Simultaneous Localization and Mapping (SLAM) on NAO

By

LEE YUAN HUANG

FINAL PROJECT REPORT

Submitted to the Department of Electrical & Electronic Engineering
in Partial Fulfilment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

UniversitiTeknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak DarulRidzuan

CERTIFICATION OF APPROVAL

Simultaneous Localization and Mapping (SLAM) on NAO

by

Lee Yuan Huang

A project dissertation submitted to the
Department of Electrical & Electronic Engineering
UniversitiTeknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Approved:

____

AP Dr AamirSaeed Malik
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

September 2012

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

Lee Yuan Huang

ABSTRACT

Simultaneous Localization and Mapping (SLAM) is a navigation and mapping method used by autonomous robots and moving vehicles. SLAM is mainly concerned with the problem of building a map in an unknown environment and concurrently navigating through the environment using the map. Localization is of utmost importance to allow the robot to keep track of its position with respect to the environment and the common use of odometry proves to be unreliable. SLAM has been proposed as a solution by previous research to provide more accurate localization and mapping on robots.

This project involves the implementation of the SLAM algorithm in the humanoid robot NAO by Aldebaran Robotics. The SLAM technique will be implemented using vision from the single camera attached to the robot to map and localize the position of NAO in the environment. The result details the attempt to implement specifically the chosen algorithm, 1-Point RANSAC Inverse Depth EKF Monocular SLAM by Dr Javier Civera on the robot NAO. The algorithm is shown to perform well for smooth motions but on the humanoid NAO, the sudden changes in motion produces undesirable results.This study on SLAM will be useful as this technique can be widely used to allow mobile robots to map and navigate in areas which are deemed unsafe for humans.

ACKNOWLEDGEMENTS

This project would not have been completed (or near completion) without the help and encouragement from my supervisor, Dr.AamirSaeed Malik and YasirSalih who has guided me by trying to explain the concepts in a simplified manner. I have indeed wasted much time overthinking instead of starting from the small parts and working on it in a sequential manner. Nevertheless, I would say, there is no definite end to this project till an accurate representation of the environment and the robot’s motion is achieved which I am unable to replicate in this project. This is indeed a regret which I hope will be solved in a much better way one day. My greatest thanks go to my two supervisors.

I would also like to thank Dr. Javier Civera for allowing his code to be used for this implementation. It is great that he took the time to answer a few of my queries often times with suggestions for other reading materials concerning the algorithm used despite his busy schedule.

I shall also thank my family members and friends for their support during the hard times of the project. Without which, I would probably have given up on the project a few months ago. My thanks also to the many researchers which materials have been helpful in helping me understand the subject of study better, and not to forget the many contributors who have enriched the Internet with their knowledge and sample codes/libraries, including those in the Aldebaran forums.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SLAM –	Simultaneous Localization and Mapping
RANSAC –	RANdom Sample Consensus
EKF –	Extended Kalman Filter
LIDAR –	Light Detection and Ranging

CHAPTER 1
INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a concept that is introduced to solve the question on how a robot can be navigated in an unknown environment. To be able to do so, the robot will need to create a map of the environment and at the same time, know its location and motion in the map.Many solutions have been proposed in this field of study and one of them for monocular vision is applied in the project here.

1.1 Background of Study

The use of mobile robots is becoming more prominent as the technology advances further thus allowing faster and heavier computation to be processed. With advancement in hardware for processing, robots nowadays can be utilized to perform heavy computations such as using vision in real-time for navigation. One of the important challenges in SLAM is to create a system which is efficient enough to be executed in real time.

Incidents such as natural disasters and the nuclear plant accident in Fukushima have shown to the world that robots are necessary for automated navigation and mapping in an unknown area in cases of emergency where it is too risky for humans. However even for the case mentioned above, remote control is still used and that posed a problem if there is a loss in connectivity. Google’s driverless car on the other hand has showcased the amazing capability possible with the use of SLAM. SLAM is also widely applicable in other fields such as subsea mapping, oil and gas exploration, or even space exploration on other planets.

1.2 Problem Statement

Although it is possible for the NAO robot to navigate by calculating its position based on its motor, the odometry is still prone to error especially after a longer distance. If motion can be modeled using visual odometry and error of the method canbe estimated, it can be used to correct NAO trajectory[1]. The robot will also need to avoid dangerous obstacles which might damage the robots. Although it can be easily done using sonar or laser sensors in an unknown environment, without a map, the robot will only perform simple obstacle avoidance without knowing its direction and position.

By implementing SLAM on the robot, the robot will be consistently aware of its position relative to the landmarks in the environment. Thus, it will be able to navigate in a map accurately as SLAM allows the robot to localize effectively using the measured data from the camera and its predicted position based on the control input given.

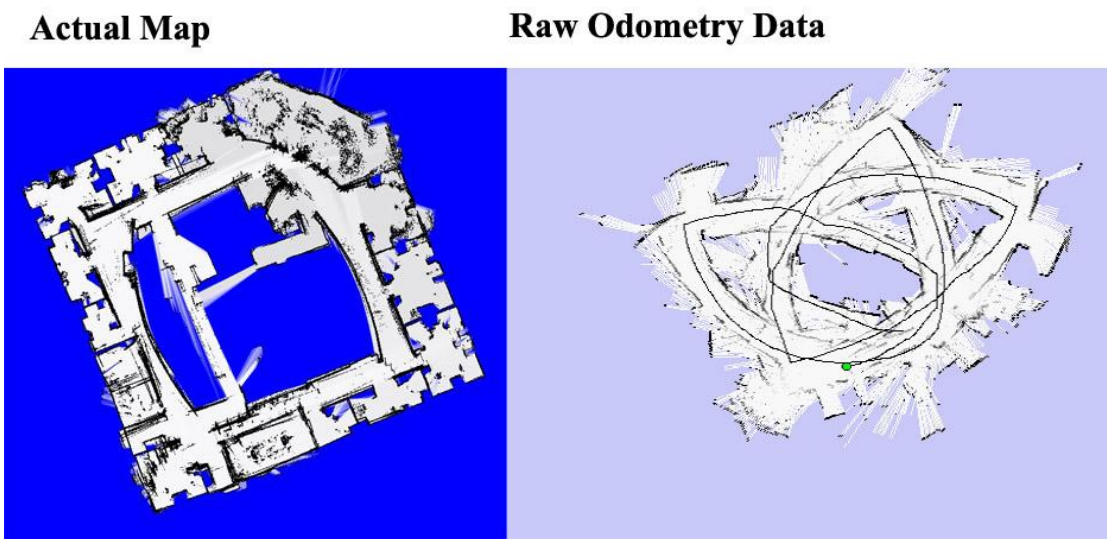


Figure 1: Error from raw odometry [2]

1.3 Objective and Scope of study

The main objective of the project is:

Implementation of SLAM on NAO which consists of these side objectives which are:

- Kinematic Modeling of the NAO Robot.
- Landmark detection and extraction from the environment.
- Data association.
- Using Extended Kalman Filter (EKF) for estimation of robot position and landmark.

The scope of study in this project involves first familiarization with the robot to program on it. It involves the use of a few programming languages which are Python, and MATLAB. The algorithms which are used will not be new but existing ones such as the Extended Kalman Filter and data association using 1-point RANSAC

The kinematic modeling of the robot is not undertaken and has been replaced with a simpler model being assumed.

1.4 Relevancy and Feasibility of Project

The project is relevant in our studies because it is one of the major problems in robotics and comprises of different fields, such as probabilistic robotics, image processing and programming knowledge which are all certainly educational. The implementation of SLAM on NAO has never really been attempted and this will be the first few tried out. The findings here would hopefully speed up the progress of others when implementing SLAM on NAO in future.

The project is feasible since SLAM has already been proven to be able to localize and map a robot’s movement in an environment. However within the time frame for a beginner, it doesn’t leave much room for trial and error to test out other algorithms if it has been implemented but with unsatisfactory results.

CHAPTER 2
LITERATURE REVIEW& THEORY

2.1 Overview of SLAM

Simultaneous Localization and Mapping (SLAM) is a problem in robotic navigation which has been explored for decades. It is not a new problem and many solutions has been proposed theoretically and also practically implemented on robots. Robotic navigation can be summarized as being concerned with the questions: “where am I?”,“where am I going?” and “how should I get there?”[3] In SLAM the same questions are posed with the focus being on the first two questions. The robot will need to know its position in an unknown environment and map it at the same time so that it will be able to head towards its destination. The question of getting there can be answered using path planning techniques instead such as the popular A* search algorithm.

The essential SLAM problem consists of producing a simultaneous estimate of the robot and the landmark locations. The true locations are not provided but are inferred based on the observations made between the robot and the landmarks. [4]

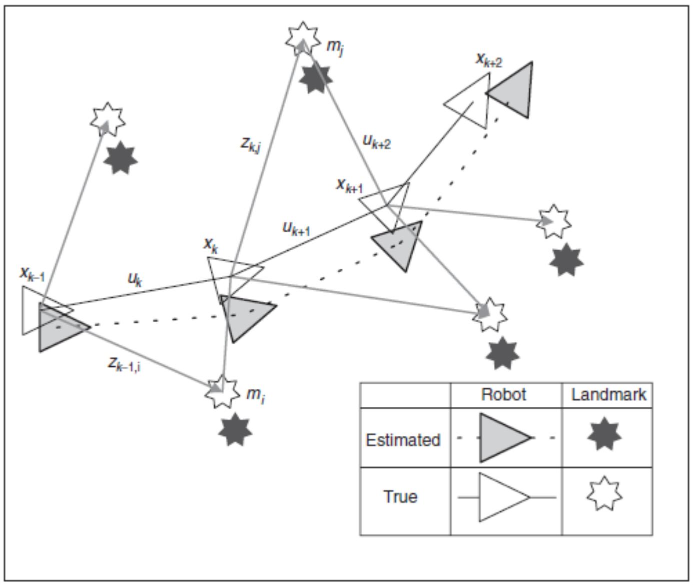


Figure 2: The essential SLAM problem. [4]

SLAM can be performed by combining the use of dead-reckoning and many different types of sensors such as sonar [5], laser range finders[6] and also the use of active vision[7]. Most of the SLAM algorithm implemented has been done on wheeled robots[8] or vehicles[9], Unmanned Aerial Vehicles (UAVs), and Autonomous Underwater Vehicle[10]. Although the implementations in humanoid robots like NAO is rare SLAM on humanoid have actually been performed[11].

The implementation of SLAM consists of several main parts which are kinematic modeling, landmark extraction, data association; state estimation and landmark update[12]. SLAM which are normally performed on wheeled robots or vehicle are usually modeled using simpler models like the bicycle model used for Ackerman steered vehicles as presented by Bailey during the SLAM Summer School 2009[13]. However for NAO robot, it utilizes a humanoid model and so other techniques like the Denavit-Hartenberg (D-H) robot modeling technique is used to create the motion model for the robot[1][14]. Landmark extraction involves capturing images of the surrounding environment, detecting and extracting the easily distinguishable features such as walls, corners or patterned objects. Data association will be used after the landmarks are extracted to match the landmarks after each repeated observation. Visual SLAM implementation utilizes different types of feature (landmark) extractors such as Harris Corner Detector[15], Kanade-Lucas tracker and SIFT[16] while for data association, the nearest neighbor method can be implemented for feature matching. A more robust method is the Jointly Compatible Branch and Bound which can be used for more accurate data association as shown in the comparison done by Temeltas[17]. These three feature extractors are compared in the report by Klippenstein[18] and the conclusion reached is that the choice of feature extractors is not critical. The estimation of states is usually performed using the most important methods which are EKF-SLAM and FastSLAM[4]. Landmark update will be done after getting the estimates for the states by repeating the observation of the surroundings and adding new landmarks to the map.

The SLAM algorithm will be implemented in our project using EKF-SLAM which has proven to produce acceptable consistent results if the heading variance remains small [19]. FastSLAM is also a popular algorithm used in SLAM and is shown to outperform the EKF in environments with ambiguous data association[20] and can be scaled to be applied in larger environment[21]. Another algorithm which is used for SLAM is Graph-based SLAM which has undergone a renaissanceand currently belongs to the state-of-the-art techniques withrespect to speed and accuracy[22].

2.2 EKF SLAM

The theory of the Extended Kalman Filter will be described in this section. The Kalmanfilter is commonly used to solve estimation problems with two main steps which are predict and update. In the prediction step, the probability distribution function p from the previous step (k-1) will be updated to the current state k based on the probabilistic dynamic model of the system of the system which could be the motion model of the robot such as the velocity and the steering angle.

$$p(x_{k|k-1}) = \int p(\, x_{k|k-1} | x_{k-1|k-1,} u_k) p(x_{k-1}) dx_{k-1}$$

In the update step, the measurements from sensors for example will be combined with the probability distribution function previously obtained using Bayes' rule.

