Sanchez Belenguer, C.; Soriano Vigueras, Á.; Vallés Miquel, M.; Vendrell Vidal, E.; Valera Fernández, Á. (2014). GEMA2:Geometrical matching analytical algorithm for fast mobile robots global self-localization. Robotics and Autonomous Systems. 62(6):855-863. doi:10.1016/j.robot.2014.01.009

GEMA^2: Geometrical Matching Analytical Algorithm for Fast Mobile Robots Global Self-Localization

C. Sánchez[a,b], A. Soriano[a], M. Vallés[a], E. Vendrell[a], A. Valera[a]
{carsanb1, ansovi, mvalles, evendrell, avalera}@ai2.upv.es
[a] Instituto U. Automática e Informática Industrial.
Universitat Politècnica de València. Valencia, Spain
[b] Institute for Transuranium Elements. European Commission
Joint Research Centre (JRC). Ispra (VA), Italy

***Abstract* – This paper presents a new algorithm for fast mobile robot self-localization in structured indoor environments based on geometrical and analytical matching, GEMA^2. The proposed method takes advantage of the available structural information to perform a geometrical matching with the environment information provided by measurements collected by a laser rangefinder. In contrast to other global self-localization algorithms like Monte Carlo or SLAM, GEMA^2 provides a linear cost with respect the number of measures collected, making it suitable for resource-constrained embedded systems. The proposed approach has been implemented and tested in a mobile robot with limited computational resources showing a fast converge from global self-localization.**

Keywords Autonomous robots, Robot's Self-Localization, Laser range finders, Geometrical matching, mobile robots.

1 INTRODUCTION

The importance of self-localization for mobile robots with limited computational resources is a well-known problem; especially in indoors environments where globally accurate positioning systems like GPS are not available. Without the information of the exact position and heading (pose) of the robot in the space, it would not be able to navigate in the environment, follow a path or go to a goal point. This issue has been widely studied in the literature ([1], [2], [3], [4], [5]) and there are two mainly strategies of self-localization: position tracking algorithms which know the initial pose of the robot at the beginning of the execution (also known as Local position estimation) and global localization algorithms which work through environment recognition without knowledge of its initial pose (Global position estimation).

Most positional tracking strategies that work with knowledge of the initial pose use Kalman filters ([6], [7]) to perform the sensor fusion to improve significantly the odometry problems adding greater efficiency to the self-localization. Simple or reduced filters can be implemented in limited resource systems but without any correction from a global position system they always have small errors that due to the feedback of these methods increase over time.

The global position strategies for indoors try to self-locate recognizing the environment detected by the robot. These algorithms try to solve problems like the kidnapped robot [8] or wake-up robot [9]. To address these problems, there are two popular methods: the first one is based on the study of algorithms that try to self-locate in a map and at the same time, they try to build the map of the environment; like the known algorithm SLAM [10] (Simultaneous Localization And Mapping). These strategies usually require a high consumption of resources due to the use of cameras or sensors to detect environmental information, which requires costly image processing and complex matching algorithms, so they are hardly applicable to real-time autonomous robots with limited resources. The second solution is based on the refinement of self-localization

within a known map. The most popular algorithm of this kind is the Monte Carlo Localization (MCL) [11] that estimates the pose of a robot as it moves and senses the environment. Typically it starts with a uniform random distribution of particles over the known map and they are resampled based on recursive Bayesian estimation whenever the robot moves or it detects changes in the environment. The particle filter's time complexity depends on the number of particles and, naturally, the more particles, the better the accuracy, so the algorithm has a compromise between speed and accuracy. In systems with limited computational resources is non-viable analyze a large number of particles each time therefore usually or the accuracy is poor, or they use communications to compute the calculations in a computer with more resources.

Self-localization techniques based on robot's observations can be understood as a specific application of surface registration techniques: by calculating the best correspondence between a given observation of the robot and a previous one (or a well-known map of the environment) the relative position of the observer can be obtained by simple triangulation. In this line, multiple descriptors that simplify the search process have been proposed based on curvature [12], [13] or integral invariants [14], [15]. To calculate the descriptor's alignment, there are techniques based in combinatory optimization [15], [16], random algorithms RANSAC [13], [17] or forward search [12]. Results achieved with these techniques provide coarse alignments that are generally refined with algorithms like ICP (Iterative Closest Point) [18], [19], which alternates between the calculation of correspondence between samples and the calculation of the alignment transformation. There are many variations of the original ICP algorithm that can be found properly classified and detailed in [20].

Considering these factors, this paper presents GEMA^2, a new algorithm for analytical self-localization based on the geometrical correspondence of the samples perceived from the environment by a laser rangefinder, in order to provide an efficient global localization algorithm compatible with local strategies and computationally implementable in resource-constrained embedded systems.

The proposed technique takes advantage of the structured nature of indoor environments. This structure is normally introduced by the presence of straight walls and, consequently, this work focuses its effort on developing a fast line extraction algorithm in order to detect them. However, the proposed algorithm can be easily extended to recognize any other type of parametrizable geometric primitives using the same basic idea.

After this review, the paper is organized as follows. In section 2 a general overview of the approach is introduced. In sections 3, 4 and 5, the three different stages of proposed technique are explained. In section 6 the experimental platform, in which this method has been implemented, is presented with detail and the experimental results are analyzed. Finally, conclusions and future works are drafted in the section 7.

2 OVERVIEW

The self-localization technique proposed in this paper runs in three different stages: calibration, segmentation and localization. *Calibration stage* is executed before the robot can move autonomously, and its results can be used in subsequent stages without re-calibrating the sensor. The goal of this stage is to calculate the error function $e(d)$ associated to the distance measurements d_i returned by the sensor.

Once the calibration has been performed, the execution cycle of the robot starts by reading the measures obtained by the laser, and converting the set of local 2D points obtained to a much

simpler representation. This task is performed in the *segmentation stage*, where points s_i are inferred into lines l_i, filtered, and intersected between themselves, in order to calculate a set of corners c_i.

The most defined corner c is then passed to the *localization stage*, where it is aligned with all the similar corners in the known map m_i. Each alignment produces one possible location for the robot $(\theta_{robot}, x_{robot}, y_{robot})$, which is evaluated using a cost function based on squared errors. The alignment with the smallest error is then returned as the location of the robot in the known map.

Figure 1 shows the complete execution cycle of the proposed technique that is explained in this section.

