's software integrates several components that combine a real time soccer playing robot. In order to do so, we designed a software that combines all the necessary functionalities into one multi-threaded program.

Our software, from design to implementation, was developed by team RoBIU without any use of software from other teams.

3.1 Multi-Threading

Multi-threaded environment let us deal with the robot's various moduls (such as brain, vision, localization, etc.) in order to be more efficient. We prefer to use multi-threading over multiprocessing so we could use the same memory space for all our software components, and in order to save the context-switch time. Our software consists of 4 main threads - Brain, Vision, Localization and Communication.

3.2 Artificial Intelligence

Artificial intelligence - AKA Brain - is the main component in our design. It is in charge of taking all important inputs from other modules, processing, understanding and deciding the next move. The brain implementation is based on a FSM (finite state machine), which computes the next state according to the current state and the various updated inputs (vision, localization, etc.).
We started by writing a simple FSM which knows how to find the ball and kick to the opponent's goal. Afterwards we gradually considered more and more factors such as Localization properties and communication between robots, in order to make the robot more intelligent.

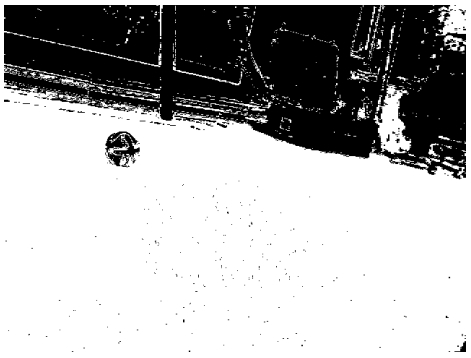
3.3 Vision

The Vision module is responsible for image processing. The main goal is to detect meaningful objects - ball, goal and white lines. (This year we will deal with corners as well)
The implementation uses some functions from the OpenCV image processing library [2].

Calibration Tool

Our design contains a separate tool that adjust our image-processing to the current environment colors. The tool "teaches" the robot how to define the green color spectrum and the white spectrum. The tool shows the user 2 images - the original image and an "only green"/"only white" image. The user clicks on the green pixels in the original image and the tool colors only these pixels in white in the "only green"/"only white" image, as can be seen in Fig. 1.

(a) Original Image



(b) "Only Green" Image

Figure 1: Calibration-Tool Example

Common Image Processing Techniques

HSV - We use the HSV (hue,saturation & value) image format[3] which is more adequate here than RGB format. RGB components of an object's color in a digital image are correlated to the amount of light hitting the object, and therefore with each other. Thus, image descriptions in terms of RGB components make object discrimination difficult. Instead, descriptions in terms of HSV are far more relevant.

Dynamic Threshold - Almost every image processing in our code starts with an "image segmentation". This is done in a very simple way, using the threshold method. The threshold uses the values that were calibrated with our calibration tool. By doing so, the threshold varies with every calibration, making it adaptive to a variety of lighting conditions, grass color, etc.

Erosion & Dilation After thresholding we use Erosion & Dilation to "fill holes" in the given B&W image. Erosion is done by applying a filter on the image that changes pixel color to white *iff* all the pixels that surrounds it are white. Dilation does the same action for black pixels. Combining these filters gives us the ability to "fill holes" and more.

Various algorithms have been tested regarding ball, goal and white-lines detection. Ultimately we've chosen the most efficient one in terms of fast computations and accuracy as described below.

We used Canny Algorithm(that use Sobel operator) in order to find edges in grayscale pictures. By using Canny algorithm we had to take care about:
-low error rate
-edge points should be localized
-single edge point response

Ball Detection

For ball identification we use HLT (Hough Line Transform) to find and erase lines. Here line deletion allows for decreasing software complexity. Since most of the contours in the image except the ball are made of straight lines, it helps to reduce the edges that are not a part of the ball contour. This method is effective only for RoboCup due to the unique object shape.

