RobôCIn @Home Description Paper

Carlos Henrique Caloete Pena[1], Heitor Rapela Medeiros[1], Lucas Oliveira Maggi[1],
Marvson Allan Pontes de Assis[1] and Raphael Cândido Brito[1]
Hansenclever de Franca Bassani[1], Edna Natividade da Silva Barros[1]

***Abstract*—The RoboCup@Home aims to develop service and assistive robot technology with relevance for future personal domestic applications. Commonly, the RoboCup@Home tasks involve object detection and recognition, mapping and navigation, and human interaction. This year is the first time that we are going to participate in RoboCup@Home, on LARC 2018, using our TIAGo from PAL Robotics and we are searching for and developing new features for it. TIAGo combines mobility, perception, manipulation and human-robot interaction capabilities. We had developed packages with ROS and Gazebo (open-source simulator), and after we test in a real scenario. In this year, we could develop an initial framework to participate in RoboCup@Home. Our future work consists in to improve our manipulation package, and speak/image datasets to perform more real scenarios tests.**

I. INTRODUCTION

The RoboCup@Home aims to develop service and assistive robot technology with relevance for future personal domestic applications. The competition of RoboCup@Home consists of tests which the robots have to solve. Commonly, the task involves object detection and recognition, mapping and navigation, and human interaction [11].

II. RobôCIn TEAM

The RobôCIn is a robotic research group from CIn/UFPE. The team was created in 2015 by students, with the aim to apply what they learn in computer engineering classes. Since 2016, we participated in Latin American Robotic Competition (LARC - http://www.cbrobotica.org), on robotic soccer competition and we achieved fifth place, among 23 teams.

In the Very Small Size Soccer category [10], we developed much expertise in computer vision systems, robotic path planning, and system control areas. The knowledge acquired is essential to recognize and interact with its surroundings using TIAGo. One of the new categories that we would like to participate is @Home Category [11], on LARC 2018, using our TIAGo - Titanium Platform, that we have acquired in our university. This year is the first time that we are going to participate in this category, and we are searching for and developing new features for TIAGo.

[1]Center of Informatics - CIn, Universidade Federal de Pernambuco, Recife, PE, Brazil, 50.740-560 Email: {chcp, hrm, lom, mapa, rcb7}@cin.ufpe.br

III. TIAGo PAL ROBOTICS Robot

TIAGo is a service robot produced to operate in indoor environments. It inherits all the technology and robustness resulting from years of development and extensive use of others robots of PAL Robotics. The company is integrating all its software in ROS, and provide comprehensive documentation and support to help researchers to start working with the robots and obtaining results in a short time [2].

Fig. 1. TIAGo Robot Description

TIAGo combines mobility, perception, manipulation and human-robot interaction capabilities. As we can see in Fig. 1, TIAGo has an RGB-D Camera to map the environment, a stereo microphone to capture audio, laser range-finder to do obstacle avoidance and 3D mapping, a differential-drive base to mobility, and 7 DoF arm with fingers to perform manipulation. It has an autonomy of 4-5 hours with one battery and 6-10 with two batteries, and the charge is provided in real-time with a display LCD [3].

TIAGo arm is composed of 4x M90 Modules motors (the firmware of the electronic boards implements control PIDs for the position, velocity, and torque), and 1x M3D wrist

with 3 DoF (the embedded electronics provide control PIDs for position and velocity). The modules of the arm provide an accuracy of 0.087°. The arm modules max speed is 100°/s. Moreover, the modules and the wrist include self-protection mechanisms, like over-temperature and over-voltage. The different motors of the robot can be controlled using ROS interfaces [2]. TIAGo head has 2 DoF, so we can move up to down and left to right, and with this, we can perform an environment scan to map the scene. The Lifting Torso can be used to position TIAGo in a higher or lower place to perform manipulation.

