SLAM using the NAO camera

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

January 22, 2014

Abstract

This paper researches the use of two 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 was used as the main source of input for gaining information about the environment. By first preprocessing the image the NAO takes to reduce noise and then using OpenCV's *Canny* and *Hough Transform* methods, the data is extracted from the image and used in the SLAM algorithms. Experiments were done mainly to estimate the influence of the noise based on the distance deviations of the movements of the NAO. The findings show that the number of time steps and correct landmark associations are essential for the performance of the SLAM algorithms. The main conclusion is that more extensive experiments need to be performed.

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, while simultaneously localizing the robot on that map. To achieve this the NAO needs to take images of its environment during runtime and analyse these in order to find the location of landmarks. Next the distance to these landmarks is calculated. These distances are used as information input for the SLAM algorithm (EKF or Graph, see section 4) which has as output an array containing the position of the robot and the positions of the observed landmarks.

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 GraphSLAM. 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 a NAO V3.2, a humanoid robot developed by the Aldebaran Robotics. This robot has been the robot used for the RoboCup Standard Platform League since 2008 [2].

2.2 Features of the environment

The soccer field that is used is the half Standard Platform League soccer field. It is a 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 Software Architecture

The software developed during this project consists of a Graphical User Interface, used for issuing commands to the robot from a computer, an Image Processing module, and a module of SLAM algorithms. All programming has been done in Python using the NAOqi Framework to control the NAO.

3.1 Graphical User Interface

The Graphical User Interface (GUI) allows for easy control of the robot. It allows the user on a laptop to send motion commands to the robot, to command the robot to take a picture, and to run the SLAM algorithms.

3.2 Image Processing Module

After issuing motion commands, the robot takes a picture of it's environment and obtains measurements from that picture. The Image Processing Module accepts the image taken by the robot's camera as input and produces a set of measurements obtained from that image as output. The algorithms used in this module are described in detail in the Image Processing section.

3.3 SLAM Module

The SLAM Module contains an abstract class for running SLAM algorithms. This abstract class takes motion and measurement data as input. It produces an estimate of the robot's location and rotation, and an estimate of landmark positions, as output. The motion data is equal to the motion commands issued to the robot by the user, and is therefore obtained directly from the GUI. The measurement data is the output of the Image Processing Module.

This module contains different implementations of the abstract class for different algorithms. The algorithms used are described in detail in section 5.

4 Image Processing

The purpose of this section 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 or field line corner points) are passed onto the SLAM algorithms, which uses this as input to compute the positions of the robot and the landmarks. 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 the corner points are extracted using Hough Transforms.

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

4.1 Preprocessing

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

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

White:

$$L_p > \beta + (L_{\max} - \beta) \times \frac{L_{\text{avg}}}{L_{\max}} \tag{4.1}$$

Green:

$$G_{\min} \leq H \leq G_{\max} \tag{4.2}$$

Yellow:

$$Y_{\min} \leq H \leq Y_{\max} \tag{4.3}$$

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

The value of β has experimentally been shown (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 a green area on the field. This image is then scanned for the minimum and maximum hue. G_{min} will be set equal to this minimum hue minus a 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 4.1) looks like Figure 4.2.

Figure 4.1: Raw image taken by NAO

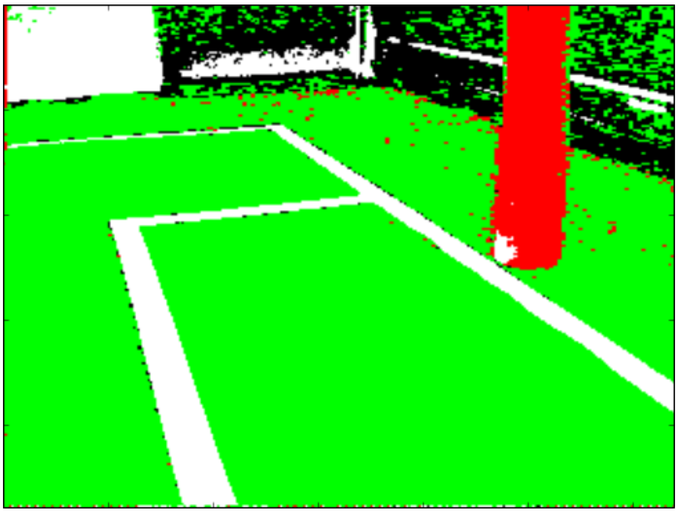

Figure 4.2: Image after color filter

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

4.2 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 not, the algorithm stops scanning this column of pixels and proceeds to the next one. This heuristic was added to avoid scanning parts of the goal's mast or noise in the background.
When the algorithm has decided that there are enough green pixels under a yellow one it continues to climb the vertical axis and counts 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 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.

