PathFinder: An autonomous mobile robot guided by Computer Vision

Andre R. de Geus[1,2], **Marcelo H. Stoppa**[1], **Sergio F. da Silva**[1,2]
[1]Modeling and Optimization Program, Federal University of Goias, Catalao, Goias, Brazil
[2]Biotechnology Institute, Federal University of Goias, Catalao, Goias, Brazil
Email: geus.andre@ufg.br, mhstoppa@gmail.com, sergio@ufg.br

Abstract— *Localization is a key research topic in mobile robotics, responsible to assist robotic systems equipped with sensors, to navigate with certain autonomy. Unfortunately, the sensors shows frequently reading errors that disturb its location. In this paper, we describe the development of a computer vision system for autonomous navigation of a robot in a simulated environment. The system uses a unattached camera to detect the robot, concentrating the localization problem in the feature extraction of the images. Also, it uses Artificial Intelligence algorithms to determine the path in order to find the best solution. Our results show that the robot was able to follow the path and reach the goal, validating the proposed method.*

Keywords: Autonomous Navigation, Path Planning, Artificial Intelligence, Computer Vision

1. Introduction

According to [10], mobile robots are automatic transport devices, indeed, mechanical platforms equipped with a locomotion system able to navigate through a given environment, endowed with a certain level of autonomy to their locomotion. Autonomy is not just about energy sufficiency issues, but also the processing capability to plan and execute some tasks. Navigation in unknown environments is one of the areas with great interest to mobile systems, offering wide range of applications, from systems that assist in-house cleaning [5] to dangerous operations of search and rescue [6].

Navigation is an intrinsic feature of robots, allowing them to move freely into its environment until reach its goal. According to [7], dead-reckoning is a classic method of navigation, whose accuracy depends directly on the quality of the sensors used. Given that its location is based in previously locations, is inevitable the accumulation of position errors.

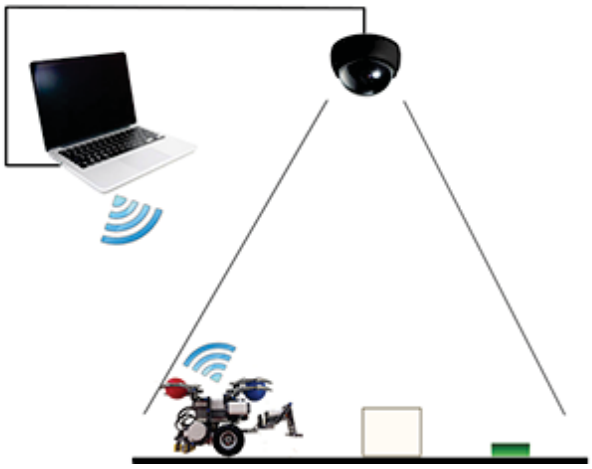
The compensation of position errors demands integration of multiple sensors. Combining their data, improves significantly the estimation of robots location in its environment. The technique proposed by [3] uses sensors that capture the direction, acceleration and engine revs. The authors demonstrate various improvements considering that heterogeneous sensors have different perceptions and can cooperate with each other.

Looking to increase the accuracy, adding information extracted from images provided by a webcam, shows to be an potential alternative to assist the path planning and execution. This is the motivation for the proposed work, that uses a webcam with panoramic view, to guide a robot to a goal, without human intervention and use of sensors.

2. Problem Description

The evaluated system consists of a robot with colourful parts that assist in determining its position, which is built under the Mindstorms NXT® platform. Big boxes are used to simulate obstacles that blocks the robots way. The goal is represented by a small sheet on the floor. A low-resolution webcam positioned on the environment's ceiling. The processing is performed by a laptop connected to the webcam, which sends the movement commands to the NXT®. Fig. 1 illustrates the environment layout and its components, where the target is a green rectangle, and the obstacle is a white rectangle.

