A humanoid robotic platform to evaluate spatial cognition models

Martin Llofriu[1,2], Gonzalo Tejera[1], Alejandra Barrera[3] and Alfredo Weitzenfeld[2]

[1]Instituto de Computación, Facultad de Ingeneiría, Universidad de la República, Uruguay
[2]Computer Science and Engineering, College of Engineering, University of South Florida
[3]Departamento de Computación, ITAM, Mexico DF, Mexico

***Abstract*—Robot self localization has been shown to be needed in both real robotic tasks and in current robotic competitions as the humanoid soccer league and Standard Platform League (SPL) of Robocup. In this paper we describe a platform to reproduce animal navigation experiments in robots that aim to develop a bio-inspired approach to the SLAM problem. The platform is based on an Aldebaran NAO robot navigating inside an open circular maze with external landmarks for the robot to perceive. We describe the robot architecture and the platform used to support spatial cognition model evaluation.**

***Index Terms*—Humanoid robot, self localization, place cells.**

I. Introduction

Self-localization in navigation has been shown to be of great importance for the development of autonomous robots, including robotic football teams, navigation in unstructured environments like underground mines and cleaning robots, to name a few. The present work presents the development of a experimental platform to perform research in the field of bio-inspired models of robot self-localization.

Animal spatial cognition experiments provide insight on the neural mechanisms underlying their navigational capabilities. By programming spatial cognition models into robots and reproducing the same experiments, researchers are capable of further develop a brain model of the animal's self localization capabilities.

Spatial cognition in rodents has been long studied. In his experiments with rats, Morris et al. have shown that the animal is capable of making decisions based on its global position within a certain environment, proposing the existence of a cognitive map[27]. Further studies have shown the existence of cells with firing patterns that correlate to absolute location information. Place cells found in the hippocampus are shown to fire when the rat is within a certain region of an environment[28]. Grid cells, found in the entorhinal cortex, on the other hand, fire when the rat is near the vertices of an absolute triangular grid laid out in the environment[24].

A robot experiment platform has been developed to reproduce navigational experiments similar, but not limited to, Morris'. The platform is based on an Aldebaran NAO robot navigating on a open circular arena with external landmarks for the robot to perceive. In this paper, we describe the robot architecture and experimental platform used to support spatial cognition model evaluation.

This article plays two roles. On one hand, it thoroughly describes the experimental platform to ensure experiment reproducibility and to share developed code. Thus, we include software references and tutorial-like explanations in the article. Additionally, technical aspects of the development are described in depth. Some mathematical and technical arguments are then presented.

Section II describes the environment that models the open maze and the landmarks. Section III describes the robotic system and the deployed software framework. Section IV describes the software deployed in the robot including the percepts, decision processes and actions. Sections V and VI show the performed preliminary tests and the obtained results. Finally, conclusions and future work is included in sections VII and VIII.

II. Experiment environment

The environment consists of a 2.5 m. diameter circle over a green carpet. The circle is delimited by white tape stripes conforming a 26-side inscribed polyhedra. Figure 1 shows the experimental environment.

Four rectangular columns are placed in four cardinal points of the circle (not real magnetic cardinal points). Each column holds a different Japanese letter that serves as an identifiable landmark to the robot. The Japanese letters are printed with a surrounding .15 m sided black rectangle, as required by ARToolkit (see Section III). These symbols where chosen because they contain complex binary image patterns, which makes them suitable for recognition. The landmarks where held .22 m over the ground by the columns.

No special lighting devices have been set and the experiments are carried out using the laboratory's illumination, which consists of regular fluorescent tubes.

III. Robotic platform

The robotic platform is based on the Aldebaran NAO v4 robot. An *Ubuntu* distribution[19] has been deployed in the robot, as well as the *Robotic Operating System (ROS)* and third-party packages.

A. Ubuntu chroot

In order to be able to deploy software more easily in the robot, an *Ubuntu* version has been deployed in it. Given the

Fig. 1: The working environment. The horizontal landmark in the ground is not yet taken into account by the perception system. Figure taken from [33].

robot has a i386 processor, it is possible to do so by building a chroot system in almost any desktop machine and then deploying it to the robot.