4.3 Field Line Corner Point Extraction

Before any fieldlines can be extracted the background noise needs to be removed. The details of this operation will not be explained here. 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 (7 pixels in this specific case), erase all the pixels above this point and continue to the next column. More details about this operation can be found in [3]. Finally, any remaining green and yellow pixels are removed because they aren't needed any more. At this point the image will look like Figure 4.3.

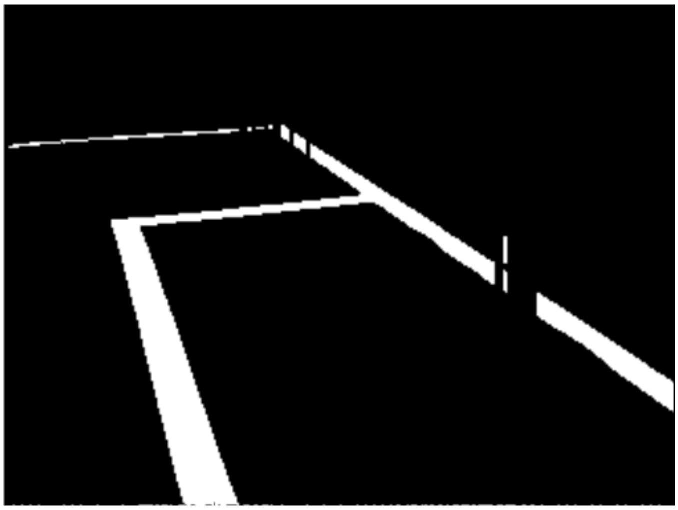

Figure 4.3: Background noise removed

There are quite some ways of obtaining the landmarks, three in this example. One in particular seems to be favoured in the world of robotics which is based on edge-detection and Hough transforms [4][5]: OpenCV's built-in *cv.Canny* and *cv.HoughLinesP* methods.
Canny is an edge-detection algorithm and returns an 8-bit image, see Figure 4.4.

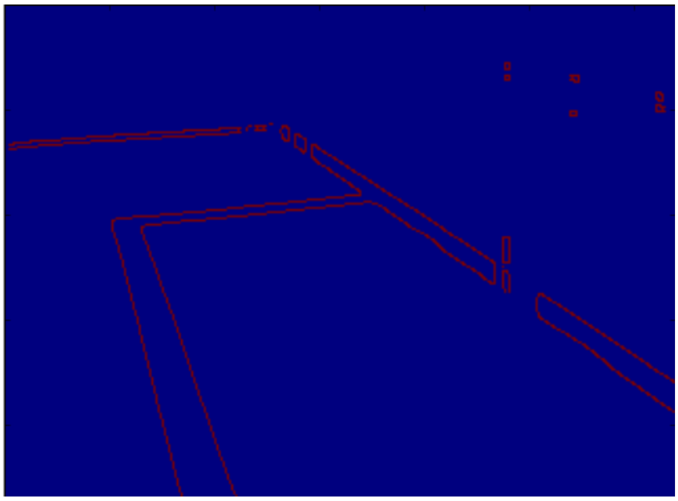

Figure 4.4: Image after applying Canny

After using the Canny method the second method is applied to the newly obtained image which returns a list of line-segments. These lines drawn on top of Figure 4.4 results in Figure 4.5.