Fig. 1: Side view PathFinder's environment layout



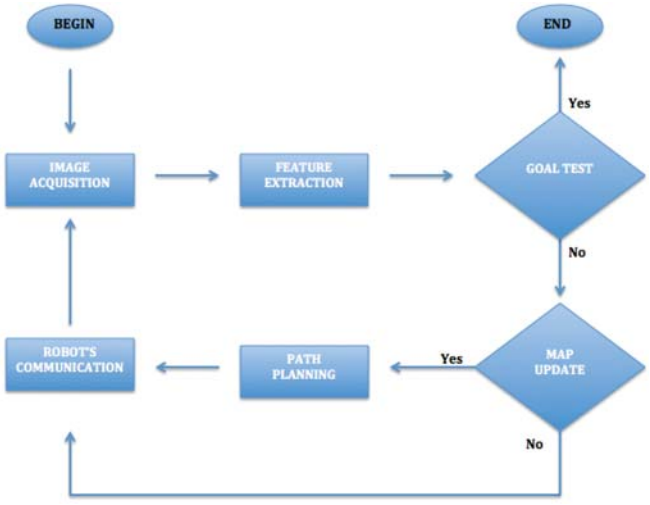
The environment layout has restrictions on the objects colours, due to the filters used to localize them. The robot has blue colour on its head and red in its tail (Fig. 2). The obstacles are white boxes of the NXT® kit and the goal is a small green sheet.

Fig. 2: PathFinder's design

The system uses Computer Vision [2], [8], [12] and Artificial Intelligence [13], [9] techniques. The system modules are illustrated in Fig. 3 and are responsible for:

1) Image acquisition using a webcam;
2) Feature extraction that determines the position of robot, obstacles and target;
3) Path planning to find the best solution avoiding obstacles;
4) Communication between the computer and robot;
5) Mapping update that also checks if the objects stay at the same place in the map;
6) Goal test to check if the robot reached the goal

Fig. 3: System Modules Flowchart



3. System Development

The autonomous control system is implemented in the programming language Python, which although slower than other languages has become very popular in a short time, due to its simplicity and clean syntax. Furthermore, it's easily integrated with C/C++ codes, allowing to involve them in Python wrappers. This provides two advantages: have a fast code as the original C/C++ code and easy syntax coding.

Two libraries are used: OpenCV (Open Source Computer Vision), coded in C and C++ with interface to Python, which has a wide range of Image Processing and Computer Vision techniques; and nxt-python, an interface designed to send commands interpreted by the NXT® brick, unlike NXC and NXT-G, compilable languages that runs on the brick.

The next subsections will describe the approaches for each module of the system shown in Fig. 3.

3.1 Image Aquisition

The system initializes sampling an image from the webcam through the OpenCV library, however this first image should be discarded because it's completely out of focus due to insufficient time to focus adjustment implemented in most webcams. To get around this problem, the system initializes the webcam and waits 30 seconds, enough time to adjust the focus.

The acquisition of image doesn't takes into consideration the quality of webcam resolution, so any low-cost webcam can be used in the system. It happens due to the reduction in the resolution of the image after the feature extraction module described below. The new resolution is calculated based on the minimum space that the robot can move in a safe way. Therefore, any higher resolution provide for a better hardware would be wasted. The next step will detail the techniques used to define the objects positions.

3.2 Feature Extraction

This step has a big importance to the system, since the navigation of the robot is based on the feature extraction of the image obtained in the previous step, without intervention of any other sensor. This step is subdivided in two groups of interest, one used for navigation and the other to construct the occupancy grid map:

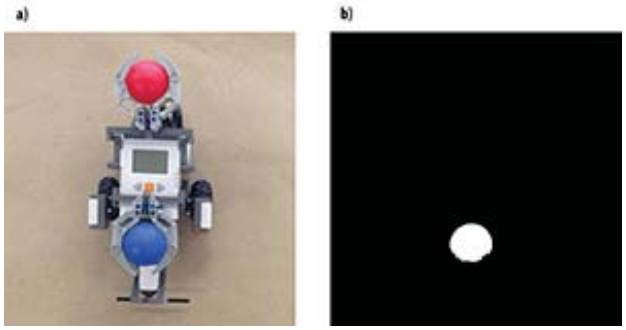
1) Robot's Position and Direction;
2) Obstacles and Target.

