Abstract—Search and Rescue (SAR) Robotics has been gaining an increasing interest in recent years. In spite of that, there is still a considerable amount of work to be done in terms of autonomy and usability. RAPOSA is a semi-autonomous, tracked wheels robot, designed for SAR operations that has a low level of autonomy. The only tasks that were performed autonomously are the inversion of the image and of the commands for the wheels when the robot “flips” upside down. The main goal of this thesis is to endow RAPOSA with a higher level of autonomy, namely by providing an hole detection mechanism and an autonomous docking to the power and communications cable. The hole detection is performed recurring to the infrared sensors. The autonomous docking is mainly based on vision using also odometry that was also developed under this work. Additional work for this thesis is described, namely a new communications protocol. The two tasks mentioned above were implemented and tested under various circumstances including outdoor environments. The new communications protocol, highly efficient (in terms of packet rate), is used nowadays, having solved some communication problems.

Keywords: Search and Rescue Robotics, Autonomous Docking, Vision based control

I. INTRODUCTION

Search and Rescue (SAR) robots have an increasing importance in our life. In scenarios of natural catastrophes, a quick response is essential but not always possible due to the building damage. Reinforcing the structure to allow a safer environment to human and dog rescue teams can take too much time to possible survivors. From this, arises a clear need for SAR robots that can act rapidly before the human and dog teams.

RAPOSA is one example of a semi-autonomous SAR with a low level of autonomy. The main features of this robot are the main body plus an articulated frontal arm, the two-side tracked wheels, the wireless communications with or without the cable (access point in the end of the cable) and the possibility of operating the robot even if it flips down. The tethered option includes both wireless communication and power. The only tasks that were performed autonomously are the inversion of the image and of the commands for the wheels when the robot “flips” upside down. Due to this fact, the main goal of this Master thesis is to to endow RAPOSA with a higher autonomy level namely by:

- detection of holes and automatic stop
- autonomous docking to the power and communications cable

As it can be seen later, other tasks were implemented: some of them as a necessary condition (such as odometry) others as a compliment to a better functioning of the robot (new communications scheme). So, in the next subsection the related work is discussed. Then, the general software architecture used to implement the mentioned goals is described. The hole detection module is presented next. Additional work is shown before the autonomous docking, namely the communications and odometry. After the description of the autonomous docking and its results, the dissemination activities are briefly discussed. In the end, some conclusions are drawn and the future work is proposed.

A. Related work

There are several well succeed examples of SAR robots not only as academic projects but also as commercial products.

One of the most well known works is developed by the Center for Robot Assisted Search & Rescue (CRASAR) led by Dr. Robin Murphy. Several robots have been tested and used in real scenarios by this Center. CRASAR has not only robots capable to be used in SAR operations but also people trained to these rescue operations. Their robots worked in September 11th, 2001 after the World Trade Center (WTC) attack, Iran earthquake in 2003 and Katrina hurricane in 2005 for instance [1]. Other very important work has been done by Adam Jacoff’s team at NIST with the USAR Performance Metrics and Test Arenas development [2]. These Arenas, real scenarios representing buildings in different stages of collapse, have been used since 2000 by the American Association for Artificial Intelligence (AAAI) and since 2001 worldwide by the RoboCup Rescue Robot League where several teams compete.

As the interest in this kind of robots was being raised up, several commercial products appeared. Many of them are used not only as SAR robots but also for pipeline inspection, bomb-disarming or military operations. For instance, both Packbot from iRobot1 and TALON from Foster Miller2 were used in the WTC attack but nowadays are used in Afghanistan and Iraq in explosive disposal.

A common feature in most of these robots is that they can operate in wireless mode or tethered mode. Although, normally when the tether is connected, the batteries are useless because it is not possible to change between the two modes without changing robot structure. RAPOSA can change between tethered and batteries power in real time. Moreover, when the tether is connected to the energy supplied is used not only to power the robot but also to recharge the batteries. The trend is to have either tethered or not-

1http://www.irobot.com
2http://www.foster-miller.com/lemming.htm
tethered according to the application and to the needs of power and mobility but RAPOSA is a good exception.

