Mobile Networks and Applications manuscript No.
(will be inserted by the editor)

The Feasibility of Navigation Algorithms on Smartphones using J2ME

André C. Santos · Luís Tarrataca · João M. P. Cardoso

Received: date / Accepted: date

Abstract Embedded systems are considered one of the areas with more potential for future innovations. Two embedded fields that will most certainly take a primary role in future innovations are mobile robotics and mobile computing. Mobile robots and smartphones are growing in number and functionalities, becoming a presence in our daily life. In this paper, we study the current feasibility of a smartphone to execute navigation algorithms and provide autonomous control, e.g., for a mobile robot. We tested four navigation problems: Mapping, Localization, Simultaneous Localization and Mapping, and Path Planning. We selected representative algorithms for the navigation problems, developed them in J2ME, and performed tests on the field. Results show the current mobile Java capacity for executing computationally demanding algorithms and reveal the real possibility of using smartphones for autonomous navigation.

Keywords Navigation algorithms · visual landmark recognition · particle filter · extended kalman filter · potential fields · smartphones · J2ME

André C. Santos
IST - Technical University of Lisbon
Avenida Prof. Dr. Cavaco Silva, 2780-990 Porto Salvo, Portugal
E-mail: acoelhosantos@ist.utl.pt

Luís Tarrataca
IST - Technical University of Lisbon
Avenida Prof. Dr. Cavaco Silva, 2780-990 Porto Salvo, Portugal
E-mail: luis.tarrataca@ist.utl.pt

João M. P. Cardoso
Departamento de Engenharia Informática
Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
E-mail: jmpc@acm.org

1 Introduction

Autonomous navigation is an important aspect for mobile robotics and for mobile devices with the goal of helping the user to navigate in certain environments. A mobile device such as a smartphone can be used to guide the user in museums, shopping centers, exhibitions, city tours, and emergency scenarios when a catastrophe occurs; to control more effectively home appliances like vacuum cleaners; to assist impaired people, etc.

Currently, most navigation problems require solutions dependent of computationally demanding algorithms. Despite their recognized complexity, those algorithms have not been fully analyzed in the context of smartphones. Thus, this paper presents a performance study of four navigation algorithms when implemented using J2ME technology for mobile devices. To test those algorithms on the field, we use a system composed by a mobile robot and two smartphones. In this system, a smartphone executes the navigation algorithms and sends control instructions to the mobile robot using *Bluetooth*. A second smartphone acts as an intelligent visual sensor, communicating processed visual information to the former smartphone. A schematic of the system organization is presented in Figure 1.

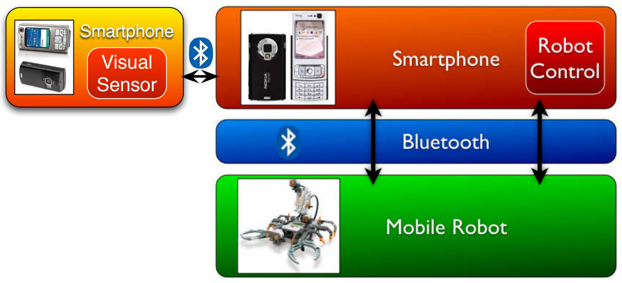

Fig. 1 Organization of the system components.

By developing and studying J2ME implementations of navigation algorithms on smartphones, we hope to be contributing to a clear understanding about the current capabilities of high-end smartphones and J2ME, and possibly to highlight future improvements on both.

The remainder of this paper is organized as follows: Section 2 gives an overview of navigation algorithms; Section 3 presents the experimental setup used, followed by Section 4 with the algorithms implemented; Section 5 shows experimental results and Section 6 presents the main conclusions.

2 Autonomous Navigation

Autonomous navigation has been widely focused by the mobile robotics area [15]. Navigation is defined as the process or activity of accurately ascertaining one's position, planning and following a route. In robotics, navigation refers to the way a robot acquires a perception of the environment where it is immersed in, finds its way and is able to move itself in that environment [15]. Navigation is a common necessity and requirement for almost any mobile robot.

Leonard and Durrant-Whyte [13] briefly describe the general problem of mobile robot navigation by three questions: ”Where am I going?”, ”Where am I?” and ”How do I get there?”, each one relative to a navigation subproblem: Mapping, Localization and Path Planning, respectively.

2.1 Mapping

The mapping problem exists when the robot starts without a map of the environment where it is immersed in and incrementally builds one as it navigates. While in movement, the robot senses the environment and identifies key features which allow it to register information of its surroundings. The main concern for the mapping problem is how the mobile robot does perceive the environment. There are many sensors used for mapping, being the most common sonar, digital cameras and range lasers. The complexity of the mapping problem is the result of a different number of factors [23], being the most important: size of the environment, noise in perception and actuation, and perceptual ambiguity.

Mapping approaches based on images captured by a visual sensor (e.g., digital camera), have been mostly accomplished through the extraction of natural features from the environment or through the identification of special artificial landmarks. Artificial landmarks mainly consist of solid-color or pattern-based features. Solid-color landmarks [18] are in most cases easily identifiable, but can sometimes be blend into the environment and become less detectable. In contrast, patterns [1,8] are less blurred with the background environment and can also be easily identified. An approach based on natural landmarks existent in the environment [14, 20] is a more computationally demanding task, but benefits from the fact that there is no need for an artificial preparation of the environment.

2.2 Localization

Localization is the process of estimating where the robot is relatively to some model of the environment, and using the available sensor measurements. As the robot keeps moving, the estimation of its position drifts and changes, and has to be kept updated through active computation [15]. These updates are performed based on the recognition of special features in landmarks, sensor data and probabilistic models.

Localization uncertainty rises from the sensing of the robot, because of the indirect estimation process. The measurements besides being noisy, because of real-world sensor characteristics, may not be available at all times. Based on the uncertainty characteristics of the localization problem, similarly to other important mobile robotics problems, localization has been tackled by probabilistic methods [23]. Amongst the most commonly used are Markov Localization [5] and Particle Filters [7].