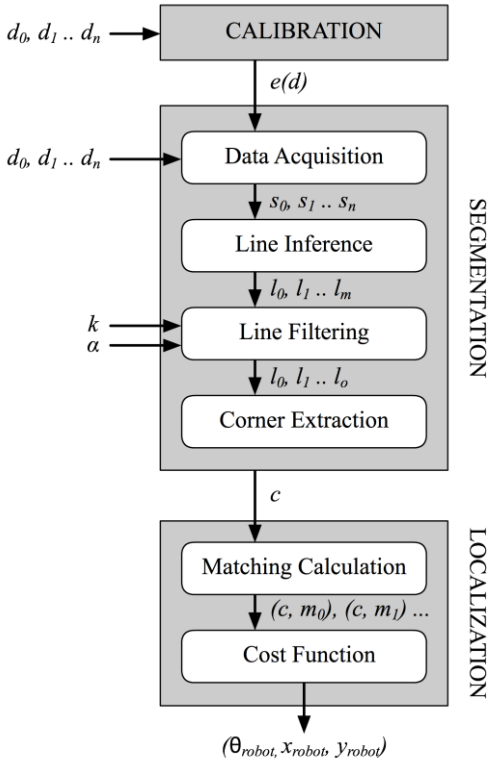

Figure 1: Execution cycle of the proposed technique

3 CALIBRATION

Calibration stage deals with the estimation of the sensor's measurement error. The result of this stage is critical, since line inference algorithm uses this value to establish if a given sample belongs to the line defined by the previous ones or if it belongs to a new line.

The measurement error $e(d)$ is a function that indicates, for each distance d_i, the uncertainty of the returned value. Its value can be a constant, or a function that depends on the measured distance and/or the orientation of the laser beam in local coordinates.

To calibrate the sensor, the robot is placed in front of a straight wall and, using least squares, the equation of the line that best fits the returned measurements, is calculated. The error associated to each distance is calculated as the geometric distance between the point defined by the distance measurement returned by the sensor, and the point defined by the intersection between its projection line and the optimal line calculated using all samples. Figure 2 illustrates this concept.

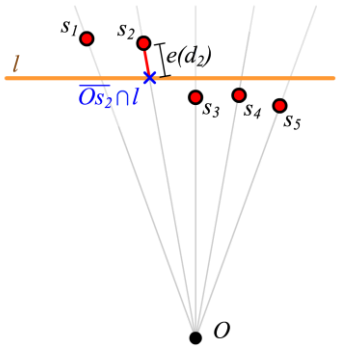

Figure 2: calculation of the error associated to each measurement. In orange is shown the line calculated using least squares.

During the empirical tests with the laser, the error remains constant, no matter the measured distance or the projection angle. This way, a error function as a constant value for all measures has been set, being $e(d) = 2cm.$, which corresponds to the maximum error obtained in the calibration stage.

4 SEGMENTATION

The goal of the segmentation stage is to reduce input data to a much simpler representation that allows to compare it with the known map in an efficient and robust way. For this, noise introduced by the sensor and real world objects imperfections has to be filtered using the error estimation calculated in the calibration stage.

To speed-up localization calculations, the input set of distance measurements returned by the sensor is inferred into a set of straight lines. These lines are intersected with their neighbors to compute a set of corners. The most accurate estimated corner will be used in the localization stage to evaluate the position of the robot in the known map.

4.1 Data Acquisition

A sensor is defined as $S(f, n, e(d))$ where f represents the *FOV* (Field Of View), n is the number of distance samples returned and $e(d)$ is the error function that estimates the measurement error associated to each distance d_i.

Values of f and n are specified by the sensor manufacturer, whilst value of $e(d)$ is calculated in the calibration stage.

The set of distance measurements obtained by the sensor $D = (d_1, d_2 \dots d_n)$ is an ordered array of decimal values. Projection angle φ_i of distance d_i can be calculated as $\varphi_i = f * \left(\frac{i}{n} - \frac{1}{2}\right)$, and its local coordinates (u_i, v_i) as follows:

$$\begin{aligned} u_i &= sin(\varphi_i) * d_i \\ v_i &= cos(\varphi_i) * d_i \end{aligned} \tag{1}$$

Given a set of distances D, a set of samples $S = (s_1, s_2 \dots s_n)$ are calculated in this stage, where each sample s_i is defined by three points in the sensor's local coordinates:

$$\begin{aligned} s_i &= (sin(\varphi_i) * d_i, cos(\varphi_i) * d_i) \\ \lfloor s_i \rfloor &= \big(sin(\varphi_i) * (d_i - |e(d_i)|), cos(\varphi_i) * (d_i - |e(d_i)|)\big) \\ \lceil s_i \rceil &= \big(sin(\varphi_i) * (d_i + |e(d_i)|), cos(\varphi_i) * (d_i + |e(d_i)|)\big) \end{aligned} \tag{2}$$

where $e(d_i)$ is the associated measurement error for each sample.

This way, each sample is characterized by its measured distance to the sensor, s_i, its minimum possible distance, $\lfloor s_i \rfloor$, and its maximum possible distance, $\lceil s_i \rceil$, according to the error estimation performed in the calibration stage.

These samples are used in next stages to perform several calculations. If the required frequency for the proposed technique cannot be ensured, due to computational restrictions, an interesting simplification can be performed at this point: each calculated sample can represent a set of distance measures. Thus, by averaging distances, the amount of samples to process in further stages can be reduced, and some random noise can be filtered.

4.2 Line inference

The goal of this stage is to calculate a set of straight lines $L = (l_1, l_2 \dots l_m)$ that best fits the set of samples calculated during the acquisition stage, considering the measurement error.

The main difficulty associated to these calculations is to establish if a given sample s_i belongs to the line obtained considering previous samples s_{i-j}, or if it belongs to a new line.

To solve this problem, an estimator called *line visibility* is associated to each line, l_k, defined by two scalar values $V_{l_k} = (d_{min}, d_{max})$. This estimator specifies the range of projective distances in which sample s_i will be considered as part of the line defined by previous samples. This way, the following can be established:

$$s_i \in l_k \leftrightarrow [\lfloor s_i \rfloor \dots \lceil s_i \rceil] \cap [d_{min} \dots d_{max}] \neq \emptyset \qquad (3)$$

being $[d_{min} \dots d_{max}]$ the range of distances specified by V_{l_k}, and l_k the line defined by previous samples $(s_{i-1}, s_{i-2}, \dots)$. Values of $[\lfloor s_i \rfloor \dots \lceil s_i \rceil]$ represent the uncertainty associated to the position of s_i, considering its associated measurement error. Figure 3 illustrates this concept.