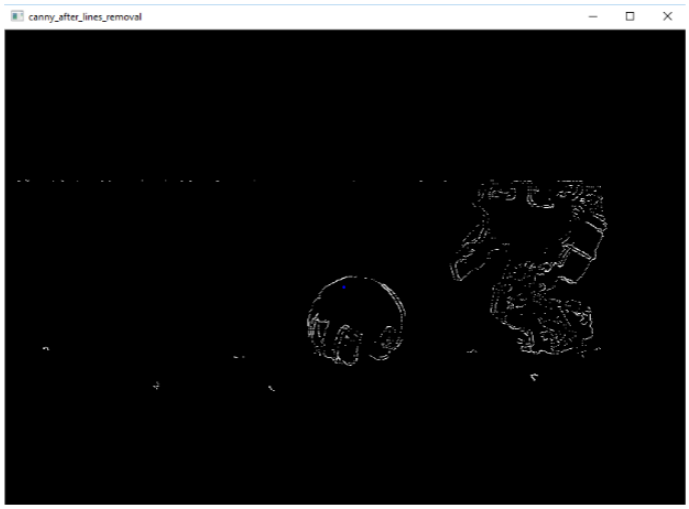


Figure 2: Image after line deletion

We can see in Fig. 2 that erasing lines meaning less edges without harming other detections.
We then use RANSAC method in order to find regions of interest (ROI).

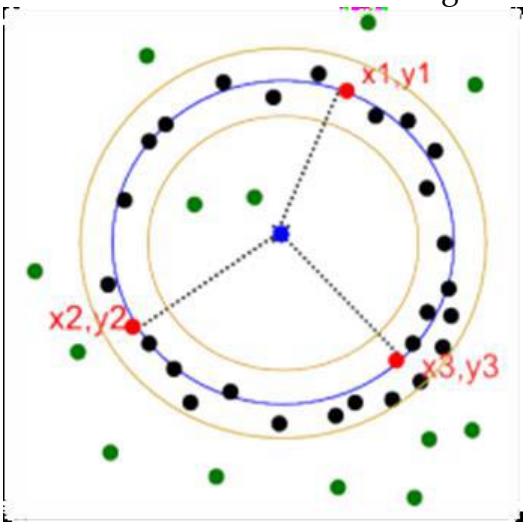


Figure 3: Supporting points for a circle shape

The algorithm will chose the circle that has the most number of supporting points according to circle fit of RANSAC (see Fig. 3). We modified the RANSAC method and added few steps in order to have more robust decision about the ball identification.

There are about 1000-1500 pixels in each image. It is a large data to handle in real time. It might cause identification of false circles due to the aliasing

of large number of points. Fig. 4 presents an example of identification of few circles after several iteration of RANSAC algorithm.

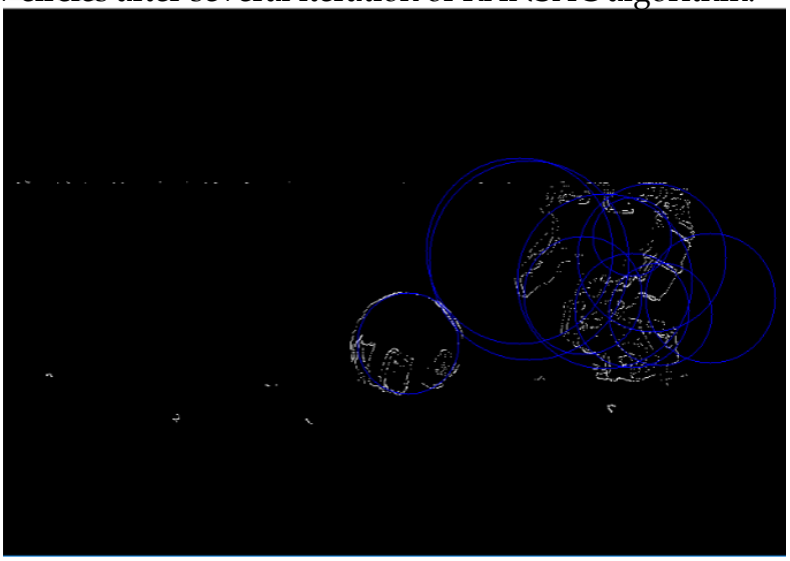


Figure 4: detection of circle shapes including false circles

It is clear that besides the ball there are several false circles.

Now the algorithm calculates the correlation between each circle color to the ball color which was defined in the calibration made beforehand (Fig. 5).

