Fast computational processing for mobile robots' self-localization

Hélder Ribeiro, Pedro Silva, Ricardo Roriz, Tiago Maia, Rui Saraiva, Gil Lopes and A.Fernando Ribeiro
Laboratório de Automação e Robótica, Grupo de Controlo Automação e Robótica
Department of Industrial Electronics, University of Minho
Azurém, Guimarães, Portugal
{ a58795, a68541, a68536, a57126, a58783}@alunos.uminho.pt, {gil, fernando}@dei.uminho.pt

Abstract—**This paper intends to present a different approach to solve the Self-Localization problem regarding a RoboCup's Middle Size League game, developed by MINHO team researchers. The method uses white field markings as key points, to compute the position with least error, creating an error-based graphic where the minimum corresponds to the real position, that are computed by comparing the key (line) points with a precomputed set of values for each position. This approach allows a very fast local and global localization calculation, allowing the global localization to be used more often, while driving the estimate to its real value. Differently from the majority of other teams in this league, it was important to come up with a new and improved method to solve the traditional slow Self-Localization problem.**

Keywords—RoboCup; MSL; Middle Size League; MINHO team; Self-Localization; Localization;

I. Introduction

MINHO team started a robotic football team in 1997 and has been participating on RoboCup scientific challenge since 1999 making improvements on their platforms and software. In 2011 that development paused and returned in 2014. The restart consisted in rebuilding both hardware and software, and there was an urgent need to improve the robots self-localization technique, and to push the development of a new method. Regarding the RoboCup MSL specific application autonomous robots need to know their position on the field (the world), to be able to move to a certain position, and to kick towards a certain direction or to perform high level agent coordination, but that implies the need of the existence of a method that allow the robot to self-localize, only using local on-board sensors. The robustness, processing time and false positives in the line point detection are a big concern, and all of them model the structure and procedure of the method. At the moment the majority of teams use the self-localization algorithm created by Brainstormers Tribots [1]. The method here presented tries to improve the computational time using a rather different approach in the error calculation procedure. The method uses a precomputed set of meaningful distances of every possible position in the 20x14m field area (standard 18x12 plus 1 meter all around the field, resulting in a 20x14m world), with a 10x10cm resolution. Then, given the robot's true orientation, the position with the least error is computed using an error modelling function, coming up with the true position of the robot on the field. In Section II, a tracking of the league's development is given, while in Section III the imaging solution (also a standard in the league) and the method used in the search for interest points in the image, is presented. Section IV presents the method itself, explaining the world view from a certain point in the field, and the error calculation procedure. Section V explains the two different methods for acquiring the robot's true heading, in relation to a known reference, one by hardware (with its complementary software) and the other by software. Section VI addresses the results achieved during the research and application of the method, concluding this work.

II. League's Development

The league evolved rapidly throughout the years, accomplishing new challenges and tasks, complying with the current advances of modern day computing technologies. With new camera technologies, new computers and better communications, the imaging quality, the processing power and the information sharing velocity has been greatly improved. Due to these facts, the league stepped up new challenges to the teams, making changes in the rules and in the composition of the field. As the global system is faster, the necessity of having robust, effective and fast algorithms (to accomplish different tasks) to fit the "processing time window" is urging. Regarding the field layout, it is now larger, with 18x12 meters playable area, only with the standard white line markings. Taking into account that the robot's catadioptric camera system only covers about 4 meters radius of field area, there is a lot of information missing, giving a lot more importance to team communication and agent coordination, to accomplish team and tactical objectives of the game. Stated that the league, and the whole game, is evolving fast, the development of the present method, represents the evolution of a well-settled algorithm, to improve, at least, the computational time involved in the self-localization process.

III. Imaging Solution and Points of Interest

A. Catadioptric Sensor Setup

When it comes to imaging, there is a standard in the league, to use a catadioptric sensor, which is obtained by pairing a catadioptric mirror and a camera, with various technologies, output types and price ranges. The catadioptric mirror [2] [3] was developed specifically for this application, using simulation tools to achieve best performances.