2.3 Simultaneous Localization and Mapping (SLAM)

SLAM [21] involves both localization and mapping and constitutes a technique used by autonomous robots to build up a map within an unknown environment while at the same time keeping track of their current position. This approach is complex since it involves both localization and mapping simultaneously, both with uncertainties associated. One main concern in SLAM is keeping the uncertainty controlled, for both robot position and landmark position, in order to diminish errors as much as possible. For this double uncertainty, SLAM normally uses methods based on Extended Kalman Filters (EKF) [9,27] and Particle Filters [7]. There are currently further developments on the SLAM technique, being FastSLAM [16], an optimization worth mentioning (which is based on EKF and Particle Filters).

2.4 Path Planning

Path Planning is the process of looking ahead at the outcomes of possible actions, and searching for the best sequence that will direct the robot to a desired goal location [15]. It involves finding a path from the robot's current location to the destination. The cost of planning is proportional to the size and complexity of the environment. The bigger the distance and the larger the number of obstacles, the higher the cost of the overall planning. Path Planning techniques for navigation can be divided into local path planning and global path planning. They mainly differ on the quantity of information of the environment they need to possess. Local techniques only need information of the environment that is near to the robot, while global techniques use full information of the environment.

There are many different approaches to path planning, as they try to solve the problem using different techniques. Two examples of Path Planning techniques are the Artificial Potential Field [10] and the technique based on Ant Colony [4].

3 Prototype

Our prototype consists of a mobile robot, a Lego Mindstorms NXT[1] kit, coupled with two smartphones (we have used the Nokia N80[2] and the Nokia N95[3]). One smartphone is coupled to the mobile robot and positioned so its built-in camera faces the front of the robot, enabling it to act as an intelligent image sensor (see Figure 2). The other smartphone, which is the main processing unit, responsible for executing the navigation algorithms, does not need to be physically coupled to the mobile robot. The robot and the two smartphones communicate with each other via *Bluetooth*, requiring always to be located within communication range.

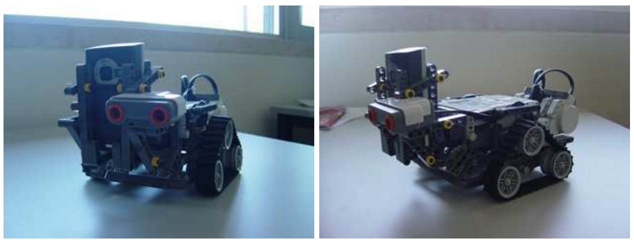
Fig. 2 Prototype mobile robot with a smartphone.

Development for the smartphones was done in Java using its Micro Edition version (J2ME[4]). The choice of J2ME development was mainly due to Java's known portability among the most common mobile phone manufacturers. Development for the mobile robot was also done using a subset of Java supported by the JVM present in the custom firmware leJOS NXJ [22], for the Lego's NXT Brick.

3.1 NXT Middleware

A middleware component is responsible for the interaction between the smartphones and the mobile robot. The navigation algorithms are executed in the central smartphone and the control instructions are passed to the robot via the middleware. Raw data from sensing by the robot is acquired by the middleware via a *Bluetooth* interface. A block diagram representation of the middleware is presented in Figure 3.

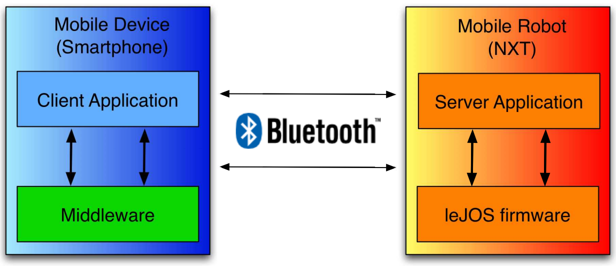

Fig. 3 NXT Middleware Platform.

The core functionality of the mobile robot middleware consists in providing abstractions for *Bluetooth* communication and also access to the mobile robot's sensors and actuators. The main operations supported include: establishing and ceasing connections; moving forward, backward and rotating; and data acquisition from the available sensors. A code snippet is provided in Figure 4. The middleware component was developed in the Java programming language and was built on top of the leJOS NXJ firmware [22].

```
MobileClientNXT nxtClient = new
MobileClientNXT(BluetoothAddress);
nxtClient.connect();
nxtClient.forward(velocity);
nxtClient.rotate(angle);
nxtClient.stop();
nxtClient.disconnect();
```

Fig. 4 Example middleware client code snippet.

4 Navigation Algorithms Considered

In order to tackle the autonomous navigation problem, we approached mapping, localization, simultaneous localization and mapping, and also path planning, each with a method chosen to be adapted to a smartphone constrained environment. For mapping, we based our technique on visual color identification of artificial landmarks. For localization, we used the particle filter probabilistic approach. For simultaneous localization and mapping, we used an Extended Kalman Filter (EKF) approach. For path planning, we focused on the potential fields technique.

4.1 Visual Landmark Recognition

For real-time mapping we rely on feature extraction by the visual sensor. With the objective of keeping the detection

[1] `http://mindstorms.lego.com/`
[2] `http://www.nseries.com/products/n80/`
[3] `http://www.nseries.com/products/n95/`
[4] `http://java.sun.com/javame/technology/`

and recognition of the landmarks as fast as possible, the approach implemented in this work uses solid-color cylindrical artificial landmarks. The approach developed is similar to the method used by [3]. In our approach, the visual system detects a landmark, recognizes its color, and calculates its distance and orientation to the visual sensor.

Color segmentation represents the first step for detecting a landmark on each captured image by the smartphone built-in camera. Previously, the landmark color features were gathered and analyzed, providing a way to empirically produce a set of rules in the RGB color space for detecting the colors used in the artificial landmarks. These rules detect the presence of a landmark in an image, thus allowing the corresponding landmark classification based on its color. E.g., for a green landmark we use the rules presented in Equation (1), where R, G and B correspond to the red, green and blue color components of the RGB color space. The value X is an adjustment value, used to increase the green color component relatively to the red and blue components.

$$(G \geq 130) \text{ and } (G > R + X) \text{ and } (G > B + X) \tag{1}$$

The color segmentation process transforms the captured image into a binary, black and white image, as can be seen in Figure 5. White color pixels indicate the presence of the green range color and black pixels the absence of it.