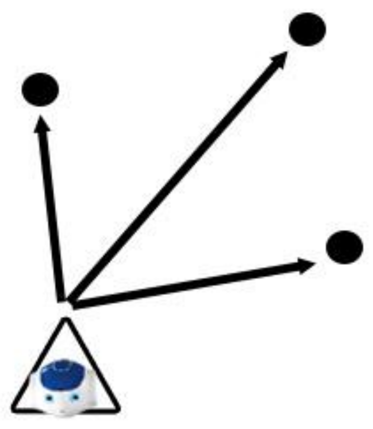
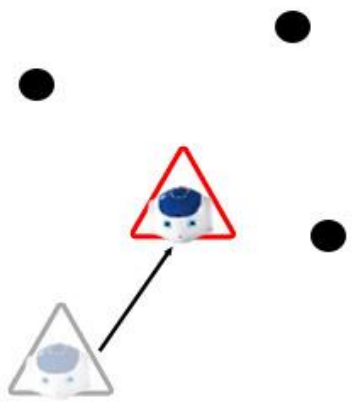
$$p(x_{k|k}) = np(z_k | x_{k|k-1}) p(x_{k|k-1})$$

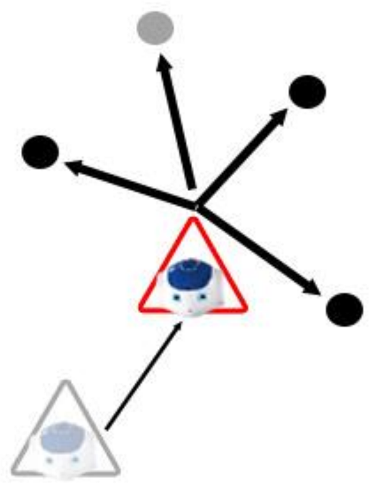
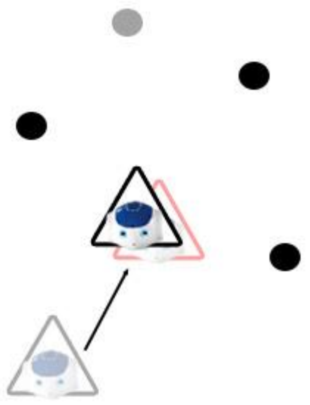
Although the Kalman filter is highly efficient due to it being a recursive algorithm, for use in SLAM, the extended Kalman Filter is used instead because of the non-linear nature of robotics system.

For EKF-SLAM, the process consists of three steps which are:

- Prediction
- Observation / Measurement
- Update

For a basic SLAM the first step consists of predicting the next position of the robot using a given control input, for example, the command to move the robot to the right by one meter and to the front by 0.5 meter which will result in a change in the state position of the robot (x + 1m, y + 0.5m). In the second step which is called observation, the landmarks surrounding the robots are detected using sensors such as cameras or LIDAR devices. The landmarks which are previously observed will be used to update the position of the robot since the landmarks are assumed to be fixed, unchanging fixtures.

	The triangle represents the robot while the ellipses are the landmarks observed. The map is initialized with the position of the landmarks based on the initial robot position which is assumed to be error free.
	The robot moves and based on the odometry it assumes that it is now in the red coded position.

	The robot measures again the landmarks by reobserving the previously initialized landmarks. New landmarks are also initialized. The result from the reobservation (measurement) from the sensors does not match the odometry results.
	The location of the robot is updated after fusing the information from the sensors of the robot and the odometry data.

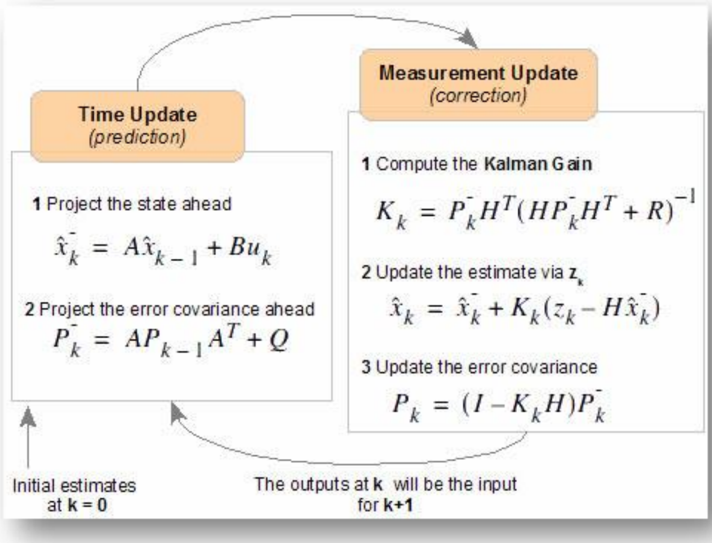

Time Update
(prediction)
1 Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
2 Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update
(correction)
1 Compute the Kalman Gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
2 Update the estimate via z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
3 Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates at k = 0
The outputs at k will be the input for k+1

Figure 3: EKF SLAM flowchart for single iteration [23]

2.3 MonoSLAM

For this project, the implementation of SLAM will focus on the usage of a single camera. For that reason, for this report, the section concerning MonoSLAM (monocular SLAM) will be added as it is a very crucial part of this final year project. The difference between MonoSLAM and the system that we are using is that MonoSLAM is implemented purely on a single camera which will be moved by the user instead of using a mobile robotics. Therefore, MonoSLAM is more complicated as it requires the motion of the camera to be estimated purely using vision.

On the other hand, on the Nao robot, we will be able to get the trajectory model of the camera movement by using the information from the movement of the robot. Nevertheless, MonoSLAM and its improved variants will be extremely useful for our case. The concept and algorithm used in MonoSLAM can be added on to our current implementation for the purpose of landmark recognition and associating the landmarks.

In the paper by Davison[24], point features are observed using a handheld camera. The system is initialized by using a known object with four initial landmarks, which are the corner points.

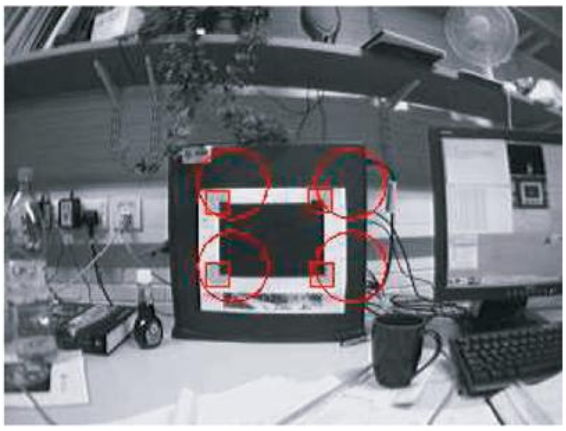


Figure 4: Known target used by Davison to initialize system

This initialization will allow the scale of the world to be known, and at the same time allow the pose of the camera to be tracked when other new landmarks are observed. The new landmarks are selected from the portion of the image that currently has no landmarks and the region where the camera is heading towards is prioritized. The point with the best Shi-Tomasi score[25] will then be chosen from that region so that the number of points will be minimized to reduce the computational effort. An 11x11 image patch centered at the chosen point is then saved for matching later using normalized cross correlation (NCC). When the camera moves, the landmarks will move out of the camera's view and so new landmarks will be initialized to ensure that a sparse set of 10 landmarks is maintained. Landmarks with too many failed observations will be removed to add in newer, more reliable landmarks. However due to the sparse sets of landmarks, the Davison system can be further improved if more landmarks can be observed without affecting the needed frame rate for real-time operation. This system can be a good starting point for our project although it is limited to a small sized room due to the limitation on the total number of landmarks.

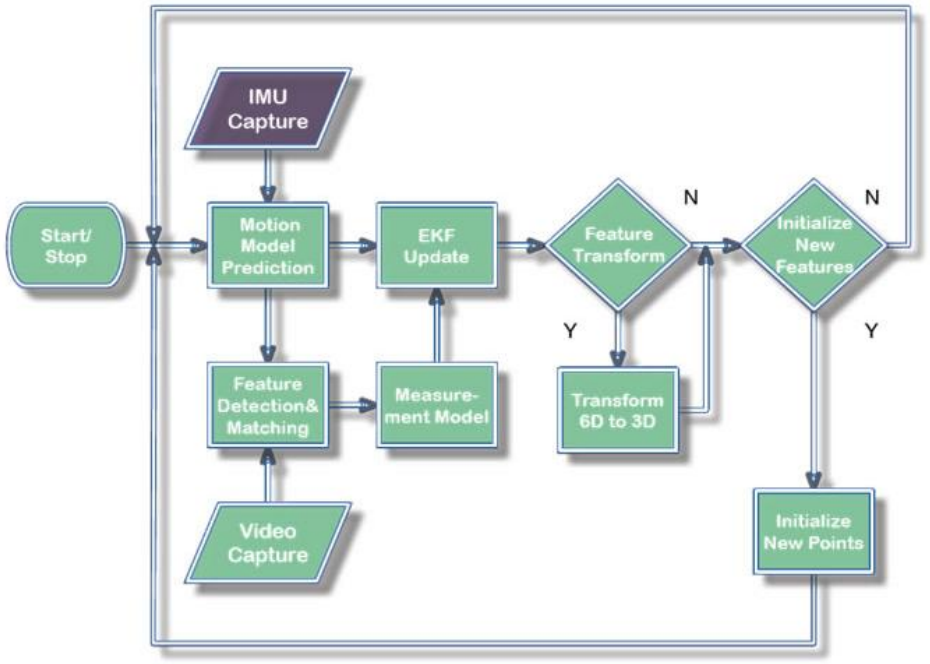


Figure 5: Flow chart of the MonoSLAM implementation [23]

The system is improved by Williams[26] and his thesis is taken as reference in our review here. In the system implemented by Williams, landmarks are represented using inverse depth parameterization[27] which allows multiple landmarks to be initialized simultaneously. This improvement eliminates the need of a known target as used in the Davison system to set the scale. The system utilizes the FAST corner detector and then it will perform the same thresholding as in the Davison system by using the Shi-Tomasi score. Active search and the JCBB algorithm are used for matching to reject false matches.

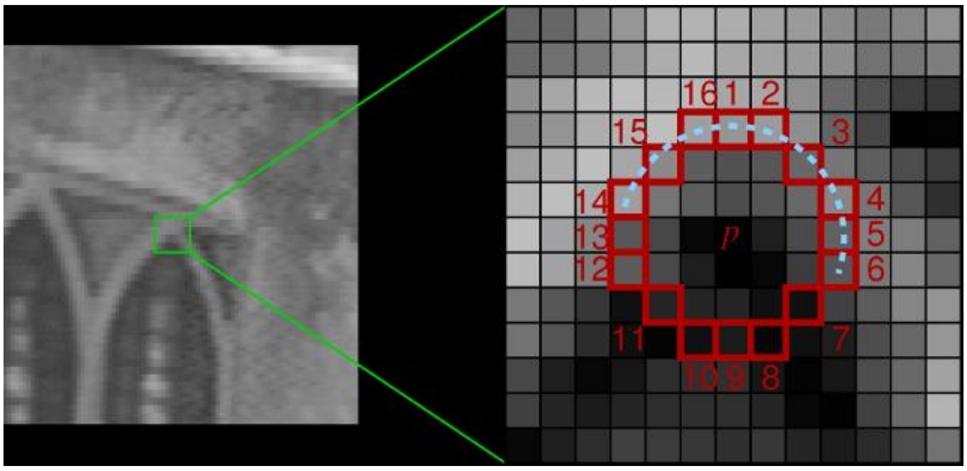


Figure 6: FAST Feature detection on image patch. The pixels used in the feature detection is highlighted[28]

Another monocular SLAM implementation which we will review is that done by Civera[29]. The contribution of this paper is in the section of data association which previously has been confined for quite some time to the Joint Compatibility Branch and Bound (JCBB) method by Jose Neira in 2001[30] especially for use with the EKF framework. JCBB eliminates spurious measurements by choosing the hypothesis that has the largest number of compatible pairings. It searches through all possible matches the largest set thatis jointly compatible and thus proves to be a much better choice than the nearest neighbor approach. However the 1-point RANSAC method proposed by Civera serves to overcome some of the shortages faced by the widely used JCBB method. In terms of computational costs, the interpretation tree search that JCBB uses to extact the largest jointly compatibleset of matches has exponential complexity when spurious observations are present. This limits the number of the observations to around 10 to 12. On the other hand, the 1-point RANSAC is linear in the state and measurement size, thus having a lower costvariation with the number of outliers. Therefore, these papers are all essential in building the concept that we would like to implement in our SLAM.