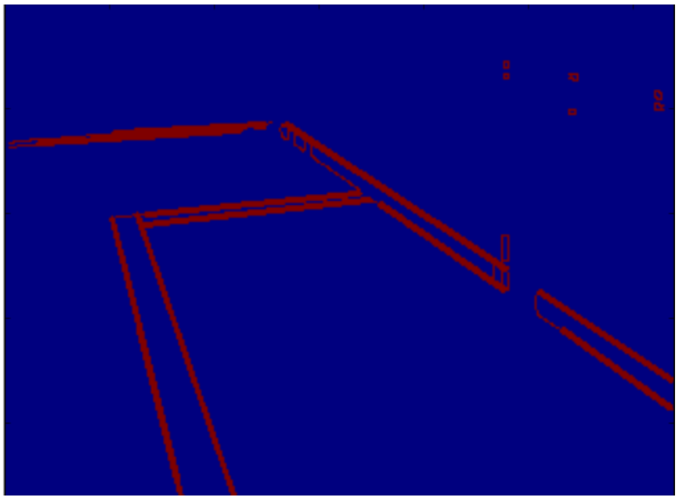

Figure 4.5: Image after applying Hough Transform

The thicker lines are the lines generated by the Hough Transform method. OpenCV's method for creating these lines takes a few parameters of which the following are of great importance because they form bottlenecks on the performance of the entire image processing operation: *maxLineGap* and *minLineLength*. *maxLineGap* is the maximum distance between two points to still be considered as a line. *minLineLength* is the minimum length a line should have. These parameters will be discussed in-depth in sections 6 and 7.
The only thing left to do is to find the intersections of these lines and deleting those that 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.

4.4 Distance Calculation

A landmark is represented by three parameters: the x and y coordinates of the landmark on the image and a boolean which is true if this landmark is a goalpost and false if it is a corner point. The last parameter has no value in calculating distance, but the first two do.
To calculate this distance the height, vertical angle and the horizontal and vertical fields of view of the camera and the dimensions of the image are required. For the sake of readability these are abbreviated as h_{c}, $\angle_{\mathrm{c}}$, fov_{h}, fov_{v}, w_{img} and h_{img} respectively.
To further ease the readability, a few prior operations are executed. First the $\angle_{\mathrm{c}}$ is adjusted such that it equals the angle of the bottom of the image:
$\angle_{\mathrm{c}} = \angle_{\mathrm{c}}$ - $\frac{fov_{\mathrm{v}}}{2}$
Then the offset of the x-coordinate from the center of the image is calculated:
x_{off} = x - $\frac{w_{\mathrm{img}}}{2}$
Next the horizontal angle of the x-coordinate with respect to the image is calculated:
$\angle_{\mathrm{x}} = \frac{x_{\mathrm{off}} \times fov_{\mathrm{h}}}{w_{\mathrm{img}}}$
Similarly the vertical angle of the y-coordinate is calculated:
$\angle_{\mathrm{y}} = \angle_{\mathrm{c}} + \frac{y \times fov_{\mathrm{v}}}{h_{\mathrm{img}}}$
Finally to get the distance:
$distance = \frac{h_{\mathrm{c}} \times tan\angle_{\mathrm{y}}}{cos\angle_{\mathrm{x}}}$ These distances combined with the third landmark parameter (boolean goalpost or not) are then passed onto the SLAM algorithm.
This concludes the section on Image Processing. The experiments on the earlier-mentioned variables and their results will be discussed in sections 6 and 7.

5 Localization and Mapping algorithms

5.1 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.

State Representation

During the runtime of the algorithm, the state is represented by a vector μ and a matrix Σ (see figure 4.1).

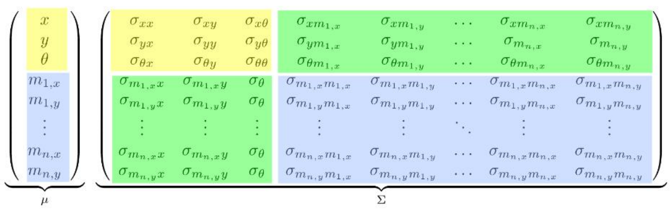

Figure 5.1: The vector of the state [10]

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, $m_{i,x}$ and $m_{i,y}$ are the x and y coordinates of the i^{th} 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 $3+2n$.

Algorithm Initialization

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 run a new iteration to compute an estimate of the new state using only the previous state estimate and the data obtained since the last state was estimated.