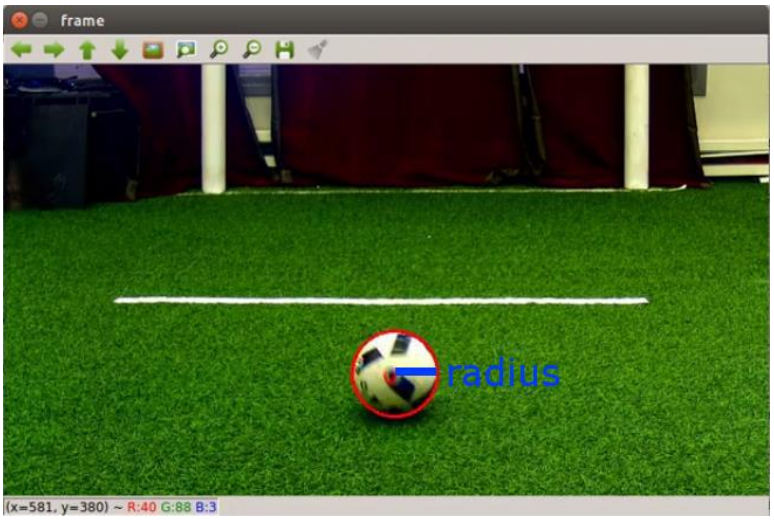


Figure 5: The robot detects the ball

Distance to ball calculation

After considering several methods like triangulation and linear approximated look-up table we decided to use the linear regression tool. 30 samples are used for building a 3rd degree polynomial function that produces the distance as a function of the ball radius.

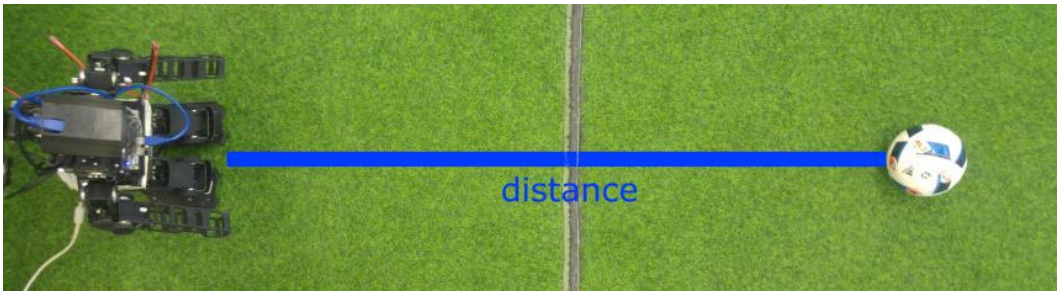


Figure 6: Distance to ball as was calculated by the robot

Scan and follow the ball

We used Gazebo simulator in order to detect the optimum scan method. It lead us to a sine scanning. It is performed by synchronizing 2 head motors for vertical and horizontal scanning (Fig. 7).

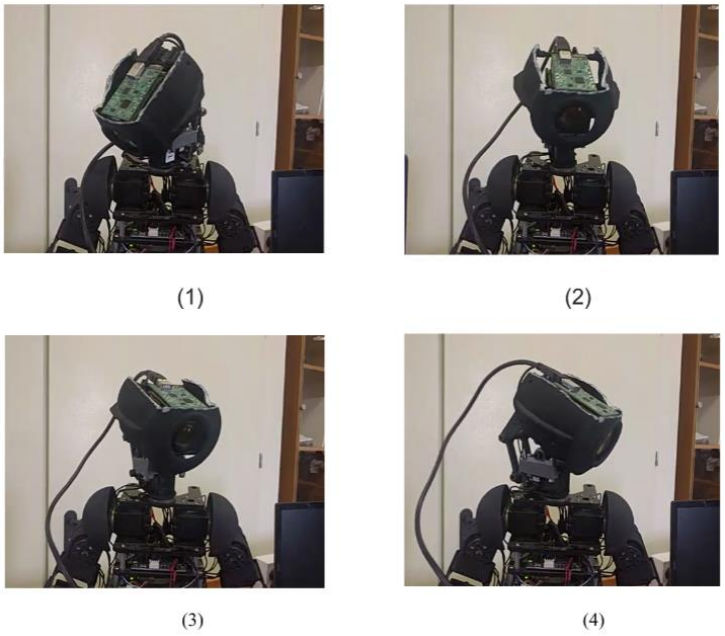


Figure 7: The robot turn his head

Goal Detection

The goal detection is based on finding objects, which are suspected to be the goal's posts, in the input image. Then, the algorithm selects the most relevant ones to be the posts (Fig. 8). If only one object was found, the algorithm will determine which one of the two posts the robot sees.