3.2.1 Robot's Position and Direction

As described before, the robot has details in red and blue on its edges, then to determine the robot's position comes down to locate the region that contains the greatest intensity of one of these colours. Our approach uses the thresholding technique described in [4], to determine the region in the image that has a given colour. To detect the blue colour, for example, he technique on channel zero index of image, representing only the intensity of blue in each one pixel. It is normal that images with regions in blue colour show greater intensity on this channel as compared to regions of other colours, which are also composed of blue. The result is a

binary image, indicating regions that present values higher than the threshold determined.

Fig. 4: a) Segmented image of the robot. b) Binary image applying threshold 70 on the blue channel.



Once the blue region of the robot is identified, the robot's position on the map is determined by the central pixel of this area. The Python function that returns the median pixel in shown in Fig. 5

Fig. 5: Python function that returns the median pixel of a colour region

```
def getMedianPixel(self, color, threshold):
    image = npy.array(self.image,int)
    pic = npy.array(image[:,:,self.color])
    averagePic = npy.array(image,int)
    pic -= npy.average(averagePic,2)
    image = npy.array(255 * (pic > threshold))
    hFlat = npy.sum(image,0)
    vFlat = npy.sum(image,1)
    hIndex = 0
    vIndex = 0
    i = 0.
    for val in hFlat:
        hIndex += val * i
        i += 1.
    hIndex /= npy.sum(hFlat)
    i = 0.
    for val in vFlat:
        vIndex += val * i
        i += 1.
    vIndex /= npy.sum(vFlat)
    return int(hIndex),int(vIndex)
```

Likewise the previous step, the median pixel of the red region is found, and assists in determining the direction of the robot. In order to select the best technique to be used in the system, two approaches were implemented and analysed.

The first approach uses SVM (Support Vector Machine), a machine learning method presented by [14], to classify which class a given image belongs. To implement this procedure, is used the scikit-learn library, that is able to classify multiple classes. Five classes were criated: Up, Down, Left, Right, and Null, corresponding to valid directions (multiple of 90 degrees) for navigation and Null for any other invalid direction (see Fig. 6).

Fig. 6: Class examples (Down(a), Null(b), Left(c))

To train the SVM, a set of several images was provided, and the training parameter was calculated by the versor $[vx, vy]$ of the median pixel of the blue region $(x2, y2)$, with origin the median pixel of the blue region $(x1, y1)$. The versor is computed as follows:

$$vx = \frac{(x2 - x1)}{\sqrt{(x2 - x1) + (y2 - y1)}}$$
$$vy = \frac{(y2 - y1)}{\sqrt{(x2 - x1) + (y2 - y1)}} \quad (1)$$

The second approach uses only geometric equations to determine the direction, using Python numpy library. The arctan2 function receives as argument two values that indicate the direction of the vector in a Cartesian plane and returns the corresponding value of the angle formed by these points in radians, in the range $[-\pi, \pi]$. With the vector direction with the coordinates to the edges of the robot, is used the *rad2deg* function to transform the result in degrees.

Fig. 7: Python function that returns the angle relative to the Y axis.

```
def findAngle(self):
    y = (self.blueY-self.redY)
    x = (self.blueX-self.redX)
    return (npy.rad2deg(npy.arctan2(x,y)))
```

With a preliminary simple comparison, it was observed that both produce equivalent results for this problem, being indifferent to to choose of one over the other. However, we opted for the second approach to avoid the SVM training step and storage of large data set required to the training step. In trivial cases like this, is not obvious improvements on the use of Machine Learning techniques, but in facial expressions recognition applications, as in [1], the use of SVM becomes indispensable.

3.2.2 Obstacles and Target

The obstacles placed in the environment block the robot's way, so it has to be identified and indicated as unavailable

places in the path planning step. The first approach tested used edge detection algorithm to find the obstacles, but ended up being discarded because it left the inside of the obstacles as free points for navigation.

The obstacles, the Mindstorms NXT® kit white boxes, such as for the robot, are localized by thresholding technique in the blue channel of the image, using a blue threshold just below the value we use to find the robot. The resulting binary image contains the area of the boxes and also, undesirable parts of the robot. This is corrected after removing a rectangle that represents the total area occupied by the robot, around its blue dot calculated previously.