IV. THE ROBOT OPERATION SYSTEM

The Robot Operating System (ROS) is an open-source framework with libraries and tools for writing robot software. In Fig. 2, we can see the ROS environment. ROS aims to simplify robot complex tasks, encouraging collaborative robotics software development. Using ROS, we can build, write and run code across multiple computers [1].

Fig. 2. ROS environment: A set of tools, packages and libraries to robot software development.

The primary goal of ROS is to support code reuse in robotics research and development. With this in mind, we could reuse code from others universities and improve it with our research. This approach contributes directly to science, where we can improve state-of-art techniques providing improvements and new points of view, focusing on the problem.

ROS currently runs on Unix-base platforms, in our work, we have used ROS Indigo Distribution with Ubuntu 14.04 LTS. The Indigo Distribution is compatible with TIAGo Pal Robotics robot.

V. GAZEBO

Gazebo is an open-source well-designed simulator that makes possible test robot tasks solutions using realistic scenarios. These scenarios can be model to be similar to real-world indoor and outdoor environments [4].

It provides a robust physics engine and a graphical interface to help the roboticist test and simulate the environment. The PAL Robotics team provides TIAGo simulation using ROS and Gazebo. The TIAGo simulation model allows a smooth transition from simulation to the robot. With Gazebo, we could make test algorithms rapidly for @Home competition outside the university, for example, in our house. In Fig. 3, we can see a TIAGo camera view inside Gazebo.

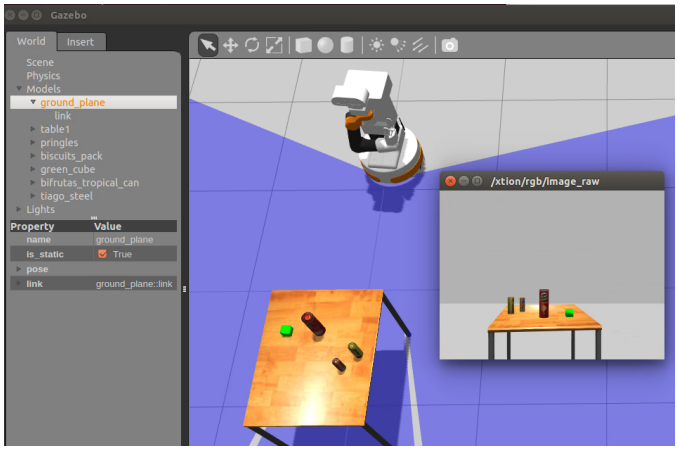

Fig. 3. TIAGo simulation inside Gazebo.

VI. TEST DESCRIPTION

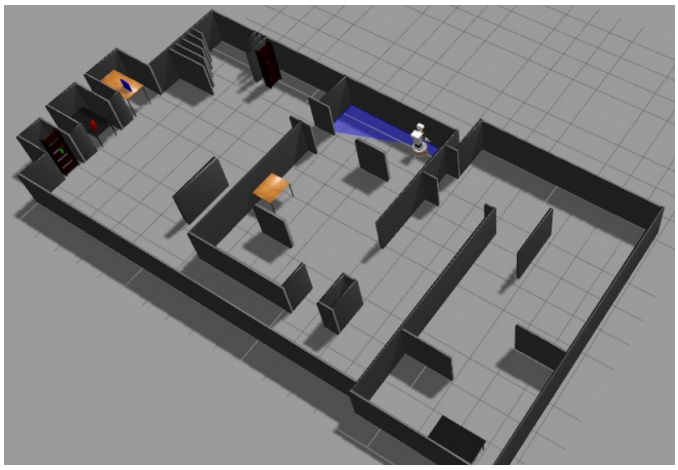

Fig. 4. Simulated map created for delivery task

VII. @HOME TASKS

A. MAPPING AND NAVIGATION

We used the gmapping package [15] for mapping and navigation to make the robot find the location of the packets, including the ones in a not predefined location. This package provides laser-based SLAM (Simultaneous Localization and Mapping). Using slam gmapping, we could create a 2-D occupancy grid map from the laser and pose data collected by a mobile robot.