Object is suspected as a post if it satisfies the following terms:

1. **White:** First, we use a simple threshold function on the HSV transform of the given image (White is easy to recognize in the HSV transform image). We get a B&W image, in which only white objects are white.
2. **Vertical:** We perform a vertical erosion algorithm on the given image to remove any horizontal white objects from the image. Only vertical white objects are left.
3. **Rectangle-shaped:** We use OpenCVs minAreaRect to surround all these objects with minimum area rectangles. We check the ratio between the output rectangle and the white-object area, and we eliminate any rectangle that does not satisfy the threshold ratio.
4. **Straight-angled:** From the robot's eyes, the posts are orthogonal to the fields plane. We check that the rectangles angle is close to zero.
5. **Inter-edge ratio:** The post's shape is characterized by long vertical edge and a short horizontal edge. We eliminate any rectangle that does not meet this characteristic.

Full goal detection: We take the 2 largest candidates to be the posts:

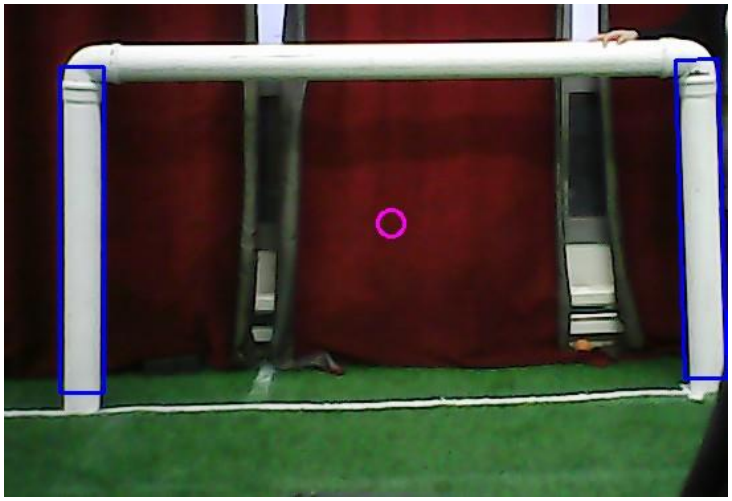


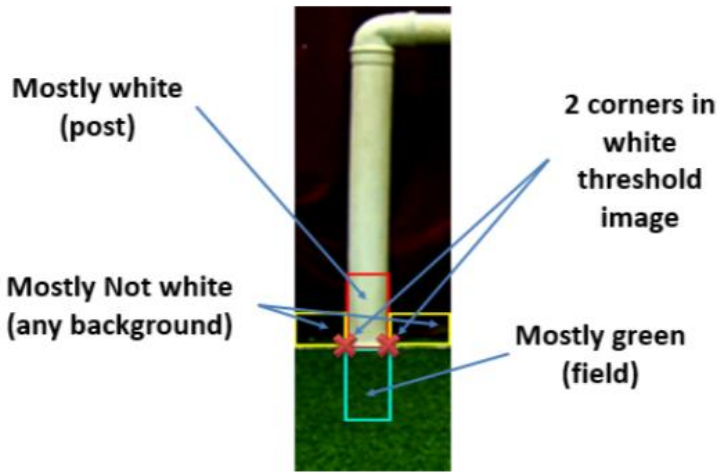
Figure 8: The robot detects the goal and its center

Single post detection: If only one object was found we need to determine which one of the post the robot sees (left or right).
We do it by detecting the crossbar. The algorithm detects the crossbar by performing horizontal erosion on the B&W image, in order to remove any vertical white object. With that being done, the algorithm counts white pixels from the left and the right of the post's top. The direction with the larger number of white pixels determine the location of the post, as presented in Fig. 9.