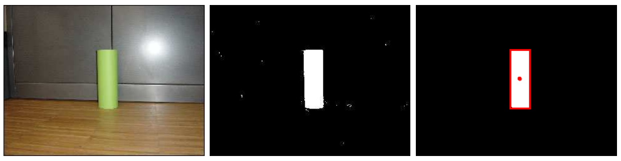
Fig. 5 Image captured with a green landmark (left image); binary image after the application of the color segmentation (middle image); and landmark boundary detection after the application of the image noise reduction filter (right image).

Reducing the noise present in an image is a necessary step for the elimination of salt and pepper noise, caused by the color segmentation stage. The noise may negatively influence results in future image processing stages and therefore needs to be removed or at least reduced. The filter implemented uses a 3×3 scanning window, that analyzes all the landmark pixels present in the image. The window checks if the pixels surrounding the current scanned pixel mostly belong to the landmark or to the background. If the neighborhood pixels are mostly background ($\geq 50\%$) then the pixel is most likely noise and is erased (see Figure 5).

For a more accurate calculations of distance and orientation, we use a minimum rectangular boundary containing the shape detected. This technique is used to help cope with some small variations in the shape's perspective, that can vary due to the angle from which the image was acquired.

Both distance (d) and orientation (θ) measures are calculated from the visual sensor to the landmark and are based on a method proposed by [28]. The landmark distance and orientation information can be inferred from the landmark's size and position in the image (see Equations (2) and (3)), by knowing the width (x'), height (y') and center point of the landmark $(x_{LandmarkCenter})$, and having previously performed measurements for camera calibration (k, m and l are scaling and adjustment factors).

$$d_y = k_y \times \frac{1}{y'} \qquad d_x = k_x \times \frac{1}{x'} \qquad d = \frac{d_x + d_y}{2} \tag{2}$$

$$\theta = m \times x_{LandmarkCenter} + l \tag{3}$$

In Figure 6 we present the pseudocode for a mapping algorithm that uses visual landmark recognition. The mobile robot has to explore all map positions in order to try to acquire the map for the whole environment. At each location, the mobile robot needs to capture an image, to perform the landmark recognition and to build the map with the new information obtained.

```
  Name  : Mapping Algorithm
  Output: Environment Map
1 for all possible positions do
2     CaptureImage ();
3     LandmarkRecognition ();
4     BuildMap ();
5 end
```

Fig. 6 Pseudocode for the Mapping algorithm.

4.2 Particle Filter

For localization, we use a Particle Filter method, based on the one presented in [19]. The environment is represented as an occupancy grid map, where each grid cell matches an area of the real environment at a specific ratio. Each grid cell can be assigned with estimation probabilities of the mobile robot's position or with a possible presence of an obstacle.

The Particle Filter method can be divided into three main stages: Prediction stage which involves a motion and noise model for movement; Update stage which concentrates on sensing the environment and altering the particles relevance weight value; and a Resample stage where the particle population is managed. At any given time a position estimate can be acquired using different techniques.

The Prediction stage involves two different models. A motion model, responsible for the robot's path planner. This model is responsible for providing at each step a path for the mobile robot's movement. In this work, two motion models were developed: an explorer type model which visits all free locations in the map, and a point-to-point model which is a predefined obstacle-free path from one location in the environment to another. A noise model, responsible for reflecting odometry errors, is added to the robot's motion and its implementation is based on the model provided in [19]. The odometry errors considered were divided into rotation errors and translation errors. Both errors were experimentally established from the real odometry errors from the mobile robot used. Translation and rotation with noise are accomplished using a pseudo-random value, drawn as a sample from a Gaussian distribution.

The Update stage is represented by a measurement model which provides, on each observation of the environment, necessary information for a function that updates the particles weights. In this implementation, a particle's weight is considered to be a numeric value *w* greater than 0. An observation consists on sensing the environment. Sensing is done by using a simulated observation from the information present in the internal map or by using the visual landmark recognition method presented earlier.

Resampling occurs when a considerable amount of particles within the particle population have weight values below a threshold and therefore have low contribution to the overall estimation of the robot's position. The resampling process recognizes particles with small weight values $(< threshold)$ and replaces them with a random particle, whose weight value is higher than the resampling threshold $(\geq threshold)$. This random replacement minimizes the problem of diversity loss. When all particles have weights below the defined *threshold* then a new random set of particles is generated.

In our approach, at a certain time, t, the position estimate is given by the best particle, i.e., the one in the the current particle set having the maximum weight value. There could have also been used estimations of the robot's position using a weighted mean considering all particles or a robust mean. The robust mean is in fact the weighted mean limited by a small window around the best particle.

The pseudocode for the Particle Filter algorithm is presented in Figure 7. The ESS function used in Figure 7 represents the Effective Sample Size and can be calculated by Equation (4). The coefficient of variation (cv_t^2) can be calculated by Equation (5). The variables M and i represent the total number of particles and the i^{th} particle, respectively. Recall that w represents a particle's weight.

$$ESS_t = \frac{M}{1+cv_t^2} \tag{4}$$

$$cv_t^2 = \frac{var(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(M \times w(i) - 1)^2 \tag{5}$$

In the algorithm, lines $4-7$ are responsible for the Resample phase; lines $8-10$ are relative to the Prediction phase; lines $11-14$ are for the Update phase; and lines $15-17$ are used to normalize the weights of the particles.

```
Name: Particle Filter Algorithm
Input: A set of Particles $i$ at $t = 0$
       $(S_i^0 = [x_j, w_j] : j = 1 \ldots M)$
1  $W = w_j : j = 1 \ldots M$;
2  while Exploring () do
3      k = k + 1;
4      if ESS (W) $< \beta * M$ then
5          Index = Resample (W);
6          $S_i^t = S_i^t(Index)$;
7      end
8      for j = 1 to M do
9          $r_j^{t+1} = \hat{f}(r_j^t, \alpha)$;
10     end
11     s = Sense ();
12     for j = 1 to M do
13         $w_j^{t+1} = w_j^t * W(s, r_j^{t+1})$;
14     end
15     for j = 1 to M do
16         $w_j^{t+1} = \frac{w_j^{t+1}}{\sum_{j=1}^{M} w_j^{t+1}}$;
17     end
18 end
```

Fig. 7 Particle Filter Algorithm (adapted from source: [19]).