Algorithm Iteration

On a high level, each iteration of the algorithm can be split up in two steps; a prediction step, in which transition models of the system are applied to make a prediction of the new state vector μ and covariance matrix Σ, and an update step, in which information on previously observed landmarks and estimates of noise are used in an attempt to counteract noise. Any matrices used in this section are defined in more detail in appendix B, unless otherwise is stated.

Prediction Step

In this step, first the new robot position is predicted as follows, and the new values are entered in the first three positions of μ:

$$\begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} d \times cos(\theta) \\ d \times sin(\theta) \\ \Delta\theta \end{pmatrix}$$

where d is the distance the robot travelled forwards. Σ_t is computed as follows:

$$\Sigma_t = G_t \times \Sigma_{t-1} \times G_t^T + R_t$$

where R_t is the covariance matrix for motion noise. It contains values which need to be experimentally tuned.

Update Step

In the update step, the algorithm loops through the landmarks observed in this time-step. The following calculations will be performed for each landmark j. First, the Kalman Gain K_t^j is computed:

$$K_t^j = \Sigma_t \times H_t^j (H_t^j \times \Sigma_t \times H_t^{jT} + Q_t)^{-1}$$

where Q_t is the covariance matrix for measurement noise. It contains values which need to be experimentally tuned.

The Kalman Gain can intuitively be thought of as a matrix with numbers indicating how much information we want to gain from differences between expected measurements and actual measurements.

Next, μ_t is updated:

$$\mu_t = \mu_t + K_t^j (z_t^j - \hat{z}_t^j)$$

where z_t^j contains the measurements concerning landmark j in the current time-step. The subtraction $(z_t^j - \hat{z}_t^j)$ is the difference between the measurement performed at this time-step and the measurement which the robot would expect given previous information, so this computation clearly shows the intuition behind the Kalman Gain described above.

Finally, Σ_t is updated for the last time in this iteration:

$$\Sigma_t = (I - K_t^j \times H_t^j)\Sigma_t$$

5.2 GraphSLAM

Introduction

GraphSLAM is an algorithm for mapping using sparse constraint graphs. The basic intuition behind GraphSLAM is simple: GraphSLAM extracts from the data a set of constraints, represented by a 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[6]. 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 :

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

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

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 noise parameters represent the amount 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 $\varepsilon_{measurement}$. Usage and approximation of these two variables will be explained more in detail in Experiments section where better values for them are obtained by experiments. 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:

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

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

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 [7], it is known that if x represents best estimates of robot poses and landmark positions, the following equation holds :

$$\Omega \times x = \xi \tag{5.1}$$

Using equation (5.1), it is possible to find x using the Ω and ξ matrices. To do so, the following computation is used :

$$x = \Omega^{-1} \times \xi \tag{5.2}$$

Equation (5.2) 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[6, 7] and it will be the one of the main focus points of experiments section of this paper as well.

5.3 Complexity Analysis

As shown in [11], the time complexity of each iteration in EKF SLAM is $O(n^2)$, where n equals the number of landmarks observed up to that point in time. The algorithm's running time is dominated by the matrix multiplications involving Σ, which is a $(3+2n)\times(3+2n)$ matrix. Σ also dominates the memory requirements of the algorithm, with all other used matrices being equal or smaller in size, and therefore the space complexity of the algorithm is also $O(n^2)$. It is important to observe that both time and space complexity of the algorithm only grow as a function of the number of landmarks observed, and not as a function of time.

In GraphSLAM, dominating matrix is Ω as all the other matrices and vectors used in computations are smaller in size. Size of Ω at any time step t_i is $(i + numLandmarks) \times (i + numLandmarks)$. This can be represented as function of time as it grows per each time step. This leads to the fact that, space complexity of GraphSLAM is $O(N(t)^2)$. As for time complexity, computationally most intensive part of GraphSLAM is inversion of Ω matrix, which leads time complexity to be $O(N(t)^3)$. It is important to observe that time and space complexity of GraphSLAM grow as a function of time and observed landmarks, which is not the case in EKF SLAM.

