Imperial College London
Department of Computing

Simple But Effective Personal Localisation Using Computer Vision

MEng Individual Project Report

Stelios Kyprou

Supervisor: Dr. Andrew Davison
Second Marker: Dr. Maja Pantic

2008-2009

Abstract

This report describes a biologically inspired approach to vision-only simultaneous localization and mapping (SLAM). A SLAM algorithm, named RatSLAM, was extended to cope with "shaky" video sequences. The main SLAM system, models a part of the rodents brain, the hippocampus. This system communicates with a fast vision system, which provides, both external and self-motion cues. The system runs completely in an online manner, where it simultaneously calculates odometric offsets, builds a map, detects loop closures and relocalises by recalling previously visited scenes. The built map is continuously calibrated, especially after a loop closure, where odometric errors are corrected. The mapping abilities of this program are demonstrated in a series of indoor and outdoor tests, using a 2.83 GHz Intel Core Duo processor, and 3 GB of RAM. The video capture device was a digital camera, capturing videos at 10 fps and a resolution of 640 $\times$ 480 pixels. The largest outdoor experiment was 23 minutes long, where a route of 750 m was travelled by a person holding a camera. All three loop closures where detected, and an accurate map was obtained at the end of the experiment.

Acknowledgements

I would like to thank my supervisor, Dr. Andrew Davison, for his support during the project development. Our frequent meetings helped me keep a steady schedule and motivation throughout the year. I would also like to thank him for lending me his video camera.

I would like to thank my second marker, Dr. Maja Pantic, for her useful comments and feedback on the early stages of my project design.

Finally,i would like to thank my parents for their continuous support during all four years i have been studying in London.

Contents

Chapter 1

Introduction

Simultaneous Localisation and Mapping is one of the most challenging problems in mobile robotics. Over the years, increase in computational power has allowed theoretical solutions to this problem to be implemented in real world experiments. Various techniques have been used, such as Extended Kalman Filters or particle filtering[1]. Even with this computational power though, the problem is still difficult to solve and make a model that would work efficiently. There are two main reasons for this. Mobile robots are basically moving sensor platforms. Different type of sensors provide various capabilities, but are far from consistently accurate. The most accurate sensor for example, the laser range finder, may still provide false data. It may 'see' through glass, and beams may bounce on multiple surfaces before returning to the receptor. Under direct sunlight, it may make faulty readings. And most importantly, a good high quality laser range finder is expensive. When using probabilistic SLAM, assumptions about the environment are made, such as landmark based techniques, to reduce the the computational processing made during the experiment. Otherwise, mapping large outdoor environments resort to off-line procedures occurring after the experiment.

There have been other studies, where the point of view for solving the SLAM problem is completely different. How can animals and humans navigate through environments, without using expensive sensory systems? Humans do not store the exact distance from a wall when they see one. But we are able to localise our selves, and remember places in a more complicated and accurate manner than a mobile robot. This inspired researchers to study the brain of animals to get a better understanding of how animals use their spacial perception within an environment. The most studied animal has been the rodent, focusing on the rodent hippocampus[2]. This yielded some ideas on how a biologically inspired system can be used to solve the SLAM problem, without using expensive sen-

sors and and complex mathematical probabilistic algorithms.

RatSLAM[8] is a system based on a system of neural networks inspired by the rodent hippocampal complex. It has been designed to work in indoor and large outdoor environments in a real-time effective way, by simulating the rodents spatial cognition described in Chapter 2. Experiments have been made with success, creating maps from small 8*8 meter offices, to a large 3.0*1.6 km suburb using a single video camera[13]. This system has also been extended to allow path planning based on the self-built map and a given goal. RatChat has also been developed to facilitate human-robot interaction,in a teacher-student manner, to provide higher level spatial conceptualization.

1.1 Contributions

RatSLAM proved to be a very good solution for the SLAM problem. It simultaneously localises and builds maps in real time, and provides path planning given a specific goal. The experiments have been proved successful, but with one drawback. In order for the experiment to prove a success, there are a few assumptions:

- The orientation of the camera being used must be forward facing with respect to motion.
- The movement of the camera has little or no translational movement parallel to the camera sensor plane.

This is the reason why when when mapping an outdoor environment, the camera was mounted on the roof of a road vehicle.
The core contribution of this project is the extension of the existing RatSLAM system, so that these constraints are removed (Chapter 3). This would result in wider range applications, such as using the system on rough terrains or on humanoid robots. The simplest implementation, would be to use the RatSLAM system for a simple but effective personal localisation. It has been observed during our experiments that when video recording while walking the movement of the camera is far from parallel to the camera sensor plane.
By successfully implementing this extension, we can use RatSLAM more broadly, but still use it effectively with the experiments that already took place in the past (Chapter 4).