4.3 Potential Fields

For path planning we use the Potential Fields approach [10]. This approach is amply used for path planning and collision avoidance due to its mathematical simplicity and elegance, providing acceptable and quick results [12] in real-time navigation. This method is based upon the concept of attractive and repulsive forces, where the goal is seen as a global minimum potential value (attractive force), and all obstacles as high valued potential fields (repulsive force). The movement of the robot is then defined by the potential values present in its path, moving ideally from high to low potentials. Examples of potential field functions are: Khatib's FIRAS function [10], Superquadratic potential function [26], and Harmonic potential function [11].

The pseudocode of the algorithm for the potential fields is depicted in Figure 8. Our approach uses as basis the potential field functions presented by [6]. The potential field functions used are defined as follows: $U_{Total}(p)$ denotes the total scalar potential field; $U_{Att}(p)$ the attractive scalar potential field; $U_{Rep}(p)$ the repulsive scalar potential field; $F_{Total}(p)$ the total vector potential force which is equal to the negative gradient (∇) of the total potential field; $F_{Att}(p)$ the attractive vector potential force; $F_{Rep}(p)$ the repulsive vector potential force; and p the position $[x, y]$ of the robot.

$$U_{Total}(p) = U_{Att}(p) + U_{Rep}(p) \tag{6}$$

$$F_{Att}(p) = -\nabla U_{Att}(p) \tag{7}$$

$$F_{Rep}(p) = -\nabla U_{Rep}(p) \tag{8}$$

$$F_{Total}(p) = F_{Att}(p) + F_{Rep}(p)$$
$$= -\nabla U_{Total}(p) = -\left[\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right] \quad (9)$$

Input : Robot Position (rp), Goal Position (gp), Obstacle Positions (op[])
Output: Directional Action Vector

```
1 while NotInGoalPosition do
2     $F_{Att}$ ← AttractivePotentialForce (rp, gp);
3     $F_{Rep}$ ← RepulsivePotentialForce (rp, op[]);
4     $F_{Total}$ ← $F_{Att}$ + $F_{Rep}$;
5     Velocity ← DetermineVelocity ($F_{Total}$);
6     Angle ← DetermineAngle ($F_{Total}$);
7     UpdateRobotPosition (Velocity, Angle) ;
8 end
```

Fig. 8 Potential Fields approach.

The most difficult problem for the Potential Field method, known as the local minima, has been addressed using escape techniques (e.g., Random Escape, Perpendicular Vector Escape [25], Virtual Obstacle Concept Escape [17]). In order to provide a smoother robot movement, a lookahead function was implemented which prevents the mobile robot from falling into local minima locations by detecting them in advance. Other recognized problems and limitations of this method include: difficulty in considering different obstacle geometry, e.g., concave shaped obstacles; obstacle grouping and closely positioned obstacles; and goals non-reachable with obstacles nearby. Note, however, that the implementation studied in this work is considered representative, although being kept as simple as possible.

4.4 Extended Kalman Filter (EKF)

The most complex problem in mobile navigation is the simultaneous localization and mapping (SLAM). In this case, the robot builds the map of the environment, while it tries to localize itself on the map. Thus, the robot starts by neither having the map of the environment nor its localization.

One of the most well-known approaches to SLAM is the use of the Extended Kalman Filter (EKF) algorithm [9, 21]. We evaluate here the use of an EKF implementation based on the C implementation proposed in [2]. The J2ME implementation uses float data types and emulates the input robot's position information from an internal odometer and a set of landmarks from sensors. As output, it produces improved robots position estimations and a stochastic map based on landmarks (features). Those features are representations of physical objects such as corners, walls, etc.

A feature extractor generates the distance, the orientation, and the signature of an observed landmark relative to the robot's local $x-y$ coordinates.

EKF is a very complex computational task. Its computational complexity scales quadratically with the number of landmarks (features) in the map. EKF requires Jacobian calculations and heavily involves matrix operations. Although improvements over EKF have been proposed (e.g., the FastSLAM [16] optimization), we adopt the EKF as a reference and representative solution.

5 Experimental Results

In this section, we present and discuss experimental results for representative approaches to the navigation problems of mapping, localization, SLAM and path planning. Here, we evaluate the performance of the algorithms developed, by comparing executions between the used smartphones and a desktop PC (equipped with an AMD Athlon 64 X2 Dual Core Processor at 2.20 GHz with 1GB of RAM), and analyzing the feasibility of using smartphones for real-time autonomous navigation. A first study of performance was done with profiling results gathered from the Sun Java Wireless Toolkit[5] PC MIDP[6] (Mobile Information Device Profile) emulator. Next, we conducted several experiments on the field (environment shown in Figure 9).

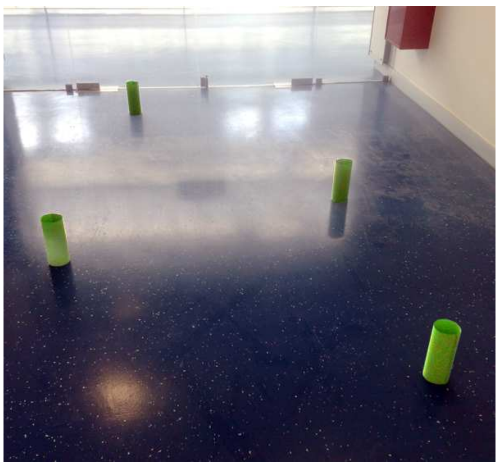

Fig. 9 Field environment for testing of the navigation algorithms.

Complementary to the evaluation of the algorithms, an assessment of the computational power for the two smartphones used was also conducted by analyzing both camera and low-level operations.

5.1 Smartphone Performance

J2ME grants access to a phone's camera through the Mobile Media API (MMAPI[7]) library, which provides audio,

[5] http://java.sun.com/products/sjwtoolkit/
[6] http://java.sun.com/products/midp/
[7] http://java.sun.com/products/mmapi/

video and other multimedia support to resource constrained devices. Applications which rely on image acquisition and processing, to be successful, need the underlying device to be able to execute heavy algorithms as well as acquire images at a significant acquisition rate. Factors like image processing and camera performance in J2ME should be carefully analyzed as they carry possible bottleneck implications for the feasibility of applications that use them.

Considering the evaluated smarpthones, the Nokia N95 boasts a Texas Instrument's OMAP2420 chipset, contrasting with the Nokia N80 which features an OMAP1710. The former, OMAP2420, features an ARM1136 processor core clocked at 330 MHz whilst the latter, OMAP1710, features an ARM926TEJ clocked at 220 MHz.

