SLAM using the NAO camera

Maastricht University
Group 2
Taghi Aliyev, Gabriëlle Ras, Dennis Soemers, Roel Strolenberg

January 20, 2014

Abstract

This paper researches the use of two Simultaneous Localization And Mapping (SLAM) algorithms, namely the Graph SLAM and the EKF SLAM. The SLAM algorithms are used for the localization and mapping of a NAO robot on a Standard Platform League soccer field [1]. The camera of the robot is used as the main source of input for gaining information about the environment. ...

1 Introduction

1.1 Problem Definition

The problem addressed in this paper is the real-time Simultaneous Localization and Mapping (SLAM) problem, using a NAO robot on a soccer field. The SLAM problem concerns building a map of the environment around the robot, whilst simultaneously localizing the robot on that map. To achieve this the Nao needs to take images of its environment during runtime and analyze these in order to find the location of landmarks. Next the distance to these landmarks is calculated. These distances are used as input for the SLAM algorithm (EKF or Graph, see section 4) which has as output the next set of moves the Nao should make to maximize the localization efficiency and to locate itself on the map it created. In this paper, it is assumed that the NAO robot will use his camera to obtain measurements of the environment, and that it only needs to be able to operate on the soccer field. The robot will also be able to keep track of the movement commands issued to the robot.

1.2 Outline of Paper

Section 2 describes the important features of the environment in which the robot will act. Section 3 describes the Image Processing techniques used to obtain useful measurements from images produced by the NAO's camera. Section 4 describes the implemented algorithms for dealing with the SLAM problem, namely EKF SLAM and Graph SLAM. Section 5 describes the experiments performed with the implemented algorithms. Section 6 describes the results of these experiments. Section 7 provides some ideas on future work. Finally, section 8 concludes the paper.

2 Environment

2.1 The NAO

The robot used for this study is NAO, a humanoid robot developed by the company Aldebaran Robotics. This robot has been the standard robot used for the standard platform competition of the RoboCup since 2008.

2.2 Features of the environment

The soccer field that is used is the half Standard Platform League soccer field. It is green field with white lines and a yellow official goal post. The aim of the robot is to localize itself within the bounds of this half soccer field using the white lines as landmarks.

3 Image Processing

This section's purpose is to explain the mechanisms, algorithms and mathematics behind the processing of images made by the NAO. After an image is processed the obtained distances to landmarks (goalposts/fieldline cornerpoints) are passed onto the SLAM algorithm which uses these distances to determine the Nao's next moves. A brief introduction to the subsections follows before heading into the details.

First the image is preprocessed. In this step the image is converted into a new image consisting only of green, yellow and white pixels. Any background noise is removed.

Second the goalposts get isolated and the location of the foot of each post - if visible - gets determined.

Third the field lines get isolated and using Hough Transforms the corner points are extracted.

Lastly the distance gets calculated using the camera angle and height, the resolution of the image and the coordinates of the previously extracted landmarks.

3.1 Image Preprocessing

Before any kind of extraction can be made, the image first needs to be preprocessed. First a random sampling of pixels is made from which the average light intensity is calculated using the luminence from the HSL color-space.

Next the white, green and yellow (goal) pixels are extracted by converting them to the HSL color-space and by using the following thresholds and bounds:

White:

$$L_p > \beta + (L_{\max} - \beta) \times \frac{L_{\text{avg}}}{L_{\max}} \tag{3.1}$$

Green:

$$G_{\min} \leq H \leq G_{\max} \tag{3.2}$$

Yellow:

$$Y_{\min} \leq H \leq Y_{\max} \tag{3.3}$$

Where
$\beta = 120$
$L_{\max}$ = *maximum luminance*
L_{p} = *luminance of pixel*
L_{avg} = *average luminance*
H = *hue of pixel*
$G_{\min}$, $G_{\max}$ = *lower upper bounds for the hue of green*
$Y_{\min}$, $Y_{\max}$ = *lower upper bounds for the hue of yellow*

The value of '120' has experimentally been shown (see section 5 and 6) to deliver the best results. The G_{min} and G_{max} are determined at the start of each run by taking a picture of the 'grass'. This image is then scanned for the minimum and maximum hue. G_{min} will be set equal to this minimum hue minus some small number ϵ in order to deal with the occasional over- and under lighted areas on the field. The same is done for G_{max}.
Y_{min} and Y_{max} are determined similarly but without adding/subtracting ϵ, because (in the case of this project) there is only one goal and thus the difference in lighting is negligible.
At this point the original image(Figure 3.1) now looks something like Figure 3.2.