II. GENERAL SOFTWARE ARCHITECTURE

In this section it is described the general software architecture of the work developed. To get a modular approach and to help the development of new modules as more independent of the present code as possible, YARP (Yet Another Robot Platform) [3] was chosen as a development tool. YARP is an open source middleware that supports an inter-process communication (IPC) transparent to the user. It has a set of libraries and protocols to signal processing and interfacing with operating system (OS) and devices. The main features include support for IPC, image processing and device drivers. Using YARP facilitates the development of code related to image processing (a significant part of the project). This is due to the internal class of the image class of YARP that is very similar to the one used in IPL (Image Processing Library) [4] and in OpenCV [5] library. The first one is an Intel library that implements basic operations on image and that at run-time is optimized to the CPU used. The last one is an open source library that implements sophisticated routines. The ideal architecture would include all the modules on the robot side. Although, due to practical reasons like lack of RAM memory, lack of free space on the flash drive and battery autonomy decrease, it was chosen to develop the modules on the console side. When these practical conditions change, the modules should be put on-board. YARP will be extremely useful in this process due to its portability. Nowadays, each module is connected to the interface by YARP links making completely independent the Hole detection and the Autonomous docking.

III. HOLE DETECTION

The first addressed problem was the hole detection, mostly because of its simplicity. Even though it is not a complex problem it can be very useful to detect holes automatically. With the sensors calibrated one can always look to the information in the interface about the distance measured by the infrared sensors. But this is not enough because that does not take into account the arm angle that changes the distance measured from the distance to the floor (in the vertical axis).

The sensor used, model GP2D120 [6] from Sharp has a detection range from 4 to 30 cm. Due to poor precision of the sensor, the detection is based on a threshold that can change with intervals of 1 cm. Experiences in outdoor and indoor environments were done to calibrate the sensor. Thus, there is a look-up table that for each sensor reading has a corresponding distance in centimeters. A way of reducing the noise is the use of a simple exponential smoothing method described in (1)

\[ y_t = \alpha \cdot \text{measurement} + (1 - \alpha) \cdot y_{t-1}, \]

where \( \alpha \) is the smoothing constant, \( \text{measurement} \) is the sensor actual reading and \( y_t \) the value used to detect the hole. This equation is recursive and so the last values accumulate but their influence decreases exponentially as time goes by. The value for \( \alpha \) chosen was 0.9 giving a much bigger importance to the new sensor reading than the older ones. This approach is very reactive but it is also safer because it ensures a quick answer to real big changes in the distance to floor.

When there is a hole, the interface displays a message “HOLE DETECTED” and it stops immediately the robot. It sets the velocity of the wheels motors to 0 and blocks the arm motor as it is maintaining its position. This is important because if one turns off the arm motor it would go to the rest position and that could be dangerous provoking a disequilibrium and consequent fall of the robot into the hole. If there is a false positive detection (a hole that does not really exist), the operator can click the button ON to OVERRIDE. It is important to notice that due to the semi-autonomous kind of robot it is essential to keep the possibility of control by the operator. So, when in “OVERRIDE” mode, the motors are re-enabled and the user can carry on with what he was doing. If the sensor is detecting too many unreal holes, the user can simply maintain the OVERRIDE mode meaning that it should be the operator the responsible for not entering inadvertently onto a hole.

As it was said before, the arm angle is considered in the hole detection. The distance used to detect holes is the biggest height that the robot is reaching with respect to the contact point of the wheel. The module was tested and worked correctly. The limited range does not allow detecting distances higher than 26cm and that is a point to improve by changing the sensors.

IV. COMMUNICATIONS

The communications protocol that was originally developed, presented congestion problems because of its huge traffic with tiny packets. So, there was a need to redesign it. It is not worthwhile to implement a completely new protocol because one can take part of the best of the previous communications scheme in order to improve it. The protocol that was in use needed 11 packets to ask values (5 packets) and send commands (6 packets) in the PC - robot way and more 5 packets to receive the responses. The ideal (and what was implemented actually) is to have only one for each way. This represents an improvement of 87.5% in the packets rate.

The original protocol it was easy to go from the packet received to the command sent to the PIC. One can aggregate all these tiny packets in one larger one and send it in each way. Then, in each side it was just necessary to unencapsulate the large packet to the tiny packets maintaining the low-level protocol with the PIC.

To implement this scheme, a Blackboard was created on the robot side (on the PC it was already implemented). When all the responses from the PICs are written in the Blackboard, the packet is sent to the robot. Although, to prevent problems resulting from some bad functioning PIC or hardware, there is a timeout mechanism. After a certain amount of time, if the Blackboard is not fully filled with the required answers, then the timeout mechanism sends a local copy of the Blackboard. It is ensured that the Blackboard...
is updated because whenever a response comes it overwrites the previous value.