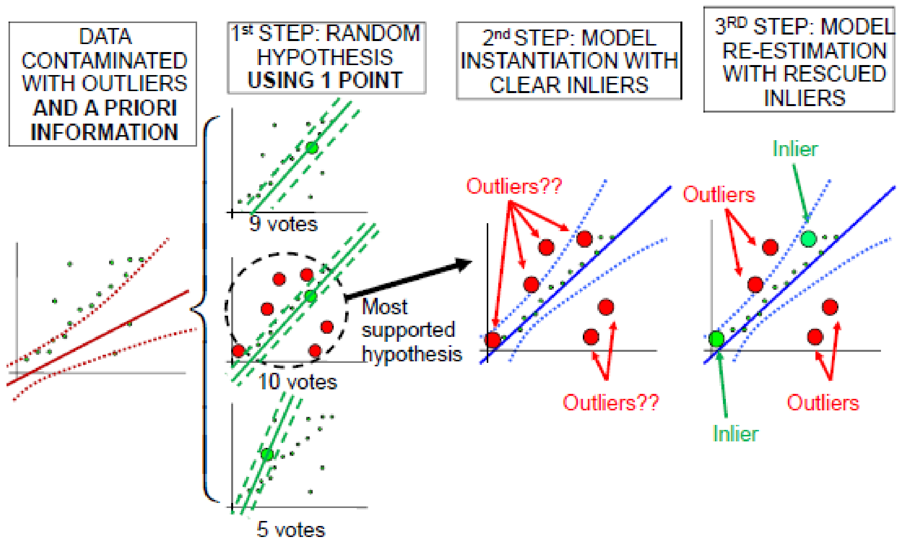
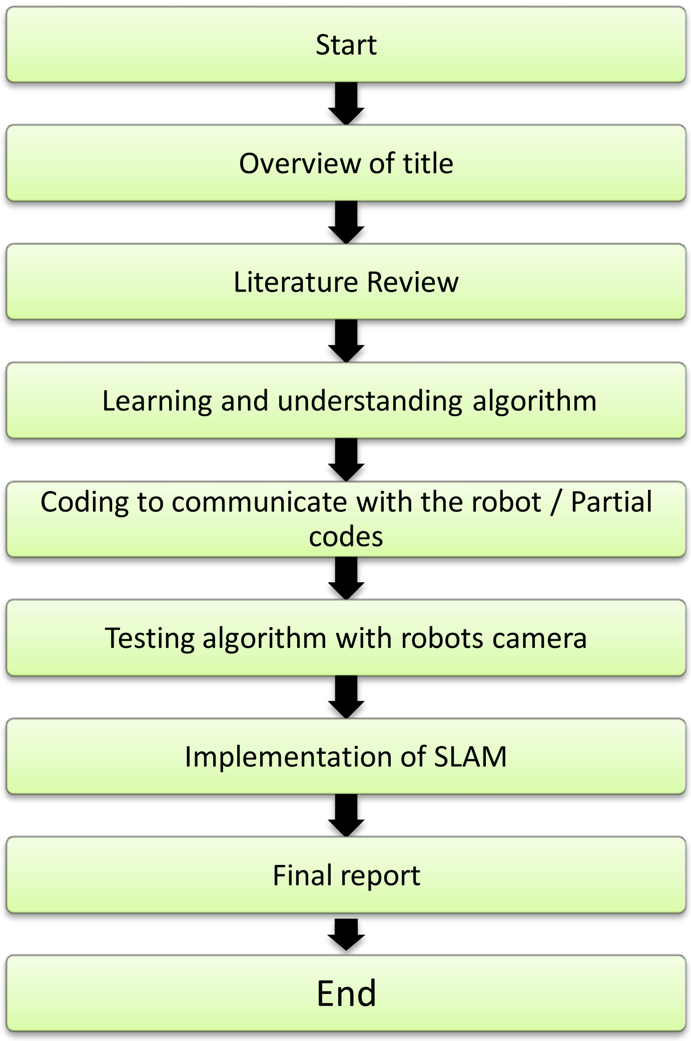

DATA CONTAMINATED WITH OUTLIERS AND A PRIORI INFORMATION
1st STEP: RANDOM HYPOTHESIS USING 1 POINT
2nd STEP: MODEL INSTANTIATION WITH CLEAR INLIERS
3RD STEP: MODEL RE-ESTIMATION WITH RESCUED INLIERS
9 votes
10 votes
5 votes
Most supported hypothesis
Outliers??
Outliers??
Outliers
Outliers
Inlier
Inlier

Figure 7: 1-Point RANSAC steps for a 2D line estimation example: Compared to the standard RANSAC, the algorithm assumes that an a priori probability distribution over the model parameters is known in advance. This prior knowledge allows computation of the random hypotheses using only 1 data point. The computational cost is lowered by the reduction of the number of hypotheses. [29]

CHAPTER 3
METHODOLOGY

3.1 Flow of activities

The flow of the methodology used for the final year project will be included in this section. Below, is a general overview of the activities conducted for FYP I and FYP II.


Start
Overview of title
Literature Review
Learning and understanding algorithm
Coding to communicate with the robot / Partial codes
Testing algorithm with robots camera
Implementation of SLAM
Final report
End

3.2 Project Activities

Research is first done to get a grasp of the rough idea behind the implementation of SLAM. One will be introduced to the different methods currently used for SLAM on autonomous vehicles or robots.

Literature review

Based on the research done from reading journals, online publications, lecture notes and tutorials, a literature review is compiled as a summary of the previous work done on SLAM and the methods used.After the literature review is done, we will then proceed with choosing the algorithm to be used such as either EKF or FastSLAM. Then, we will perform the necessary reading and learning to familiarize with the mathematical concepts, and how it works. The use of Monocular SLAM using EKF is chosen because of the less complexity in implementing since the robot motion is not taken into account for the odometry. A motion model is instead assumed for the movement of the camera, an assumption which will prove costly for our implementation.

Learning and understanding the algorithm

The chosen algorithm 1-Point RANSAC Inverse Depth EKF Monocular SLAM is read and based on the papers and the codes, I try to understand the algorithm used. This part, I would say that as of now, I have an overview of the algorithm but not really too detailed into the mathematic behind it

Coding to communicate with the robot /Partial codes

The programming on the robot initially started with the use of choreographe to move the robot, and using NAO simulator to run and test the functions in the room. However, I soon realized that Choreographe (the software GUI for NAO) is practically not that useful for the work intended. With the switch to the use of the python scripts, it is easier to code and control the robot to obtain image sequence, videos and to move the robot. It is also in this part where I tested with partial parts of the necessary stuffs needed in the algorithm. A few camera calibration methods are tested out to verify the accuracy of those methods. The calibrated parameters are important as they determine the accuracy of the SLAM later on.

Testing algorithm with robots camera

With the calibrated camera parameters, the algorithm is tested on video sequences recorded by using the camera on the robot. The corner detection algorithm is benchmarked using a same image to compare the performance of these algorithms. Using mex files, the FAST corner detector used in the earlier MATLAB codes performs significantly faster. Even in this stage, a few methods have been attempted to speed up the coding, and one was to convert the whole MATLAB coding platform to use C++ which was never completed due to insufficient time and some programming shortcomings.

Implementation of SLAM

For the implementation, the python code is no longer used, and instead, the MATLAB code is used from the libraries provided by Aldebaran Robotics to communicate with the robot. Images are captured from the robot while it is moving and the SLAM accuracy is evaluated.

Final Report

In this section, it comprises of the final stages whereby the report is prepared. Before the report is done, a presentation was also done for the Electrex and the SEDEX exhibition in UTP.

Tools/Hardware/Softwareused:

- NAO V3 robot with camera (OV7670 VGA)
- Firmware 1.12.5
- Computer running on Windows (execution of main programs) and Ubuntu (accessing files from memory stick)
- Python, Visual Studio C++, and MATLAB.

3.3 Gantt Chart

3.3.1 FYP I

No	Detail/Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Selection of Project Topic: Visual SLAM on NAO	▨	▨ ●												
2	Preliminary Research Work: Research on literatures related to the topic		▨	▨	▨	▨									
3	Submission of Extended Proposal						▨ ●								
4	Kinematic Modeling of Robot						▨	▨ ●							
5	Familiarize and coding on NAO						▨	▨	▨	▨	▨	▨	▨	▨	▨
6	Preparation for Presentation							▨	▨						
7	Proposal Defense									▨ ●					
8	Draft Report											▨	▨	●	
9	Interim Report													▨	▨ ●

● **- Key Milestone**

3.3.2 FYP II

No	Detail/Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Algorithm read and understand			▨	▨											
2	Testing and profiling algorithm					▨	▨	▨								
2	Progress Report								●							
3	Implementation of SLAM								▨	▨	▨	▨	▨	▨	▨	
4	Final results / Findings											▨	▨	▨	●	
5	Pre-Edx											●				
6	SEDEX												▨			
7	Final Report													●		
8	Viva														▨	●

● **- Key Milestone**

CHAPTER 4
RESULTS AND DISCUSSIONS

4.1 Calibration

The parameters of the camera are needed to initialize the camera for the algorithm. Discrepancies were noticed when the calibration method is repeated using the OpenCV libraries with python using the webcam on my laptop.
With the open CV function, FindChessboardCorners, the script automatically detects the chessboard image and search for the corners. The script automatically calculates the calibration parameters when a set amount of frames have been captured and redisplay two windows, one with the actual view and the other after undistorting.

Basically the intrinsics and distortion obtained through this method fluctuates and changes by a lot in the subsequent tries. Therefore, I decided to try out a few other methods to validate the results.
Using the MATLAB Camera Calibration Tool Box done by Jean-Yves Bouguet.

The first method was by using the MATLAB Calibration Tool Box which is considered a classic in camera calibration.

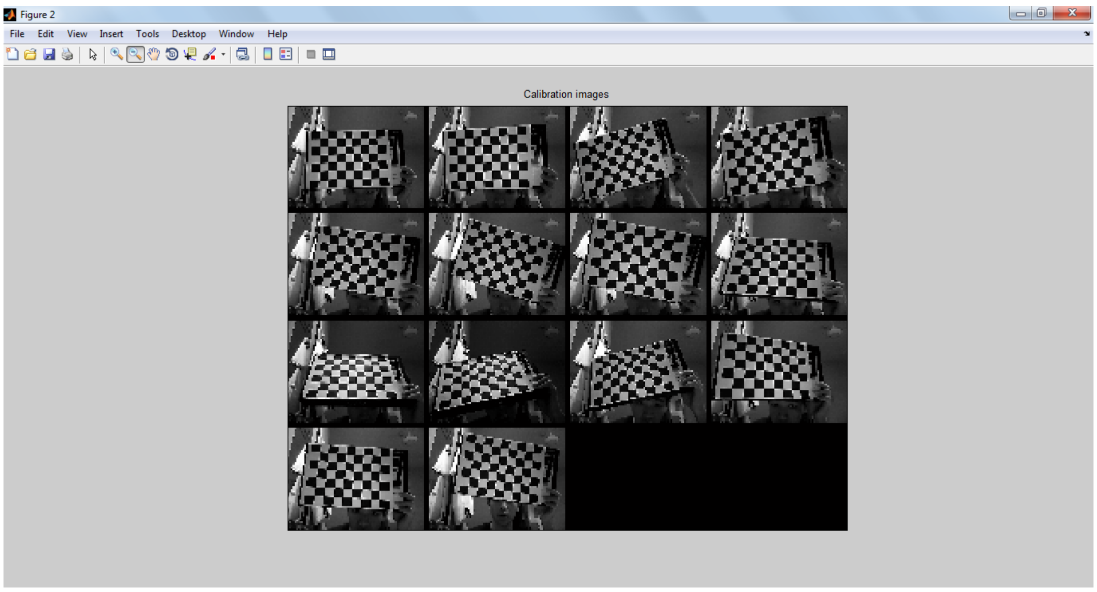


Figure 8: Mosaic of 14 pictures taken using imtool

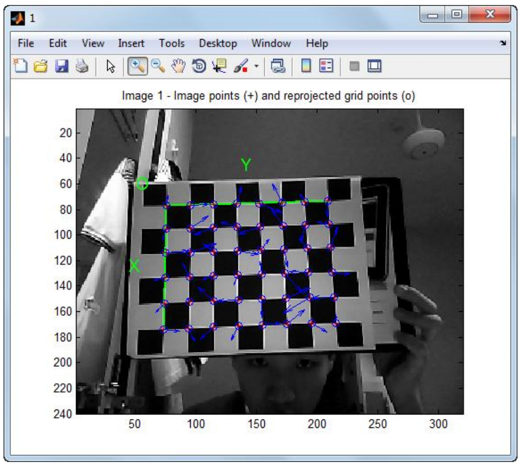

File Edit View Insert Tools Desktop Window Help
Image 1 - Image points (+) and reprojected grid points (o)
20
40
60
80
100
120
140
160
180
200
220
240
50
100
150
200
250
300
Y
X
O

Figure 9: Reprojection on image

Calibration results after optimization (with uncertainties):

Focal Length:	fc = [286.95150 287.46227] ± [3.49466 3.43038]
Principal point:	cc = [155.31871 115.93520] ± [1.91445 2.23545]
Skew:	alpha_c = [0.00000] ± [0.00000]
Distortion:	kc = [0.33039 -1.08948 -0.00347 -0.00541 0.00000] ± [0.03168 0.19624 0.00288 0.00353 0.00000]
Pixel error:	err = [0.15615 0.12837]

GML C++ Camera Calibration Toolbox[29]

The GML Calibration toolbox utilizes multiple calibration patterns and for this try, two templates are used, 6x5 and 6x9 is used with the squares at 30mm apart.