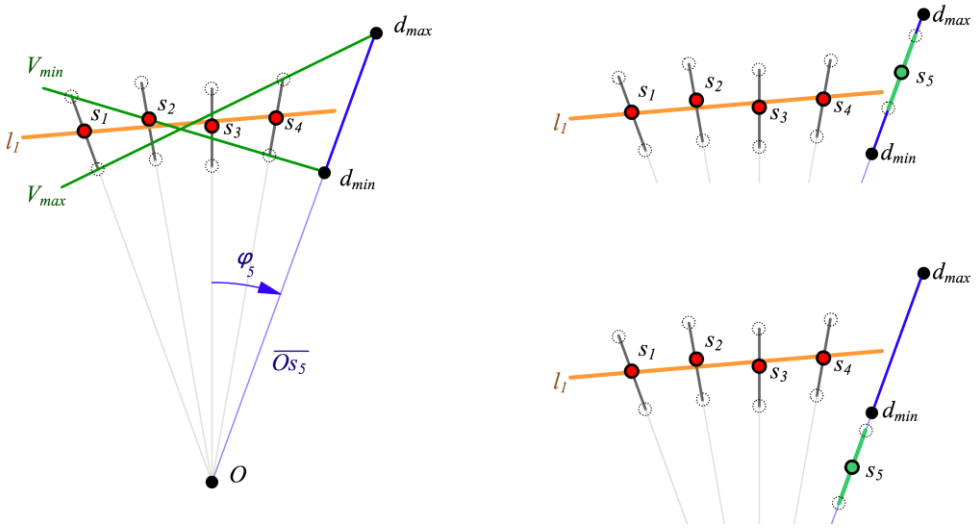

Figure 3: use of the *line visibility* concept. Red circles correspond to s_i values, dashed circles correspond to $\lfloor s_i \rfloor$ and $\lceil s_i \rceil$ values. Dark grey lines represent the uncertainty associated to each sample $[\lfloor s_i \rfloor \dots \lceil s_i \rceil]$. (Left) graphical representation of V_{l_1} considering all illustrated samples. (Right-up) a new sample s_5, represented in green, that satisfies equation *(1)* and, thus, belongs to the line l_1. (Right-down) a new sample s_5 that does not satisfy equation *(1)*, so it does not belong to the line l_1.

Given a set of consecutive samples $S = (s_{i-j} \dots s_{i-2}, s_{i-1})$, to compute values of d_{min} and d_{max} for the next sample s_i, it is necessary to find the minimum/maximum slope lines, $V_{min}$and V_{max} respectively, that intersects all ranges $[\lfloor s_k \rfloor \dots \lceil s_k \rceil], s_k \in S$.

The intersection points between these lines and the line $\overline{Os_i}$, defined by φ_i and marked in blue in Figure 1 (left), correspond to d_{min} and d_{max}, being $d_{min} = V_{min} \cap \overline{Os_i}$ and $d_{max} = V_{max} \cap \overline{Os_i}$.

This way, if a sample s_i fails equation (1), it means that it cannot exist one line that intersects all ranges $[\lfloor s_k \rfloor \dots \lceil s_k \rceil], s_k \in (S \cup s_i)$ and, consequently, the sample must belong to a new line.

To infer L, all samples are iterated in order, creating new lines when it is necessary. Code 1 illustrates this process.

If a sample s_i passes equation (1), defined by the line that considers previous samples $(s_{i-j} \dots s_{i-2}, s_{i-1})$, it is then included in the line, and values of d_{min} and d_{max} are updated considering the new sample.

If s_i fails the test established in equation (1), a new line is created that only includes s_i, and the equation of the previous line is calculated using least squares for fitting, according to the projective distance measured for each included sample (s_i).

```
L ← {∅};
l ← new Line(∅);
foreach Sample s_i in S do
    d_min ← V_l.min;
    d_max ← V_l.max;
    if ([⌊s_i⌋ ... ⌈s_i⌉] ∩ [d_min ... d_max] ≠ ∅) then
        l.AddSample(s_i);
        l.RecalculateV_l();
    else
        l.LeastSquareFit();
        L.add(l);
        l ← new Line(s_i);
    end if
end foreach
l.LeastSquareFit();
L.add(l);
```

Code 1: proposed clustering algorithm to infer the set of lines *L* that best fits the set of samples *S*

This way, by using $\lfloor s_i \rfloor$ and $\lceil s_i \rceil$, the *line visibility* estimator that splits the original sample set into smaller clusters of aligned samples is calculated. Then, by using s_i distances, the line that best fits all samples inside each cluster is calculated too and this set of lines is the result of this stage.

One of the advantages of this technique is that it first produces the set of aligned samples, and then, it calculates the equation of the optimal line that best fit them. This is important, since Ordinary Least Squares (or Linear Least Squares), minimizes the sum of squared vertical distances between observations and the responses predicted by linear approximation. Thus, the more horizontal samples are distributed, the more accuracy. In order to optimize the results of this technique, the set of samples of a cluster can be rotated, then approximated, and then the optimal line is rotated back again.

To calculate values of d_{min} and d_{max}, minimum and maximum slope lines that intersect all $[\lfloor s_i \rfloor \dots \lceil s_i \rceil]$ ranges have to be calculated. Given that the way of calculating these lines is symmetrical, the calculations are focused on finding out the value of V_{max}.

The first thing to consider when updating V_{max} consists in differentiating two different cases, according to the new sample (s_j) position: (A) if $[\lfloor s_j \rfloor \dots \lceil s_j \rceil]$ intersects V_{max} no update is necessary, since $V_{max}$is already the maximum slope line. (B) If not ($\lceil s_j \rceil < d_{max}$), V_{max} is no longer the maximum valid slope line, and has to be re-calculated. In this case, it is important to notice that the new maximum slope line is defined by two points: $\lfloor s_p \rfloor$ and $\lceil s_q \rceil$, which are the most restrictive projective distances. Figure 4 illustrates this concept.