Considering camera performance it is important to notice that current MMAPI implementations do not support the access to frames in video capture mode, preventing a stream-oriented video acquisition. Nevertheless, MMAPI enables video snapshot acquisition. MMAPI also allows the access to RGB pixel values from an image in order to proceed with data processing. These correspond to the steps required in order to execute some useful computation over a given image, and as such the combined total time can be interpreted as representing the acquisition time [24]. Figure 10 illustrates the average acquisition times obtained for Nokia's N80 and N95 models.

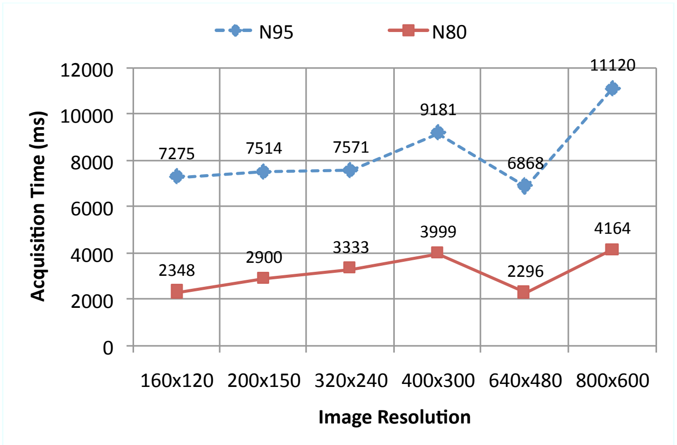

Fig. 10 Image acquisition time for different image resolutions.

It should be noted that the N80 and N95 smartphones boast different cameras, respectively with 3 and 5 megapixels, and also with unique sets of characteristics, such as auto-focus and red-eye reduction, that could not be directly manipulated using the API. In order to provide accurate results regarding camera performance, it would have been important to control such features. We were unable to achieve image resolutions superior to 800 $\times$ 600, since higher resolutions requests generated a media exception.

Both models present relatively high acquisition times even for such low resolutions as 160 $\times$ 120, making it impossible to meet real-time requirements, and showing the need for fast video acquisition in order to access individual frames at high rates. Also noteworthy and without apparent justification is the fact that both models present an acquisition time drop for the 640 $\times$ 480 resolution.

Regarding J2ME processing performance in the N80 and N95, we measured the processor's execution time for basic low-level operations to determine the overall speed and to identify the fastest and slowest operations. Operations tested considered integer data types and consisted on: access an array; increment a variable; add two variables; use bit shift operators to multiply and divide by a power of two; and compare two variables (equal, less or equal, less).

The results presented in Figure 11 reveal average execution times for each operation, obtained by measuring individual repetitive runs.

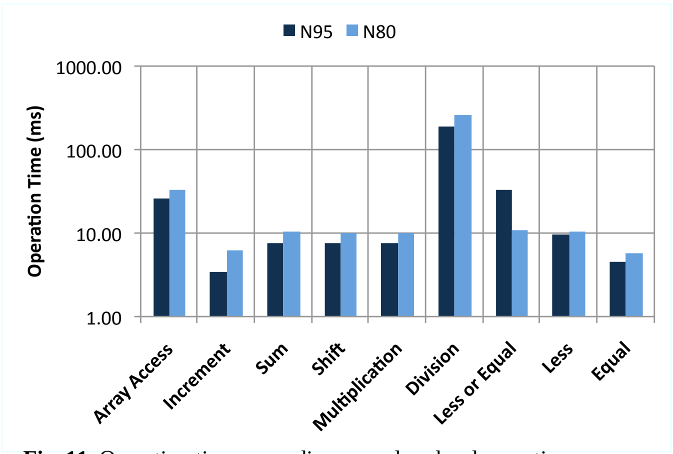

Fig. 11 Operation times regarding some low-level operations.

For the operations considered, division is the slowest operation in both smartphone models followed respectively by array accesses, comparison operators and finally the other arithmetic operations. Although the N80 and N95 feature different base ARM processors and one expected to see better performance from the latter, it was still surprising to see that regarding the less or equal operation the N95 was more than three times slower than the N80. Regarding the remaining operations it is possible to see a clear performance superiority from the N95.

5.2 Mapping

We show here mapping experiments that test the application of the Visual Landmark Recognition method while trying to map the environment presented in Figure 9. Tests executed indicated good identification of the landmark colors, as well as possible problems due to illumination variations, which were the main sources for the incorrect identifications.

Figure 12 presents profiling results for capturing an image and applying the landmark recognition stages. As can be seen, most of the time was consumed on two stages: 51% of the execution time was spent on color segmentation, and 21% on camera image capture.

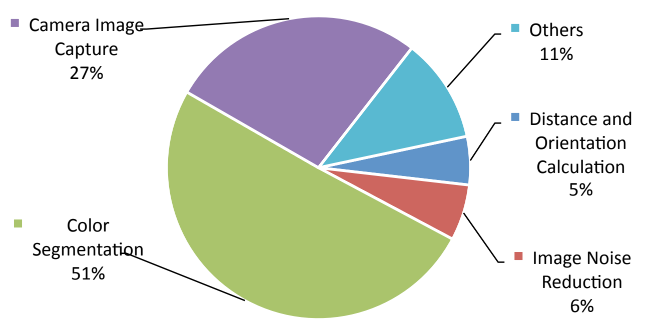

Fig. 12 Contribution to the overall execution time of each step associated to the Visual Landmark Recognition algorithm.

Table 1 compares the execution time obtained when running the algorithms on the PC, on a Nokia N80, and on a Nokia N95. Obviously, the PC is the fastest to execute the application. Comparing the two smartphones, execution time is slower in the Nokia N95 compared to the N80. As seen in previous testing, the N95 has a more complex built-in camera with higher resolution, making it slower when capturing an image with J2ME.

Table 1 Execution time measurements for the Visual Landmark Recognition method.