Figure 3.1: Raw image taken by NAO

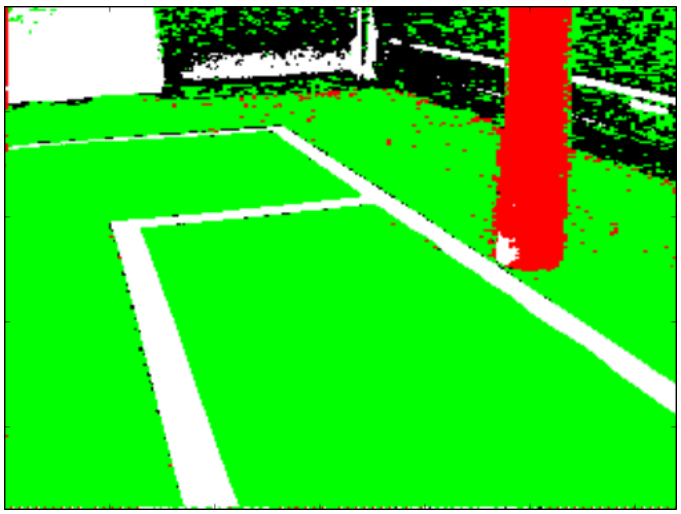

Figure 3.2: Image after color filter

Before continuing preprocessing, the goalposts need to be extracted first.

3.2 Goalpost Extraction

The only parts of the goal that are of interest for this project are the locations of each foot of the posts, because the localisation algorithms use a 2D map. To find these the image is scanned from bottom to top. If a yellow (in this case red because of optimisation) pixel is spotted the algorithm looks a few pixels down to see if there are some green pixels. If there are sufficient green pixels the algorithm continues, if there aren't the algorithm stops scanning this column and proceeds to the next one. This heuristic was added to avoid getting parts of the goal's mast or some random noise in the

background.
When the algorithm has decided there are enough green pixels under a yellow one it continues to climb the vertical axis and count the amount of adjacent yellow pixels. If this counter surpasses a certain predefined threshold (somewhere between 10 and 20 percent of the height of the image) the first-encountered yellow pixel is now saved in a list of goalpost-coordinates and the algorithm continues to the next column.
After the entire image is scanned the list of goalpost-coordinates is clustered and averaged to prevent multiple landmarks at one goalpost.

3.3 Fieldline Cornerpoint Extraction

Before any fieldlines can be extracted the background noise needs to be removed. The details of this operation will not be explained here, but the general idea is to scan the color-filtered image from top to bottom for the color green. Once a safe amount of green pixels in a row has been encountered, 'erase' all the pixels above this point and continue to the next column [2]. To finish up we will remove any remaining green and yellow pixels because they aren't needed anymore. At this point the image will look something like Figure 3.3.

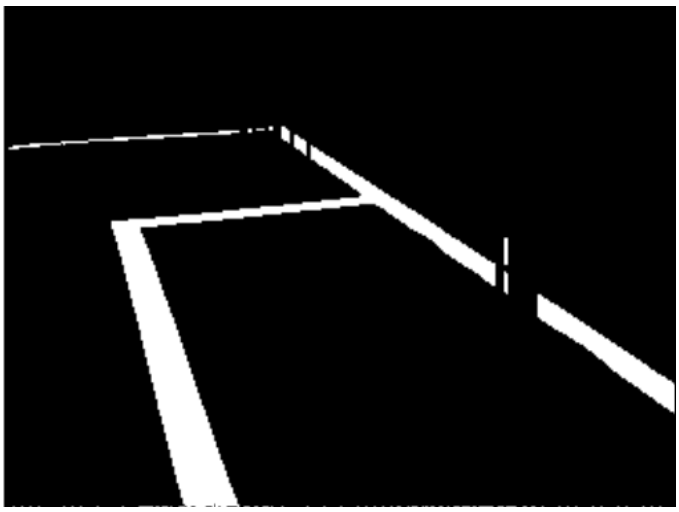

Figure 3.3: Background noise removed

There are quite some ways of obtaining the - in this example - three landmarks, but one in particular seems to be favored in the world of robotics [3][4]: OpenCV's built-in Canny and Hough Transform methods. To illustrate Canny's use see Figure 3.4.