During the development of the new protocol, the original one failed in an exhibition at Central Building due to the congestion problem. As what concerns to the new one, it never failed until the present moment under similar situations. Moreover, it was measured the number of packets using the tool tcpdump and the theoretical expectations were confirmed by the results. The timeout condition proved to be useful and works as well.

V. ODOMETRY

In order to do some of the autonomous docking, there was the need of using dead reckoning. Odometry was not already implemented in the robot, so there was the need to develop it. First of all, it is extremely unadvised in the literature to use dead reckoning in this kind of tracked vehicles [7], [8]. Although the robot is a differential drive vehicle, it is a particular case named skid steering. In skid-steer vehicles, there is always a large slippage whenever the vehicle turns. In the case of RAPOSA, the center of rotation also changes, depending on the position of the frontal arm. One possible solution to estimate a more accurate kinematic model could be to use the approach described in [9], but it would need additional external sensors that are not feasible to introduce in RAPOSA.

Considering the classic unicycle kinematic model (Fig. 1), the equations relating the vehicle velocity and the speed of each wheel are the following:

\[ V = h \frac{\omega_r + \omega_l}{2} = \frac{V_r + V_l}{2} \]
\[ \omega = h \frac{\omega_r - \omega_l}{d} = \frac{V_r - V_l}{d} \]

where \( V \) and \( \omega \) are respectively the linear and angular velocities, \( V_r \) and \( V_l \) denote the right and left wheel speeds, \( h \) is the wheel radius (assumed equal to both wheels), and \( d \) is the distance between the wheels. As it is described in [7], the distance \( d \) used for the kinematic model of the unicycle, in the case of a tracked wheel vehicle, lies within an interval between \( d_{\text{min}} \) and \( d_{\text{max}} \) (see Fig. 2). This distance is not actually fixed, depending for instance on the ground physical properties. However, in this implementation it was empirically estimated and considered a constant.

\[ \text{Fig. 1. Kinematic model of a unicycle vehicle.} \]

Fig. 2. Range of the \( d \) distance for the case of RAPOSA.

Low level issues, such as non-accurate access to each wheel velocity, together with an analog drift, brought some difficulties to the implementation, and introduced a significant error margin to the odometry measurement.

VI. AUTONOMOUS DOCKING

Several methods for automatic docking can be found in the literature. The use of computer vision has some advantages: on the one hand, it is a passive sensor eliminating the need of an emitting source, and on the other hand, other sensors like ultrasonic ones or infrared are not so accurate. In this work, only one camera is used to provide feedback to the control algorithm. Color segmentation, together with some geometric considerations, are used to perform the docking.

The first successful attempts to an autonomous mobile robot recharge were in the late 40’s with the Elsie and Elmer the so called "Machina Speculatrix" by Walter [10]. These turtle-shaped robots had a light following behavior into a hut that contained the battery recharger and a light bulb. In the last decade, there were several docking strategies for recharging purposes, as well as other purposes. In [11] a Nomad XR4000 used a IR beacon, a sonar, and a Sick laser range finder to dock to a recharging station. More recently, a Radio-Frequency IDentification (RFID) docking was presented in [12]. Among the ones that use computer vision, in [13] it is described a docking system for marsupial robots with an imprinting biological inspired mechanism. Vision and a laser range finder were used in [14] for a Pioneer 2DX. In [15], the Nomad has an omnidirectional camera and uses 3 non-collinear landmarks and their bearings to drive autonomously to the one that has the docking station below it. In a work described in [16], a circle on a wall is used to aid the docking. Other example of a docking purely relying on vision is the one described in [17]. It extracts a set of features that are compared to a database model involving machine learning.