The *chroot* can be built following the tutorial found in [2]. The *Precise* (12.04) *Ubuntu* distribution is recommended, while the *Raring* distribution (13.04) is not recommended, due to the fact that *ROS Groovy* (stable) version does not work in that environment.

To use the newly deployed *Ubuntu* distribution, one has to access the robot console through the *ssh* command and use the *chroot* command[2] to change from the native operating system to the new one.

B. Robotic Operating System

This new operating system has been equipped with the *Robotic Operating System (ROS)*[15] platform. Using *ROS* has many benefits. To begin with, plenty of developed software can be used, as has been done in the developed software. Apart from that, the *ROS* framework facilitates the development of decoupled architectures by providing services like interprocess communication. Additionally, the developed software is more likely to be reusable when programmed in a standard platform. Finally, *ROS* eases the development process of robotic software by providing debugging and visualization tools like those shown in Section IV

ROS can be deployed in the chroot system through the use of the apt-get command[9]. This deployment method has been developed in this work and has the advantage of deploying a full *Ubuntu* system on the robot. This platform allows for an easier deployment of *ROS* third party packages through the use of the *apt-get* tool, avoiding the tedious task of cross-compiling each one of them by hand.

After deploying *ROS*, a workspace is to be laid out, which can be done following the tutorial in [6]. Additionally, a *rosbuild* workspace is needed to deploy old third-party packages as explained in the next section. *Rosbuild* workspaces can overlay catkin ones[7]. Once this has been set up, the *rosbuild* environment will have access to the packages both *catkin* and *rosbuild* workspaces, as well as the packages installed by default with *ROS*.

C. Third-party packages

Some third-party ROS packages have been installed in the system. The best way to do this is through the use of the *wstool*[20], which has the same syntax as the old *rosws* command[16]. These commands take care of downloading the package's last version once the repository has been established using the *wstool set* command.

1) nao_robot: This package interfaces with the *naoqi* program running in the robot, providing an extra layer of abstraction. It provides useful functionality for software development for the *NAO* robot:

- Publish of controlling service that allows to move the robot using *ROS* messages
- Camera driver that polls images continuously and publishes them as *ROS* messages
- Robot URDF model[12] and a joint state publisher that may be used by the *tf*[18] and *robot state publisher*[**?**] packages to compute transformation between the different joint coordinate frameworks

As part of the present work, this *ROS* stack has been modified to include a camera driver that polls both images at the same time, called *nao_camera_both.py*. This version of the stack is available as a git repository[11].

2) camera_info_manager_py: This package provides a python interface for the *ROS* camera info manager functionality[4]. It is used by *nao_cam_both.py* to publish camera intrinsic parameters to the *camera_info topic*.

As part of the development process, a slight change has been made to this package in order for it to allow publishing more than one service. However, the updated version has been merged to the main branch.

3) ccny: ccny package[8] provides functionality for recognizing patterns like the one used in the landmarks. It makes use of the *ARToolkit* software[21]. Besides visually detecting the patterns, it provides information about the pattern's relative location to the camera, given that the camera is properly calibrated.

The repository pointed out in the documentation does not contain working code. The *IHeartEngineering* git repository[**?**] should be used instead.

IV. Robot Software

The robot program process sensorial information into higher level features and use them to make navigation decisions. It is composed of two interacting systems: the a*gent_ap*i python package that runs locally in the robot and the *Mobile Internet Robotics (MIRO)*[35] program running in a remote computer. These two systems communicate with each other using the *protobuf* software[14].