(a) The original image

(b) The crossbar is detected (colored in blue), and the algorithm detects that the post in the image is the left post.

Figure 9: Single post detection



Line Detection

For line detection the algorithm uses 2 matrices: one for detecting the field's area (grass), and one for holding the potential lines (the area that is colored white).
Then, the algorithm extracts the boundaries of the field from the first matrix, and creates a new matrix that will help to determine which parts are white according to the observations and insert black value to the remaining cells.

Next step is performing an AND operation between the white matrix and the boundaries matrix. Then, in the last step, HoughLines transform is used to detect the lines.

Corner Detection

L junction detection:

In Cartesian coordinate system a straight line can be represented by the equation:

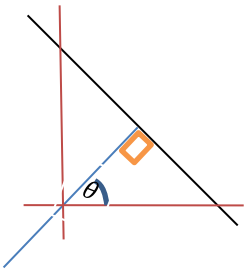
$$y = mx + n$$

Where m is the line slope and n is the Y Intercept (where the line crosses the Y axis). M and n are constant and (x, y) represents an arbitrary point of the line (line 1).

It is possible to convert to polar coordinate system such that (x, y) coordinates can be converted into (θ, ρ) coordinates.

In order to demonstrate the conversion, a line that pass through the axis and is perpendicular to the first line will be added (line 2). The equation of this line is:

$$y = -\frac{1}{m}x$$



We choose θ to be the angle between the perpendicular line and the positive x-axis. ρ is defined as the distance between the straight line and the origin (which is exactly the length of the perpendicular line between the origin and the intersection with the straight line). It is shown in the following drawing:

It can be shown that:

$$\theta = \arctan\left(-\frac{1}{m}\right) \rightarrow m = -\frac{\cos(\theta)}{\sin(\theta)}$$

and therefore: $y = mx + n \rightarrow y = -\frac{\cos(\theta)}{\sin(\theta)}x + n \rightarrow ysin(\theta) + xcos(\theta) = nsin(\theta)$

since $\rho = nsin(\theta)$:

$$\rho(\theta) = ysin(\theta) + xcos(\theta)$$

Running through $0 \leq \theta \leq 2\pi$ will produce the curve of $\rho(\theta)$.

(Fig. 10)

It can be done for all the detected lines. To every (x, y) there will be a joint point (θ_1, ρ_1). For a different straight line that cross the same (x, y) point, there will be a different polar coordinates joint point (θ_2, ρ_2). (Fig. 11)

We know that the lines are vertical <=> $\frac{\pi}{2} = |(\theta_1) - (\theta_2)|$