A. Docking system

The docking system is composed by two parts: the grabbing mechanism (inside the robot) and the cable. The cable is flexible but at the end it has a solid pyramidal structure designed to provide a fixed inclination of it while laying on the ground (Fig. 3). To perform the docking the robot should lower its back and enter the metal guide with a certain orientation. Fig. 4 shows the docking socket at RAPOSA’s back, with the camera visible inside the robot. In its back part, there are two sliding doors that grab and pull the pyramid into the robot (an effect of bi-conical shape at the end).
Comparing with the state of the art, with the exception of [13], most of the publications on this subject use a visual target as an auxiliary tool to make the docking possible. In [13], the robot moves towards the mother robot but the marker is very big (a bigger area is easier to detect) and it is always visible. In RAPOSA, the situation is much more complex because the docking station enters the robot by the hole that is used by the back camera to look for it. Moreover, the optical axis of the camera is not aligned with the height of the station and so control in the y-axis is also done. It could not be found any other work that controls in 3D the robot to enter the docking station recurring only to vision.

**B. Solution proposed**

The goal is to start the autonomous docking algorithm from the moment that the pyramid is visible by the back camera. The algorithm ends with the fastening of the sliding doors. Due to the location of the camera and the insertion hole size, the useful Field of View (FOV) is approximately $25^\circ$. To minimally add visible marks to aid the vision algorithm, the bi-conical metal guide was painted with orange, and a few edges of the pyramid were painted with blue (Fig. 4). This allows to both detect the cable end, and to estimate the relative orientation of the pyramid to the robot.

**1) Docking regions:** Fig. 5 depicts a top view of the geometry of the problem. The robot is assumed to have the pyramid within the FOV of its back camera. The relative angle $\alpha$ that the pyramid axis makes with the robot position is crucial to the docking process: when this angle is small, the robot is able to dock in a straightforward fashion, by going forward$^3$ while performing small turns. Otherwise, an alignment maneuver is necessary to bring it aligned with the pyramid axis, thus reducing $\alpha$ to a small value. The strategy followed to tackle this problem consists in dividing the space around the pyramid in angular sectors, relative to the pyramid axis, as shown in Fig. 6.

---

$^3$In fact, the robot moves backwards relative to its front, since the robot docks its back with the docking station. We will, however, adhere to the word forward to avoid confusion.
2) Mathematical description: All the computation is based on the pin-hole model of the camera and on the perspective projection. The angle $\alpha$ is estimated from the detected blue rectangle (pyramid corners) and orange ellipse (ball at the pyramid end). The distance is estimated based on the geometric properties of the detected blue regions, whenever the robot is not aligned with the pyramid. Otherwise, it can use simply the orange ellipse area because the ellipse will maintain its shape as long as the robot moves in the direction of the pyramid. The blue color is not used here because occlusions occur for some distance range, while the orange is essential and enough for little misalignments.

From the side view of the robot (Fig. 7), one can get by trigonometric relationships:

$$\frac{g}{f} \approx \frac{h'}{d}, \quad (4)$$

where $f$ denotes the focal length of the camera, $d$ is the distance between the focal point and the vertical passing through the top blue vertex of the pyramid, $g$ is the height of the blue rectangle detected in pixels, while $h'$ is an approximation of the length of the blue edge. While $h$ is the real length of the edge, this approximation $h' \approx h$ has a reasonable small error (max 1.5%). So, (4) is approximated here by

$$\frac{g}{f} \approx \frac{h}{d}. \quad (5)$$

__Fig. 7. Side view from the docking pyramid and its corresponding projection__

From the top view (Fig. 8), the geometric relationship to estimate $c$ is given by

$$
\frac{c}{f} = \frac{a' + b'}{d} = \frac{a + b}{d'},
$$

However, since $d'$ cannot be accurately estimated, we use the relation

$$
\frac{c}{f} = \frac{a'' + b''}{d} \approx \frac{a + b''}{d}, \quad (7)
$$

while assuming $a'' \approx a$. The variables $a$ and $b''$ are given by

$$a = m \sin(\alpha), \quad (8)$$

$$b'' = \frac{h}{2} \cos(\alpha), \quad (9)$$

where $m$ is the distance between the midpoint of a blue edge and the center of the ellipse and $\alpha$ the angle to be estimated.