We followed the TIAGo guide to navigate autonomously. In this step, TIAGo uses his RGBD camera and laser to avoid obstacles. TIAGo begins in a pre-defined position of the world which does not correspond to the map origin. After that, the probabilistic localization system spreads particles all over the map. Now, we need to move TIAGo to help the particle filter converge to the right pose estimate. So we start the teleop [13] command to use left and right

arrows to guide TIAGo. With this approach, the invalid particles are removed because the lase scan matches with the map and the localization converge to TIAGo correct pose. After that, we clean the cost maps as it contains erroneous data due to a rough localization of TIAGo. Now, the map only has the obstacles, and the system can perform the navigation. In Figure 5, we can see the map generated by the gmapping package after we clean the cost map.

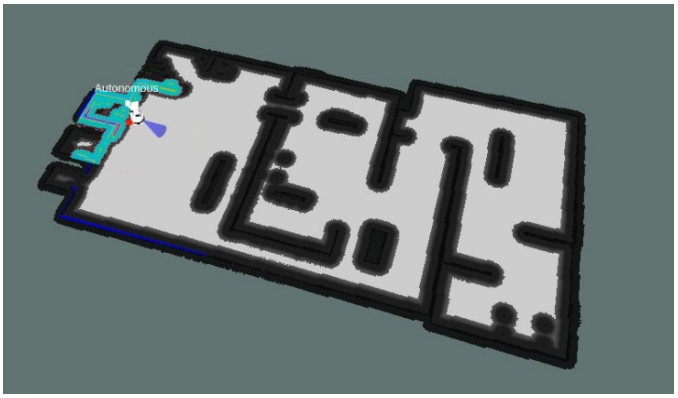

Fig. 5. Map generated by gmapping package with cleanned cost.

In this stage, we need to kill the teleop node and start the autonomous navigation.

B. AUTOMAP

1) Getting the map: We started to implement and test an automatic mapping algorithm to an closed environment. This algorithm is implemented over gmapping package, using the results of the cleaned costmap from *gmapping* SLAM. Then we get the image of the costmap Figure 6 calling the rosservice *pal_map_manager*.

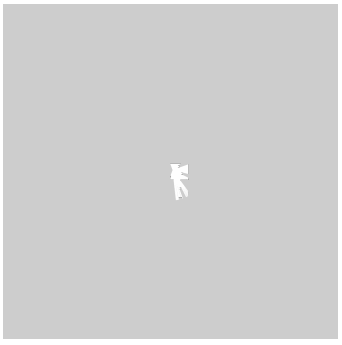

Fig. 6. Raw costmap.

2) Processing the map: The process can be seen in Figure 7, where we have 4 stages that follows: (a) We cut out the region of valid data from the raw costmap.