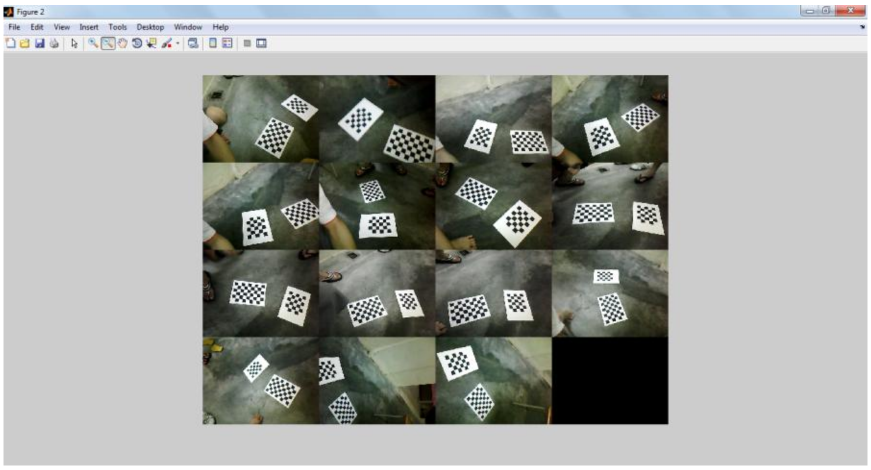


Figure 10: Two chessboard templates used

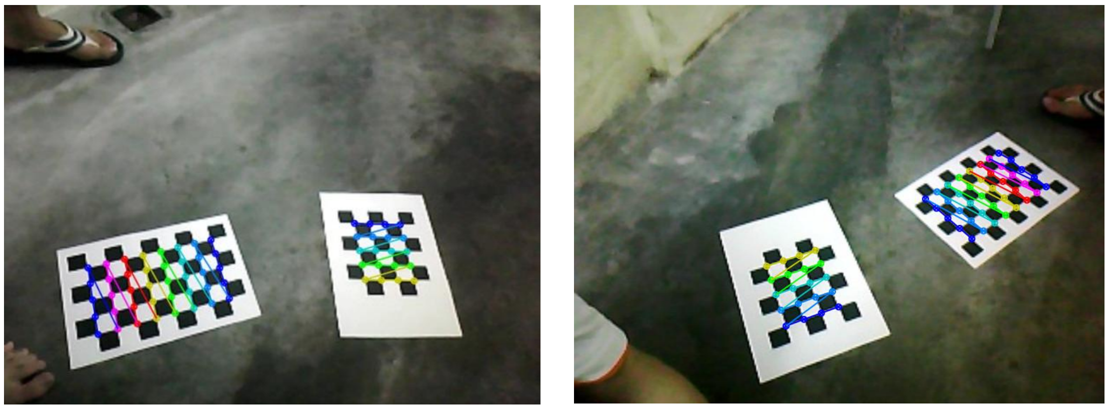


Figure 11: Detection of corner points

Focal Length: fc = [286.697 286.336] ± [5.682 6.080]
Principal point: cc = [164.604 100.874] ± [4.200 6.767]
Distortion: [0.196733 -0.410724 -0.014089 0.009308] ± [0.074513 0.469068 0.008955 0.00571]

Pixel error: err = [0.34 0.63]

The two results are slightly different but in comparison with the inaccuracy from the python script, the two methods, the Matlab Toolbox and the GML toolbox are feasible methods.

The inaccurate result from the python open cv implementation might me due to the fact that the board moves when the capture is taking place, thus causing blurry motions to be captured.

Finally the MATLAB Calibration ToolBox is the chosen method to calibrate the camera on the robot NAO.

Obtained **OV7670 camera** calibration parameters from Bouguet's toolbox:

Calibration results after optimization (with uncertainties):

Focal Length: fc = [382.05271 380.50844] ± [2.18708 2.15287]

Principal point: cc = [159.59442 110.47495] ± [2.28538 2.10549]

Skew: alpha_c = [0.00000] ± [0.00000]
Distortion: kc = [0.27363 -0.99625 0.00047 -0.00523 0.00000] ± [0.02088 0.11953 0.00276 0.00280 0.00000]

Pixel error: err = [0.19699 0.18773]

4.2 Feature detector benchmark

In a vision based SLAM system, feature detectors are needed to search for easily traceable points in the images captured. In comparison to edges and lines, corner points are small and so require less memory and computation to maintain. A few methods were taken to the test with the same image. The importance of this part is learning to implement and to use the codes and also to test the speed of the algorithm.

The test image is grayscale with a dimension of 768 x 288.

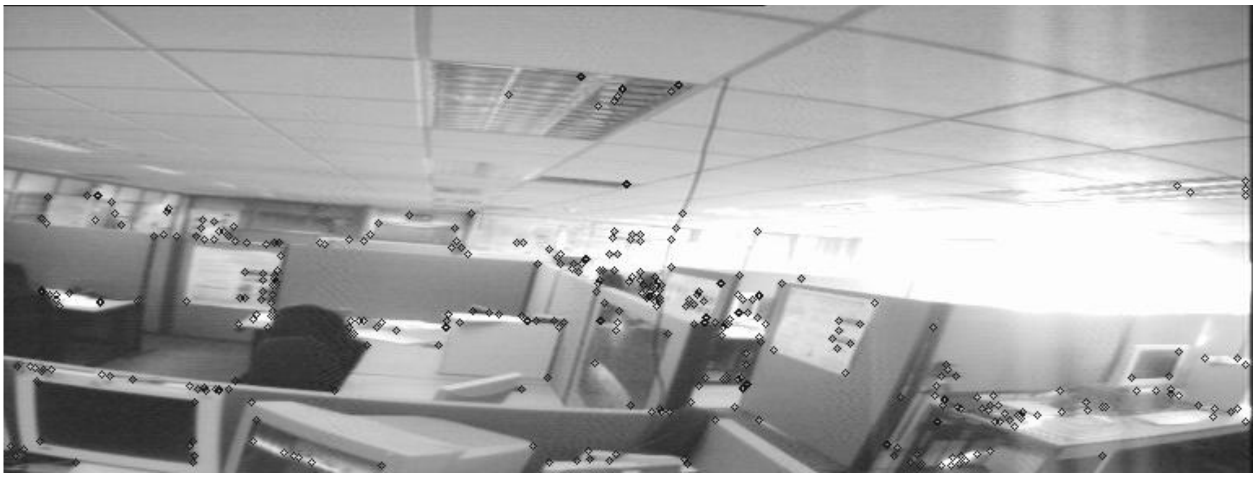


Figure 12: Grayscale image of a lab[26]

FAST corner detector

Test 1: MATLAB (m-file)

Elapsed time: 4.90 seconds

Test 2: Python

Elapsed time: 7.36 seconds

Test 3: MATLAB (mex-file)

Elapsed time: 0.032seconds

SURF

Test 1: MATLAB (mex-file)

Elapsed time is 0.11 seconds.

Harris Corner Detector

Test1: Python/OpenCV

Elapsed time: 0.225 seconds

Test 2: MATLAB (mex-file)

Elapsed time is 0.119475 seconds.

Corner detector	m-file	Python	Mex file
SURF	-	-	0.11 s
Harris	-	0.225 s	0.12 s
FAST	4.90 s	7.36 s	0.032 s

Table 1: Comparison of detectors

4.3 MATLAB profiling

By first profiling the MATLAB code given online, I am able to identify the functions where the program requires the most time in. Here, the top 12 functions arranged based on total time is shown for 10 frames, 20 frames and 60 frames runtime respectively.

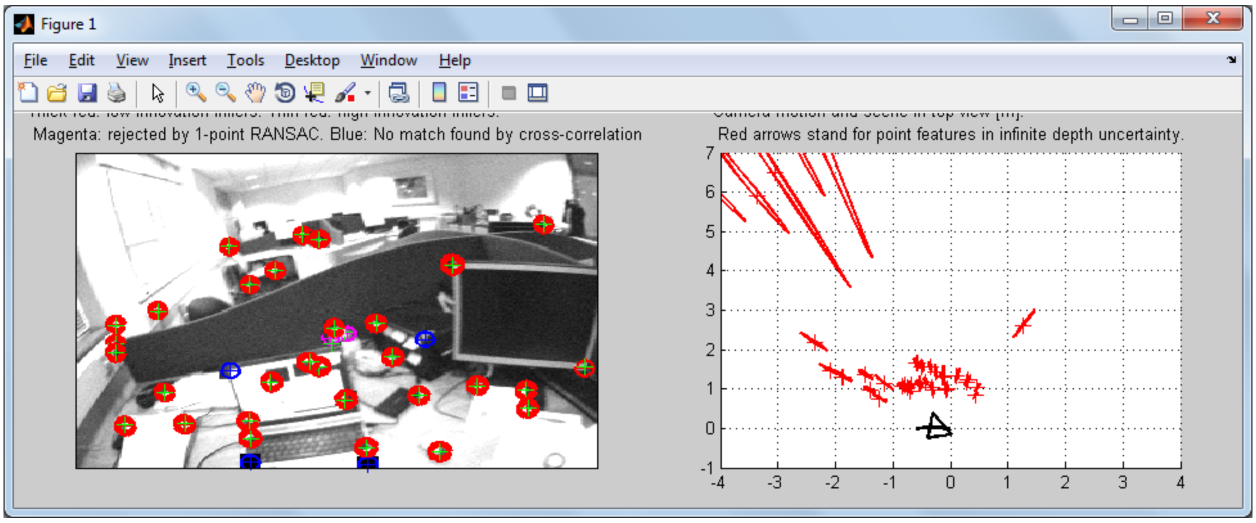
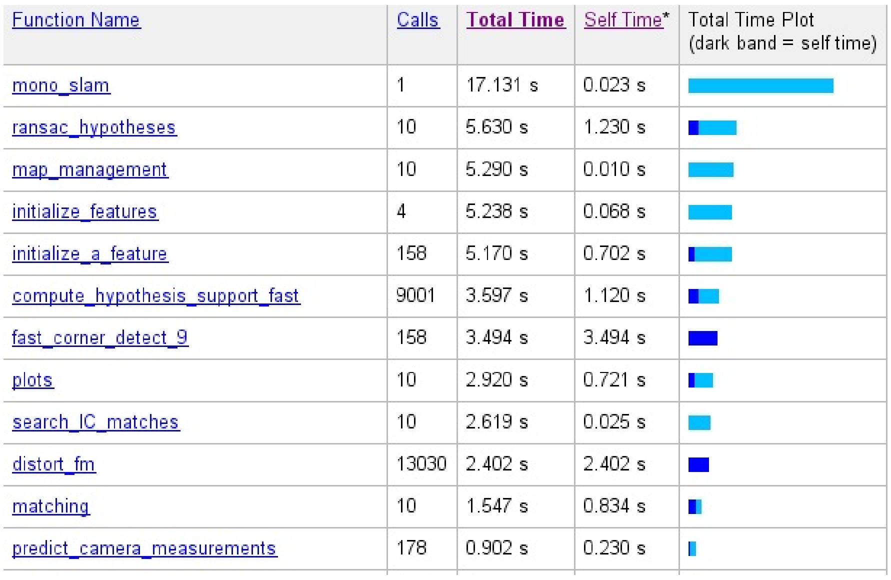


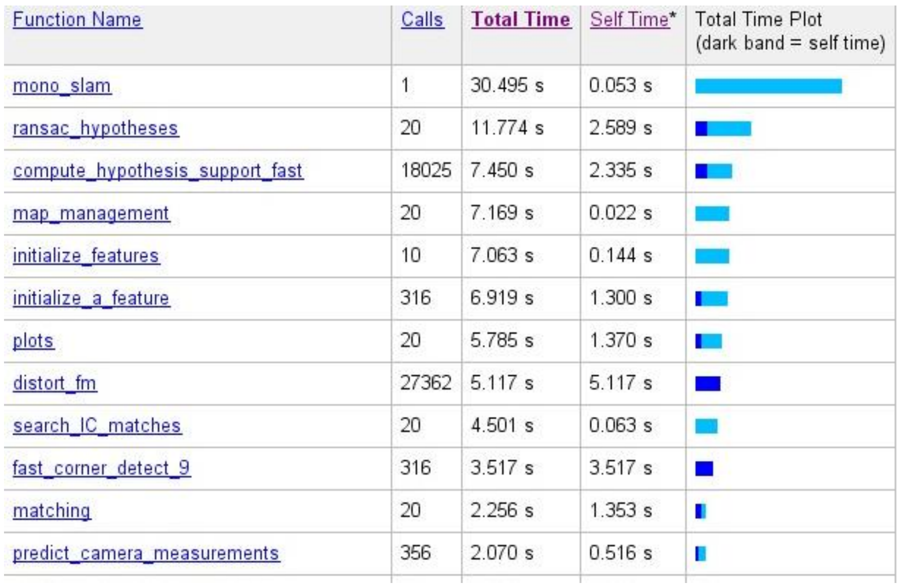
Figure 1
File Edit View Insert Tools Desktop Window Help
Magenta: rejected by 1-point RANSAC. Blue: No match found by cross-correlation
Red arrows stand for point features in infinite depth uncertainty.
7
6
5
4
3
2
1
0
-1
-4
-3
-2
-1
0
1
2
3
4