6 Experiments

6.1 Image Processing

As was announced in section 3, the following variables will be experimented with:
β (minimum luminance of pixels to classify as 'white')
The parameter values of the Hough Transform *maxLineGap* and *minLineLength*
These variables were chosen for experimentation because these were the bottleneck-points of the image processing. In order to measure performance, the cost matrix in Table 6.1 is applied to each test-instance where the goal is to minimize the cost.

Table 6.1

		Landmark exists: true	Landmark exists: false
detected landmark	true	-1.333	5
	false	1	0

Note that classifying something as a landmark even though it's not gets a relatively big penalty whereas failing to detect a landmark gets a lower penalty. This is done because the SLAM algorithms are capable of dealing with the latter (false negative), but have a lot more trouble handling the former (false positive).
Thirty images were made from random locations on the soccer field as test-instances with a resolution of 320×240 pixels. Because the three mentioned variables are all dependent on each other they all have to be tested simultaneously. Because of this, some pre-testing has been done by hand to find a region of convergence in order to reduce the table's size. The table containing the final experiments and their results can be found in Table A.1 in the appendix and will be discussed in the next section.

6.2 Noise Parameters

Both of the implemented SLAM algorithms have parameters which need to be experimentally determined, to give the algorithms information on how much noise we expect in motion and measurements.

In order to estimate the influence of noise on motion, the robot was ordered to move a variety of distances forwards. When the robot finished his movement, the actual distance he moved was measured and compared to the given command.

In a similar way, the influence of noise on measurements from image processing was estimated. The robot was tasked to print distances to landmarks obtained through image processing, and these distances were compared to distances measured in reality.

6.3 Running SLAM

After image processing and noise parameters are experimented and values for parameters are found, last experimentation step is to run SLAM algorithms on NAO. Couple of experiments are done and results are plotted on User Interface. Those pictures will be included in results section. One important thing to recognise is that, number of experiments are not high meaning more improvements could have been made on parameters and computations which will be the topic of next section. Three types of setup for experiments were created:

In first setup, noise parameters were set to 1, meaning measurements and motion motors are completely trusted. So, assumption was that there is no noise in world. Landmark association was set to 30cm, meaning if two landmarks were at most 30 cm off from each other, they will be recognized as being same landmark.

In second setup, noise parameters were adjusted to better value using results of previous experimentations. In this setup, noise was introduced to computations. However, data association value was not changed.

In third setup, data association value is adjusted to more realistic value as well using both results of previous two setups and results of previous experiments.

7 Results

7.1 Image Processing

The best results (least $\overline{cost}$) are obtained for $\beta = 120$, $maxLineGap = 5$ pixels and $minLineLength = 40$ pixels. Note that -as described in the previous section- the resolution of an image is 320×240 pixels, so the values of $maxLineGap$ and $minLineLength$ will probably not apply to other resolutions. The reason the $maxLineGap$ is relatively low and the $minLineLength$ relatively high w.r.t. the resolution is because the cost matrix nudges the variables towards a safe value. It rather sees a landmark undetected than a misclassified landmark, which is -as explained in the previous section- preferred.

7.2 Running SLAM

GraphSLAM

First, GraphSLAM was experimented as an offline solver for SLAM problem. It should be mentioned that number of time steps was too small for GraphSLAM to yield any good results but algorithms still managed to improve their results even with small number of time steps with adjustments made in each test setup.

In first experiment, as mentioned, noise parameters were set to 1.0, meaning no noise is expected in computations. Results, as expected, were off. Following image shows result of this first experiment.