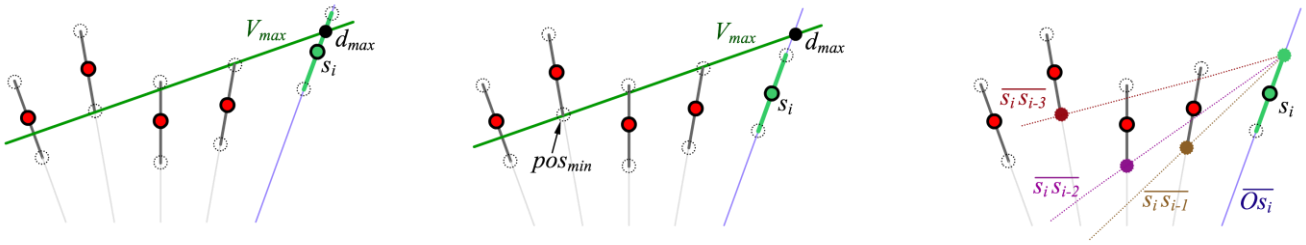

Figure 4: updating of V_{max}: (Left) update case A, where no computation is required. (Center) Update case B, where V_{max} has to be re-calculated. (Right) Calculation of the new value for V_{max}. In the proposed example, V_{max} is the line defined by $\overline{\lceil s_i \rceil \lfloor s_{i-3} \rfloor}$.

Calculating the value of $\lceil s_q \rceil$ is immediate ($\lceil s_q \rceil = \lceil s_j \rceil$), since $\lceil s_j \rceil < d_{max}$.

To calculate the value of $\lfloor s_p \rfloor$, it is necessary iterate through each sample included in the current line, looking for the point $\lfloor s_i \rfloor$ that minimizes the angle between the line $\overline{\lceil s_j \rceil \lfloor s_i \rfloor}$, and the projection line defined by $\overline{Os_j}$.

Considering that the addition of samples to the line is an iterative process, an interesting optimization can be performed: once a sample has been identified as the most restrictive one, none of the previous samples needs to be considered in subsequent iterations. This way, by storing the index of the most restrictive sample, a lot of unnecessary evaluations can be avoided, ensuring a correct result. Code 2 illustrates this process.

```
function CalculateV_max(s_j: Sample, V_max: Line)
    d_max ← V_max ∩ Os_j;
    if (⌈s_j⌉ < d_max) then
        angle_min ← ∞;
        for pos in pos_min .. j − 1 do
            if (Angle(⌈s_j⌉⌊s_pos⌋, Os_j) < angle_min) then
                angle_min ← Angle(⌈s_j⌉⌊s_pos⌋, Os_j);
                pos_min ← pos;
            end if
        end for
        V_max ← ⌈s_j⌉⌊s_pos_min⌋;
    end if
    return V_max;
end function
```

Code 2: pseudo-code to update value of V_{max}. Consider that pos_{min} is a global variable that keeps its value after each invocation of the function.

In terms of computational complexity, the worst-case scenario for the proposed algorithm consists in a set of samples perfectly aligned: since all of them will pass the visibility test, there will only be one line. When updating V_{min} and V_{max}, all new samples will force to recalculate the lines, and the most restrictive sample will always be the first one, so the final execution cost is expected to be $O(n^2)$, being n the number of samples. However, the worst-case scenario is extremely unusual to happen, even impossible when using a sensor with FOV values similar or greater than 180º.

In the best-case scenario, each sample included in a line will not force to update values of V_{min} and V_{max} so, the execution cost is expected to be $O(n)$, being n the number of samples.

4.3 Line filtering

The goal of this stage is to add robustness to the proposed technique, considering new uncertainties not contemplated when calculating the error function in the measures of the sensor $e(d)$. These uncertainties are mainly related to physical imperfections in both, the sensor and the environment.

During the empirical tests, these imperfections have been classified in two cases: (a) interferences in the laser rangefinder due to the topology and material of the measured area and, (b) imperfections in the walls that lead to split them in several separated lines.

Interferences in the scanner happen when the laser beams intersect reflective/refractive surfaces. The resulting distance measures are unpredictable, and add a lot of noise to the input data. Also, at practice this precision decreases significantly when scanning a sharp corner.

Imperfections in the walls make theoretically planar surfaces to be curved. This affects the line inference technique by making that, eventually, one sample of the same wall fails the visibility test defined by the previous ones. The result is that a wall is then characterized by more than one line.

By detecting and properly filtering these imperfections, the quality of results improves significantly. Figure 5 illustrates these two kinds of interferences using real data, some examples of them, and the result of applying the proposed filtering.

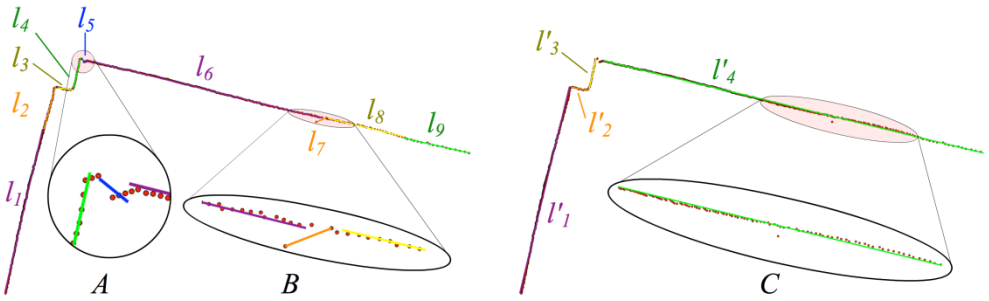

Figure 5: imperfections in the sensor measures that affect the line inference calculations. (left) result of the previous stage without considering these imperfections. (right) result after filtering. Case *A* shows the error of the sensor when scanning a sharp corner. Case *B* the consequences of reflections in the line inference algorithm. Case *C* shows a wall that deviates in both directions.

To filter interferences in the scanner, the fact that an anomalous measure seriously penalizes the *line visibility* estimator it is considered. Thus, a line containing a *bad sample* has a very limited amount of samples in it (cases *A* and *B* in Figure 1), so all lines containing less than k samples are discarded ($|l_i| < k$), and so are the samples within them. The value of k is a parameter specified by the user, and empirical tests have proven that values in the range $[4 \dots 6]$ provide good results.

After removing the lines created by interferences in the scanner, imperfections in the walls are filtered using a second parameter, α, that indicates the maximum tolerance for wall deviations. All consecutive lines with an angular deviation smaller than α, are combined and recalculated considering all samples.