Use the same process of thresholding to determine the goal's central pixel, represented by the green sheet.

Once all the obstacles and goal are detected, the map represented as an occupancy grid is constructed, as described by [11]. All the available and unavailable indexes to navigate and the goal are represent by 0, 1 and 99, respectively.

In this step, with obstacles and targets detected, the map is constructed represented by occupation grid [11], assigning 1 to spaces occupied and 99 for the target in a two-dimensional array.

3.3 Map Update

This step is important for saving a considerable amount of processing time in the system, avoiding unnecessary path planning calculations every time a new image is acquired. This analysis process compares the current positions of the objects with their positions when the path is previously planned, determining if the path needs to be recalculated. Therefore, in case the objects move for some reason, or any intervention in the environment occurs, the robot is able to recalculate the path and reach the goal.

Fig. 8: Python function that indicates a map change and updates it

```
def mapChange(self):
    newMap = self.readMap()
    mapDiff = npy.abs( newMap - self.map )
    if ( npy.sum(mapDiff) > self.mapThreshold ):
        self.map = newMap
        return True
    else:
        return False
```

3.4 Path Planning

Once the map is constructed, indicating the position of the robot, obstacles and target, we used the A-Star algorithm to find the best path to be executed by the robot. As demonstrated by [9], it is necessary to estimate a consistent heuristic function to have a optimal results. The cost function F is calculated by G + H, where G is the exact cost of the starting point to the current point and approximate heuristic function H is calculated by the number of movements required to reach the goal, excluding the obstacles, moving only horizontally and vertically.

Fig. 9: Python function to calculate the A* cost function F

```
def G(pos):
    return Dg * (abs(pos[0] - startX) + abs(pos[1] - startY))

def H(pos):
    return Dh * (abs(pos[0] - targetX) + abs(pos[1] - targetY))

def F(pos):
    return G(pos) + H(pos)
```

In order to facilitate the execution of the path, the function returns a new two-dimensional array with same size of the constructed map. It contains only the path, indicating the goal by the number 99, decreasing 1 each movement toward the starting point. The Fig. 10 shows an example of the path returned.

Fig. 10: Path planned by A* algorithm, with starting point [2,3] e goal [15,15]

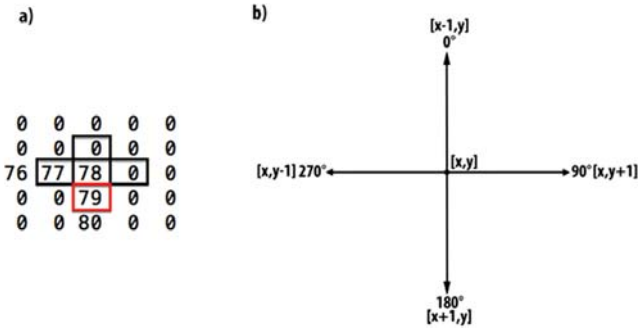
```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 74 75 76 77 78  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 79  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 80  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 81  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 82  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 83  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 84  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 85  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 86 87  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 88 89  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 90 91 92 93 94  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 95  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 96 97  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 98 99  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
```

3.5 Robot's communication

Using nxt-python interface, control commands to move the servo motor are given through USB or Bluetooth connections. The use of wires is not feasible in this context, since it influence the feature extraction step and obstruct the movement of the robot in its environment. Therefore, even with the delay in wireless connections, we opted to use Bluetooth.

As proposed by the model, the entire path execution is based on visual information obtained by a webcam. Therefore, every new image acquired, compares the neighbours values of the robots current, considering only horizontal and vertical movements. The highest value determines the direction that the robot must follow, Fig. 11 (a) illustrates an example where the robot is at position [x,y], 78 value, and determines that the highest neighbour value is in the position below, [x + 1, y] highlighted in red, readjusting his new direction to 180 degrees relative to the Y axis.

Fig. 11: a) Robot's direction adjustment, considering the robot is at the position with value 78. b) Angle correction based on the highest nighbour



Our developed approach is able to reach the goal even with uncertainty and variability of servo motors described by [13], which causes deviations even in simple commands like move forward.