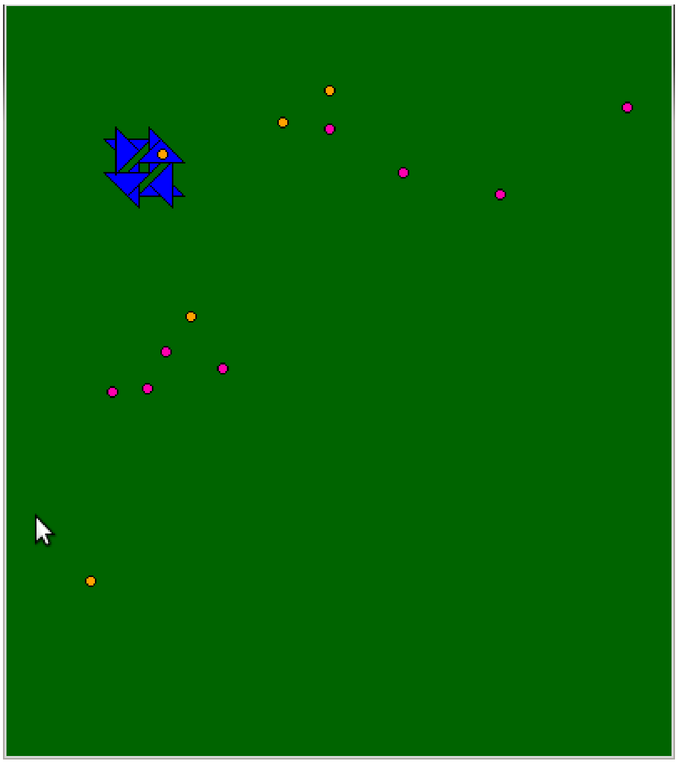

Figure 7.1: Result of first GraphSLAM experiment

As it can be seen from image, results are quite off and it sees more than two goal posts. These relates to the fact that, image processing sometimes accidentally recognises walls of room as being goal post as their colours are very similar. Also, as noise is not introduced, placement of objects do not make much sense at this point.

In second experiment that we ran using GraphSLAM yielded a little bit more stable results.

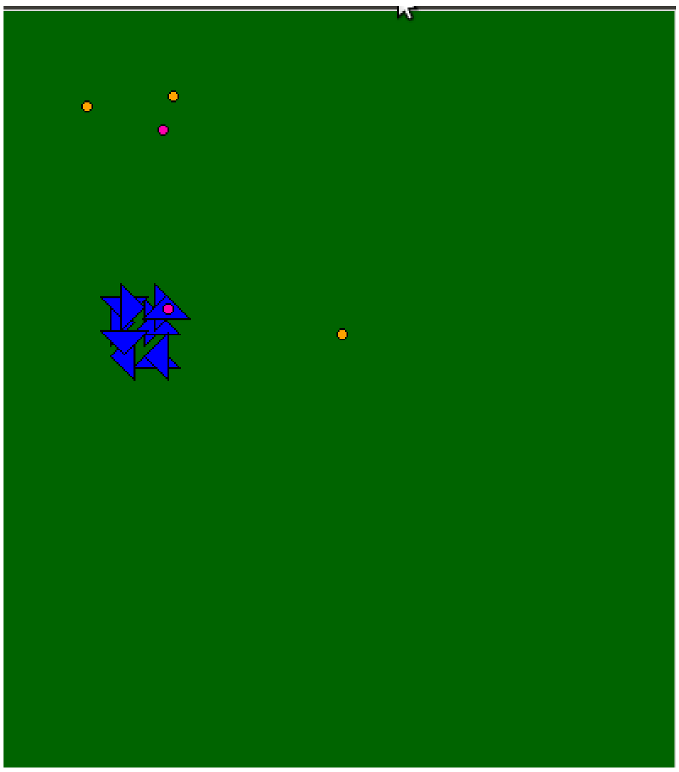

Figure 7.2: Result of second GraphSLAM experiment

As it can be seen from this image, results are making more sense, as data association starts to work better as number of landmarks and goal posts that are very close to each other are counted as being one landmark, noise is introduced helping GraphSLAM localize and create map of surrounding better.

In last experiment performed using GraphSLAM results are also compared to actual football field in order to see if results can be related to actual field in some way.