	PC	Nokia N80	Nokia N95
Execution time (*ms*)	453.00	3,079.40	5,824.30

Using a single captured image and considering a good landmark detection and color segmentation process, the distance calculation revealed quite accurate presenting an average relative error of 5.72%. The angle orientation measurement revealed reasonably accurate with an average relative error of 10.06%. When testing on-the-field, the robot's physical movement and variable lighting conditions prevent the method from achieving its best results. Although this solution cannot be considered a very reliable method for accurate mapping purposes in real-time mobile robot navigation, it presents typical mapping tasks and it is used here as a benchmark for studying the performance obtained by the two smartphones used.

Figure 13 shows the achieved mapping accuracy using Visual Landmark Recognition on the environment presented in Figure 9. The grid depicts the obstacles as black colored cells, obstacle estimates calculated in the cells marked with an "X", and the path taken by the mobile robot is presented marked with numbers.

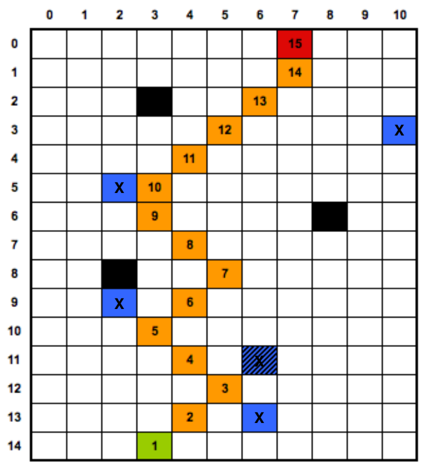

Fig. 13 Mapping results with estimated landmark positions.

The visual sensing revealed itself as time demanding and cannot, without further optimizations, be used to navigate mobile robots at high speed. Nevertheless, considering a slower motion, this solution was able to provide a mechanism for mobile robot mapping.

5.3 Localization

Experiments for Localization were conducted considering only a global localization approach based on the Particle Filter method.

For profiling the Particle Filter implementation, we considered a total number of $1,000$ particles and using the environment in Figure 14.

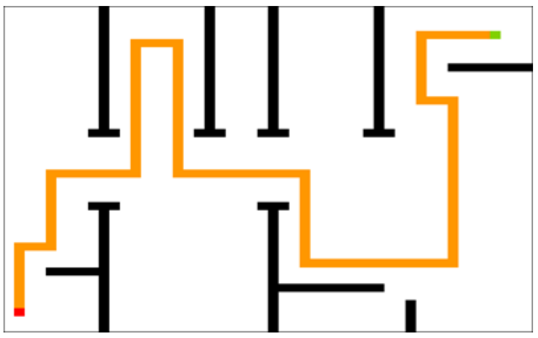

Fig. 14 Occupancy grid map for Particle Filter (50×40 with robot initial location at the top right and target location at the bottom left).

Also, we consider that the robot position estimation is only performed at the end of the mobile robot's predefined movement. Figure 15 presents the percentages of execution time of the main phases of the Particle Filter method. Table 2 presents the execution time comparison when running the implemented localization approach on a PC, on a Nokia N80, and on a Nokia N95.

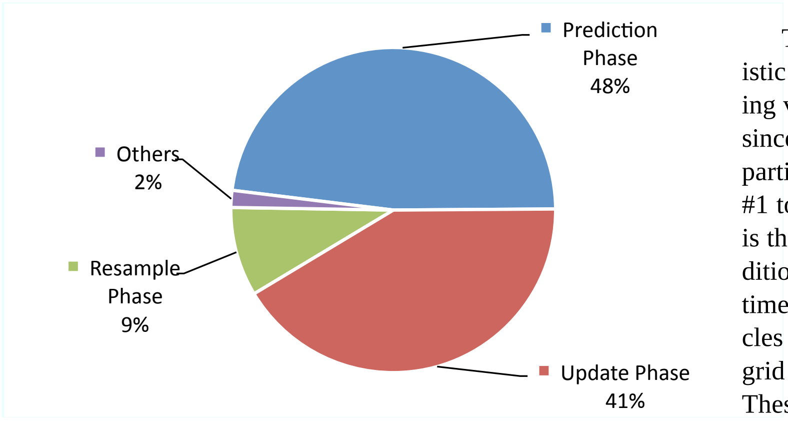

Fig. 15 Contribution to the overall execution time of each phase of the Particle Filter.

Table 2 Execution time measurements for the Particle Filter method.

	PC	Nokia N80	Nokia N95
Time (ms)	78.00	3,618.00	1,725.00

According to the experiments, the phase responsible for the highest percentage of execution time was the Prediction phase with 48%. The Update phase followed with 41%. Finally, and considering the number of particles used and their distribution within the environment, the Resample phase took 9% of the total execution time. The last 2% is spent by auxiliary tasks and by the attainment of the pose estimate (calculating the best particle).

Our next experience uses the Particle Filter method to localize the mobile robot in the environment presented in Figure 9. The localization approach is implemented as a distributed system, were the Particle Filter approach is executed on a Nokia's N95 smartphone, considering 1,000 particles; and the measurement model as a visual sensor with the landmark recognition method, running on the Nokia N80.

Results for five executions of this field experiment are presented in Table 3. Consider that the positions are given as $x-y$ coordinates and θ orientation: $[x;y;\theta]$. The robot's real position at the end of the predefined path is $[7;0;90°]$.

Table 3 Estimations for the same real position ($[7;0;90°]$) for tests on-the-field using the Particle Filter method.

Experiment	Best Particle Position
#1	$[4;0;90°]$
#2	$[7;0;90°]$
#3	$[9;1;90°]$
#4	$[4;0;90°]$
#5	$[0;11;180°]$

By analyzing Table 3, we can observe that one of the experiments estimated the robot to be at its exact physical location. In the other four experiments, three were relatively close to the robot's real position, and the last one was very far from it.

The random initialization of the particles is a characteristic that makes the method difficult to predict, by providing very different results on different runs of the algorithm, since areas in the environment can be highly populated with particles while others deprived from them (see experiments #1 to #5 in Table 3). One possible solution to this problem is the increase of the number of particles, but with high additional computational costs. Figure 16 shows the elapsed time for each loop iteration considering three sets of particles (100, 1000, and 10000) for five sizes of the occupancy grid (50×40, 64×64, 128×128, 256×256, and 500×400). These measurements are related to the execution of the particle filter algorithm with the Nokia N95.