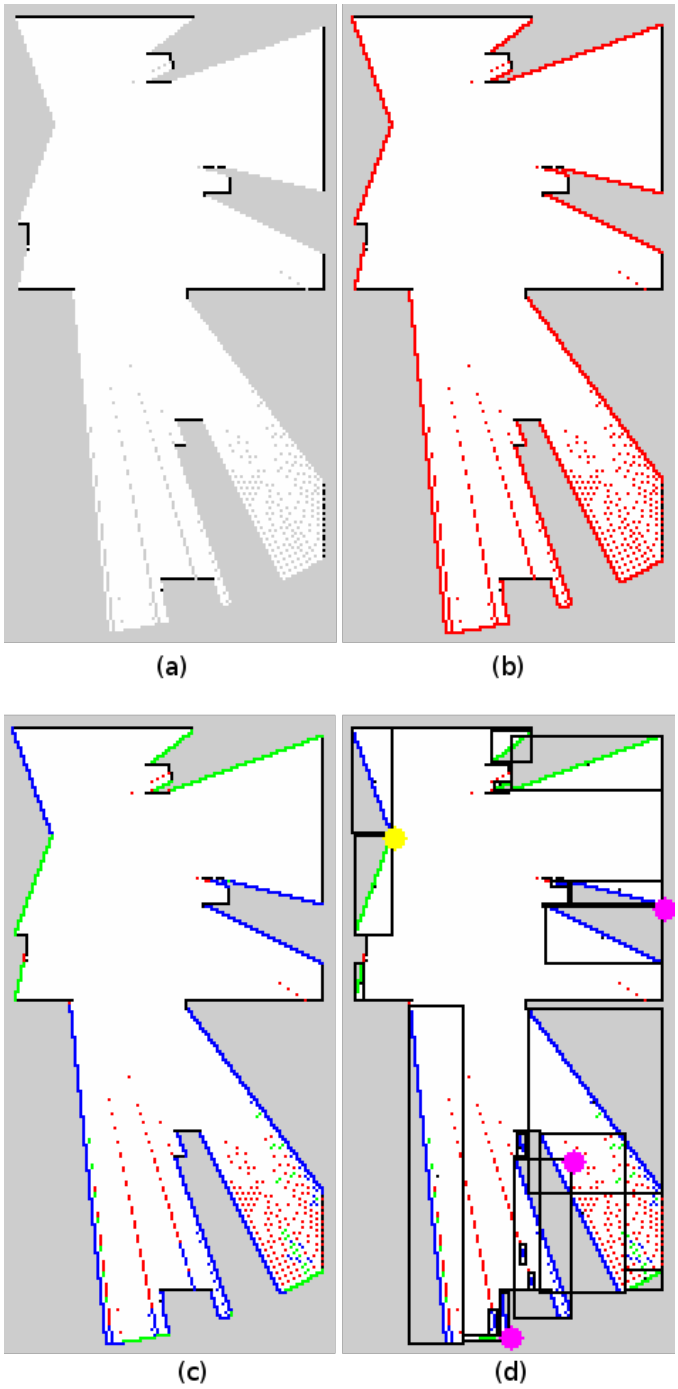

Fig. 7. Stages of the automatic mapping algorithm. (a) Cutted map. (b) Borders of the floor. (c) Extracted probable frontiers inclinations. (d) Extracted containing rectangles.

(b) We detect and mark the borders(in red) of the unknown region (in grey) and the floor region (in white). (c) We infer the inclination of the border, and mark then with blue if the slope is negative, and green if the slope is positive (in the bottom-left corner of origin for the coordinate system). (d) We draw the smallest rectangles that contains the individual segments of borders that are

not too small(frontiers), ignoring the noise from the map. And with it's centroids we search for the nearest white pixel, if not already on a white pixel, to assume that position as a possible destination for mapping(possible navigation points, this is repeated for all the frontiers.

3) Fiding TIAGo's origin on the map: We assume that the first map we get the robot will always be oriented East on the map. And it always generates an arrow in the map, as seen in Figure 7.c, there's an arrow made by and frontier of two slopes, blue and green. Then, we do a collision test of the rectangles that contains the segments of frontiers, and if they are of different colours we make the righteous point of both rectangles in X axis, but with the same postition of contact in Y axis. Because of that, we check the possible regions of arrow convergence (as seen in Figure 7.d) in yellow and pink filled circles, looking for the Western possible origin, being that represented by the yellow filled circle in Figure 7.d. And with that position of the cutted costmap, we convert these coordinates to the full costmap coordinates, and saving it's global origin's position, for further usage in navigation points.

4) Converting frontiers into navigation points: After we extract the frontier's possible's destinations for mapping, or the possible navigation points, we convert the coordinates from the cutted costmap, to the full costmap coordinates, and we make the TIAGo's first origin (in yellow on the first map, as seen in Figure 7.d) a translation vector, to translate every point to TIAGo's first position, because the navigation system uses the first position as the origin, then we apply the scale factor of pixels/meters to get the final coordinates for the navigation system's coordinates.