The image provided by this imaging setup arrives at a frequency of 30Hz, with a cropped resolution of 480x480, in YUV 411 format. Using camera's setup tool, parameters can change setup to overcome some lighting problems, achieving a good and stable image quality that is further ahead used in the calibration process, of world parameters (like ball and line size versus distance) and colour labelling through colour segmentation.

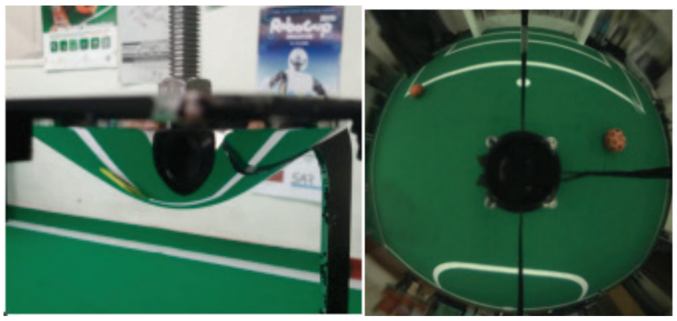
Figure 1 - Catadioptric Mirror and image captured by the catadioptric sensor

B. Image Labelling and line point extraction

The colours in the image are segmented using a 16MB YUV Look up Table [4] that is stored in memory. During the calibration procedure, the RGB space is labelled using the YUV colour space, allowing to correctly identify the nature of each pixel. To extract features used in the self-localization algorithm, instead of analysing all of the 230400 pixels, performing Hough-Transforms or other line detecting algorithms that involve Canny-Edge detector, scan lines are used. For identifying the lines, using key line points, 72 radial scan lines (spaced by 5º) and 72 spiral scan lines [5] (36 in one direction, 36 in the opposite direction, spaced by 10º) cover a large and meaningful area of the image, using only 63378 pixels, which are 27.5% of the image. Inspired on the ASML Falcons this team decided to use spiral scan lines because they guarantee that point are found when a robot is on top of a line, while the radial lines might not see them. This searching process usually takes 5 milliseconds to run, under an Intel Pentium 3805U, leaving 25 milliseconds in the "processing time window", to be occupied by other processes. After detecting field-line transitions, it is possible to analyse the detected line points in the image.

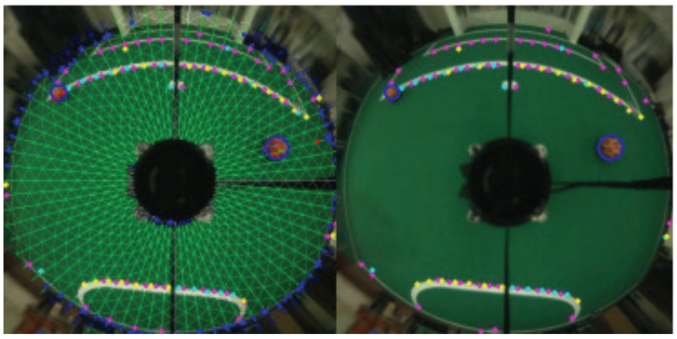
Figure 2 - Detection of interest points using radial and spiral scan lines.

The field shown in this example is half the size of an official MSL field. Each pixel is then mapped using a non-linear transformation, specific for each catadioptric sensor setup that is calibrated beforehand. The proposed method already brings the advantage that no line points need to be discarded, because, when using the method that is most widely used in the league, there is situations when points have to be discarded due to the excessive increase in processing time. Despite the fact that, in the proposed method, the processing time also increases with the number of detected line points, the increase is not significant, as the processing time per line point is very small. In the next section, first it is explained the field model built to compute the error, then, it is presented the method to compute the error, given a certain set of line points, in a certain time step.

IV. CALCULATING THE ROBOT'S POSITION

As any other self-localization method, it is required to precompute a "map", or to say, a point of view from a localization point. Given the method proposed by Tribots, their point of view is the nearest distance to the closest white marking. Instead, the proposed method defines a set of distances, acquired by the same radial and spiral scan lines. Each radial scan line can provide up to 4 distances, given the maximum distance that the catadioptric sensor can detect, and each spiral scan line can provide only one distance. This distances are the distances where the scan lines encounter white line markings. Three subsections are presented here, as they are the core procedures to obtain the robot localization on the field.