Using (7) and (5), we can obtain

$$
\sin(\alpha) = \frac{chm}{g(m^2 + (\frac{h}{2})^2)} \pm \frac{\sqrt{(chm)^2 - (m^2 + (\frac{h}{2})^2)((\frac{ch}{g})^2 - (\frac{h}{2})^2)}}{(m^2 + (\frac{h}{2})^2)}, \quad (10)
$$

As any quadratic equation, (11) has always 2 solutions (ignoring singularities). So, one has to chose one of them. If one equals the term $(\frac{ch}{g})^2 - (\frac{h}{2})^2$ to 0, $\frac{h}{2}$ is 0.5 meaning that the distance from the ellipse center to the edge end is half the width of the edge. This situation occurs at $0^\circ$ so it implies $\sin(\alpha) = 0$ which requires choosing the minus sign on the solution of the quadratic equation. For continuity reasons, the solution used will be always with the minus sign. Even though, there are still 2 possible solutions. The inflexion point corresponds to about $73^\circ$ which means that the angles estimated will be in the range $[0, 73^\circ]$ (for both sides). For angles bigger than that inflexion point there is an estimation error. Although, the robot will always get closer to the pyramid and will enter a position where $\alpha < 73^\circ$ eliminating the error after one step for initial angles lower than $110^\circ$ which is also the maximum angle for which the robot can see both colors with no occlusions. When the robot is not aligned (a significant $\alpha$), there is the need to align it. To do so, and taking in consideration the geometry formulation in Fig. 5, the robot rotates $-\beta$ at a first step (relative to referential where the robot is looking straightforward to the pyramid), travels the distance $d$ and then rotates $\varphi$. This will position the robot at a distance $\delta$ of the pyramid aligned with it.

3) Vision algorithm: The high level vision algorithm could be generically described in 3 steps:

- color based image segmentation
- shape based landmark extraction
- landmark recognition

For the color based segmentation, the color space chosen was HSV (Hue, Saturation, Value). A color is specified by a volume in this space, defined by the intersection of one interval for each axis H, S, and V. These slices offer a more robust detection performance, since RGB is very sensitive to lighting [18]. Still, lighting changes prevent the use of fixed and unchangeable thresholds to define the bands of H, S and V components. So, it is given to the user the possibility of adapting the thresholds in real time. The use of HSV allows for an intuitive tuning, instead of a RGB adaptation that could be fastidious.

For the shape based analysis, several metrics are used both for elimination and for choosing the best candidate among several ones. For the orange color there is an ellipse fitting for all the orange blobs and then some ellipses are filtered...
out first by area and then by ratio between axes as it is in Fig. 9. For the blue color, something similar is done. But here, the fitting of the rectangle is not by the circumscribed rectangle of minimal area but by its bounding region. This is because using the bounding region is better in the presence of a bad detection (or occlusions) as it is shown in Fig. 10.

![Fig. 9.](image1.png)

After this, other geometric constrains are used to eliminate pairs of rectangles and ellipses that cannot correspond to the physical dimensions of the object. There is a maximum deviation both for height and width scaled to the detection distance. There is also an area ratio criterion between the ellipse and the rectangle. A tracking of the ellipse is done and any pair with an ellipse out of a ball of 20 pixels (configurable) around the previous ellipse is filtered out. After all these filters, if there is more than one possible pair, the same criteria are used but this time tighter. In this way, one is more sensible to noise but in the case of a bad detection where there is only one pair, there is no risk of eliminating it by too strict constrains. Fig. 11 is an example of a well chosen candidate among 3 possible ones.

![Fig. 10.](image2.png)

After detecting the best orange and blue regions candidates, one of 7 states is chosen. Each state determines a specific course of action, as described in the next section. The possible states and its correspondence with the Docking regions are:

- state 1 corresponding to region I, low angle of deviation, perceives the orange and/or blue colors. It uses the information of the orange ellipse to the motor schema.
- state 2 corresponding to region II, large angle of deviation, perceives both colors. It uses vision and odometry.
- state 3 corresponding to region III or to bad detections on the other regions. It perceives only orange color.
- state 4 corresponding to region III or to bad detections on the other regions. It perceives only blue color.
- state 5 corresponding to region III or to bad detections on the other regions. It does not perceive any color.
- state 6 corresponding to region I. State used for distances close to the docking station. It perceives only orange and it is the final state.
- state 7 corresponding to region II. First part of region II where the robot rotates a certain angle closing the control loop based on vision. It perceives both color but uses the orange ellipse to the motor schema.