The *agent_api* takes the images captured by the cameras and process them to extract high level features. These include the identifiable landmarks and the lines that delimit the environment. Additionally, this high level features are processed to infer higher level information required by the *MIRO* system. The detected lines are projected to a 3D space and the robot

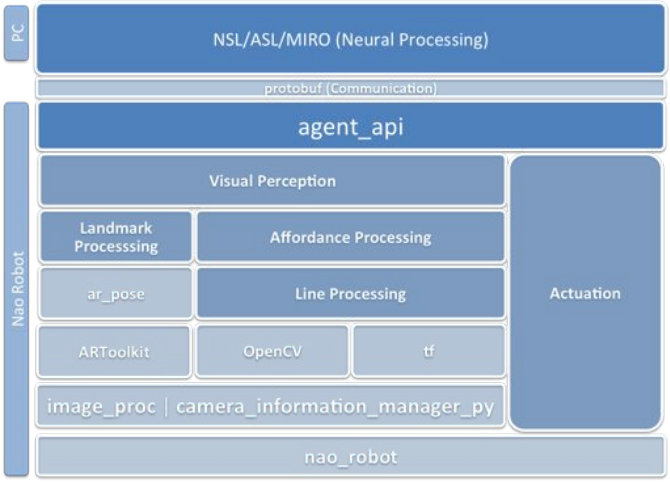

Fig. 2: Architecture diagram. Light shaded rectangles represent third party packages whereas the dark shaded ones are developed in this work.

affordances are computed using this information. Besides, the relative orientation and distance information of the landmarks is translated to the perceptual schema required by the *MIRO* system.

The *MIRO* system is used to implement a rat simulation system that makes the navigational decisions based on the inputed information. To do this, neural models are implemented using the *Neural Simulation Language (NSL)*[37] and *Abstract Schema Language (ASL)*[36]. Once the neural models make the navigational decisions, high level control command are sent back to the *agent_api* package to execute them.

Figure 2 summarizes the developed architecture.

Following, the main aspects of the *agent_api* package are explained. A detailed description of the rat simulation system can be found in previous articles[22, 23].

A. Image capturing

Image capturing is done through the implemented *ROS* node *nao_camera_both.py*. The images are inputed to the *image_proc ROS* module[10], which corrects the images using an offline calibrated distortion model.

The camera was calibrated using the *camera_calibration ROS* module[3]. The output model was saved as a yalm file. The *camera_info_manager_py ROS* module[4] is used to publish the information in this files.

Image capturing is made at a resolution of 640×480 pixels at a rate of 5 frames per seconds.

B. Line detection

1) Image line detection: The line detection module (linedetector.py) applies two *OpenCV*[13] filters to the corrected image from the bottom camera. First, a *Canny* edge detector[5] is applied. Then, the probabilistic version of the *Hough* filter[26] is applied. All parameters are set empirically to improve results in the current environment.

2) Coordinate transformation: When working with single camera visual features, there are a wide variety of options regarding which coordinate framework to use and how to map image coordinates to the chosen coordinate system. Tasse et al. work with polar coordinates of the detected features, which can be derived from the image in a straightforward way[32]. Silva et al. use the image coordinates and the known size of a detected object to compute the egocentric cartesian coordinates of the object[31]. Others use the fact that the object is in the floor plane to derive egocentric cartesian coordinates from the image coordinates[29, 30, 38], as does the present work. They do not fully specify how this is done, though.

The detected lines' image coordinates are transformed to a 3D space. This transformation can be done with a monocular camera because the lines are known to be in the floor level, as done by Jazmad et al. [25]. Thus, the z coordinate is known to be 0, if a coordinate framework on the floor is used.

The transformation is done in two steps. First, the 3D line of projection of each of the lines ends is found. Then, the line is intercepted with the floor plane to find the 3D coordinates of the point.

The projection line is found in the camera's coordinate framework first. This is accomplished by finding two points in the line. The first point p^{cam} is the origin of the camera framework, considered to be the focal point of the pinhole camera. The other point q^{cam} has the coordinates $(img_x/f_x, img_y/f_y, 1)$ where img_x and img_y are the image coordinates for the point and f_x and f_y are the calibrated focal distances. Both points belong to the projection line due to the pinhole model.

Those two points are then transformed to the floor coordinate framework using the *tf ROS* package[**?**][18], obtaining two new points p^{floor} and q^{floor}.