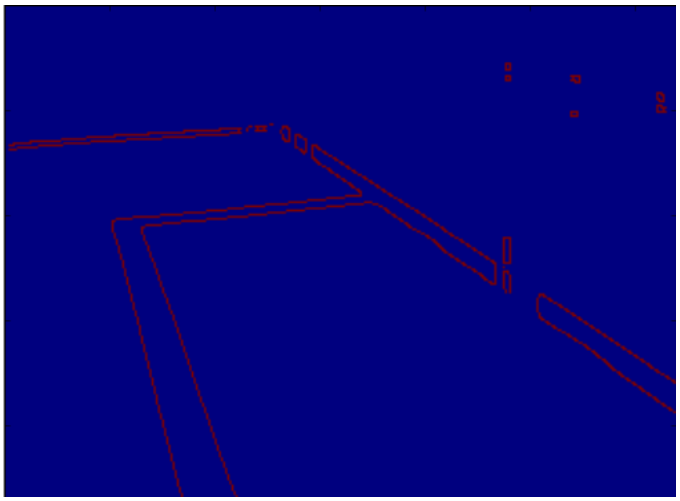

Figure 3.4: Image after applying Canny

After using the Canny method the Hough Transform is applied to the newly obtained image resulting in Figure 3.5.

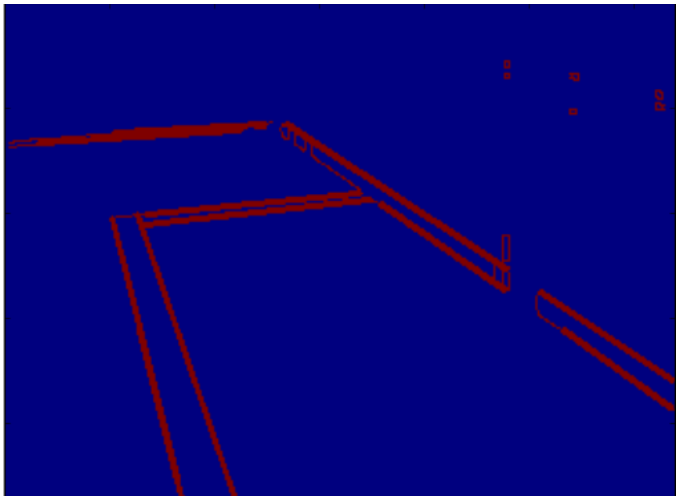

Figure 3.5: Image after applying Hough Transform

The thicker lines are the lines generated by the Hough Transform method. The only thing left to do is finding the intersections of these lines and deleting intersections which are relatively close to each other in order to avoid double landmarks.

Now that all the landmarks have been extracted the distance to these can be calculated.

3.4 Distance Calculation

4 Localization and Mapping algorithms

4.1 EKF SLAM

Extended Kalman Filter SLAM is an algorithm which uses an Extended Kalman Filter (EKF) to solve the SLAM problem. EKF is an enhanced version of the Kalman Filter, which is an algorithm that estimates the current state of a linear system based on (an estimate of) the previous state and a set of transition models. Those transition models contain the physical properties of a system and describe the transitions between states. The normal Kalman Filter requires these transition models to be linear, but EKF no longer has this restriction.

During the runtime of the algorithm, the state is represented by a vector μ and a matrix Σ (see figure 3.1).

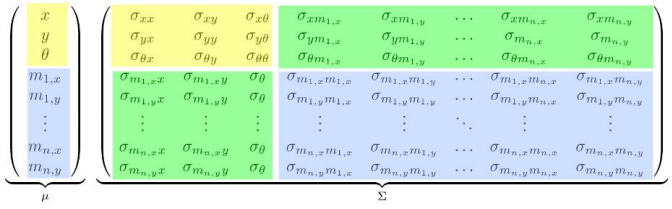

Figure 4.1: The vector of the state [8]

. The first three elements of μ are the estimated x and y coordinates of the robot and the robots estimated rotation θ. The following elements, mi,x and mi,y are the x and y coordinates of the ith landmark. If n equals the number of different landmarks observed so far, this means that μ has a size equal to 3 + 2n.

The matrix Σ contains covariances between the 3 + 2n elements of μ. This means that it is a square matrix with dimension 3 + 2n.

The algorithm is initialized with both μ and Σ being filled with all zeros. This implies that all coordinates of landmarks and future robot position are relative to the robots starting position.

EKF is an iterative algorithm. Whenever new data on the robots motions or observed landmarks is available, the algorithm can compute an estimate of the new state using only the previous state estimate and the data obtained since the last state was estimated.

The first step in such an iteration of the algorithm consists of using the known physical motion model to estimate the new location of the robot. This is computed using simple geometrical calculations and information of the movement commands given to the robot.

Next, the new values for the covariance of the robot position are predicted. These values are computed using the changes in robot position and an estimate of the motion noise. That estimate of the motion noise should be determined experimentally.