The choice of the state is performed as follows. After choosing the best pair, the state is chosen between 1 or 2 according to the angle and distance. If the robot is too close to the pyramid, the estimation of the angle is not accurate and then state 1 is chosen. That implies the deadzone mentioned. For angles bigger than 15° and/or distances bigger than 34 cm, state 2 is chosen. If there is no pair at the end of the detection, then the state is chosen by the orange ellipses detected at a first option. The best ellipse is chosen (by the same criteria) and depending on the distance it is chosen state 1, 3 (big distance) or 6 (small distance). In the case of the ellipse is not detected, then if there is any blue rectangle state 4 is chosen, otherwise state 5 is the one used. When the robot is at either state 6 or 7, further state choices are inhibited, i.e., it remains in that state until the corresponding behaviour finishes (namely, the completion of a docking or a rotation). The end of state 7 is detected based on a predictor to prevent the loss of the ellipse (in the case that the robot has to rotate all the FOV) due to the delay on the control cycle. The predictor uses the difference between the present and the previous frame to predict where the robot will be in the next one.

C. Motor schema

For each state, the velocities are computed in different ways. The arm position is also controlled but only at the end of state 1 near the docking pyramid. In all other states, the arm is kept at a constant position.

For region I states 1 and 6 are used. In state 1, the linear velocity is constant when the robot is above a certain distance. Below that distance, linear velocity is given by \( K_{\text{linear}} \sqrt{A_{\text{ellipse}}} \), where \( A_{\text{ellipse}} \) is the ratio between the ellipse area at 1 m (offline reference image) and the ellipse area at the current frame, and \( K_{\text{linear}} \) is a gain. The square root is used to make the velocity approximately linear with the distance, because the area of an ellipse decreases...
quadratically with the distance. The angular velocity is given by \(-K_{\text{angular}}x_{\text{deviation}}\), where \(K_{\text{angular}}\) is a gain, while \(x_{\text{deviation}}\) is the shift in pixels in the x-axis from the image center.

The arm position is set at the end of state 1 to \(-K_{\text{arm}}y_{\text{deviation}}\), where \(K_{\text{arm}}\) is a gain, while \(y_{\text{deviation}}\) is the shift in pixels in the y-axis from the image center. At the same time that the arm starts to move, the lights are turned on. This is because the state 6 is used when the area detected is higher than a certain threshold. In this situation the lights help the detection nearby. In this final state (6), the velocity is kept constant at a low value until a certain amount of time. After this, the robot stops and starts closing the doors.

For region II, RAPOSA cannot keep visual contact with the pyramid at all times, and thus it has to rely on odometry. Odometry is used only whenever the pyramid is out of the robot FOV.

The rotation based on vision corresponds to state 7. The first step of the algorithm for region II is a pure rotation of \(-\beta\). Whenever the angle \(\beta\) lies within the FOV of the image, the first step in region II will use only state 7, using visual servoing alone. When it is not, the angle to rotate is divided in two segments: the first part is limited by the FOV (state 7) and the second part is an odometry based rotation. The velocities used in state 7 have a low constant value so it can track the ellipse during the rotation. But, if the robot looses the ellipse for some moments, it stops. If the number of sequenced frames without identifying again the ellipse is too high (threshold configurable), it exits state 7 and recomputes a new state.

To reduce the odometry error, it is done an odometric calibration and the origin is reset after the end of state 7 with the angle rotated in that state. The inverse controller technique is used then in state 2. The velocities resulted from the controller are saturated and scaled and different configurable tolerances are used for each step mentioned at the end of subsubsection VI-B.2. The \(x_{\text{ref}}\) and \(y_{\text{ref}}\) of Fig. 5 are the references for traveling distance \(d\). The use of a bi-dimensional controller is justified by the possibility of correcting some orientation error in the middle step of going straight distance \(d\). The last rotation is not done based on vision due to the blurring noise and to prevent the following of other object. It is preferable to rotate based on odometry and then, after the robot is stopped, if it does not identify the pyramid, rotate slowly until it finds.

States 3, 4 and 5 implement this searching mechanism (slow rotations). For state 3, there is a simple scheme of start rotating for one side searching for anything else at a low speed (\(\omega_3\)). When the robot goes out of this state (e.g. by loosing the orange detected), \(\omega_3\) changes its sign and the robot starts searching in the other direction. Even if it enters other state, the change of sign is useful in case of returning later to state 3 because of, e.g., bad color detection. If it continued with the same sign it would be getting away of the target. For state 4, the same scheme used for state 3 is applied but this time to the blue color (\(\omega_4\)). In state 5, the robot does not perceive any color but uses the previous state to decide the search direction. If the previous state is state 3, then it will start to move in the opposite direction that it was moving on state 3. The same for state 4. If it came from other state, it will use other constant \(\omega_5\).