A. Building the Field Map

As briefly explained in the previous chapter, in order to be possible the use this algorithm, it was necessary to create a virtual field model capable of carrying out detection point operations. The model has been designed to match as much as possible the software that will run in the robot in real time. So it was created a model of the field, fully configurable using the Processing development IDE. This model is capable of detecting the line intersection points equally to that described in section III, i.e. trough 72 radial scan lines (spaced by 5º) and 72 spiral scan lines (36 in one direction, 36 in the opposite direction, spaced by 10º). It is also introduced in the model an offset from the central point of search, with the radius corresponding to the space occupied by the robot in order to create a closer approximation to reality. First the model performs a search for intersection points using the radial lines. It consists of a transition from the RGB color (0,255,0) corresponding to the field to a RGB color (255,255,255) corresponding to the lines. For that radial line creation and search is used the following equations:

$$x = x_i + Inc_{point} * \cos(Angle * \pi/180) \tag{1}$$

$$x = x_i + Inc_{point} * \cos(Angle * \pi/180) \tag{2}$$

Where:

- x_i and y_i make the central search point.
- x and y make the next search point.
- Inc_{point} define the distance to next search point
- Angle define the angle of the radial (spaced by 5º)

After this, the search is done by spiral lines using the following equations:

$$x = x_i + MASK*\cos(Spiral_{Angle} * \pi/180) \quad (3)$$

$$y = y_i + MASK*\sin(Spiral_{Angle} * \pi/180) \quad (4)$$

Where:

- x_i and y_i make the central search point.
- x and y make the next search point.
- MASK simulates the robot radius.
- $Spiral_{Angle}$ defines the angle of the next spiral point

In both search methods it is defined a search distance limit for each one of the lines, which is the maximum robot visualization distance as shown in Figure 3. Important to notice that the dimensions of this field model corresponds to half of the official field size.

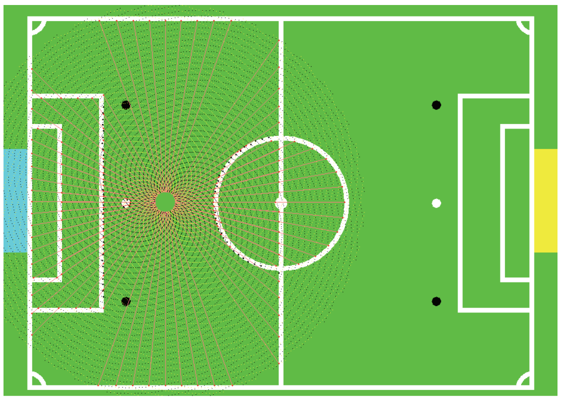
Figure 3 - Virtual points' extraction

In order to get all the intersection points a full scan of the field is carried out, with increments of 10cm. At the end, a file is created with all these values from the radial and spiral lines properly divided, in order to facilitate its use in the error computing method described in the following section.

B. *Computing and Modelling the Error*

At the beginning of the error computation algorithm the previously created file is read into memory and stored in the form of structures to facilitate and increase calculation speed, between the real points acquired by the robot and the ones contained in the file. For this comparison, two fundamental aspects are taken into account. One is that the virtual part is more accurate in detecting the interceptions points in comparison to the segmentation algorithm in real time, i.e. for each of the lines used in the virtual search there is a high probability of detecting at least one transition. But, in real time on the robot vision algorithm, the detected transition points will be lower. This occurs because the color segmentation algorithm is not perfect due to brightness changes in the field or the imperfections in the mirror, or even due to dynamic obstacles present on the playing field (other robots or humans), not all the search lines will return transition points. So the error computation algorithm ignores rows that do not meet transition points. Another very important aspect to consider is the confidence that is given to the distance calculated from the transition point to the robot center image. The transition points detected in shorter distances will have higher weight in the calculation error than points detected at a larger distance from the center of the image (center of the robot). The weight assigned to each comparison point is calculated based on the following equation and graphic:

$$Weight = 1-(c^2)/(c^2+e^2) \quad (5)$$

Where e is the error and c is the value of e where the weight becomes constant.