Then, the algorithm deals with observed landmarks. If a landmark has not been observed previously, its coordinates and covariances are computed in a similar way to the values of the robot position in the previous steps. This time, models and noise parameters specific to measurements are used.

Finally, landmarks which have already been observed in previous iterations and have now been re-observed are dealt with. The same computations as in the step above are used to compute the coordinates where the landmark is now measured to be. These are compared to the coordinates where the algorithm would expect to see that landmark given the estimate of its current position, and the estimate of the location where that landmark was previously observed. This difference between expectation and reality, combined with parameters indicating how much noise we expect there to be in motion and measurement data respectively, allow the algorithm to compute how much each value of μ should be adjusted, based on how much we trust each piece of data.

4.2 Graph SLAM

Introduction

GraphSLAM is a novel algorithm for mapping using sparse constraint graphs. The basic intuition behind GraphSLAM is simple: GraphSLAM extracts from the data a set of soft constraints, represented by a sparse graph. It obtains the map and the robot path by resolving these constraints into a globally consistent estimate. The constraints are generally nonlinear, but in the process of resolving them they are linearized and the resulting least squares problem is solved using standard optimization techniques[5]. For this project, GraphSLAM is used as a technique populating sparse "information" matrix of linear constraints.

Building up matrices

As is the case with many other SLAM techniques, the first process that is performed by GraphSLAM is the

creation of information matrices. For future reference, they will be called Ω and ξ for ease. Here, Ω corresponds to the so-called "information" matrix and ξ represents motions. Easier way to see it is to look any type of constraint. Ω keeps information about which poses and landmarks are represented in given constraint and ξ keeps information about right hand side of these constraints.

In order to create Ω and ξ matrices, first of all data from environment is collected. Data in this case is the collection of three type of constraints: Initial position, relative motion and relative measurement constraints. One example for this type of constraint and addition of that constraint information into Ω and ξ matrices is given below:

Constraint : robot moved 10 steps forward:

$$x_i = x_{i-1} + 10$$

There are two equations that we can get from this constraint:

$$x_i - x_{i-1} = 10$$

$$-x_i + x_{i-1} = -10$$

Afterwards, we add both constraint informations into the matrices in following fashion :

For row and column corresponding to x_i and x_{i-1} we add 1 and we subtract 1 from row and column corresponding to relation between x_i and x_{i-1}. To explain better, let's take a look at following image which shows where what should be added :

$$\Omega = \begin{array}{c} \\ \vdots \\ x_{i-1} \\ x_i \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} \dots & x_{i-1} & x_i & \dots \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & +1 & -1 & \vdots \\ \vdots & -1 & +1 & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \end{array}$$

$$\xi = \begin{array}{c} \\ \vdots \\ x_{i-1} \\ x_i \\ \vdots \end{array} \begin{array}{c} \\ \begin{pmatrix} \dots \\ -10 \\ +10 \\ \dots \end{pmatrix} \end{array}$$

Ω and ξ are populated in this fashion with all the collected constraints.

Introducing noise to environment

Above given description of matrices assume perfect world and perfect motors/sensors and do not include any error values or noise. However, in real-life environment there is going to be noise caused by either motion or measurement motors of robot. So, noise should be added and handled by GraphSLAM algorithm. Noise is handled by adding approximated error to Ω and ξ matrices. Those error/noise values represent how much measurements and motions are trusted. For GraphSLAM, there are two types of errors, namely motion noise and measurement noise. For future reference, motion noise will be represented as ε_{motion} and measurement noise will be represented as $\varepsilon_{measurement}$. Noise is added to the computations by adjusting Ω and ξ matrices in following fashion. Let's take the same constraint used in section before. Constraint is:

$$x_i = x_{i-1} + 10$$

For this constraint, noise is taken into account by changing set-up of Ω and ξ matrices in following way:

$$\Omega = \begin{array}{c} \\ \vdots \\ x_{i-1} \\ x_i \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} \dots & x_{i-1} & x_i & \dots \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ \vdots & +1 \times \frac{1}{\varepsilon_{motion}} & -1 \times \frac{1}{\varepsilon_{motion}} & \vdots \\ \vdots & -1 \times \frac{1}{\varepsilon_{motion}} & +1 \times \frac{1}{\varepsilon_{motion}} & \vdots \\ \dots & \dots & \dots & \dots \end{pmatrix} \end{array}$$

$$\xi = \begin{array}{c} \\ \vdots \\ x_{i-1} \\ x_i \\ \vdots \end{array} \begin{array}{c} \\ \begin{pmatrix} \dots \\ -10 \times \frac{1}{\varepsilon_{motion}} \\ +10 \times \frac{1}{\varepsilon_{motion}} \\ \dots \end{pmatrix} \end{array}$$

In cases of relative measurement constraints, $\varepsilon_{measurement}$ is used instead of ε_{motion}.