Figure 13: Example of SLAM on MATLAB



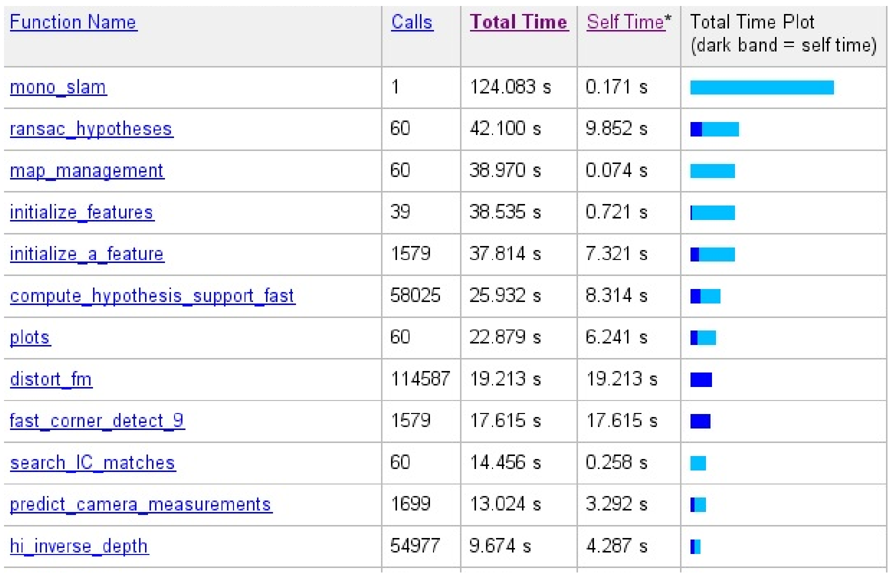
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mono_slam	1	17.131 s	0.023 s	
ransac_hypotheses	10	5.630 s	1.230 s	
map_management	10	5.290 s	0.010 s	
initialize_features	4	5.238 s	0.068 s	
initialize_a_feature	158	5.170 s	0.702 s	
compute_hypothesis_support_fast	9001	3.597 s	1.120 s	
fast_corner_detect_9	158	3.494 s	3.494 s	
plots	10	2.920 s	0.721 s	
search_IC_matches	10	2.619 s	0.025 s	
distort_fm	13030	2.402 s	2.402 s	
matching	10	1.547 s	0.834 s	
predict_camera_measurements	178	0.902 s	0.230 s	

Figure 14: 10 frames



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mono_slam	1	30.495 s	0.053 s	
ransac_hypotheses	20	11.774 s	2.589 s	
compute_hypothesis_support_fast	18025	7.450 s	2.335 s	
map_management	20	7.169 s	0.022 s	
initialize_features	10	7.063 s	0.144 s	
initialize_a_feature	316	6.919 s	1.300 s	
plots	20	5.785 s	1.370 s	
distort_fm	27362	5.117 s	5.117 s	
search_IC_matches	20	4.501 s	0.063 s	
fast_corner_detect_9	316	3.517 s	3.517 s	
matching	20	2.256 s	1.353 s	
predict_camera_measurements	356	2.070 s	0.516 s	

Figure 15: 20 frames



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
mono_slam	1	124.083 s	0.171 s	▬
ransac_hypotheses	60	42.100 s	9.852 s	▬
map_management	60	38.970 s	0.074 s	▬
initialize_features	39	38.535 s	0.721 s	▬
initialize_a_feature	1579	37.814 s	7.321 s	▬
compute_hypothesis_support_fast	58025	25.932 s	8.314 s	▬
plots	60	22.879 s	6.241 s	▬
distort_fm	114587	19.213 s	19.213 s	▬
fast_corner_detect_9	1579	17.615 s	17.615 s	▬
search_IC_matches	60	14.456 s	0.258 s	▬
predict_camera_measurements	1699	13.024 s	3.292 s	▬
hi_inverse_depth	54977	9.674 s	4.287 s	▬

Figure 16: 60 frames

The results above were before the implementation of the MEX-files for the corner detection. However, after the MEX implementation, the algorithm is still slow although the fast_corner_detect_9 time lag is now negligible. This is due to the computational cost spent on matrix multiplications mostly in the EKF update step. It is suggested that a better library is used for the multiplication process.

4.4 Implementation of SLAM

The SLAM algorithm is then tested in a controlled environment. The figure below shows the top view of the arrangement for the environment in the lab.

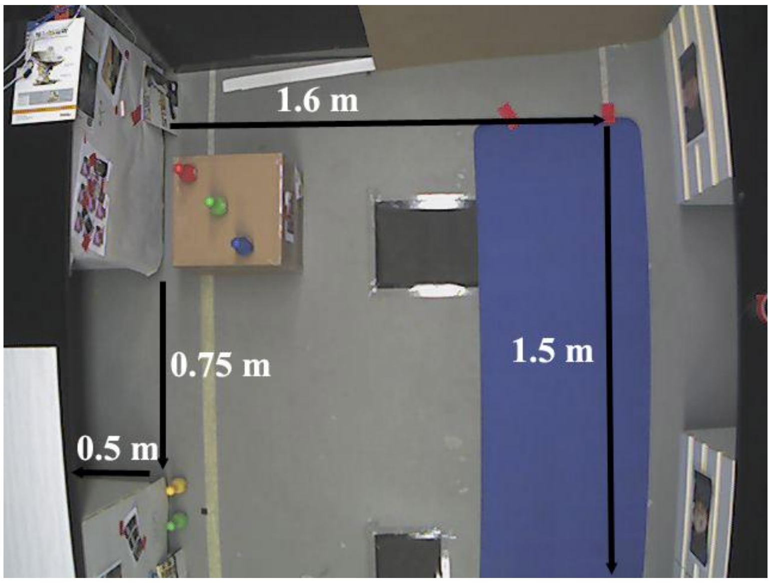


Figure 17: Top view of map

Having a controlled environment will provide us with a ground truth to compare our obtained results with.

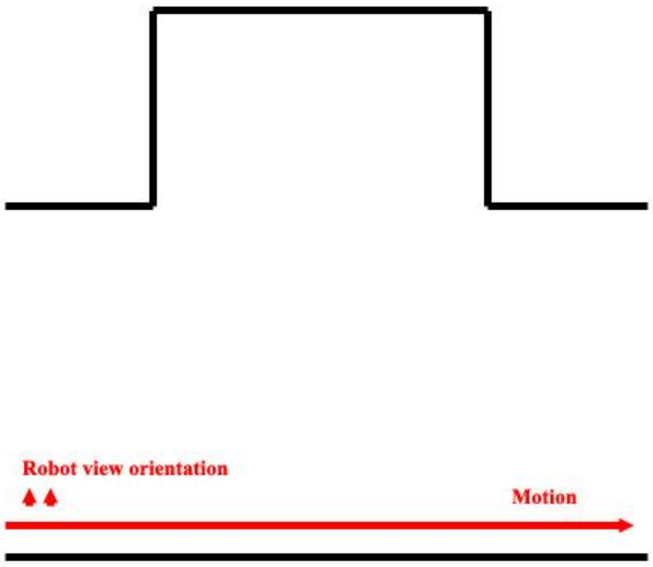


Figure 18: Estimated result if accurate

4.4.1 *Result obtained from a smooth displacement (offline)*

The robot is manually moved to the right for 1.5 meters. Throughout the movement, the head of the robot is facing towards the feature filled environment.

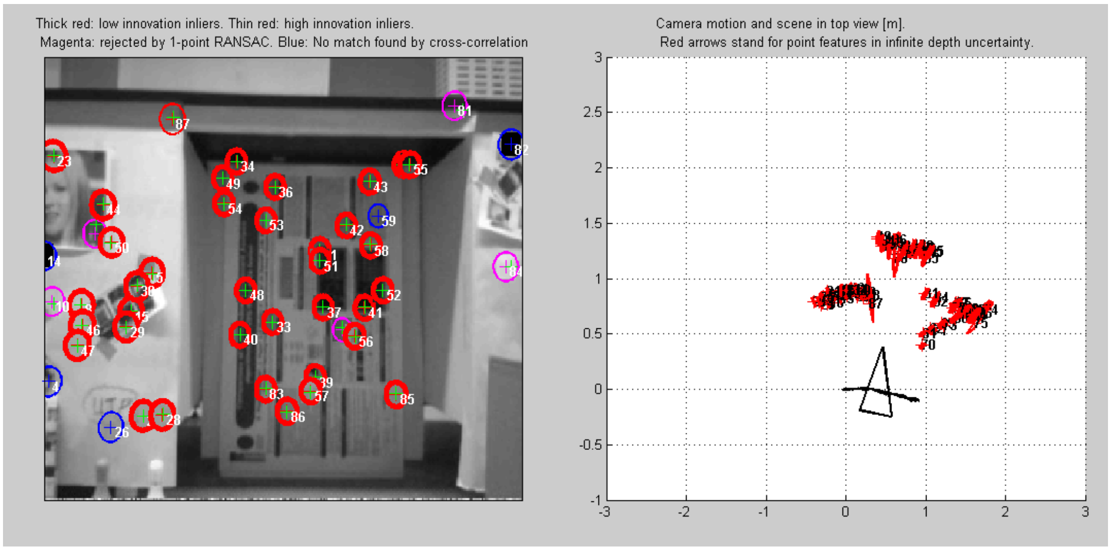

Thick red: low innovation inliers. Thin red: high innovation inliers.
Magenta: rejected by 1-point RANSAC. Blue: No match found by cross-correlation
Camera motion and scene in top view [m].
Red arrows stand for point features in infinite depth uncertainty.

Figure 19: Smooth displacement (offline)

As seen in the figure, the linear motion is accurately estimated. The landmarks are also correctly plotted since there is a section with landmarks which are situated further in and the objects that are placed closer to the robot's path (the box with three bowling pins) are correctly positioned. Due to the use of a monocular camera, the scene and the camera motion can only be recovered up to a scale factor and here, the appropriate scale factor is around 1.5x since the 1.5 meters displacement have been taken as a 1 meter displacement. This applies also to the distance to the landmarks.

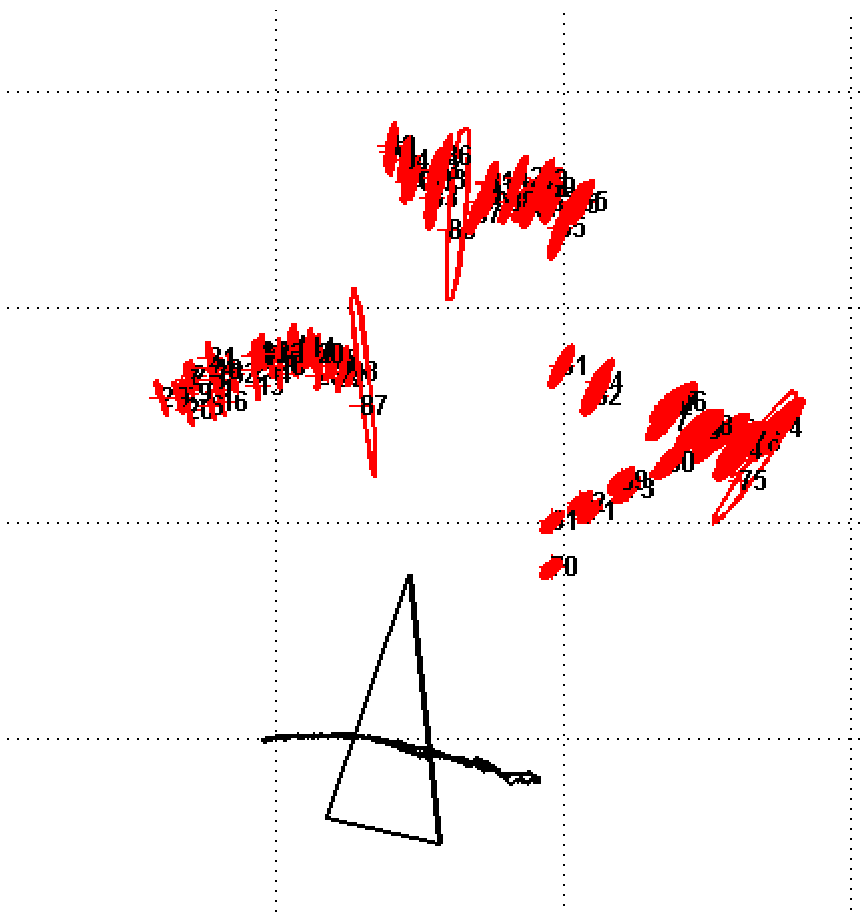
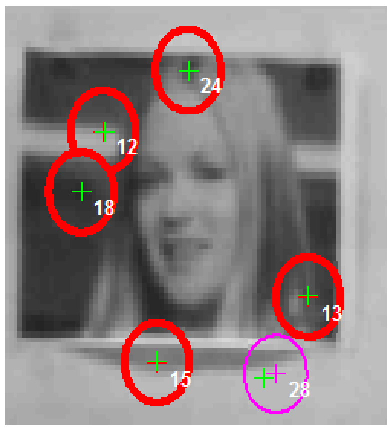
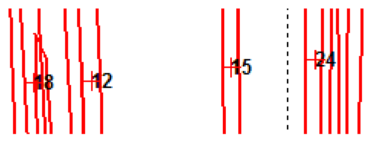
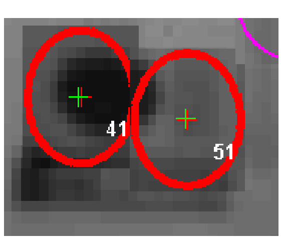
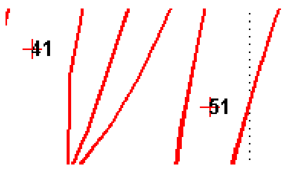


Figure 20: Zoomed in









4.4.2 Results obtained from a displacement from robot walk (real time)

Real time capability is obtained not through the speed of the computation but by setting the robot to take frames on each footstep. This results in large frame skips and cause difficulties in matching the points in subsequent frames. The result from this is unsatisfactory and since it is impossible to reduce the speed of NAO's footstep any further, it is not possible for the results to be improved unless the computation speed is increased to allow in between frames to be captured.