Chapter 2

Background

2.1 Spatial encoding in Rodent Brains

2.1.1 Overview

In the field of biological mapping and navigation, rodents are one of the most studied animals. In addition, the rodent hippocampus is one of the most studied brain regions of any mammal. This helped to establish a good understanding of how rodents navigate and create models of this behaviour. While observing individual cell activity in the rodent hippocampus, three type of cells where identified that helped in navigation:

- Place cells
- Head direction cells
- Grid cells

Place cells appeared to respond to the spatial location of the rodent, head direction cells responded to the animal's orientation, and recently grid cells where discovered, firing at regular grid-like intervals in the environment. It was noticed that grid cells where firing only when certain place cells where active, in conjunction with relative active head direction cells. These inventions are very important to understand how rodents navigate, based on their self-motion information and external cues, used in conjunction to keep their spatial cognition accurate.

Cells	Represent	Fired when:
PC	Location	Rat is in the corresponding location
HDC	Orientation	Head turns in the corresponding orientation
GC	Grid-like Intervals in environment	Specific PC's **and** HD cells are active

Table 2.1: Cell Types: Place Cells(PC), Head Directions Cells(HDC) and Grid Cells(GC)

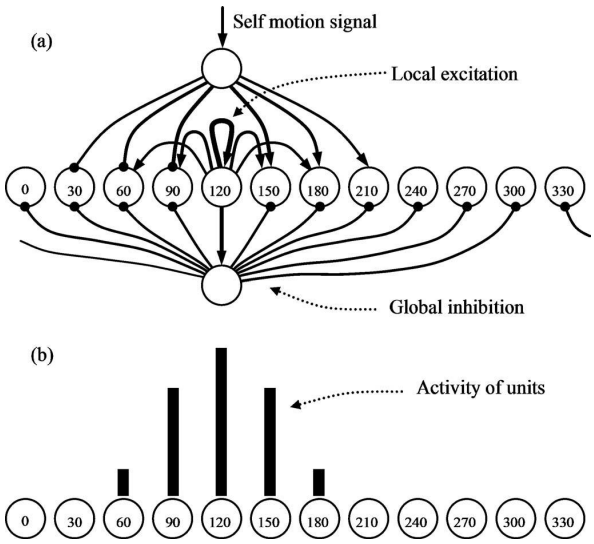

Figure 2.1: Head Direction cell representation

2.1.2 Continuous Attractor Networks

Continuous attractor networks are often used to model the behaviour of the rodent's place cells, head direction cells and grid cells [16],[17]. They represent neural units linked with each other with excitatory and inhibitory connections. The activity in these units, is calculated by computing the sum of activity based on these connections. These connections are recurrent and work rather differently than the standard feed-forward network. In the following subsections, we explain how these attractor networks are used to represent the rodent hippocampus.

2.1.2.1 Attractor Dynamics

For explaining the properties of a continuous attractor network, we will use the Head Direction cells as an example [5].

In figure 2.1 we see a set of head direction cells in a line. Each cell has a set of links. In this example we will use the 120 degree cell. Links are represented

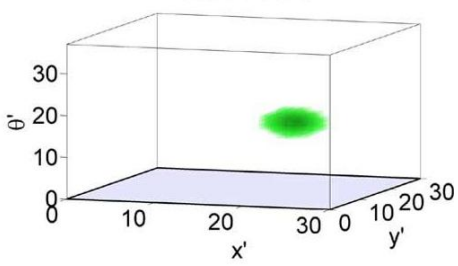

Figure 2.2: A 3-D continuous attractor network, with it's activity packet. This packet travels in the network unaltered, based on self-motion cues

with lines, those ending with a circle being inhibitory links, and those with arrows being excitatory links. When the head of the rodent turns at 120 degrees, a series of events happen. The 120 degree cell is locally excited (injected with energy). The cell excites it's self, and cells around it, through the excitatory links. It also activates the Global Inhibition cell, which in it's turn inhibits cells that is connected to. This will eventually result in a single peak activity, with the most active cell being the 120 degree cell. It is important to say, that if no excitation is provided for a long period of time, all cells will lose their energy. Also in figure 2.1 we see only the connections between the 120 degree cell and the activity level when that cell is excited, and no other self-motion cues are occurring. The rest of the cells have similar connections as well.

2.1.2.2 Path Integration

The activity shown in figure 2.1(b) represents the activity of the head direction cells, in a fraction of the time. With the presence of self-motion cues, this activity packet is shifted, according to the cue. It was shown that this activity packet is transferred while at the same time being unaltered. The activity peak can be moved without deformation using weights that are derivative of the local excitatory weights[19]. This shift in activity, imposes that when errors occur, are not shown in the activity packet, which stays constant both in width and heigh like for example in figure 2.2 . Unlike this method, it is known that in traditional SLAM methods, uncertainty is taken into account, and dealt with continuously, to eliminate false representations.