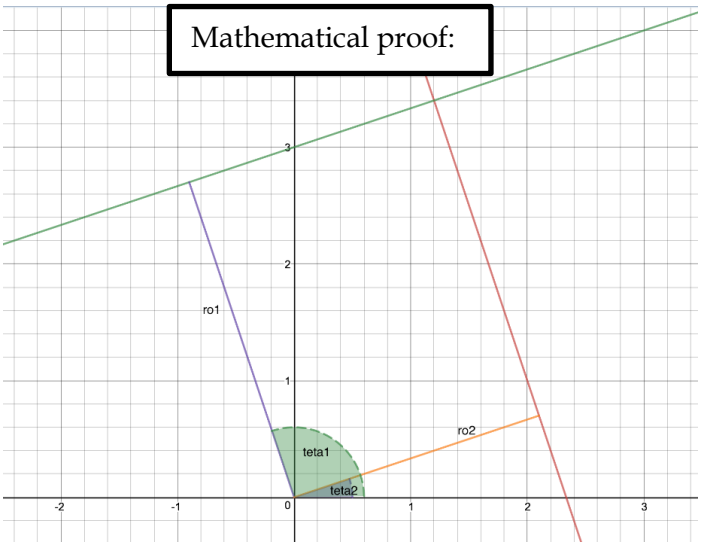


Figure 10: The process of the corner detection

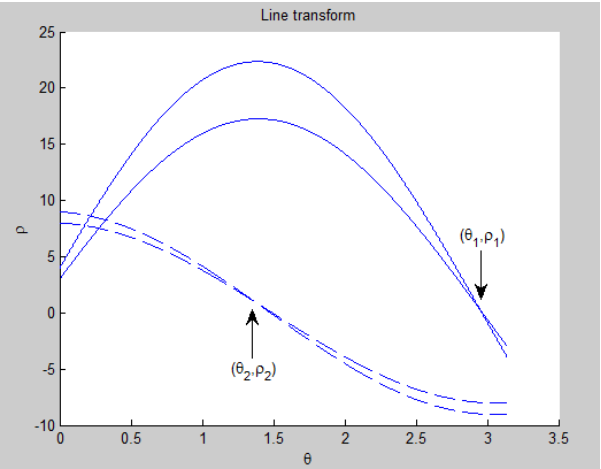


Figure 11: The joint points

T junction detection:

With the image processing algorithm, the program receives a list of L junctions. Some of these junctions might actually be T junctions (2 Two L-junctions could be actually a T junction). This leads to check of the L junctions for possible discover of T junctions.

For this purpose, first the algorithm takes two points that define line 1 and two points that define line 2:

line 1: (x1, y1) -> (x2, y2)

line 2: (x3, y3) -> (x4, y4)

Consequently their slope are respectively:

$m_1 = \frac{y_2 - y_1}{x_2 - x_1}$ $\quad m_2 = \frac{y_4 - y_3}{x_4 - x_3}$

Next step is to calculate their intersection using the following formula:

$$x_{intersection} = \frac{-x_3 * m_2 + y_3 + x_1 * m_1 - y_1}{m_1 - m_2}$$

$$y_{intersection} = (m_1 * x_{intersection} - x_1 * m_1 + y_1)$$

The algorithm keeps a certain threshold radius around the intersection (Fig. 12):

Apparently there are 5 scenarios:

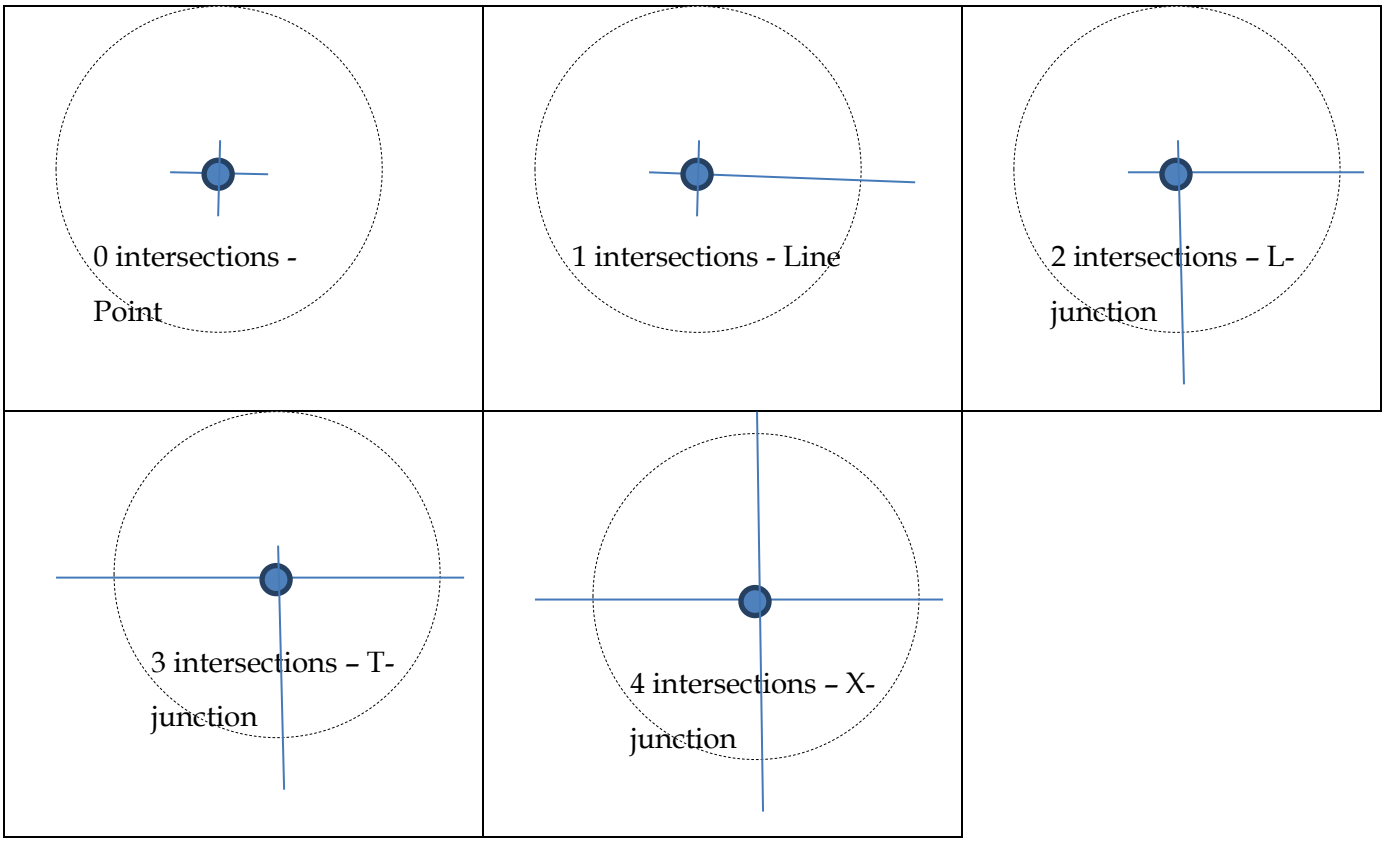


Figure 12: Five possible scenarios

The algorithm then checks how many points are outside the threshold radius. If there are 3 points outside the threshold radius, it is determined to be a T-junction.

3.4 Localization

Localization of the robot is one of the main features for its successful functionality. It means that the robot understands and decides where it is located and it consequences.
The problems related to the localization algorithms include improper image data, symmetric playing field - e.g. in case the robot sees only a white line, it won't be able to decide which line in which side of the court it is, if localization is not used.
In addition, localization is required for planning the robot's next move. According to its location, the robot can determine what should be its next move; whether to go to ball, come back to help the defense or move to the

center of the goal (in case it is a goalkeeper).

There are various and different ways to implement localization. Most of them are saving and using former data and current sampling in order to conclude the current localization. We decided to use "*Particle Filter*" in our project, which can help us to resolve the main following issues:
Position tracking - In this scenario, we want the robot to find its location, as the initial location of the robot is known, as well as its control data since it started. The Particle Filter can solve this problem simply, by only changing the initial distribution.
Initial localization - We want to be able to find the robot position in the field when there is no an initial position. We can use a uniform distribution as a starting point and rely on the "Particle Filter" to converge given enough parameters.
Kidnapped robot problem - This problem is the hardest. In this scenario, the robot can be 'teleported' at any time (e.g. the robot being moved by the referee) and the robot still need to be able to find its position after few iterations of the filter. To solve this scenario we use the Monte-Carlo Particle Filter.
As an input for the localization, we mainly use vision along with the obstacles the robot identified so far. In addition, we take an advantage of the *Gyro*, which enables us to configure in which directions the robot turned and walked.

Parenthetically, we should take into consideration that the data on which the localization algorithm relies, namely Gyro and image processing, is not clean as it contains sample noise and other erroneous data. Consequently, the resolution of the robot's location will need to be a "wise decision", relying on a high probability. Another parameter we took into consideration is the type of environment. In a static environment, the one of which the Particle filter was developed for, the only changing variable is the robot's position. However, in our case, there are other moving objects (e.g. other robots). Despite of this fact, we can still consider our environment static since we can detect the field (a static variable) even though there are objects on it using image processing. Thus, we can still use the Particle Filter.

4 Conclusions

In our document we've introduced the hardware structure and software design of our robots. Although it will be the 6th year that Bar-Ilan University will participate in the competition, all the team members have changed (except of the mentors), so this will be our first chance to take part in RoboCup. We look forward to participate in the RoboCup competition this year, and are determined to play as worthy competitors.

References

[1] Robotis Product Information, http://www.robotis.com.

[2] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, October, 2008.

[3] R. Gonzalez and R. Woods, *Digital Image Processing*, Third Edition, Pearson Education, 2008.

[4] S. Thrun, W. Bugard and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.