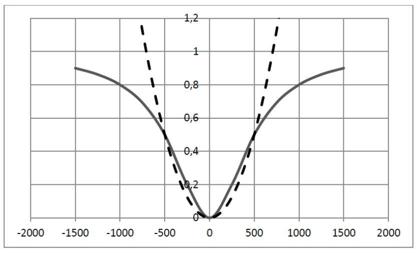

Figure 4 - Error modelling function

After running through the positions that are being searched, carrying out a local or global search, the current position of the robot is the one that minimizes the error term, being the error term, the accumulation of the error between the ideal (model) distances, and the distances provided by the detected line points. After an estimation from the vision system, it is necessary to fuse it with hardware data, preferably, odometry.

C. *Correcting the estimates using Kalman Filtering*

Kalman Filtering [6] [7] is for sure one of the most used prediction and correction mechanisms, in the whole field of engineering. Its capabilities to predict the state of a system, given its past and its mathematical model, and correct that estimate using sensors feedback, makes it one of the most famous methods in engineering for tracking, prediction, even used in airplanes and their missiles to track and chase their targets, based on their measures on it and the mathematical model of a moving object.

Two individual Kalman Filters must be computed, then a merging Kalman Filter enters in place, building an estimation of the robot position that is based both on software (vision) and hardware (odometry). First, the separate Kalman Filters shall be presented, concluding this subsection with the fusion algorithm for the individual Kalman Filters.

1) *Individual Filter to Correct Vision Estimates*

When the vision system outputs an estimate of the robot's current position, it has to be corrected before it is merged with the odometry. Given the output sent to the omnidirectional motor controller of the robot, applying the omnidirectional mathematic model, an estimate of the position of the robot relative to its previous position is computed. Then, using standard Kalman Filtering equations, the vision estimate (that acts as the sensor component) corrects the estimate, yielding a better estimative to start.

2) Individual Filter to Correct Odometry Estimates

In the same manner that the vision estimate is the sensed correcting component, the odometry also is the sensing component. Using the communication lines to the hardware modules, one can retrieve the values given by the encoders, performing matrix calculations, converting motor velocity (given by encoder ticks versus time) into robot's angular and linear velocities, having also its direction of movement. Again, using the omnidirectional mathematical movements and the inputs given to the motor controller, it is possible to predict the state of the system, relative to the previous position and then correcting it with the odometry measures. This individual Kalman Filtering procedure, assures that the values input to the fusion algorithm are the best ones possible, as applying Kalman Filtering yields better results than any of the sources alone, meaning that, the output of the filter is more reliable and trustworthy than the output of the mathematical model or the sensors (vision and odometry) alone.

To better comprehend the process, it is possible to summarize the previously described methods using the following diagram.

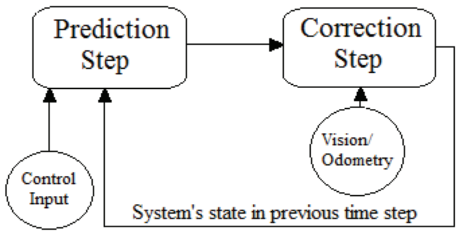

Figure 5 - Individual Kalman Filters procedure

3) Fusion Algorithm to provide Final Estimate

After improving individual estimates' performance with standard Kalman Filters, it is time to fuse their estimates, in order to yield a more robust, stable and accurate estimate. There are several methods to perform the fusion of two stable measures, like a standard weighted average, an Information Filter, etc. The fusion is carried out using a Decentralized Kalman Filter [8] (DKF). A Decentralized Kalman Filter consists of a network of nodes in which the merging/fusion process occurs in each node, sharing the information with the neighbor nodes. This allows the computational stress of a centralized fusion processor to be eliminated.

The sensing node performs the local (individual) Kalman Filter and shares the information with the other nodes, all of them also performing fusion algorithms. After that, it assimilates the information received by the neighbors, producing and improved local estimate, using both local and global information. This architecture brings the modularity concept to the table, regarding the Kalman Filtering computational "scene", given the fact that no a priori knowledge is needed and the local estimates are corrected using global information. Another improvement is in the robustness of the system, where the performance of the system does not depend on any of the processors, but on the communication link instead, being the system flexible to loss or addition of nodes, since there is no need to know the configuration of the network, only its state. The DKF is represented by the following equations, indexed by node i.