The result of this stage is a new set of lines $L' = (l'_1, l'_2 \dots l'_o)$ that improves the segmentation obtained in $L = (l_1, l_2 \dots l_m)$, and that verifies that $o \leq m$.

4.4 Corner extraction

To efficiently compare the resulting set of lines L' with the known map, a global registration between the two datasets is necessary. One of the most common techniques to perform this operation is to simplify the representation of both datasets into simpler ones. This way, alignment calculations are considerably accelerated.

Instead of working directly with lines, the use of *corners* is proposed to perform this operation. A corner is defined by the intersection between two consecutive lines, and characterized by its inner angle (β), its position (x, y) and its orientation (θ).

To obtain the set of corners $C = (c_1, c_2 \dots c_p)$ defined by $L' = (l'_1, l'_2 \dots l'_o)$, all lines are intersected between them, storing only the intersections that are close enough to the endings of both lines. All calculated corners maybe do not exist in the real world but they can be very helpful in the location algorithm (e.g. this allows to avoid interferences caused by small pillars like the one shown in Figure 5).

Given the interest in accelerating global registration, instead of using all corners, only the best characterized is used: c. To find this one among all C, the *quality* of a corner is defined as $|c_i| = \min(|l_1|, |l_2|)$, being l_1 and l_2 the lines that generate it, and $|l_1|$ and $|l_2|$ the amount of samples of each line, respectively.

$$\begin{aligned} c &= max(|c_i|)\,, c_i \in C \\ |c_i| &= min(|l_1|, |l_2|)\,, l_1 \in c_i, l_2 \in c_i \end{aligned} \tag{4}$$

5 LOCALIZATION

The localization stage takes, as input data, the resulting corner obtained in the segmentation stage and a pre-calculated map, where all the corners have been identified. By aligning these corners with the one obtained in the previous stage, all possible locations for the robot are inferred.

For each location, a measure of correctness is calculated, so they can be sorted according to the quality of achieved results. The most correct result is expected to be the robot localization in the real world.

5.1 Matching calculation

This stage deals with the calculation of all possible locations of the robot in the real world. To do so, all corners in the known map with a similar inner angle to the one obtained in the segmentation stage are aligned, producing each alignment a localization for the robot.

In order to accelerate calculations, the known map has been pre-processed so all corners have been obtained and sorted according to their inner angle. This way, finding similar angles computational cost is reduced to $O(\log_2(n))$, being n the total number of corners in the map. In

case the map is too big and, to simplify this search, once the location of the robot has been calculated only corners around the last known position have to be checked.

Given an alignment between corners, the robot orientation (θ_{robot}) is calculated as follows:

$$\theta_{robot} = \frac{\pi}{2} + \theta_m - \theta_c - \frac{(\beta_m - \beta_c) * |l_{1_c}|}{|l_{1_c}| + |l_{2_c}|} \quad (5)$$

where θ_m is the orientation of the selected corner in the map (in global coordinates), θ_c is the orientation of the corner calculated in the segmentation stage (in local coordinates), β_m is the inner angle of the selected corner in the map, β_c is the inner angle of the corner calculated in the segmentation stage, and $|l_{1_c}|, |l_{2_c}|$ are the amount of samples contained in the lines that define the corner calculated in the previous stage.

By calculating θ_{robot} as proposed, the difference between inner angles is compensated proportionally to the amount of samples of each line that define the corner c. It is expected that, the more samples a line contains, the more accurate the estimation is.

Then, the robot position (x_{robot}, y_{robot}) is inferred by simple triangulation, as shown below:

$$\begin{aligned} x_{robot} &= -x_c * \cos(\theta_{robot}) + y_c * \sin(\theta_{robot}) + x_m \\ y_{robot} &= -x_c * \sin(\theta_{robot}) - y_c * \cos(\theta_{robot}) + y_m \end{aligned} \quad (6)$$

Where x_m and y_m are the global coordinates of the intersection point defined by the selected corner in the map, and x_c and y_c are the local coordinates of the intersection point defined by the corner calculated in the segmentation stage. Figure 6 illustrates these concepts.

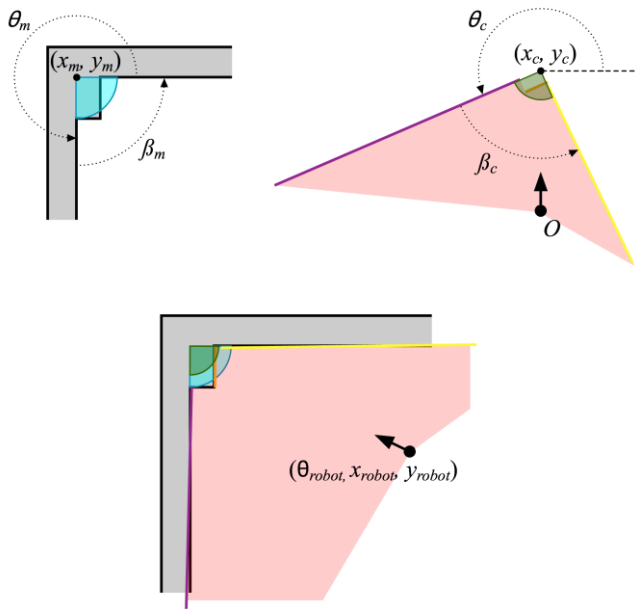

Figure 6: alignment between corner c and a corner from the known map m. (Top-left) characterization of m, in global coordinates. (Top-right) characterization of c in local coordinates. (Bottom) correspondence between both corners and robot pose calculation.

5.2 Cost function

This is the last stage in the proposed technique, and it is responsible to evaluate the quality of each pose calculated in the previous one. The final result of the localization algorithm is a sorted list of possible poses, where the first element is expected to be the real world's robot position.

To quantify the quality of each possible pose calculated using the known map, the set of measures $D' = (d'_1, d'_2 \dots d'_n)$ that the sensor should have produced in ideal conditions (no measurement error), given the position and orientation calculated.

The total error of a given orientation can then be calculated as:

$$error = \sum_{0 \le i < n} (d_i - d'_i)^2 \tag{7}$$

being d_i the original measures returned by the sensor.

In case the computational cost of these calculations does not satisfy the execution frequency required, as it occurs at some robots with limited computational resources only a subset of equispaced distances can be used.