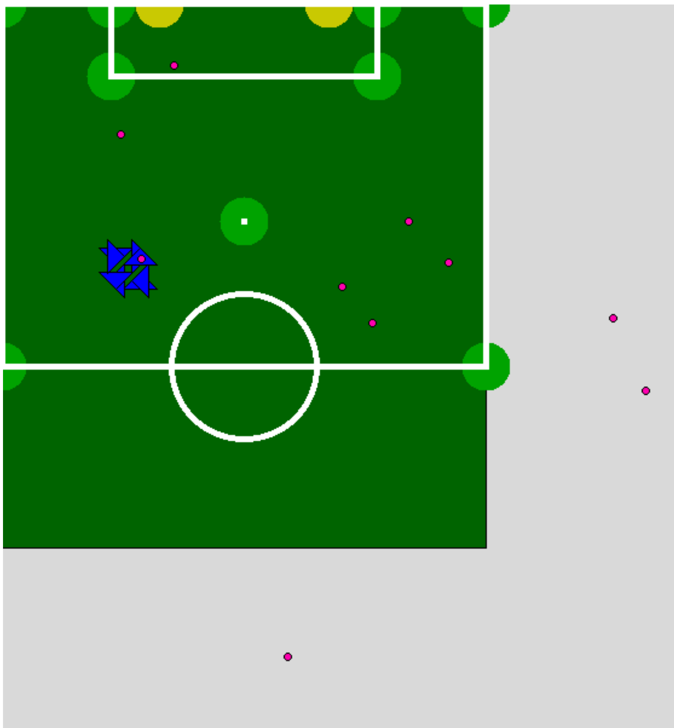

Figure 7.3: Results of last experiment using Graph-SLAM

Even though, NAO did not see any goal posts during the experimentation(again because of small number of time steps), landmarks found by it seem to be better mapped onto the world. As expected, there is still noise in results.

As already mentioned, the biggest reason why experiments did not yield more meaningful results was the lack of more time steps during experimentations.

EKF SLAM

Testing the entire software package on the NAO on the football field did not result in accurate results. Many coordinates of both the robot and the landmarks would diverge to lie outside the actual dimensions of the soccer field. Similar errors were observed in earlier experiments using a simulated world when the noise parameters were incorrect. After tuning the noise parameters for the simulated world, the EKF SLAM algorithm did give accurate result. It is likely that, with more testing for noise parameters, accurate results could be obtained with the NAO.

8 Future Work

8.1 Image Processing

A way to decrease the noise obtained from approximating the landmarks during image processing is to apply particle filtering. Particle filtering is a general Monte Carlo (sampling) method for performing inference in state-space models where the state of a system evolves in time and information about the state is obtained via noisy measurements made at each time step [8].

Measuring the distance to a landmark is often noisy due to slight deviations in the vertical angle of the camera and the torso of the Nao. To approach this problem it is suggested to at the same time as the current camera, take a picture with the other camera and compare these images to obtain the distance to a landmark using static distances (absolute and angular distances between cameras).

8.2 SLAM algorithms

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

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 [9]

Landmark Association

One of the problems faced by both SLAM algorithms discussed in this paper is about association of landmarks. Using euclidean distances between approximated landmarks in order to find which ones might be same does not work in all the cases. One possible improvement to this problem is to use Mahalanobis distance. Mahalanobis distance is a descriptive statistic that provides a relative measure of a data point's distance from a common point. Its main difference from Euclidean distance that it takes into account the correlations of the data set and is scale-invariant. In other words, it has a multivariate effect size.

Intuitively, Mahalanobis distance estimates the probability of some point belonging to set of points. So, first step is to find center or mass of the set of points and then comparing given point with those centers. One issue with this approach is the fact that set might be spread out over large or small ranges. In order to fix this problem, simplistic approach that is used in many cases is estimation of standard deviation of the distances of sample points from center of mass. If, distance is smaller than one standard deviation then we can conclude it is highly probable that point belongs to that set.

Active and Fast SLAM

Also, there exist some other approaches to solve SLAM problem. One of them is Active SLAM. Main advantage of Active SLAM is that it also tries to solve exploration problem by finding best next move in order to build the map as efficiently as possible. Second possible algorithm is FastSLAM. Advantage of FastSLAM over GraphSLAM and EKF SLAM is that it also deals with kidnapping problem. Kidnapping problem refers to the case where robot is placed somewhere else in the world during the simulation without robot knowing about it. So, these two approaches might be interesting alternatives to solution of SLAM problem.