5) Navigating to map the environment: We use the service interface of move_base to send the navigation points, using a script in python, wich send the coordinates to the navigation service and wait for the return of the script execution to start the process of obtaining the navigation points.

C. DETECTION

A lot of progress has been made in computer vision through the use of convolution neural networks (CNNs). This kind of approach has been used in many object detectors. For detection task, we decide to use a You Only Look Once (YOLOv2) [7], That choice was based principally in two characteristics of YOLO. First, it supports multiple objects per image, over then 9000. Second, for the result showed in the YoloV2 paper, this network is less likely to provide false positives errors, And finally, it archived better speed than others like Fast R-CNN [8], RetinaNet [6], and SSD [5]. In Fig. 8, we can see a image passing by YOLO architecture and it provides the object boundbox.

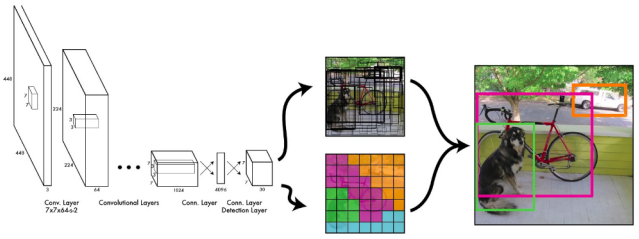

Fig. 8. YOLO pipeline for object detection. Image from YOLO authors blog.

This network was trained with our dataset to provide from both the real and simulated environments as showed in picture 9

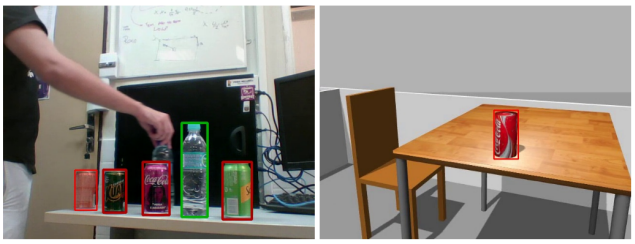

Fig. 9. Example of RoboCIn Dataset. In (a) ground truth of real data, and (b) ground truth of simulated environment data.

After we have the boundbox and class of the object in pixels with his respective class represented by (x_{pixels}, y_{pixels}, id_{class}). This pixel position points to the object center. For acquiring the camera's depth the distance from the object, a built-in function need the (x_{pixels}, y_{pixels}) coordinates, and returns the distance of this pixel in meters. Now we have (x_{pixels}, y_{pixels}, z_{meters}, id_{class}). The next step is to normalized the (x_{pixels}, y_{pixels}) measured using the intrinsic parameters from the camera. Finally, TIAGo moves his head (using move head package [14]) to minimize ($x_{normalized}$, $y_{normalized}$) distance between the object and TIAGo's head center, so the object is now center with the camera image. The manipulation is done with ($x_{normalized}$, $y_{normalized}$, z_{meters}, roll, pitch, yaw, id_{class}).

1) DEPTH CAMERA PACKAGE: The TIAGo depth RGBD camera (xtion) provides a roscpp node to extract the z component, but we have done a version in rospy to extract the z and help the community to integrate with others python modules.

D. MANIPULATION

We used the MoveIt! ROS package [9] for picking the packages and taking them to the delivery spots. Our approach consists in inverse kinematics, so we realized the planning in Cartesian space. With the MoveIt! package,

we could plan a joint trajectory to reach a given pose in Cartesian space [16].

In order to realize manipulation, firstly, we need to detect the delivery package. The bound box of packages and the depth from the camera is provided by the detection module described before. Afterwards, it is estimated the center of the object and a group of joints from TIAGo is choosen to get the delivery package. With the reference frame, we send the message to MoveIt! to find a path to the object. Once the path was found, it executes it.