6 EXPERIMENTS

In order to verify the correct functionality of GEMA^2 algorithm, it has been implemented and analyzed in an experimental platform. This platform is based on a Robotino® mobile robot equipped with a laser rangefinder sensor.

Robotino is a FESTO mobile robot system with three omnidirectional drives [21]. The robot controller consists of an embedded PC running with a Linux operating system installed on a compact flash card. In order to improve its functionality, Robotino is also equipped with several types of sensors, like infrared distance measuring sensors, incremental encoders, anti-collision sensor, analogue inductive proximity sensor, camera module, etc.

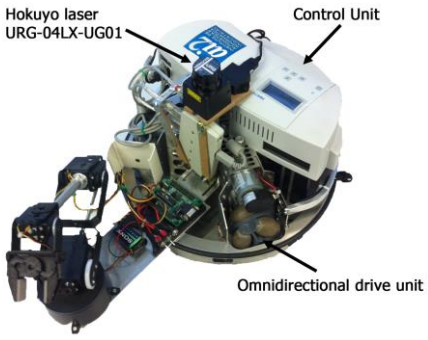

Figure 7: Robotino with Hokuyo laser rangefinder installed.

Robotino is driven by 3 independent, omnidirectional drive units. They are mounted at an angle of 120° to each other. The drive units allow for motion in all directions –forward, backward and sideways – and the robot can be turned on the spot as well. Each of the 3 drive units consists of a DC motor, a gear unit, an all-way roller, a toothed belt and an incremental encoder unit. Actual motor speed can be compared with desired speed by means of the incremental encoder, and can then be regulated with a PID controller.

The controller unit consists of 3 components:

- PC 104 processor, compatible with MOPSlcdVE, 300 MHz, and Linux operating system with real-time kernel, SDRAM 128 MB and Ethernet, USB and VGA interfaces.
- Compact flash card that contains the operating system, the functions libraries (C++ API) and the included programs for controlling Robotino.
- Wireless LAN access point.

For accessing the functions, libraries, execution programs and Linux environment on the PC 104, a terminal program via the WLAN connection has been used. With this program it is possible to edit the source code, compile and execute the Robotino control functions.

In order to read the measures for the proposed automatic localization algorithm, the Hokuyo URG-04LX-UG01 has been used [22]. It is a laser sensor for area scanning. The light source of the sensor is an infrared laser, and the scan area is a 240º semicircle with a maximum radius of 5600mm. Pitch angle is 0.36º and sensor outputs the distance measured at every point (maximum: 684 steps). Principle of distance measurement is based on calculation of the phase difference, due to which it is possible to obtain stable measurements with minimum influence from object's color and reflectance.

The Hokuyo laser rangefinder is connected via USB to the controller unit of Robotino. For software integration, a library offered at [23] has been used, which provides the basic methods of connection and receiving laser measurements. The connection is made using the COM port assigned to the laser, and measures are asynchronously received with a period of 400ms per event. At each event 684 measures are stored and processed.

Several tests with the Hokuyo with Robotino were performed to calculate the actual computational cost of the algorithm based on the number of samples collected by the laser. Figure 8 shows the relation between the time used by the algorithm according to the number of measures processed by Robotino's processor and the global correction of the technique respect to the sampling ratio, in an outlier-free environment.

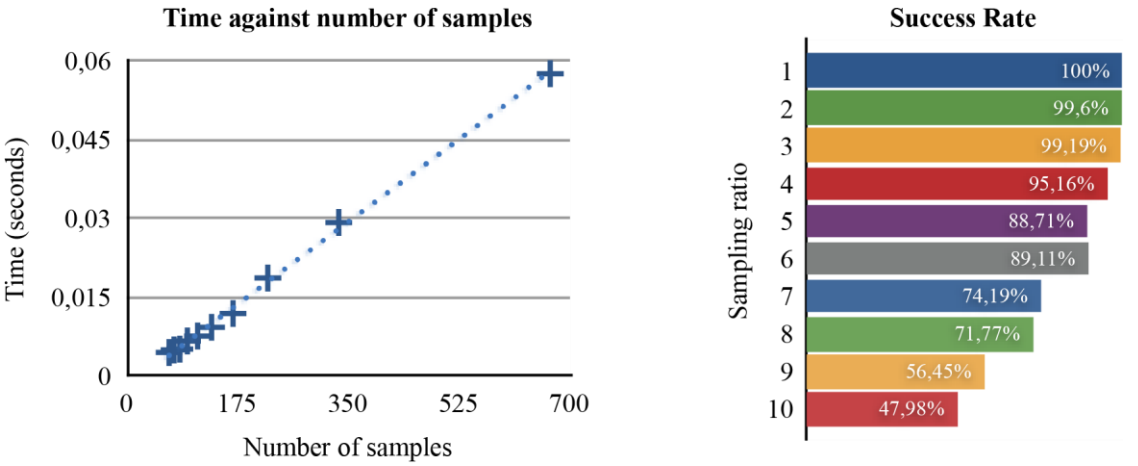

Figure 8: (left) Several tests to show the relation between the execution cost of the algorithm in Robotino processor and the number of samples collected by the Hokuyo sensor. (right) Global correction of the achieved results, depending on the sampling ratio used in an outlier-free environment.

As it can be seen, the computation cost is linearly related to the number of samples. In the worst case, using 668 measures from the laser, GEMA^2 takes 0,058s to run, which is not a problem for Robotino processor and it allows to locate the robot in each iteration of the control loop (control loop period is set at 100ms). Also, global correction of the technique decreases, as the number of samples considered is reduced.

To analyze in depth the computational cost of each phase of the algorithm, Figure 9 shows the cost distribution of all stages for each test performed according to the number of samples. For example, for 668 samples, the total time is 0.058s. The data acquisition stage occupies 0.00464s (8% of the total time), the lines inference algorithm 0.0377s (65%), filtering, corner detection and localization 0.00116s (2%) and the cost function occupies the rest: 0.0145s (25%). However, for 66 samples the total time is 0.0044s, with a 0.00136s for the cost function (31%), 0.00015s for filtering, corner detection and localization (3.5%), 0.00242s for line inference (55%) and 0.00046s for acquisition (10.5%).

Analyzing the complete graph, the line inference stage is the heaviest one, with 65% of time on average. The cost function stage occupies 25% on average and the acquisition 10%. The filtering, corner detection and localization phases are the fastest, with 2% of the algorithm execution time.