Then, the lambda equation of the line is considered, as shown in Eq. 1.

$$r = p^{floor} + \lambda(q^{floor} - p^{floor}) \tag{1}$$

Given that the z coordinate is known to be zero, Eq. 2 is obtained, from where λ is derived.

$$0 = p_z^{floor} + \lambda(q_z^{floor} - p_z^{floor}) \tag{2}$$

Then, the x and y coordinates of the point can be computed as shown in Eqs. 3 and 4.

$$r_x = p_x^{floor} + \frac{p_z^{floor}}{p_z^{floor} - q_z^{floor}}(q_x^{floor} - p_x^{floor}) \tag{3}$$

$$r_y = p_y^{floor} + \frac{p_z^{floor}}{p_z^{floor} - q_z^{floor}}(q_y^{floor} - p_y^{floor}) \tag{4}$$

C. Landmark detection

Landmark detection is delegated entirely to the ar_*pose ROS* package[1]. The corrected image from the top camera is passed to the *ar_pose* module.

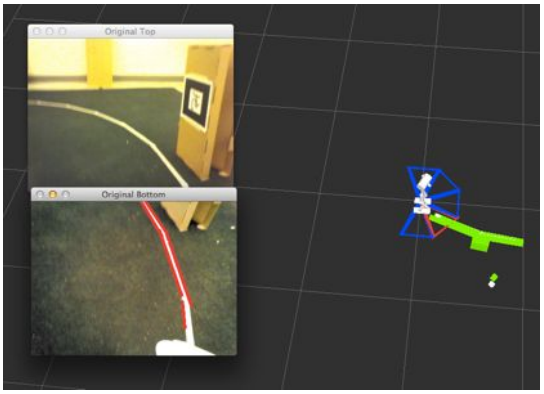

Fig. 3: The *rviz* visualization of the affordance calculation. The robot model is shown in white. Blue triangles represent navigable affordances and red triangles non-navigable ones. The green lines show all the lines taken into account in the affordance calculations. The output of both cameras is also shown in the left margin and the output of the Hough algorithm is shown for the bottom camera.

Prior to the current robot architecture, color matching has been used in our laboratory to detect landmarks[34](submitted). The use of black and white symbols as landmarks, as opposed to colored objects, turns the landmark recognition process less dependent from color calibration and from the environment illumination. Besides, the pattern allows for a better estimation of both the 3D coordinates, as well as the 3 degrees of rotation of the landmark. Finally, the use of complex patterns lowers the probability of false positives due to similar objects in the background, that are usual with color matching techniques.

In the future, automatic landmark extraction such as *SIFT* should be used. This would allow for the system to be executed in real environments, but requires the system to be able to deal with ambiguity.

D. Affordances processing

The rat simulation system takes the possible movement information, which are referred as *affordances* in the present work, into account when making the navigational decisions. The python module calculates each affordance by intersecting a triangle with all the detected lines. The triangle is isosceles and the angle adjacent to the equal sized sides is attached to the origin of the floor coordinate framework. The triangle then models the possible ending positions of a movement in that direction, given a certain movement error.

Having no intersection implies that the robot can move in the direction of the triangle.

Figure 3 shows the visualization of the affordances triangles using the *ROS* package *rviz*[17].

V. Tests

The implemented software was tested by comparing the estimations against the ground truth. Two experiments were designed to test the landmark detection and the line detection systems.

Fig. 4: Landmark detection experiment design from an aerial perspective. The rectangle represents the landmark and the circle represent the different robot positions.

A. Landmark experiments

A landmark known to the robot was placed in a fixed location. Then, the robot was required to estimate the distance to the landmark from different locations. Each position was 0.1 meters away from the following and previous ones, and they lie over one of two lines, one being perpendicular to the landmark and another having a 45 degrees skew, as shown in Figure 4. At each position, 20 distance measures of the landmark were recorded.

B. Line experiments

Two experiments were designed to test line recognition and the estimation of their location in a 3D coordinate system relative to the robot.

The first experiment resembles the landmark recognition one. A line was laid out in the floor and the robot position was varied along a line perpendicular to it. At each position, 20 measures of the distance to the line were recorded.