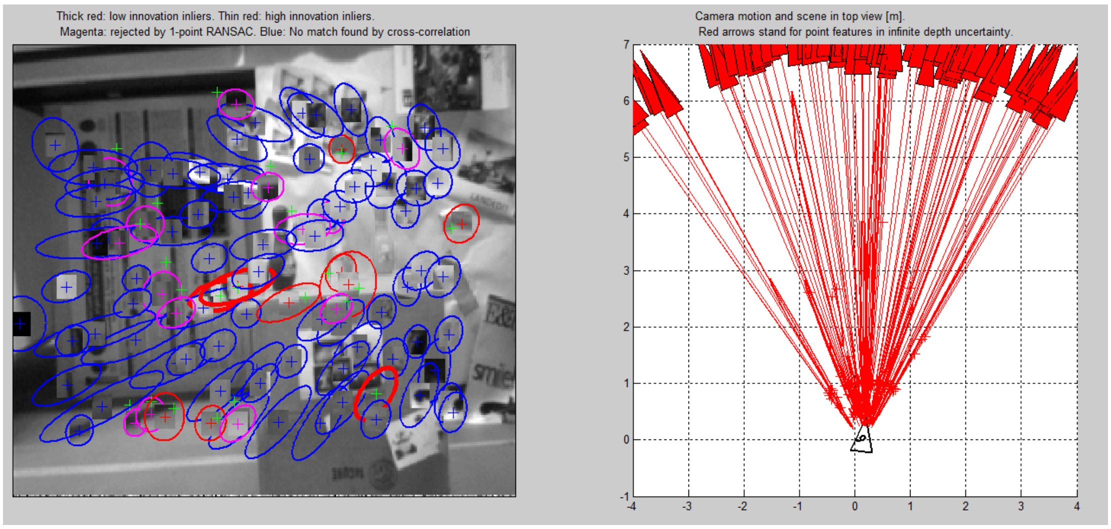

Thick red: low innovation inliers. Thin red: high innovation inliers.
Magenta: rejected by 1-point RANSAC. Blue: No match found by cross-correlation
Camera motion and scene in top view [m].
Red arrows stand for point features in infinite depth uncertainty.
7
6
5
4
3
2
1
0
-1
-4
-3
-2
-1
0
1
2
3
4

Figure 21: Realtime through frame skips

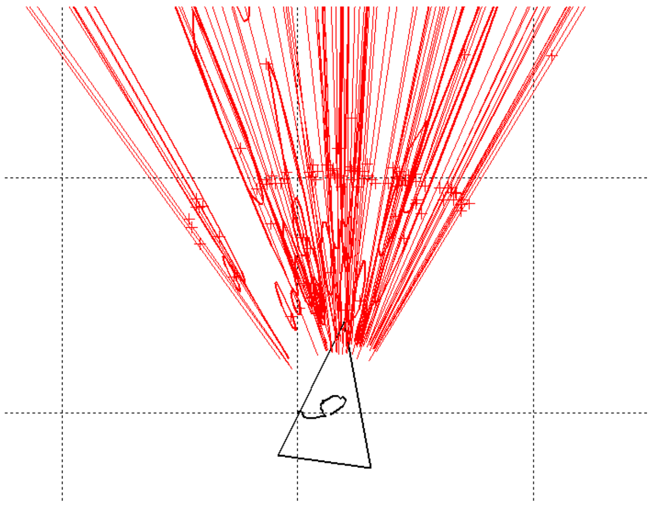


Figure 22: Inaccurate Mapping and Localization

The algorithm has a good approximation on the direction of the landmarks but the scaling factor and the motion estimation is really bad. Structure from motion might be a better solution for a condition with large frame skips.

4.4.3 Results obtained from a displacement from robot walk (offline)

In this third test, the robot is programmed to walk straight, but during the walking sequence, with NAO's terrible motion control, the motion was actually not that straight. The video sequence recorded during the walk is tested to simulate the resulting SLAM if the algorithm can be implemented in a real time 30Hz environment.

From a same video sequence of 1300 frames:

Without discarding the blur images:

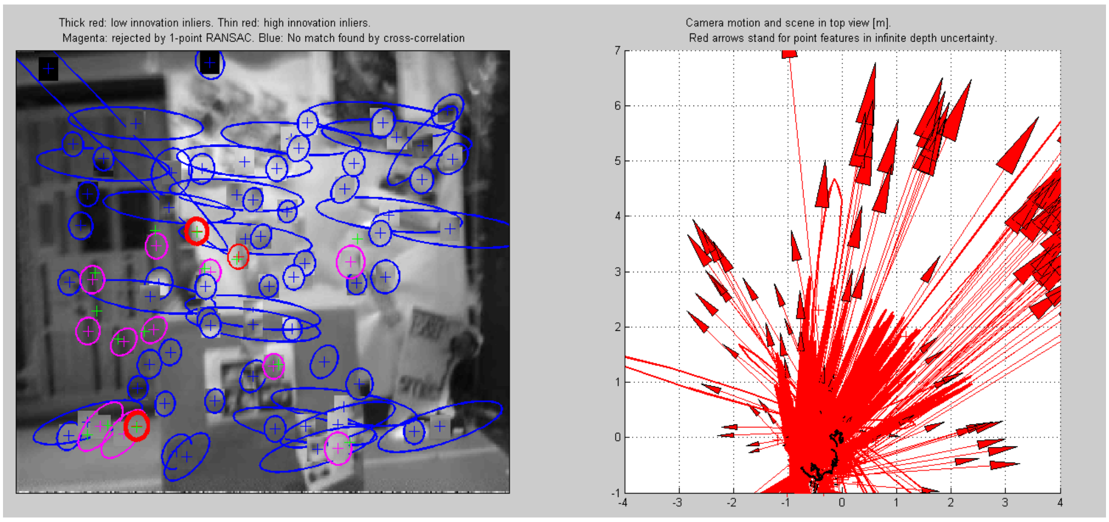

Thick red: low innovation inliers. Thin red: high innovation inliers.
Magenta: rejected by 1-point RANSAC. Blue: No match found by cross-correlation
Camera motion and scene in top view [m].
Red arrows stand for point features in infinite depth uncertainty.
7
6
5
4
3
2
1
0
-1
-4
-3
-2
-1
0
1
2
3
4

Figure 23: With blurry frames accounted

The results show that the robot has moved all over the area and the landmarks observed are scattered around the robot when in fact the landmarks should only be in front of NAO.

The blue ellipse represents the detected features/landmarks which are not able to be matched with any previous detection. As a result, many landmarks and initialized and more points are added to the map although they might be the same observed points from previous frames.

No valuable information could be interpreted from this trial when the blurred images are not removed.

With the blur images discarded:

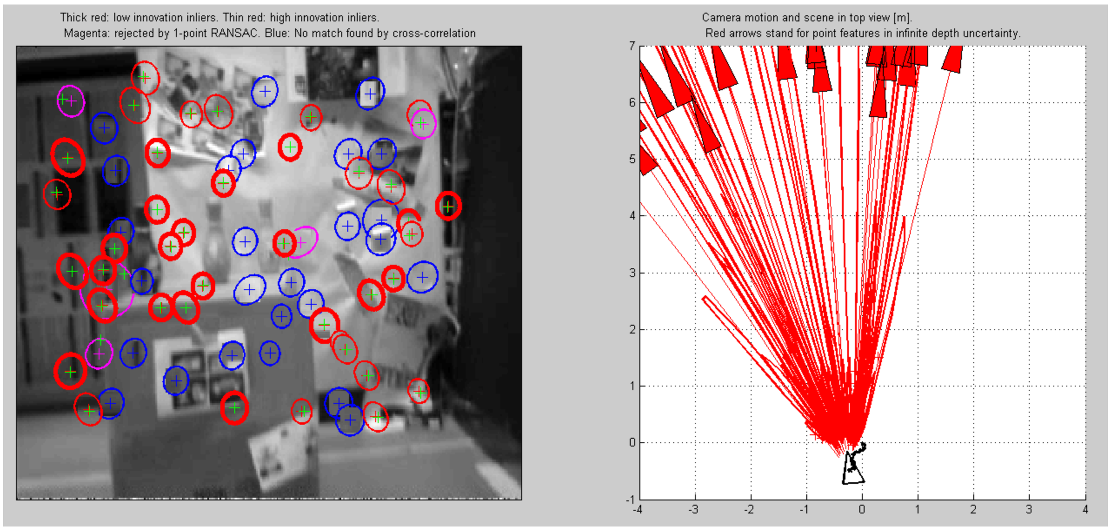

Thick red: low innovation inliers. Thin red: high innovation inliers.
Magenta: rejected by 1-point RANSAC. Blue: No match found by cross-correlation
Camera motion and scene in top view [m].
Red arrows stand for point features in infinite depth uncertainty.
7
6
5
4
3
2
1
0
-1
-4
-3
-2
-1
0
1
2
3
4

Figure 24: Blur frames above a certain threshold is removed

The blur metric is utilized to set a certain threshold for the frames so that blurry images are discarded. The threshold used here is 0.45.

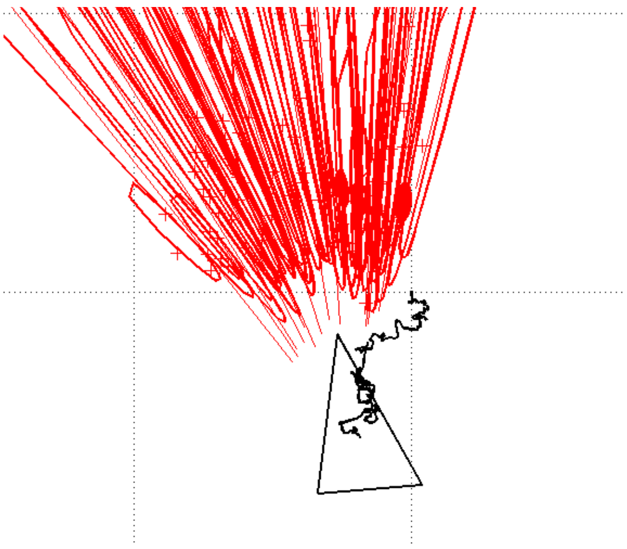


Figure 25: Blur frames removed

The results are better when the blurred frames are removed but still insufficiently accurate for the landmarks and the motion estimation.

Summary of results:

Situation	Verdict
Smooth displacement of the robot through a horizontal motion.	Accurate representation of motion and landmarks to a certain scale
Real time online processing with frame skips (robot walk).	Inaccurate motion and insufficient landmarks but with a little resemblance to the motion and map.
Offline processing of video recorded for all frames (robot walk)	Inaccurate motion and landmarks scattered. Total loss of any direction or localization for the robot.
Offline processing of video recorded with blurred frames discarded (robot walk)	Better than processing of all frames but the motion is inaccurately represented in the opposite direction. The landmarks are in the right direction but inaccurately placed in terms of distance.

Table 2: Summary of results

Note: The processing time remains slow at an average rate of around 0.5 frames per second.

CHAPTER 5
CONCLUSION & RECOMMENDATIONS

From the early findings, it is proven that the best and fastest feature detector to be used is the FAST corner detector as long as it is in the form of a compiled MEX file and not in a python script or a MATLAB file. The use of different calibration methods also proves that the calibration method in Bouguet's toolbox is accurate and reliable. Although, not mentioned, the calibration parameters used for the algorithm in MATLAB needs to be converted to the Tsai model.