Getting results from Ω and ξ

After matrices are created, last part of GraphSLAM can be executed. Referring to [6], it is known that if x represents best estimates of robot poses and landmark positions, the following equation holds :

$$\Omega \times x = \xi \tag{4.1}$$

Using equation (4.1), it is possible to find x using the Ω and ξ matrices. To do so, the following computation is used :

$$x = \Omega^{-1} \times \xi \tag{4.2}$$

Equation (4.2) is the main calculation that returns best estimates for robot poses and landmark positions and it shows the ease of using and implementing GraphSLAM. Additionally, its computational power is proven to be

quite high through experimentations[5, 6] and it will be the one of the main focus points of experiments section of this paper as well.

5 Experiments

6 Results

7 Future Work

One of the possible future enhancements for SLAM is improvement of robot's exploration skills. One possible solution for this problem is to use Fourier detection/exploration algorithm which will be explained very briefly in next sub-section.

7.1 Frontier detection/exploration

Overall, the exploration problem deals with the use of a robot to maximize the knowledge over a particular area. Frontier detection algorithm/approach tries to make use of *frontiers* which are the regions on the border between open space and unexplored space [7]

8 Conclusion

9 Dennis matrices

$$\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \mu_{j,x} - \mu_{t,x} \\ \mu_{j,y} - \mu_{t,y} \end{pmatrix}$$

$$q = \delta^T \times \delta$$

$$z_t^i = \begin{pmatrix} \sqrt{q} \\ atan2(\sigma_y, \sigma_x) - \mu_{t,\theta} \end{pmatrix}$$

$${}^{low}\mathrm{H}_t^i = \frac{\delta h(\mu_t)}{\delta \mu_t} =$$

$$= \frac{1}{q} \times \begin{pmatrix} -\sqrt{q}\times\delta_x & -\sqrt{q}\times\delta_y & 0 & +\sqrt{q}\times\delta_x & \sqrt{q}\times\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}$$

$${}^{low}\mathrm{H}_t^i = \frac{1}{q} \times \begin{pmatrix} -\sqrt{q}\times\delta_x & -\sqrt{q}\times\delta_y & 0 & +\sqrt{q}\times\delta_x & \sqrt{q}\times\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}$$

$$\mathrm{H}_t^i = {}^{low}H_t^i \times F_{x,j}$$

$$F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & \underbrace{0\cdots0}_{2j-2} & 0 & 0 & \underbrace{0\cdots0}_{2N-2j} \\ 0 & 1 & 0 & 0\cdots0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 1 & 0\cdots0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 0 & 1 & 0\cdots0 \end{pmatrix}$$

References

[1] RoboCup Technical Committee, Standard Platform League (Nao) Rule Book, *www.tzi.de/spl/pub/Website/Downloads/Rules2012.pdf*, (May 2012)

[2] Amogh Gudi, Patrick de Kok, GeorgiosK. Methenitis, Nikolaas Steenbergen, "Visual Goal Detection for the RoboCup Standard Platform League" *X WORKSHOP DE AGENTES FISICOS*, 2009.

[3] Jose M. Canas, Domenec Puig, Eduardo Perdices and Tomas Gonzalez, "Feature Detection and Localization for the RoboCup Soccer SPL" *University of Amsterdan*, 2013.

[4] Lukasz Przytula, "On Simulation of NAO Soccer Robots in Webots: A Preliminary Report" *Department of Mathematics and Computer Science University of Warmia and Mazury*, 2011

[5] Thrun, S. and Montemerlo, M., "The GraphSLAM Algorithm With Applications to Large-Scale Mapping of Urban Structures," *International Journal on Robotics Research*, pp. 403–430 Volume 25 Number 5/6, 2005.

[6] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, Wolfram Burgard, "A tutorial on Graph SLAM," *http://ais.informatik.uni-freiburg.de/teaching/ws10/praktikum/slamtutorial.pdf*, last accessed in 2014.

[7] Brian Yamauchi, "Frontier based exploration," *http://robotfrontier.com/frontier/index.html*, accessed in 2014.

[8] Cyrill Stachniss, "Robot Mapping EKF SLAM", *http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam04-ekf-slam.pdf*