The second experiment consisted on recording line measurements while varying the robot's head pitch angle. The location of both the robot and the line were fixed. At each head pitch angle, 20 line measurements were recorded. This experiment was designed to test the integration of the robot's joint angles into the estimation of the line's position. Namely, the change in the camera's position must be computed correctly in order to derive the line's position in the floor coordinate frame.

Given that lines are not identifiable and many lines were detected at once, some work had to be done to ensure that the distance to the intended line was actually being measured. No line was placed between the robot and the reference line, so the minimum distance to any line could be taken as the desired measure. This had some negative consequences, as explained in the next section.

VI. Results

The landmark distance estimation results are depicted in Figures 5 and 6. Even though the standard deviation of the 20 measurements at each ground truth value is plotted, they are not visible due to their small relative values.

Figure 7 shows the results for the line distance estimation tests. A greater measurement variance was observed in these experiments, especially for the head pitch angle experiment. Some noise in the line detection algorithm was observed at

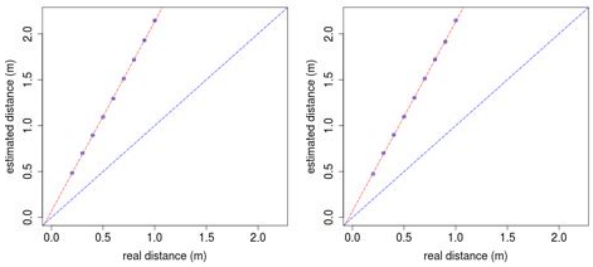

Fig. 5: The estimated distance to the landmark as a function of the ground truth for positions in the perpendicular line (left) and in the skewed line (right) with respect to the landmark. The blue line represents the identity function (expected behavior). The red line shows a least squares regression of the data.

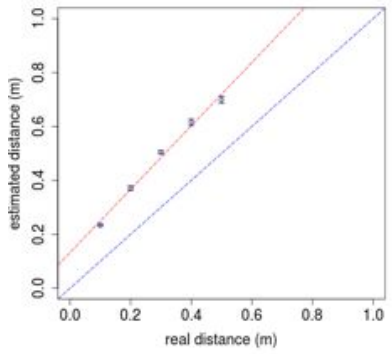

Fig. 6: The estimated distance to the line as a function of the ground truth for positions in the 45 degree skew line with respect to the landmark. The blue line represents the identity function (expected behavior). The red line shows a least squares regression of the data.

angles 10 and 15 , which could explain the higher variance in the measured distance. Given that the distance to the closest line was measured, a failure to detect the line intended to be detected implied that the distance to other background lines would be measured, interfering with the measurement process. Some false positives detections close to the robot were detected at those angles as well, which also interfered with the measuring process.

It can be appreciated from the plots that, although the estimations results in a linear response to distance changes, the actual estimated distance is usually not accurate. However, given the resulting linear behavior, the errors can be corrected by scaling all estimations by a linear transformation. Thus, it is the linear behavior what shows the correctness of the implemented system, rather than accurate measurements which can be obtained by calibrating that linear transformation.

VII. Conclusions

A robot platform for spatial cognition navigation experiments has been implemented using a *NAO* humanoid, developed using the *ROS* platform. The software was installed within the robot in a novel way, improving the development efficiency and reusability. Several third party packages have been integrated successfully and new ones have been developed or

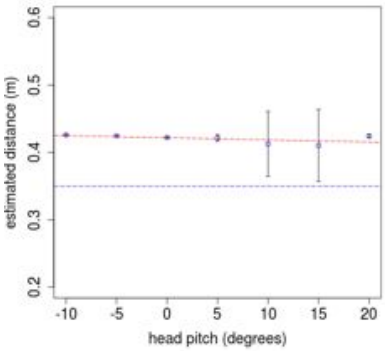

Fig. 7: The estimated distance to the line as a function of the ground truth for the head pitch angle experiment. The blue line represents the constant function at the real distance (expected behavior). The red line shows a least squares regression of the data.The angle values correspond to those accepted by the *NAO* API.

modified to provide the functionality needed to reproduce the experiments.