The algorithm itself works great for handheld cameras and motions which are smooth as shown but using the biped movement of the robot, the results were unfortunately not favorable.

The implementation of the algorithm on a humanoid robot proves to be more complicated than expected. The sudden jerks and unreliable motion from the footsteps of the robot creates a sequence of frame which are blurry if the motion is too quick and also sudden changes in the direction of the movement. The head (camera) attached on the robot tend to sway drastically back and forth while it is walking. With huge frame skips, the SLAM estimation will be badly affected for the EKF monocular SLAM using 1-P RANSAC. It is the wrong choice of algorithm to be used but due to the shortage of time, this is the only algorithm which was tried. For the case of frame separation EKF relies on first order linearization that will have big errors if the predicted values are far from the actual matched ones. In this particular case where it is not possible to grab close, sequential frames, perhaps Structure from Motion will be a better technique to be used.

Nevertheless, if we take into assumption that real time processing at 30fps is possible once the coding is ported into the more efficient C++ language; further tests are conducted on the videos recorded from the camera on NAO which is set to record at

30 fps. Testing with the algorithm and the recorded video from the robot, the results were at first extremely bad due to the blurry frames captured. These frames cause the data association between frames to be inaccurate which then results in greater errors.

Motion deblurring methods have been recommended to solve this issue and will be look into. However, before implementing motion deblur algorithm, one will need to know which frames are sharp and which are blur. To do so, the perceptual blur metric [32] is used to detect blur frames and for now, the frames will be discarded instead of corrected using filters such as Wiener or Inverse filtering.

The results after the blurry frames are discarded were better than when all frames are used in the process, but still inaccurate if compared with the results obtained from the smooth camera motion on the robot. The algorithm will be better if used with other robots such as the wheeled wifibot as the movements of the robot will be smooth and predictable.

<u>Recommendations</u>

- The algorithm used is suitable only for smooth robot motion so the use of wheeled robots or vehicle is highly recommended or perhaps the motion model will need to be changed to a more complicated one based on the movement and joints of the robot.
- The MATLAB codes will need optimization to make it less computational intensive or by having itrecoded in a faster language.
- Motion deblur techniques could be implemented to recover the bad frames instead of simply discarding the frames resulting in some gaps in the image sequence.

REFERENCES

[1] T. González Sánchez, "Artificial Vision in the Nao Humanoid Robot," Universitat Politècnica de Catalunya, 2009.

[2] "Bachelor Artificial Intelligence – Probabilistic Robotics," [Online]. Available: http://staff.science.uva.nl/~arnoud/education/ProbabilisticRobotics. [Accessed 18 March 2012].

[3] J. J. Leonard and H. F. Durrant-Whyte, "Mobile Robot Localization by Tracking Geometric," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. I, NO. 3,* pp. 376-382, June 1991.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *Robotics & Automation Magazine, IEEE* , pp. 99 - 110, June 2006.

[5] L. Kleeman, "Advanced sonar and odometry error modeling for simultaneous localisation and map building," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 2003.

[6] A. Diosi and L. Kleeman, "Advanced sonar and laser range finder fusion for simultaneous localization and mapping," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* , 2004.

[7] A. Davison and D. Murray, "Simultaneous localization and map-building using active vision," *Pattern Analysis and Machine Intelligence, IEEE Transactions,* vol. 24, no. 7, pp. 865 - 880 , 2002.

[8] A. Davison and D. Murray, "Simultaneous localization and map-building using active vision," *Pattern Analysis and Machine Intelligence, IEEE Transactions,* vol. 24, no. 7, pp. 865-880, 2002.

[9] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *Robotics and Automation, IEEE Transactions,* vol. 17, no. 3, pp. 229-241, 2001.

[10] P. Newman and J. Leonard, "Pure range-only sub-sea SLAM," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference* , 2003.

[11] O. Stasse, A. J. Davison, R. Sellaouti and K. Yokoi, "Real-time 3D SLAM for Humanoid Robot considering Pattern Generator Information," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference*, 2006.

[12] S. Riisgaard and M. R. Blas, "MIT OpenCourseWare," 2005. [Online]. Available: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/. [Accessed 23 February 2012].

[13] T. Bailey, "SLAM Summer School 2009," January 2009. [Online]. Available:

http://www.acfr.usyd.edu.au/education/summerschool.shtml. [Accessed 3 March 2012].

[14] J. Denavit and R. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices.," *Trans. of the ASME. Journal of Applied Mechanics,* vol. 22, pp. 215-221, 1955.

[15] C.Harris and M.Stephens, "A combined corner and edge detector," in *Proc. of Fourth Alvey Vision Conference*, Manchester, United Kingdom, 1988.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* vol. 60, no. 2, pp. 91-110, 2004.

[17] H. Temeltas and D. Kayak, "SLAM for robot navigation," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 23, no. 12, pp. 16 - 19, 2008.

[18] J. Klippenstein and H. Zhang, "Performance evaluation of visual SLAM using several feature extractors," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference*, 2009.

[19] T. Bailey, J. Nieto, J. Guivant, M. Stevens and E. Nebot, "Consistency of the EKF-SLAM Algorithm," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference*, 2006.

[20] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference,* 2003.

[21] M. Montemerlo, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association," Robotics Institute, Carnegie Mellon University, 2003.

[22] G. Grisetti, R. Ku□mmerle, C. Stachniss and W. Burgard, "A Tutorial on Graph-Based SLAM," *Intelligent Transportation Systems Magazine, IEEE* , vol. 2, no. 4, pp. 31 - 43 , 2010.

[23] "Monocular SLAM," [Online]. Available: http://vision.ia.ac.cn/Students/gzp/monocularslam.html. [Accessed 14 August 2012].

[24] A. Davison, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 29, no. 6, pp. 1052 - 1067 , 2007.

[25] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[26] B. Williams, "Simultaneous Localisation and Mapping Using a Single Camera," University of Oxford, 2009.

[27] a. A. J. D. J. Civera, "Unified inverse depth parametrization for monocular

SLAM," in *Proc. Robotics Science and Systems*, 2006.

[28] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking.," in *IEEE International Conference on Computer Vision*, 2005.

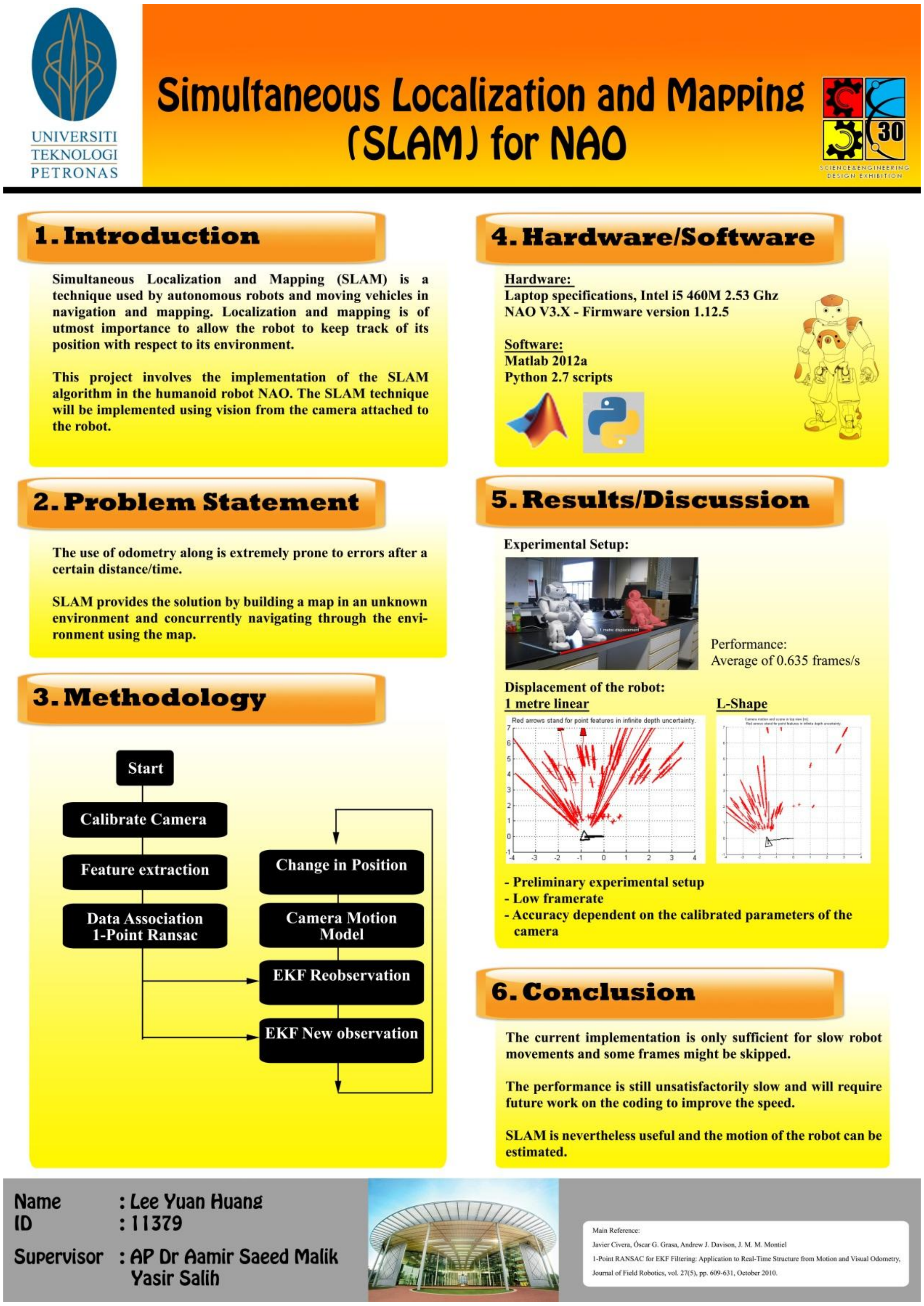
[29] J. Civera, O. Grasa, A. Davison and J. Montiel, "1-point RANSAC for EKF-based Structure from Motion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009.* , 2009.

[30] J. Neira and J. Tardos, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation,* vol. 17, no. 6, pp. 890 - 897 , 2001.

[31] A. N. A. V.Vezhnevets, *GML C++ Camera Calibration Toolbox,* 2011.

[32] F. Crete, T. Dolmiere, P. Ladret and M. Nicolas, "The Blur Effect: Perception and Estimation with a New No-Reference," in *SPIE Electronic Imaging Symposium Conf Human Vision and Electronic Imaging*, San Jose, 2007.

APPENDIX A
Presentation Poster SEDEX 30



UNIVERSITI TEKNOLOGI PETRONAS

Simultaneous Localization and Mapping (SLAM) for NAO

SCIENCE&ENGINEERING DESIGN EXHIBITION 30

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a technique used by autonomous robots and moving vehicles in navigation and mapping. Localization and mapping is of utmost importance to allow the robot to keep track of its position with respect to its environment.

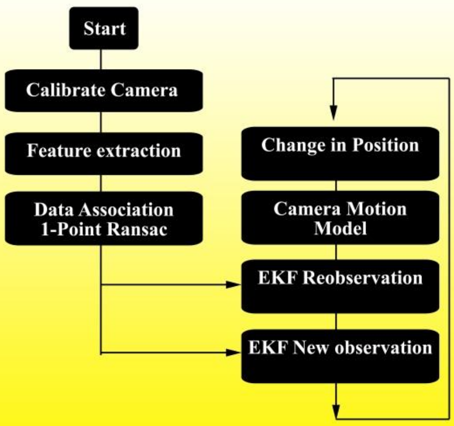
This project involves the implementation of the SLAM algorithm in the humanoid robot NAO. The SLAM technique will be implemented using vision from the camera attached to the robot.

2. Problem Statement

The use of odometry along is extremely prone to errors after a certain distance/time.

SLAM provides the solution by building a map in an unknown environment and concurrently navigating through the environment using the map.

3. Methodology



4. Hardware/Software

<u>Hardware:</u>
Laptop specifications, Intel i5 460M 2.53 Ghz
NAO V3.X - Firmware version 1.12.5

<u>Software:</u>
Matlab 2012a
Python 2.7 scripts

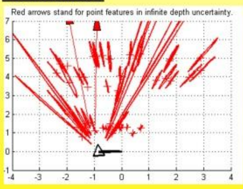
5. Results/Discussion

Experimental Setup:

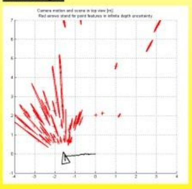
Performance:
Average of 0.635 frames/s

Displacement of the robot:

<u>1 metre linear</u>



<u>L-Shape</u>



- **Preliminary experimental setup**
- **Low framerate**
- **Accuracy dependent on the calibrated parameters of the camera**

6. Conclusion

The current implementation is only sufficient for slow robot movements and some frames might be skipped.