First, one should compute both the state error information and the covariance error information.

$$e_i(k) = P_i(k)^{-1}X_i(k)-P_i(k-1)^{-1}X_i(k-1) \qquad (6)$$

$$E_i(k) = P_i(k)^{-1}-P_i(k-1)^{-1} \qquad (7)$$

Then, using the previous equations one can compute the global (fused/merged) covariance (8) and state (9). As the sensors are independent and do not interfere with each other, the process covariance variable Q, should be near zero, but never zero.

$$P(k) = (P(k-1)^{-1}+\Sigma(E_i))^{-1} \qquad (8)$$

$$X(k) = P(k)(P(k-1)^{-1}X(k-1) + \Sigma(e_i)) \qquad (9)$$

With the application of the Kalman Filtering, the estimates become a lot more stable, using only simple mathematic operations, not consuming much processing time, yielding a rather good estimate about the localization of the robot.

V. Finding Robot's Orientation

The robot's orientation is a major component of the proposed method, as the method capabilities rely on the quality of estimation of the robot's heading direction, to estimate the robot's position. In this Section, two different methods are presented, complementary if needed, to estimate the robot's heading in the field. As described in the team description paper [9], every MINHO team robot is equipped with a 9 Degrees of Freedom IMU – Inertial Measurement Unit – that provides orientation, Yaw, Pitch and Roll. This four values are used to achieve the best orientation possible, computing the values using a DCM – Direction Cosine Matrix- algorithm [10], while using its other features to help improving the self-localization algorithm. The Yaw component is very important to correct and improve the value given by the compass, being also used to detect collisions with the robot, together with the Pitch and Roll components.

A. Robot orientation using an Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is required for the proper operation of the localization algorithm, as described in the previous Section. Although you can get the robot's heading using only a compass, it was decided to use an IMU for two main reasons. First, for its ability to adapt to surrounding magnetic fields, secondly, due to the possibility of detecting collisions between robots. The IMU uses an implementation of Direction Cosine Matrix (DCM). The motivation for using DCM, was the need for greater stability when obtaining XYZ values. Without the need to go into further explanations and details, as it is not the main focus of this work, the basics of this algorithm will be explained. Essentially what DCM does, is to represent the orientation of the robot in relation to the orientation of the Earth, using the following rotation matrix:

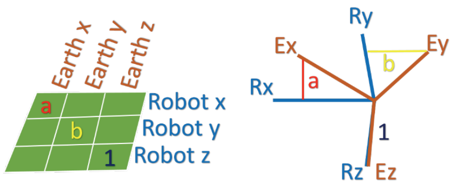

Figure 6 - DCM Rotation Matrix

It is possible to transform a system vector into another system, by multiplying it by the rotation matrix. The reverse could also be achieved simply by multiplying with the inverse matrix, taking advantage of the rotation matrix key properties, i.e., its orthogonality. After the calculations and conversions of matrices, a direct relationship between the created DCM and the Euler angles is given, using the following equation.

$$R = \begin{bmatrix} \cos\theta\cos\psi & \sin\theta\sin\phi\cos\psi & \sin\theta\cos\phi\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \sin\theta\cos\phi\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \quad (10)$$

Where:

- ϕ is the angle between the x axis and the N axis.
- θ is the angle between the z axis and the Z axis.
- ψ is the angle between the N axis and the X axis

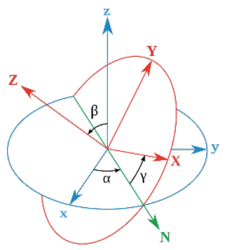

Figure 7 - Proper Euler angles representing rotations about axis z, N and Z.

Using the Yaw, Pitch and Roll data gathered by the IMU (X, Y and Z), one can estimate the position of the platform in relation to the ground plane, knowing if the robot is titled in a certain axis. As described in Section IV, the odometry position estimation is used to complement the self-localization algorithm, providing additional information to it. When two robots collide, usually it is the case when the obstacle detection and avoidance failed, and the robots will still be driving (wheels moving). At least one of the motors will cease to make contact with the ground, becoming a motor with free-spinning or skidding motion, introducing large errors to the odometry estimation process.