The robot platform showed positive results in the performed tests. The observed linear errors are expected due to differences in the measuring units scales. Namely, the notion of one meter in the robot does not correspond a real meter. The error can then be fixed with a linear transformation.

The landmark distance estimation stability is attributed to the maturity of the underlying ARToolkit software and the fact the the landmark image provides a lot of information to derive its location in 3D space. However, these results validate the camera calibration process as well because a good calibration is needed to get accurate estimations. Besides, the integration of all joint transformations from the floor coordinate framework to the *NAO* top camera is shown to be working correctly, as these are needed to coherently transform the camera coordinates to the floor coordinates in which the distance is measured.

The line distance estimation showed positive results as well. The algorithm showed a linear response to distance variations as well. In addition, the computation of the camera's position and its calibration were again validated by these experiments. A greater measurement variance was observed in this case though, which is attributed mainly to the noise in the probabilistic *Hough* line detection algorithm.

The inclusion of the black and white pattern landmarks allows the system to scale to a larger number of different landmarks, as opposed to color matching where the color calibration can only be made for a few different colors.

The line detection and position calculation, affordance calculation and landmark detection were developed as a ROS package and they publish their results in reusable *ROS* messages. The existing nao_*robot ROS* package was modified to use both cameras concurrently. The *camera*_info_*manager_py* package was modified and the modifications included in the main branch by the original authors.

VIII. Future work

The first step is to incorporate the constants derived from the experiments to calibrate the estimated distances to reflect real world values.

Secondly, the line detection system is to be improved by lowering the image noise that produces the variation in the Hough line detection algorithm outcomes. Objective measures of this noise will be included to assess the improvement in performance.

Other future work include recognizing ARToolkit patches in the floor that would represent a hidden goal to the robot, which would complete the platform required to reproduce Morris' and other experiments. Using automatic feature extractors such as SIFT as landmark detectors would be also be desirable.

References

[1] ar_pose - ROS wiki. URL http://www.ros.org/wiki/ar_pose.

[2] BasicChroot - community ubuntu documentation. URL https://help.ubuntu.com/community/BasicChroot.

[3] camera_calibration - ROS wiki, . URL http://www.ros.org/wiki/camera_calibration.

[4] camera_info_manager_py - ROS wiki, . URL http://ros.org/wiki/camera_info_manager_py.

[5] Canny edge detector — OpenCV 2.4.6.0 documentation. URL http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[6] catkin/Tutorials/create_a_workspace - ROS wiki, . URL http://www.ros.org/wiki/catkin/Tutorials/create_a_workspace.

[7] catkin/Tutorials/using_rosbuild_with_catkin - ROS wiki, . URL http://www.ros.org/wiki/catkin/Tutorials/using_rosbuild_with_catkin.

[8] ccny_vision - ROS wiki. URL http://www.ros.org/wiki/ccny_vision.

[9] groovy/Installation/Ubuntu - ROS wiki. URL http://www.ros.org/wiki/groovy/Installation/Ubuntu.

[10] image_proc - ROS wiki. URL http://www.ros.org/wiki/image_proc.

[11] mllofriu/humanoid. URL https://github.com/mllofriu/humanoid.

[12] nao_description - ROS wiki. URL http://www.ros.org/wiki/nao_description.

[13] OpenCV | OpenCV. URL http://opencv.org/.

[14] protobuf - protocol buffers - google's data interchange format - google project hosting. URL https://code.google.com/p/protobuf/.

[15] ROS/Tutorials - ROS wiki, . URL http://www.ros.org/wiki/ROS/Tutorials.

[16] rosws: A tool for managing source code workspaces — rosinstall 0.6 documentation, . URL http://www.ros.org/doc/independent/api/rosinstall/html/rosws.html.

[17] rviz - ROS wiki. URL http://www.ros.org/wiki/rviz.

[18] tf - ROS wiki. URL http://www.ros.org/wiki/tf.

[19] The world's most popular free OS | ubuntu. URL http://www.ubuntu.com/.

[20] wstool - ROS wiki. URL http://ros.org/wiki/wstool.

[21] ARToolkit. ARToolKit home page. URL http://www.hitl.washington.edu/artoolkit/.