When it is in state 5, the robot will not do a 360° degrees rotation in the very beginning. Instead, it will rotate a certain angle in one direction and if, after this, it did not find anything, it will start to rotate to the other direction but now with the angle doubled. Every time it changes the direction it doubles the angle. In this way, it can be guaranteed that the whole 360° degrees are covered incrementally. In the end, it will take longer to cover the whole turnaround. However, if the robot deviates from the pyramid when it is in the final step of state 2 and enters state 5 in the wrong direction, after a certain amount of time, it will change its direction and find rapidly the pyramid covering much less area than the whole turnaround.

### D. Results

In order to compare the results of this work with others, the benchmark used in [13] and shown in Fig. 12 was chosen. These points are the starting points for the robot and in each point the robot is pointing towards the docking station.

![Fig. 12. The grid of points used in both works.](image)

However, this grid of points is not completely comparable. Their robot had a FOV of 70° but for reliability reasons they defined an area with a maximum of 60°(for each side). RAPOSA’s FOV is 25° so using a region of about 120° is a bigger challenge. The results for this grid of points are presented in table I and Fig. 13, showing the mean time to dock, the mean velocity \((v_{\text{mean}})\), the standard deviation \((\sigma)\), and the percentage of successful dockings \(\%_{\text{success}}\). The number of trials was 4 for all the points.

Points 1 and 3 are not presented here because the robot was not able to dock in any of the trials. These points are inside the deadzone discussed previously, and so this was expected to occur. For point 4 it was not possible to dock in one of the trials.

It is clear that for the 0° angle the standard deviation is kept very low (less than 2.5s) but the mean velocity increases. This can be understood under the light of the motors schema presented: recall that the velocity is linearly
TABLE I  
RESULTS AS A FUNCTION OF THE ANGLE.

<table>
<thead>
<tr>
<th>Angle (°)</th>
<th>Points</th>
<th>Mean Time to Dock (s)</th>
<th>σ (s)</th>
<th>v_{mean} (cm/s)</th>
<th>%success</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>19.41</td>
<td>2.04</td>
<td>1.7</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>24.29</td>
<td>2.41</td>
<td>2.1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>37.88</td>
<td>2.04</td>
<td>3.5</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>46.06</td>
<td>2.48</td>
<td>4.3</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>8.10</td>
<td>45.08</td>
<td>3.98</td>
<td>3.0</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>13.15</td>
<td>51.10</td>
<td>5.24</td>
<td>3.9</td>
<td>100</td>
</tr>
<tr>
<td>60</td>
<td>4.6</td>
<td>92.77</td>
<td>25.18</td>
<td>0.5</td>
<td>87.5</td>
</tr>
<tr>
<td></td>
<td>7.11</td>
<td>73.00</td>
<td>30.72</td>
<td>1.82</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>12.16</td>
<td>68.34</td>
<td>12.18</td>
<td>2.9</td>
<td>100</td>
</tr>
</tbody>
</table>

![Fig. 13. Mean time to dock as a function of the starting point](image)

proportional to the distance except when it is too far and when it is too close, where it is kept constant at a moderate speed and very low speed, respectively. This fact explains the results, since for points 9 and 14 the robot will move at a constant speed in the beginning and decreasing its speed only after a while. In point 2 the weight of the low speed mode is bigger which explains a lower mean velocity. For the 30° angle the situation is the same for the mean velocity (increasing here $\sigma$). The reasons here are different, since for a higher distance, the movement is softer and more reliable because vision is used longer, and the generated trajectory has smaller rotations. Using vision produces less error at the end of the robot, which contributes to a faster convergence towards region I. Moreover, for a greater distance, the angle estimation error is smaller because the approximations used have smaller errors. For the 60° angle, the analysis for the mean velocity is the same as for the 30° angle. The standard deviation is much larger than before even though at 2m is significantly lower than for the other distances. Softer movements and low estimation error explain that fact. The worst situation occurs for points 4 and 6 in terms of mean velocity. The lower $v_{mean}$ makes sense, after considering the fact that the robot has to move backwards and that increases the traveled distance.