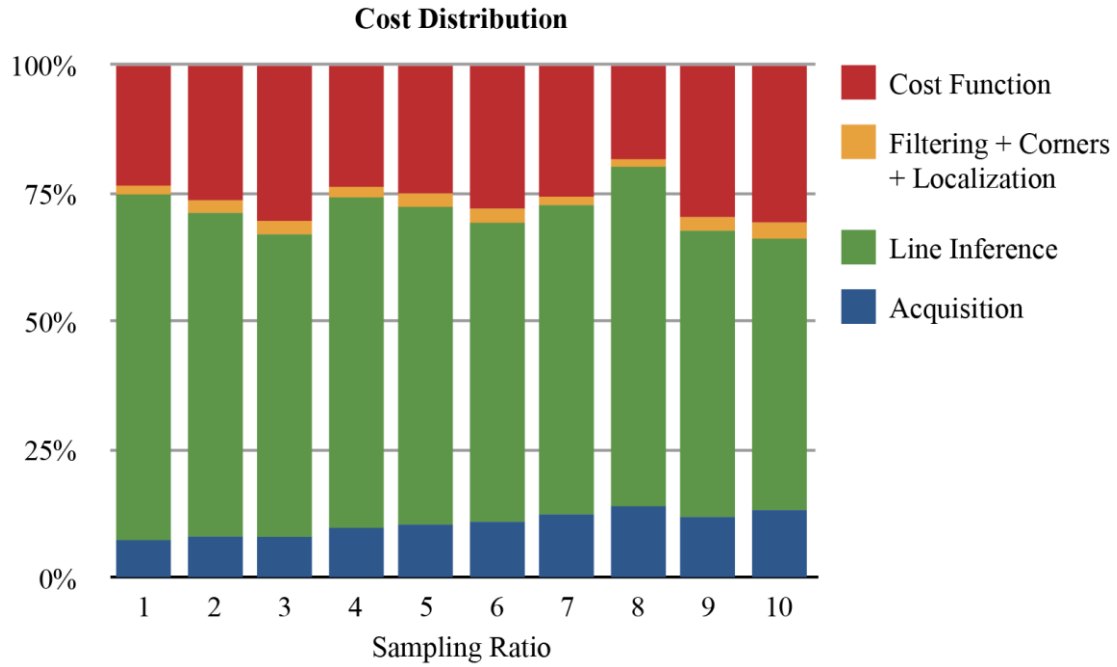

Figure 9: measures about the cost distribution of the algorithm for different number of samples collected by Hokuyo.

To estimate the accuracy of the proposed technique, an offline registration algorithm that uses as many ICP iterations, has been applied using the same data collected by the robot in actual testing. This algorithm iterates over itself until it converges to the actual position of the robot.. By checking that all paired points of the ICP algorithm in the last iteration are correct, it ensures that the achieved solution is the global optima, so it can be used as ground truth to evaluate the accuracy of the technique. Table 1 shows the actual accuracy in different scenarios calculated by this technique. These scenarios include an outlier-free environment, an environment with moving people in front of the robot, a scenario with some pieces of furniture, not included in the ground truth map and, finally, a scenario with moving flat outliers.

On the other hand, to evaluate the robustness of GEMA^2, the impact of outliers in several scenarios where the ground truth map does not fully correspond with the sensor readings, has been tested. Achieved results show that the effect of outliers can be classified in two major groups: (1) the presence of an outlier “breaks” a wall, so the total amount of samples used to infer the line is reduced causing several non-consecutive segments. (2) If the outlier is a flat surface it can be considered as a false wall leading to a misinterpretation of the environment. Figure 10 shows both cases.

Post-processing the extracted lines and merging together aligned segments can attenuate the effects of case 1. This, of course, will require extra computing time. However, empirical tests have shown that results of the proposed technique without this post-processing stage are very robust when using a full sampling ratio as Table 1 shows.

The tests performed to evaluate this case include static and dynamic outliers, with the robot in static and dynamic locations. It is important to notice that, since time is not considered (each iteration previous data is ignored), there is no distinction between moving and static outliers.

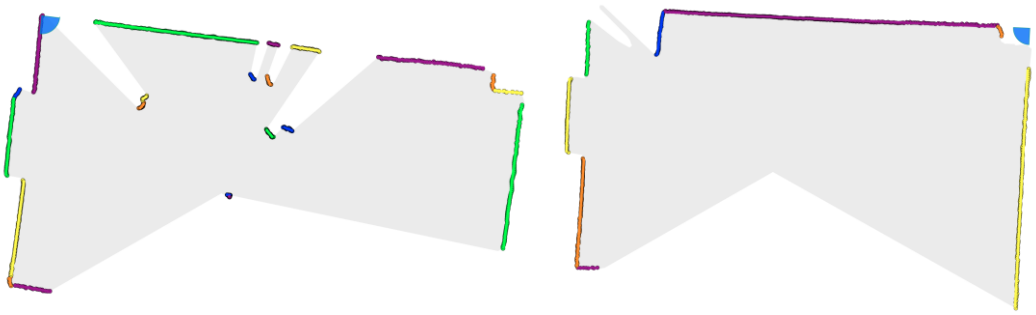

Figure 10: (left) Three people standing in front of the robot. Notice how the longest wall is splitted into four different segments. (Right) A box, represented with a blue line, leaning against a wall and misinterpreted as a wall.

For case 2, the effect of misinterpreting a flat outlier as a wall is very dependent on several factors: the portion of the environment visible for the sensor, the size and distance of the outlier respect to the sensor and the ambiguity introduced by the location of the outlier: if the outlier is isolated in the middle of an empty space, or if it makes an angle with other line segment that does not exist in the ground truth map, the risk of misinterpretation is very low.

Next table shows the results achieved for the different evaluation scenarios, considering all the samples collected by the sensor (sampling ratio=1). Success Rate measures the robustness of the technique, since a result is only considered successful if the paired corners are correct. Accuracy is the Euclidean distance between the estimated location, and the correct one. Only correct results are considered for estimating the accuracy.

Test scenario	Success Rate	Accuracy (cm.)
Without outliers	100%	3,16
Moving people	97,67%	7,87
Furniture	88,66%	5,13
Flat outliers	95,99%	5,75

Table 1: results achieved for the different evaluation scenarios with sampling ratio =1.