9 Conclusions

9.1 Image Processing

The biggest issue with the Nao in general is noise. In the image processing phase this noise is minimized by increasing the resolution of the images the Nao takes. For calculating the distance to a landmark, the noise becomes a bigger obstacle which is caused by mainly two factors: deviation in the by the Nao measured vertical angle of the camera and deviation in the angle of the torso of the Nao (not standing entirely straight when taking picture). The combination of these two sometimes balance each other out, but mostly give a noise between -2 and +2 degrees in vertical camera angle. For landmarks that are far away this often leads to distances that are 10 - 30 cm off. To reduce this error the second camera also has to take a picture at the same time as the camera currently in use as described in the section on future research.

9.2 SLAM Algorithms

As described in section 5.3, the running time and memory requirements of the EKF SLAM algorithm are affected by the number of landmarks, whereas the running time and memory requirements of the Graph SLAM algorithm are mostly affected by the number of time steps. Because the number of landmarks on the football field is relatively small, EKF SLAM does not have any difficulties with respect to running time and memory. In the experiments carried out for this project, there was no noticeable difference in running time between EKF SLAM and Graph SLAM, which can be explained by the fact that the experiments had a relatively low amount of time-steps.

Ideally, the algorithms should be tested with a much larger number of time-steps. This would allow both algorithms to deal better with noisy data. This was not possible to do due to time concerns. Such experiments would likely show a more noticeable difference in running time between EKF SLAM and Graph SLAM. It would most likely also lead to higher quality maps.

References

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

[2] For RoboCup History, *http://www.aldebaran-robotics.com/For-Robocup/history.html*,last accessed on 22-01-2014

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

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

[5] 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

[6] 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.

[7] 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.

[8] Emin Orhan, "Particle Filtering" *http://www.cns.nyu.edu/ eorhan/notes/particle-filtering.pdf*, 2012

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

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

[11] S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics. Cambridge, MA, USA: MIT Press*, 2005.

A Image Processing

Table A.1

β	*maxLineGap*	*minLineLength*	$\overline{cost}$	σ
110	4	30	6.233	1.720
120	4	30	5.333	1.566
130	4	30	4.755	1.322
110	5	30	6.831	2.008
120	5	30	5.822	1.611
130	5	30	5.133	1.568
110	7	30	9.655	2.634
120	7	30	7.328	2.304
130	7	30	5.487	1.703
110	4	40	4.223	0.982
120	4	40	3.655	0.623
130	4	40	3.754	0.809
110	5	40	3.124	0.510
120	5	40	2.122	0.322
130	5	40	2.433	0.389
110	7	40	4.032	1.343
120	7	40	3.422	0.780
130	7	40	3.971	1.084
110	4	50	5.322	1.472
120	4	50	4.932	1.328
130	4	50	5.143	1.407
110	5	50	5.122	1.902
120	5	50	4.722	2.103
130	5	50	5.002	1.514
110	7	50	5.483	1.495
120	7	50	4.820	1.278
130	7	50	5.103	1.300

B EKF SLAM Matrices and Equations

This appendix provides a detailed view of some of the matrices and equations used by the EKF SLAM algorithm.

B.1 Prediction Step Matrices

$$G_t^x = \begin{pmatrix} 1 & 0 & -\Delta y \\ 0 & 1 & \Delta x \\ 0 & 0 & 1 \end{pmatrix}$$

where Δx and Δy represent the distances travelled by the robot along the 2 axes.

$$G_t = \begin{pmatrix} G_t^x & 0 \\ 0 & I \end{pmatrix}$$

B.2 Update Step Matrices

Let $\begin{pmatrix} \mu_{j,x} \\ \mu_{j,y} \end{pmatrix}$ be the current estimate of landmark j's position, and let $\begin{pmatrix} \mu_{t,x} \\ \mu_{t,y} \end{pmatrix}$ be the current estimate of the robot's position. Then $\hat{z}_t^j$ can be computed as follows, giving the distance and relative angle at which we would expect to observe landmark j.

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

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

$$\hat{z}_t^j = \begin{pmatrix} \sqrt{q} \\ atan2(\sigma_y, \sigma_x) - \mu_{t,\theta} \end{pmatrix}$$

Some of the values computed above can now be used to compute H_t^j for landmark j as follows:

$$^{low}H_t^j = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}$$

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

$$H_t^j = {}^{low}H_t^j \times F_{x,j}$$

where N is the number of landmarks observed so far.