This grid of points could not explore all the possibilities of the algorithm including the recovery of the estimation error in one step for angles lower than about 110°. So, other points were tested including angles of 80°, 90° and 110°. For all these points, the algorithm worked although sometimes with too much time ranging from 1min till almost 4min for distances of about 1.5m. The minimum iterations on region II was 2 as expected and the maximum was 4. The problem was not with the angle estimation after the first iteration. The problems are mostly related with the odometry and with the noise in states 3, 4 and 5. Remember that when there is an estimation error, the robot might not be aligned after the first iteration, and might enter one of those states (3, 4 or 5). The robot could manage to identify correctly the pyramid but sometimes took longer because it entered state 3 or 4 with other objects before finding the pyramid. The maximum distance was also tested at 0° angle and after adding a point at 2.5m with 100% of success, other distances were tested. At 2.7m the algorithm worked well, at 2.8m worked only in 75% of the trials and at 3m does not work at all. At 3m it is not possible to extract enough information from the image received. For other angles, maximum distance is the 2m radius discussed previously. Finally, the robustness to experimental conditions was tested and the robot was able to dock in an outside environment even though the odometry was not corrected for that soil. Fig. 14 shows a very noisy example for which the algorithm worked. Even for an angle different of 0° (about 25°) it managed to dock in less than 80s even though there are big differences in the terrain.

![Fig. 14. An example of the color detection and segmentation in outdoor environment.](image)

VII. DISSEMINATION ACTIVITIES

There were several dissemination activities including dozens of demonstrations and exhibitions, TV reports and participation on the C-ELROB 2007, Civilian European Land-Robot Trial. Some hardware and software modifications were done in order to prepare C-ELROB 2007. A new wireless antenna was used and a GPS device was put in the robot. C-ELROB was a trial that consisted of searching several ERICards in a urban area of about 1.5ha with several obstacles like cars, army trucks, stairs and gratings. CEFIC Emergency Response Intervention Cards (ERICards) are intended to help firemen on a quick response to chemical transport accidents. These are the cards that are present at trucks for transportation of chemical products. The ranking system took into account the ERICards photographs and GPS coordinates and also the running time of the trial (till a maximum permitted). Among 12 teams, the Idmind-ISR/IST
team was placed in 6th place. It is important to notice that the result achieved had to do with some problems with the original communications protocol. It was in this competition that this problem was much more exposed than in other situations making impossible to operate the robot for time intervals bigger than 15 minutes due to the network congestion. There was a huge amount of tiny packets that, with the retransmissions lead to network congestion. The rhythm of images transmission was decreased from 10 frames per second to only 2 to help diminish the congestion leaving more bandwidth to the sensors and actuators packets but that was not as useful as it would be expected. With fewer images, one has to teleoperate the robot at a slower speed to maintain a good level of responsiveness. Operating at a slower speed led to a more difficult identification of obstacles by the operator and consequently less time to discover ERICards. Besides, the images quality was not very good because of its automatic white balance and the sunny weather. Some plastic protections were tried to cut some sunlight which improved the visibility. Keeping in mind all these constrains, the 6th place was a satisfactory result.

VIII. CONCLUSION AND FUTURE WORK

This thesis presents a broad range of aspects to be dealt with but the main goal of giving more autonomy to the robot was achieved. It implements several improvements that can be very useful to the user. For instance, detecting holes frees some attention of the user to search for victims. In sum, the main contributions of this thesis are:

- new communications scheme with an improvement of 87.5% in the packets rate
- autonomous hole detection
- autonomous docking to the docking mechanism
- hardware and software GPS integration

The hole detection works well and prevents the robot to fall into holes. The autonomous docking module works quite well according to the existent restrictions. Even though, there are things that should be changed starting from the hardware and the electronics.

In terms of future work, the ideal architecture presented before can be useful in developing other modules taking advantage of the modularity of code provided by YARP. The odometry should be a point to improve but that is not possible without changing the electronics to get access to the encoders. With access to the encoders, the $d_{real}$ should be estimated for other kinds of soils. Another important issue for the autonomous docking is the color segmentation. Improving it so that the need to adjust manually the HSV bands thresholds, e.g. because of the lighting conditions, can be eliminated is a good start in order to have better results. The infrared sensors can be changed to others with a bigger range because when the arm is raised up, the detection range diminishes too much. With the GPS, one can use the information of the docking pyramid GPS coordinates in order to get closer when it is not possible to perceive it with the camera. In outdoor environments, a map of the route taken by RAPOSA can be built which can be extremely useful if combined with a GIS (Geographic Information System) system.

REFERENCES