3.6 Goal test

To determine if the goal is reached, we compare the coordinates of the robot and the goal, acquired in the feature extraction step. If they have the same value, the system finished. Otherwise, the system keeps sending commands to move the robot towards the goal. The function is shown below:

Fig. 12: Python goal test function

```
def checkFinish(self):
    if (self.targetX == self.blueX and self.targetY == self.blueY):
        cv2.VideoCapture(0).release()
        return True
    else:
        return False
```

4. Conclusions

The present work showed that, through a computer vision system, a mobile robot in a restricted environment layout can be guided from one point to another without needing human intervention. To perform this task, studies were carried out in search algorithms, machine learning, computer vision, image processing and robotic navigation to ensure the robot is able to plan and execute the path satisfactorily.

Using thresholding techniques to object detection and A* search algorithm to path planning, it was possible the system, validating the proposed autonomous navigation method. However, some aspects should be carefully analysed, such as the computational cost of image processing and the processing capacity of the hardware involved, since they are determining factors to the system execution.

During the experiments, where the feature was evaluated to determine if the robot had reached the target or not, undesirable characteristics were observed in the system, which can be improved in a future work. As an example, one can cite that, in some cases the system drew trajectories very close to obstacles. Even having enough space to keep distance, providing possible collisions when doing bends near the edges of obstacles, or even in cases which the robot had totally unexpected behaviour, caused by different light conditions in the environment forcing the recalculation of the thresholds for the correct identification of the objects position.

As possible improvements for future work, we would like to:

- Use HSI colour system instead of RGB, aiming to get better results in different light conditions.
- Use a safety margin around the obstacles to avoid collisions.
- Change the path planning algorithm to select the best path with the minimum possible number of turns, in order to save time on the robot's readjustment direction.
- Create a graphical user interface to simulate objects position identified by the system.

5. Acknowlegment

The authors would like to thank the financial support of the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Research Support of the State of Goias (FAPEG) and the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES).

References

[1] Ari, I., Uyar, A., and Akarun, L. Facial feature tracking and expression recognition for sign language. In *Computer and Information Sciences, 2008. ISCIS'08. 23rd International Symposium on* (2008), IEEE, pp. 1–6.

[2] Forsyth, D. A., and Ponce, J. *Computer Vision: A Modern Approach*, 2nd ed. Pearson, 2012.

[3] Fuke, Y., and Krotkov, E. Dead reckoning for a lunar rover on uneven terrain. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on* (1996), vol. 1, IEEE, pp. 411–416.

[4] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing*, 3rd ed. Pearson/Prentice Hall, 2007.

[5] iRobot. irobot roomba: Vacuum cleaning robot. Online : http://www.irobot.com/For-the-Home/Vacuum-Cleaning/Roomba.aspx, 2015.

[6] Kadous, M. W., Sheh, R. K.-M., and Sammut, C. Caster: A robot for urban search and rescue. In *Proceedings of the 2005 Australasian Conference on Robotics and Automation* (2005), pp. 1–10.

[7] Nelson, W. L., and Cox, I. J. Local path control for an autonomous vehicle. In *Autonomous robot vehicles*. Springer, 1990, pp. 38–44.

[8] Prince, S. J. D. *Computer vision : models, learning, and inference*. Cambridge University Press, 2012.

[9] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*, third edition ed. Prentice Hall, 2010.

[10] Secci, H. A. *Uma Introdução a Robôs Móveis*, 2008.

[11] Souza, A., Maia, R., Aroca, R., and Goncalves, L. Probabilistic robotic grid mapping based on occupancy and elevation information. In *Advanced Robotics (ICAR), 2013 16th International Conference on* (Montevideo, 2013), pp. 1–6.

[12] SZELISKI, R. *Computer Vision: Algorithms and Applications*. Springer, 2011.
[13] THRUN, S., BURGARD, W., FOX, D., ET AL. *Probabilistic robotics*, vol. 1. MIT press Cambridge, 2005.
[14] WESTON, J., MUKHERJEE, S., CHAPELLE, O., PONTIL, M., POGGIO, T., AND VAPNIK, V. Feature selection for svms. In *NIPS* (2000), vol. 12, pp. 668–674.