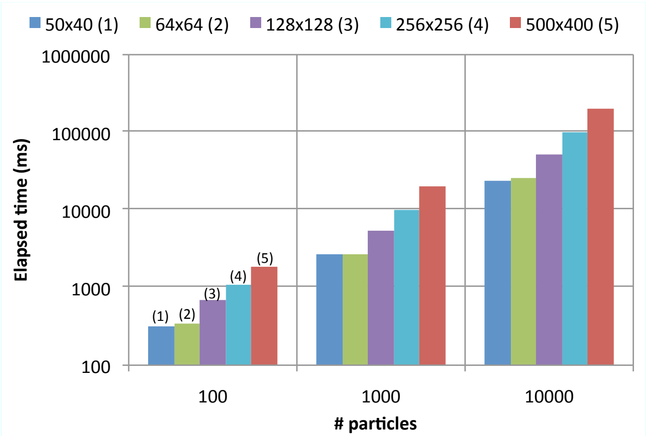

Fig. 16 Particle Filter iteration elapse time for a variable number of particles and map size.

For a 500×400 map and using 10,000 particles, the algorithm requires 1.4s to perform the particle filter stages. This result makes evidence that the smartphone is able to effectively use the particle filter algorithm in real-time for maps of this or lower dimension and using a significant number of particles.

5.4 Simultaneous Localization and Mapping

As previously referred we tested an EKF solution to deal with the SLAM problem. The EKF uses heavily 2-D matrix operations such as multiplications, transpositions, etc. Figure 17 shows the execution time of each iteration of the main loop of the EKF varying the number of features (landmarks).

As can be seen, the number of features influences tremendously the execution time (a quadratically increase with the number of features, as expected) and may impose unacceptable delays between successive readings of sensor data. Despite the heavy use of 2-D matrixes and operations between matrixes, the EKF implementation was very stable for the two smartphones used in the experiments. Once again the N95 model achieved better performance than the N80.

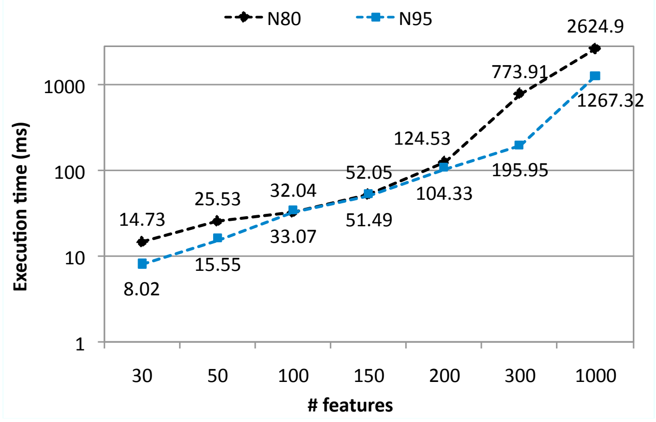

Fig. 17 EKF execution time for a variable number of features.

The profiling results show the heavy impact of the matrix multiplications (92% of the overall execution time is typically spent by this operation). Figure 18 illustrates the contribution of the more significant functions of the EKF when not considering the 92% spent by the matrix multiplications. The most time consuming function of the second group are the matrix subtractions (30%), followed by matrix prediction (14%), addition of features (14%), transpositions (13%), and realloc 2D (12%).

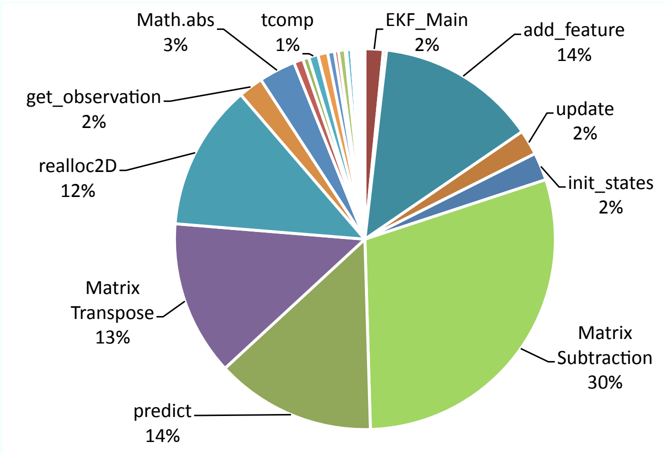

Fig. 18 Contribution of the EKF functions regarding the 8% of the overall execution time (the remainder 92% is spent with matrix multiplications).

5.5 Path Planning

The next experiments analyze the Potential Fields. For this particular implementation, we used a lookahead value of 5 for local minima detection and the Virtual Obstacle Concept escape technique. This preemptive detection is responsible for about 79% of the overall execution time, while the potential calculations for the effective next movement occupies the remaining 21% of the total time. Table 4 shows the execution time for the path presented in Figure 19. As expected, the PC presents the lowest execution time, followed by the Nokia N95 and finally by the Nokia N80.

Table 4 Time measurements for the Potential Fields algorithm.

	PC	Nokia N80	Nokia N95
Avg. Step Time (*ms*)	11.00	377.00	278.75
Total Time (*ms*)	7,664.60	253,442.75	187,882.75

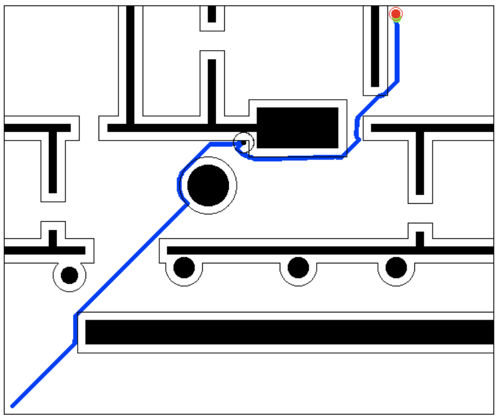

Fig. 19 Environment used for Potential Fields profiling (map size of 600×500, with robot initial location at the bottom left and target location at the top right).

When performing experiments on-the-field, the robot revealed some strange orientation changes when avoiding obstacles. This fact was never very noticeable in the simulations performed. We concluded that, even in the absence of local minima locations, some raw directional vectors cannot be directly applied for the robot's movement. Some of these directional vectors force the robot to perform expensive rotations that need to be smoothen beforehand.