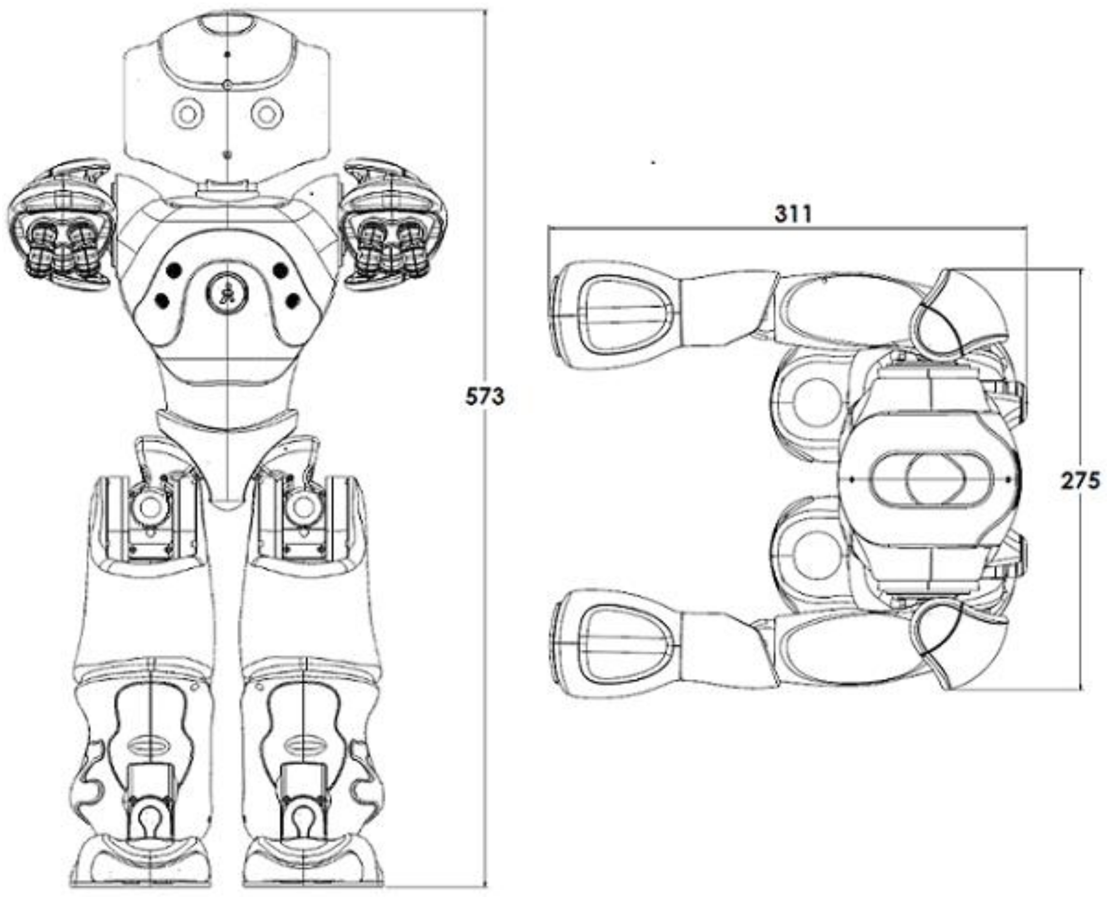
The performance is still unsatisfactorily slow and will require future work on the coding to improve the speed.

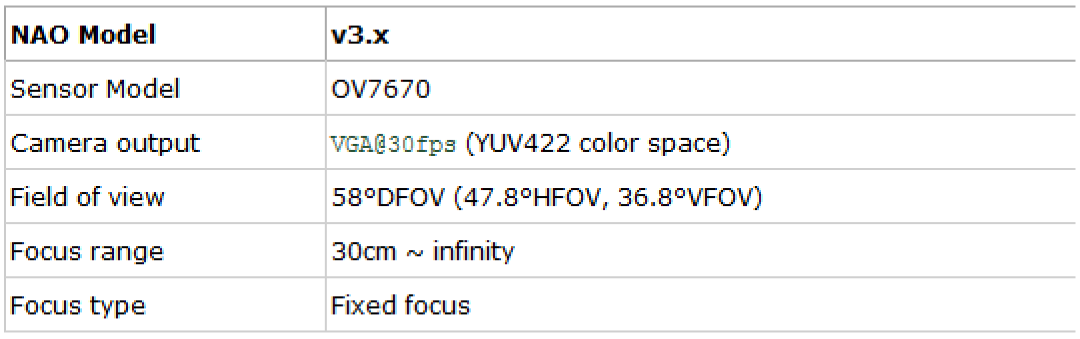
SLAM is nevertheless useful and the motion of the robot can be estimated.

Name	: Lee Yuan Huang
ID	: 11379
Supervisor	: AP Dr Aamir Saeed Malik Yasir Salih

Main Reference:
Javier Civera, Óscar G. Grasa, Andrew J. Davison, J. M. M. Montiel
1-Point RANSAC for EKF Filtering: Application to Real-Time Structure from Motion and Visual Odometry, Journal of Field Robotics, vol. 27(5), pp. 609-631, October 2010.

APPENDIX B
Related Documentations& Code Snippets

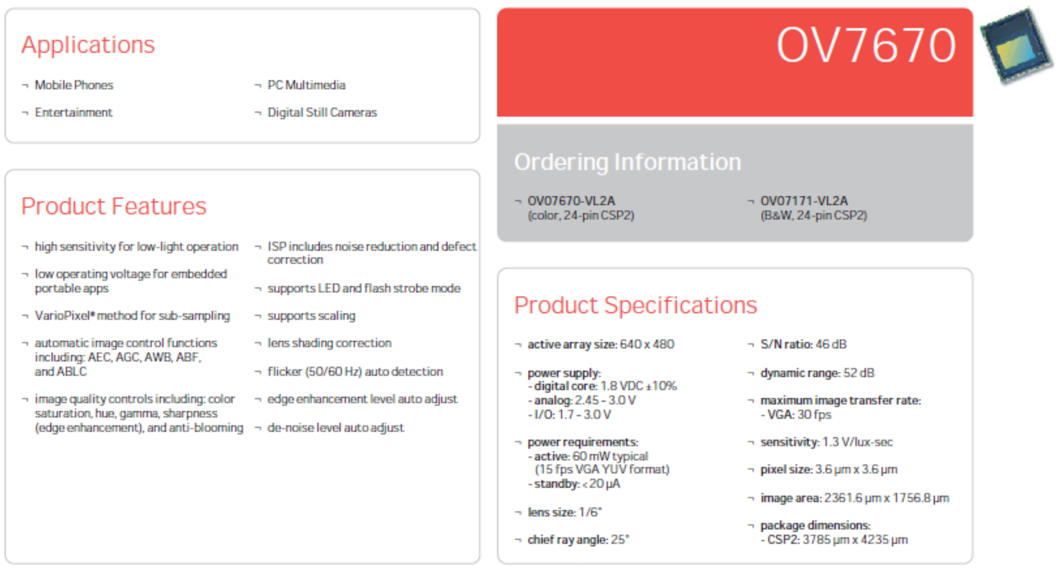




NAO Model	v3.x
Sensor Model	OV7670
Camera output	VGA@30fps (YUV422 color space)
Field of view	58°DFOV (47.8°HFOV, 36.8°VFOV)
Focus range	30cm ~ infinity
Focus type	Fixed focus

NAO V. 3.x

- x86 AMD GEODE 500MHz CPU
- 256 MB SDRAM / 2 GB flash memory



OV7670

Applications

- Mobile Phones
- Entertainment
- PC Multimedia
- Digital Still Cameras

Product Features

- high sensitivity for low-light operation
- low operating voltage for embedded portable apps
- VarioPixel® method for sub-sampling
- automatic image control functions including: AEC, AGC, AWB, ABF, and ABLC
- image quality controls including: color saturation, hue, gamma, sharpness (edge enhancement), and anti-blooming
- ISP includes noise reduction and defect correction
- supports LED and flash strobe mode
- supports scaling
- lens shading correction
- flicker (50/60 Hz) auto detection
- edge enhancement level auto adjust
- de-noise level auto adjust

Ordering Information

- **OV07670-VL2A**
  (color, 24-pin CSP2)
- **OV07171-VL2A**
  (B&W, 24-pin CSP2)

Product Specifications

- **active array size:** 640 x 480
- **power supply:**
  - **digital core:** 1.8 VDC ±10%
  - **analog:** 2.45 - 3.0 V
  - **I/O:** 1.7 - 3.0 V
- **power requirements:**
  - **active:** 60 mW typical
    (15 fps VGA YUV format)
  - **standby:** < 20 µA
- **lens size:** 1/6"
- **chief ray angle:** 25°
- **S/N ratio:** 46 dB
- **dynamic range:** 52 dB
- **maximum image transfer rate:**
  - **VGA:** 30 fps
- **sensitivity:** 1.3 V/lux-sec
- **pixel size:** 3.6 µm x 3.6 µm
- **image area:** 2361.6 µm x 1756.8 µm
- **package dimensions:**
  - **CSP2:** 3785 µm x 4235 µm

Main mono_slam.m (modified)

```
initIm = 1;
lastIm = 850;
vidName = '15m.avi';

robot = 0;

%switch to capture from robot if robot == 1
if (robot == 1)
robotIP = '192.168.0.103';
motion = ALMotionProxy(robotIP,9559);
Nao_head(robotIP );
motion.setStiffnesses('Body',1.0);
motion.walkInit();
pause(5)
im = takeImageNao(robotIP);
else
im = takeImageFromVideo(vidName,initIm);
end

tic;
% Camera calibration
cam = initialize_cam;
% Set plot windows
set_plots;

% Initialize state vector and covariance
[x_k_k, p_k_k] = initialize_x_and_p;

% Initialize EKF filter
sigma_a = 0.007; % standar deviation for linear acceleration noise
sigma_alpha = 0.007; % standar deviation for angular acceleration
noise
sigma_image_noise = 1.0; % standar deviation for measurement noise
filter = ekf_filter( x_k_k, p_k_k, sigma_a, sigma_alpha,
sigma_image_noise, 'constant_velocity' );

% variables initialization
features_info = [];
trajectory = zeros( 7, lastIm - initIm );
% other
min_number_of_features_in_image = 25; %25
generate_random_6D_sphere;
measurements = []; predicted_measurements = [];



%-----------------------------------------------------------------
% Main loop
%-----------------------------------------------------------------

%im = takeImage( sequencePath, initIm );
step2 = 2;
siz = -0.05;

for step=initIm+1:1:lastIm
```

```
% Map management (adding and deleting features; and converting
inverse depth to Euclidean)
[ filter, features_info ] = map_management( filter, features_info,
cam, im, min_number_of_features_in_image, step );

% EKF prediction (state and measurement prediction)
[ filter, features_info ] = ekf_prediction( filter, features_info );

if (robot ==1)

if (step > 4 )
if mod(step,4)
    step2 = Walk_noblockNao(motion,robotIP,step2,siz);
end
end
pause(1)
im = takeImageNao(robotIP);

else
im = takeImageFromVideo(vidName,step);

end


% Search for individually compatible matches
features_info = search_IC_matches( filter, features_info, cam, im );

% 1-Point RANSAC hypothesis and selection of low-innovation inliers
features_info = ransac_hypotheses( filter, features_info, cam );

% Partial update using low-innovation inliers
filter = ekf_update_li_inliers( filter, features_info );

% "Rescue" high-innovation inliers
features_info = rescue_hi_inliers( filter, features_info, cam );

% Partial update using high-innovation inliers
filter = ekf_update_hi_inliers( filter, features_info );

% Plots,
plots;
%stepcount = step;

% Save images
saveas(figure_all, sprintf( '%s/3image%04d.fig', 'figures/', step ),
'fig' );


end
toc;
if (robot==1)
motion.stopWalk();
end
```

<u>TakeImageNao</u>

```
functionim = takeImageNao(robotIP)
```

```
camProxy = ALVideoDeviceProxy(robotIP,9559);

resolution = 1;     % kQVGA
colorSpace = 11;    % RGB

camProxy.subscribe('matlab',int16(resolution), int16(colorSpace),
int16(30));

naoImage = camProxy.getImageRemote('matlab');

camProxy.unsubscribe('matlab');

IMarray = naoImage{7};

length_n = size(IMarray);

imredstring = IMarray(1:3:length_n);
imgreenstring = IMarray(2:3:length_n);
imbluestring = IMarray(3:3:length_n);

imred = reshape(imredstring,320,240);
red_im = imred';
imgreen = reshape(imgreenstring,320,240);
green_im = imgreen';
imblue = reshape(imbluestring,320,240);
blue_im = imblue';

%imgray = reshape(imgraystring,320,240);
%gray_im = imgray';

rgbimage = uint8(rand(240,320,3));
rgbimage(:,:,1) = red_im;
rgbimage(:,:,2) = green_im;
rgbimage(:,:,3) = blue_im;

im=rgb2gray(rgbimage);

end
```