[22] Alejandra Barrera and Alfredo Weitzenfeld. Biologically-inspired robot spatial cognition based on rat neurophysiological studies. *Auton. Robots*, August 2008.

[23] Alejandra Barrera, Alejandra Cáceres, Alfredo Weitzenfeld, and Victor Ramirez-Amaya. Comparative experimental studies on spatial memory and learning in rats and robots. *Journal of Intelligent & Robotic Systems*, September 2011.

[24] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I. Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, August 2005. ISSN 0028-0836. doi: 10.1038/nature03721. URL http://www.nature.com/nature/journal/v436/n7052/abs/nature03721.html.

[25] Mansour Jamzad, B. S. Sadjad, V. S. Mirrokni, Moslem Kazemi, Hamid Reza Chitsaz, A. Heydarnoori, Mohammad Taghi Hajiaghayi, and Ehsan Chiniforooshan. A fast vision system for middle size robots in RoboCup. In *RoboCup 2001: Robot Soccer World Cup V*. Springer-Verlag, 2002.

[26] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78(1):119–137, April 2000. ISSN 1077-3142. doi: 10.1006/cviu.1999.0831.

[27] Richard G.M. Morris. Spatial localization does not require the presence of local cues. *Learning and Motivation*, 12(2):239–260, May 1981. ISSN 0023-9690. doi: 10.1016/0023-9690(81)90020-5.

[28] J O'Keefe and J Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 34(1), November 1971. PMID: 5124915.

[29] Thomas Röfer, Tim Laue, and Dirk Thomas. Particle-filter-based self-localization using landmarks and directed lines. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, number 4020, pages 608–615. January 2006.

[30] Hannes Schulz, Weichao Liu, Jörg Stückler, and Sven Behnke. Utilizing the structure of field lines for efficient soccer robot localization. In Javier Ruiz-del Solar, Eric Chown, and Paul G. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, Lecture Notes in Computer Science, pages 397–408. Springer Berlin Heidelberg, January 2011.

[31] Hugo Silva, André Dias, José Almeida, Alfredo Martins, and Eduardo Silva. Real-time 3D ball trajectory estimation for RoboCup middle size league using a single camera. In Thomas Röfer, N. Michael Mayer, Jesus Savage, and Uluc? Saranlı, editors, *RoboCup 2011: Robot Soccer World Cup XV*, number 7416, pages 586–597. January 2012.

[32] Stefan Tasse, Matthias Hofmann, and Oliver Urbann. SLAM in the dynamic context of robot soccer games. In Xiaoping Chen, Peter Stone, Luis Enrique Sucar, and Tijn van der Zant, editors, *RoboCup 2012: Robot Soccer World Cup XVI*, Lecture Notes in Computer Science, pages 368–379. Springer Berlin Heidelberg, January 2013.

[33] Gonzalo Tejera, Alejandra Barrera, Jean-Marc Fellous, Martin Llofriu, and Alfredo Weitzenfeld. Spatial cognition: robot target localization in open arenas based on rat studies. pages 87560O–87560O, May 2013.

[34] Gonzalo Tejera, Alejandra Barrera, Martin Llofriu, and Alfredo Weitzenfeld. Solving uncertainty during robot navigation by integrating grid cell and place cell firing based on rat spatial cognition studies. 2013.

[35] Alfredo Weitzenfeld. *MIRO: An Embedded Distributed Architecture for Biologically inspired Mobile Robots*.

[36] Alfredo Weitzenfeld. NSL/ASL: a framework for modeling and simulation of biologically inspired neural based adaptive autonomous robotic agents. In *EMCSR 2002*, Vienna, Austria, 2002.

[37] Alfredo Weitzenfeld, Michael A. Arbib, and Amanda Alexander. *The Neural Simulation Language: A System for Brain Modeling*. MIT Press, Cambridge, MA, USA, 2002.

[38] Thomas Whelan, Sonja Stüdli, John McDonald, and Richard H. Middleton. Line point registration: A technique for enhancing robot localization in a soccer environment. In *RoboCup 2011: Robot Soccer World Cup XV*. January 2012.