5.6 Summary

Globally, the experiments revealed a considerable robustness of the current JVM available in the mobile devices used and the potential for those devices to execute complex navigation algorithms. However, the current state of the J2ME platform makes it difficult to provide more efficient implementations of the algorithms used, especially when video acquisition is needed, which can be seen by the execution times presented. Nevertheless, the execution of complex algorithms is possible and there is still room for further improvements (e.g., code optimizations).

6 Conclusions

The work presented in this paper focused on a study of the viability to accomplish autonomous navigation with smartphones and J2ME. Tests with well known navigation algorithms (e.g., potential fields, particle filter, and extended Kalman filter) have been performed. To achieve realistic experiments, we use a mobile robot controlled by a smartphone, which is able to execute complex and computationally intensive navigation algorithms and communicate with the robot via *Bluetooth*. The other smartphone is used as an intelligent visual sensor.

The mobile implementation of the algorithms revealed high consistency and robustness. The experiments on-the-field show that it is feasible to execute navigation algorithms in high-end smartphones, especially with soft real-time requirements. The current processing capabilities of smartphones and J2ME can fully fulfill acceptable time requirements in environments where the smartphone might be used to assist user navigation (e.g., tourist exploring an unfamiliar city, customer looking for a store at a shopping center).

From the experiments performed for visually recognizing a landmark, it is clear that future enhancements of J2ME should include the capability to acquire video streaming and to access individual frames. The current implementation is required to perform single image capture, which is too slow for most requirements needing real-time video processing.

Acknowledgements We would like to acknowledge the donations of smartphones by the Nokia Corporation. A special thanks to Vanderlei Bonato for making available the C code of the EKF implementation.

References

1. Baczyk, R., Kasinski, A., Skrzypczynski, P.: Vision-based mobile robot localization with simple artificial landmarks. Prepr. 7th IFAC Symp. on Robot Control, Wroclaw pp. 217–222 (2003)
2. Bonato, V., Peron, R., Wolf, D., de Holanda, J., Marques, E., Cardoso, J.: An fpga implementation for a kalman filter with application to mobile robotics. In: Industrial Embedded Systems, 2007. SIES ’07. International Symposium on, pp. 148–155 (2007). DOI 10.1109/SIES.2007.4297329
3. Bruce, J., Balch, T., Veloso, M.: Fast and inexpensive color image segmentation for interactive robots. In Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS ’00) **3**, 2061–2066 (2000)
4. Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. thesis, Politecnico di Milano, Italy (1992)
5. Fox, D.: Markov localization: A probabilistic framework for mobile robot localization and navigation. Ph.D. thesis, Institute of Computer Science III, University of Bonn, Germany (1998)
6. Goodrich, M.A.: Potential fields tutorial. Class Notes (2002)
7. Gordon, N., Salmond, D., Smith, A.: Novel approach to nonlinear/non-gaussian bayesian state estimation. Radar and Signal Processing, IEE Proceedings F **140**(2), 107–113 (1993)
8. Jang, G., Lee, S., Kweon, I.: Color landmark based self-localization for indoor mobile robots. Robotics and Automation, 2002. Proceedings. ICRA ’02. IEEE International Conference on **1**, 1037–1042 (2002)
9. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering **82**(Series D), 35–45 (1960)
10. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. The International Journal of Robotics Research **5**(1), 90–98 (1986)
11. Kim, J.O., Khosla, P.: Real-time obstacle avoidance using harmonic potential functions. IEEE Transactions on Robotics and Automation **8**(3), 338–349 (1992)
12. Lee, L.F.: Decentralized motion planning within an artificial potential framework (apf) for cooperative payload transport by multi-robot collectives. Master’s thesis, Department of Mechanical and Aerospace Engineering (2004)
13. Leonard, J.J., Durrant-Whyte, H.F.: Mobile robot localization by tracking geometric beacons. IEEE Transactions on Robotics and Automation **7**(3), 376–382 (1991)
14. Lowe, D.G.: Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision **2**, 1150–1157 (1999)
15. Matarić, M.J.: The Robotics Primer, 1st edn. The MIT Press (2007)
16. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: Fastslam: A factored solution to the simultaneous localization and mapping problem (2002)
17. Park, M.G., Lee, M.C.: Artificial potential field based path planning for mobile robots using a virtual obstacle concept. Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003) **2**, 735–740 (2003)
18. Prasser, D., Wyeth, G.: Probabilistic visual recognition of artificial landmarks for simultaneous localization and mapping. Proceedings of the 2003 IEEE International Conference on Robotics and Automation **1**, 1291–296 (2003)
19. Rekleitis, I.: Cooperative localization and multi-robot exploration. Ph.D. thesis, School of Computer Science, McGill University, Montréal (2003)
20. Se, S., Lowe, D., Little, J.: Vision-based mobile robot localization and mapping using scale-invariant features. Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on **2**, 2051–2058 (2001)
21. Smith, R., Self, M., Cheeseman, P.: Estimating uncertain spatial relationships in robotics. In: Autonomous robot vehicles, pp. 167 – 193. Springer-Verlag New York, Inc., New York, NY, USA (1990)
22. Solórzano, J., Bagnall, B., Stuber, J., Andrews, P.: lejos: Java for lego mindstorms. `http://lejos.sourceforge.net/` (last visited in June 2008)
23. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT Press (2005)
24. Tierno, J., Campo, C.: Smart camera phones: Limits and applications. Pervasive Computing, IEEE **4**, 84–87 (2005)
25. Veelaert, P., Bogaerts, W.: Ultrasonic potential field sensor for obstacle avoidance. IEEE Transactions on Robotics and Automation **15**(4), 774–779 (1999)
26. Volpe, R., Khosla, P.: Manipulator control with superquadratic artificial potential functions: Theory and experiments. IEEE Transactions on Systems, Man and Cybernetics **20**(6), 1423–1436 (1990)
27. Welch, G., Bishop, G.: An introduction to the kalman filter. Tech. rep., Department of Computer Science, University of North Carolina at Chapel Hill, USA (2006)
28. Yoon, K.J., Jang, G.J., Kim, S.H., Kweon, I.S.: Fast landmark tracking and localization algorithm for the mobile robot self-localization. in IFAC Workshop on Mobile Robot Technology pp. 190–195 (2001)