It must be taken into account that the GEMA^2 algorithm is not using any feedback. Each iteration the previous estimated position is ignored and a new one is calculated. Thus, the algorithm does not accumulate any error between iterations and it does not need to know any previous estate. Considering previous estimations might accelerate the execution, evaluating only the part of the map close to the latest known position. This is being considered for future work, as well as the extension for recognizing other different primitives rather than lines.

7 CONCLUSIONS

A global self-localization fast analytical algorithm for resource-limited systems has been proposed: GEMA^2. The algorithm needs a laser or any other environment analysis sensor for autolocate within a known map. Contrary to other algorithms as Montecarlo, which uses particle

filters hardly implementable in embedded systems, or SLAM algorithm which requires many resources to run, GEMA^2 algorithm allows to autolocate analytically by a geometric matching, achieving a linear cost in relation to the number of measurements taken.

Given the low computational cost, this algorithm can be used cyclically in a separate thread or it can be launched by events, for example when the positioning error by odometry is too high, because it does not need any feedback.

Although the algorithm gives in most cases a unique positioning, in cases where there is an ambiguity between multiple solutions, results of the algorithm can be merge by Kalman filters with those obtained by odometry or other sensors such as gyroscopes, accelerometers, compass…

To illustrate the practical experiment with Robotino platform and some robustness tests with different scenarios with outliers, two videos ("GEMA2-Demo.mp4" and "GEMA2-OutliersTests.mp4" respectively) are available and accompany the electronic version of this manuscript. Supplementary material related to this article can be found online at "link to electronic version of the paper".

ACKNOWLEDGEMENTS

This work has been partially funded by FEDER-CICYT projects with references DPI2011-28507-C02-01 DPI2010-20814-C02-02 and HAR2012-38391-C02-02 financed by Ministerio de Ciencia e Innovación and Ministerio de Economía y Competitividad (Spain).

REFERENCES

[1] R. Siegwart and I. Nourbakhsh, *Introduction to autonomous mobile robots*. The MIT Press, 2004.

[2] I. Skog and P. Handel, "In-car positioning and navigation technologies - a survey," Intelligent Transportation Systems, IEEE Transactions on, vol. 10, no. 1, pp. 4 –21, march 2009.

[3] Praneel Chand, Dale A. Carnegie, Mapping and exploration in a hierarchical heterogeneous multi-robot system using limited capability robots, Robotics and Autonomous Systems, Volume 61, Issue 6, June 2013, Pages 565-579.

[4] I. Loevsky, I. Shimshoni, Reliable and efficient landmark-based localization for mobile robots, Robotics and Autonomous Systems, Volume 58, Issue 5, 31 May 2010, Pages 520-528

[5] Mohsen Mirkhani, Rana Forsati, Mohammad Shahri, Alireza Moayedikia, A novel efficient algorithm for mobile robot localization, Robotics and Autonomous Systems, Available online 30 April 2013.

[6] M. S. Grewal and A. P. Andrews., Kalman Filtering: Theory and Practice Using Matlab. John Wiley & Sons, 2001.

[7] G. Welch and G. Bishop, An Introduction to the Kalman Filter, University of North Carolina at Chapel Hill,, http://www.cs.unc.edu/~welch/kalman, March 2007.

[8] S.P. Engelson and D.V. McDermott (1992). "Error correction in mobile robot map-learning". In: *ICRA 1992. Proceedings of the international conference on robotics*. ISBN 0-8186-2720-4. pp. 2555-2560.

[9] R. Negenborn (2003). Robot localization and kalman filters. Institute of Information and Computing Sciences, Utrecht University.

[10] R. Smith, M. Self and P. Cheeseman, "Estimating uncertain spatial relationships in robotics". IEEE International Conference on In Robotics and Automation (1987).

[11] F. Dellaert, D. Fox, W. Burgard and S. Thrun. MonteCarlo localization for mobile robots. Proc. IEEE International Conference on Robotics and Automation (ICRA-99), Detroit, MI (1999).

[12] Ran Gal and Daniel Cohen-Or. 2006. Salient geometric features for partial shape matching and similarity. ACM Trans. Graph. 25 (January 2006), 130–150. Issue 1.

[13] Xinju Li and Igor Guskov. 2005. Multi-scale features for approximate alignment of point-based surfaces. In Proceedings of the third Eurographics symposium on Geometry processing. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, Article 217.

[14] Qi-Xing Huang, Simon Flöry, Natasha Gelfand, Michael Hofer, and Helmut Pottmann. 2006. Reassembling Fractured Objects by Geometric Matching. ACM Trans. Graphics 25, 3 (2006), 569–578.

[15] Natasha Gelfand, Niloy J. Mitra, Leonidas J. Guibas, and Helmut Pottmann. 2005. Robust global registration. In Proceedings of the third Eurographics symposium on Geometry processing. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, Article 197.

[16] Ikuko Shimizu Okatani and Akihiro Sugimoto. 2005. Globally convergent range image registration by graph kernel algorithm. In In International Conference on 3D Digital Imaging and Modeling. 377–384.
[17] Y. Shan, B. Matei, H. S. Sawhney, R. Kumar, D. Huber, and M. Hebert. 2004. Linear Model Hashing and Batch RANSAC for Rapid and Accurate Object Recognition. In IEEE International Conference on Computer Vision and Pattern Recognition. 121–128.

[18] Paul J. Besl and Neil D. McKay. 1992. A Method for Registration of 3-D Shapes. IEEE Trans. Pattern Anal. Mach. Intell. 14 (February 1992), 239–256. Issue 2. DOI:http://dx.doi.org/10.1109/34.121791

[19] Yang Chen and Gérard Medioni. 1992. Object modelling by registration of multiple range images. Image Vision Comput. 10 (April 1992), 145–155. Issue 3. DOI:http://dx.doi.org/10.1016/0262-8856(92)90066-C

[20] Szymon Rusinkiewicz and Marc Levoy. 2001. Efficient Variants of the ICP Algorithm. In Third International Conference on 3D Digital Imaging and Modeling (3DIM).

[21] Education and Research Robots: Robotino® (2013). Available on line: http://www.festo-didactic.com

[22] Scanning Laser Range Finder URG-04LX-UG01, Hokuyo Automatic co., ltd. (2009). Available on line: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html

[23] The Robotino forum (2013). Available on line: http://openrobotino.org/