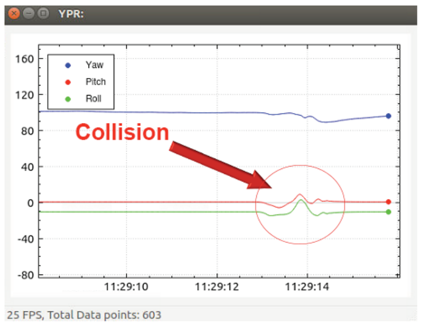

Figure 8 - Collision detection using the IMU

So, this feature provided by the IMU and the transformation algorithm, allows to halt the fusion algorithm that merges the vision and odometry estimates, only considering the vision estimates, not taking into account the errors introduced by the odometry estimate, when a collision happens. This helps the fact that, the robot will not lose the position tracking, and there is no need to perform a global localization, continuing to correct and to self-localize using only local searching patterns.

B. *Robot's orientation using Histograms*

Despite the fact that the IMU and the algorithm described before take into account external magnetic fields, avoiding reading errors due to the presence of other magnetic fields, there is always the necessity to have a complementary system, to provide the most important information, in this case, using reliable imaging and feature extraction algorithms. One of the most important achievements of the older era of MINHO team, was to develop an algorithm that provide the robot's orientation, with the detected line points, using histograms [11].

First, a histogram is built, counting the number of line points detected in a certain direction, vertically and horizontally. In order to calculate the orientation of the robot, the histogram is rotated from θ−40º to θ+40º, being θ is the last known orientation, and 40º the maximum rotation possible between frames. Verifying the histogram maximum value (peak value) for each rotation, the angle in which the histogram's value is the maximum one, summing both vertical and horizontal histograms' values, is the new orientation.

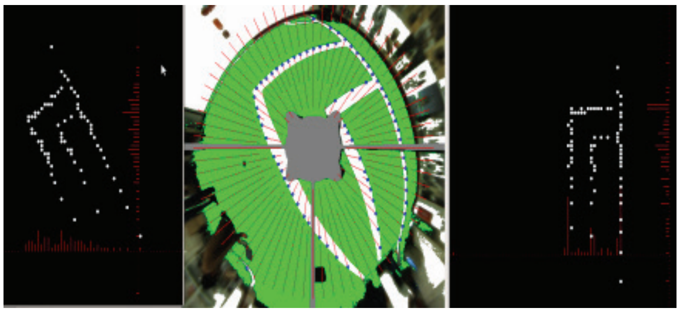

Figure 9 - Robot's orientation using Histograms example.

A graphic representation of the rotation of the histograms and the values obtained, show the maximum values and the new orientation.

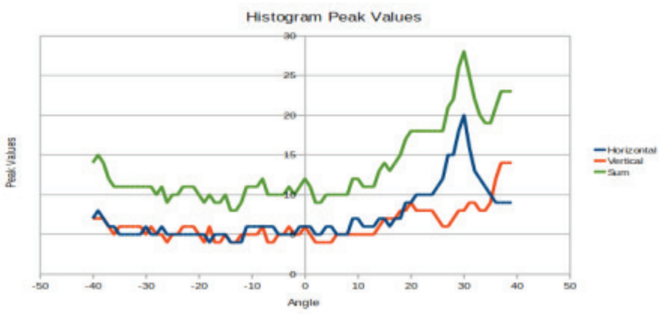

Figure 10 - Graphical representation of rotation results and maximum value.

As shown, the new orientation value has a very pronounced maximum, being set at 30º. This method allows to compute the new orientation between frames, using already gathered data, and simple numeric operations and comparisons.