E. SPEECH SYNTHESIS

We used the sound play ROS package for speech synthesis. This library is the straightforward way to make speech synthesis using ROS. The sound play package uses the Festival Speech Synthesis System [17] that currently only supports English and Spanish. The speech synthesis occurs when a text is provided for our node responsible for synthesis through a topic.

F. SPEECH RECOGNITION

For speech recognition, we used the pocket sphinx python library. This library is a wrapper for the CMU Sphinx base and the pocket sphinx library, for mobile devices. The continuous recognition module returns a list of hypothesis for the speech. The hypothesis that is unlikely, which means that the probability of the given phrase is below a threshold, are discarded. The probability of each phrase is calculated using the NLTK python library. Finally, the best hypothesis is filtered using named entity recognition and relationship extraction. The kind of answer is also identified in this phase, and an answer is formulated and sent to the speech synthesis node.

The best hypothesis is the one that best fits a known command for our robot. To identify the best hypothesis, a knowledge base is available through the knowledge base node. It is possible to make queries to the knowledge base using named entities identified by the others modules and also this one. If no hypothesis fits a known command, a question using the named entity identified and the relationship is sent to the speech synthesis node.

VIII. CONCLUSION

Using the methodology described in the Section VII, we could develop an initial framework to participate of RoboCup@Home. Our future work consists in to improve our manipulation package, and speak/image datasets to perform more real scenarios tests.

ACKNOWLEDGMENTS

This work was developed using the TIAGo titanium provided by the UFPE and the resources of the CIn-UFPE.

References

[1] Robot Operating System (ROS). Available in: http://www.ros.org/about-ros/. Last Accessed 26 June 2018.
[2] TIAGo PAL Robotics. Available in: http://tiago.pal-robotics.com/. Last Accessed 26 June 2018.
[3] TIAGo Robot Wiki. Available in: http://wiki.ros.org/Robots/TIAGo. Last Accessed 26 June 2018.
[4] Gazebo Robot Simulation. Available in: http://gazebosim.org/. Last Accessed 26 June 2018.
[5] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Springer, Cham, 2016.
[6] LIN, Tsung-Yi et al. Focal loss for dense object detection. arXiv preprint arXiv:1708.02002, 2017.
[7] REDMON, Joseph; FARHADI, Ali. YOLO9000: better, faster, stronger. arXiv preprint, 2017.
[8] GIRSHICK, Ross. Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision. 2015. p. 1440-1448.
[9] MoveIt!. Available in: http://moveit.ros.org/. Last Accessed 15 June 2018.
[10] IEEE Very Small Size Soccer. Available in: http://www.cbrobotica.org/?page_id=81. Last Accessed 15 June 2018.
[11] RoboCup@Home. Available in: http://www.cbrobotica.org/?page_id=132. Last Accessed 15 June 2018.
[12] OpenCV. Available in: http://opencv.org/. Last Accessed 15 June 2018.
[13] TIAGo Teleop. Available in: http://wiki.ros.org/Robots/TIAGo/Tutorials/motions/key_teleop/. Last Accessed 15 June 2018.
[14] TIAGo Move Head. Available in: http://wiki.ros.org/Robots/TIAGo/Tutorials/motions/head_action/. Last Accessed 15 June 2018.
[15] TIAGo GMapping package. Available in: http://wiki.ros.org/Robots/TIAGo/Tutorials/Navigation/Mapping/. Last Accessed 15 June 2018.
[16] Planning in cartesian space. Available in: http://wiki.ros.org/Robots/TIAGo/Tutorials/MoveIt/Planning_cartesian_space/. Last Accessed 15 June 2018.
[17] Speech synthesis system. Available in : http://www.cstr.ed.ac.uk/projects/festival. Last Acessed 28 June 2018.
[18] Speech recognition library. Available in: https://github.com/bambocher/pocketsphinx-python. Last Acessed 28 June 2018.
[19] Natural Language Toolkit. Available in: http://www.nltk.org/. Last Acessed 28 June 2018.