VI. Discussion and Results

The main objective of this work was to improve the computational efficiency and speed of the Self-Localization algorithm, used in all RoboCup's MSL Robots, while trying to keep the robustness and accuracy of the existent methods. It was also presented a fusion algorithm using a Decentralized Kalman Filter, in order to perform fusion of software and hardware sensors, yielding smooth and accurate estimates of the robot's position, while preventing local erroneous estimates by any of the sources of the fusion algorithm. The application of Kalman Filtering in sensor fusion is widely used, proven to be a very robust, efficient and versatile method, performing complex filtering with simple mathematical operations, allowing to compute three Kalman Filters, without disturbing the "processing time window". The fusion algorithm also brings an improvement to the standard sensor fusion algorithms used, while, using an Inertial Measurement Unit brought other benefits, that were not possible before and also, were not used in any platform in the league. The proposed method achieved the task of reducing the algorithm computational time, reducing dramatically the global localization processing time to only 150 milliseconds, regarding an Official RoboCup MSL field. When performing local searching, in an area of $4m^2$ around the centre of the robot, the proposed method only takes 3 to 4 milliseconds. The use of two different methods to acquire the robot's orientation, one by hardware and the other by software, makes the task of estimating the robot's heading very efficient and trustworthy, while taking advantage of other capabilities of the IMU, like detecting collisions in order to eliminate the noise introduced by the odometry in those situations, when calculating the position of the robot. Although the method assures a very accurate and fast localization estimate, the identification of line points needs to be carried out correctly, which represents a down-side when comparing to the most common method in the league.

Acknowledgment

This work was developed at the Laboratório de Automação e Robótica by MINHO team´s researching and developing team, at University of Minho, under the supervision of Professor A. Fernando Ribeiro and A. Gil Lopes. The knowledge exchanging between the RoboCup's MSL teams and community contributed greatly for the development of this work. This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.

References

[1] Martin Lauer, Sascha Lange, and Martin Riedmiller, "Calculating the Perfect Match: an Efficient and Accurate Approach for Robot Self-Localization", In A. Bredenfeld, A. Jacoff, I. Noda and Y. Takahashi, editors, RoboCup 2005: Robot Soccer World Cup IX, LNCS. Springer, 2005.

[2] Gil Lopes, Fernando Ribeiro, Nino Pereira, "Catadioptric system optimisation for omnidirectional RoboCup MSL robots", T. Röfer et al. (Eds.): RoboCup 2011, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7416 LNCS, pp. 318-328. Springer-Verlag Berlin Heidelberg 2012.

[3] P. Lima, A. Bonarini, C. Machado, F. M. Marchese, C. Marques, F. Ribeiro, D. G. Sorrenti, "Omni-Directional Catadioptric Vision for Soccer Robots", Special Issue of the Robotics and Autonomous Systems Journal, Elsevier, Amesterdão, Volume 36, nº 2, 31 de Agosto de 2001.

[4] Gourab Sen Gupta and Donald Bailey, "Discrete YUV look-up tables for fast colour segmentation for robotic applications", IEEE, Conferenced in May 2008.

[5] Jaap Vos, et al, "Team Description Paper", ASML Falcons, Veldhoven, The Netherlands, 2016.

[6] Kalman, R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems", Transaction of the ASME--Journal of Basic Engineering, pp. 35-45 (March 1960).

[7] Greg Welch and Gary Bishop,"An Introduction to the Kalman Filter", Department of Computer Science University of North Carolina at Chapel Hill (2001&2006).

[8] Z. Hidayat, R. Babuska, B. De Schutter, and A. Nunez : "Decentralized Kalman filter comparison for distributed-parameter systems: A case study for a 1D heat conduction process" (2011).

[9] F. Ribeiro, G. Lopes, H. Ribeiro, P. Silva, T. Maia, R. Roriz, A. Gomes and N. Ferreira, "Minho Team '2016: Team Description Paper", University of Minho, Guimarães, Portugal, 2016.

[10] Premerlani, W., Bizard, P.: "Direction cosine matrix IMU: theory", http://gentlenav.googlecode.com/files/ DCMDraft2.pdf

[11] Fernando Ribeiro, Gil Lopes, Bruno Pereira, João Silva, Paulo Ribeiro, Joao Costa, Sérgio Silva, João Rodrigues, and Paulo Trigueiros, "Robot Orientation with Histograms on MSL", (2012) T. Röfer et al. (Eds.): RoboCup 2011, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7416 LNCS, pp. 507-514. Springer-Verlag Berlin Heidelberg 2012. DOI: 10.1007/978-3-642-32060-6_43.