2.1.2.3 Local View Calibration

So if the attractor dynamics do not take into consideration errors, how will the attractor network consistently represent the position of the rodent? This is a

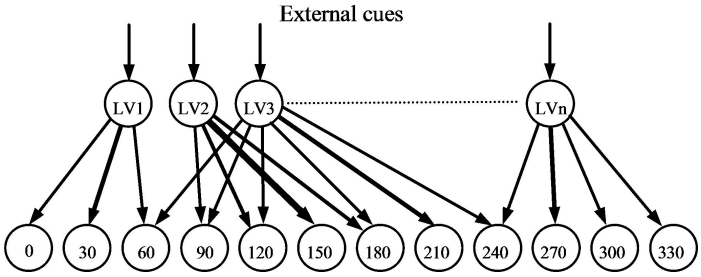

Figure 2.3: Local View cells and their links working in parallel with Head directions cells in figure 2.1 to calibrate the activity packet

common problem in robotics, where a robot cannot accurately know it's position in an environment, by only using it's internal motor sensors. The solution is simple and trivial. Use more sensors, that record environmental cues and make corrections. This is exactly what an attractor network does. It uses local view cells. These cells, represent environmental cues when the rodent is at specific locations. These cells make connections with the head direction cells that where active when the the environmental cue was present. These connections become stronger by using methods such as Hebbian learning.
By using Local View cells (LV), the continuous attractor network will correct the position of the activity packet. If a LV cell is fired randomly, even if the rodent is not at the place where the LV cell was initially activated, activity will be injected in a place where the activity packet is absent. In this case, the Global Inhibition cell, shown in figure 2.1,will filter and inhibit this activity, thus not affecting the network. If on the other hand the rodent is at the position where the LV cell was initially activated, and the current head direction has strong links with the LV cell, this will result in positioning the activity packet where it should be in the first place, thus eliminating all errors. Local View cells work in parallel with the Head Direction cells to provide this functionality [10] .

2.1.2.4 2-D Continuous Attractor Networks

Thinking in the same way as with the Head Directions cells, we can create an attractor network for Place cells as well. This time though. instead of a one dimensional Attractor network, we create a two dimensional attractor network. It's a sheet of neurons, representing the position of the rodent in space. Again neurons have excitatory and inhibitory links, to guide the energy injected to them by self-motion cues. They have links with the Local View cells, just like the Head Direction cells do, to calibrate the activity packet based on external environmental cues. Thinking of the sheet of neurons though,like in figure 2.4,

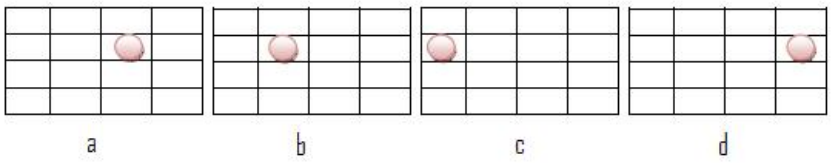

Figure 2.4: A 2-D continuous attractor network, representing Place cells. If the activity packet moves in one direction and reaches the edge of the network, it will continue on the opposite side of the network

what happens if the activity packet reaches the edge of the sheet and still needs to move beyond the edge? This implies that the area a rodent can travel, is restricted by the amount of neurons available. To improve the model, A. Samsonovich and B. L. McNaughton said that the attractor network should be continuous[18]. This means that neurons that are on the edge, are connected with neurons on the opposite side of the sheet. So in figure 2.4, when the activity packet reaches a situation like 2.4(c), it will continue on the other side of the sheet, just like in 2.4(d). This way the rodent is not limited to a particular environment size. It simply means that each neuron, will represent multiple situations in the environment [6] , since it might become active in more than one places in the world.

2.2 RatSLAM

2.2.1 Overview

The RatSLAM model is designed to simulate the rodent hippocampus, and be used in robotic applications. It can be described as a system, containing several major components, shown in figure 2.5. The robots pose is encoded in a single Continuous Attractor Network module, known as the pose cell network. Both self-motion and external cues influence the activity in the pose cells through their respective processing modules.

2.2.2 A model of Spatial Pose

In robotics, it is unusual to use separate representations of robot orientation and spatial location in mapping and navigation algorithms. This is why RatSLAM made an extension to the hippocampal model. It combines the concept of head direction and place cells[3][4] to form the new type of cell, known as a

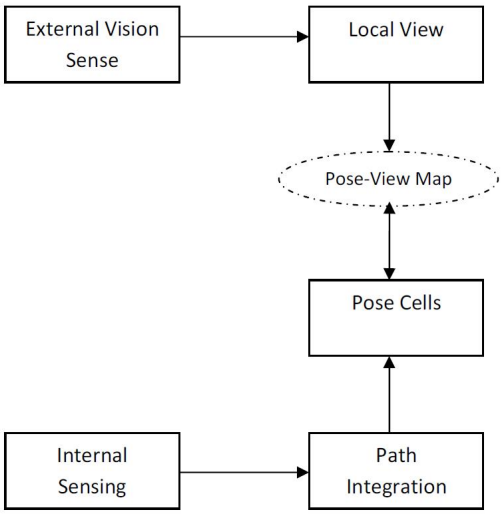

Figure 2.5: The RatSLAM system structure

pose cell.Since place and head direction cells are combined, RatSLAM uses a three dimensional Continuous Attractor Network, that is consisted of Pose Cells. Each Pose Cell is arranged in an (x',y',θ') pattern. This way, all robot pose estimates can be represented in x' , y' , and θ', like in figure 2.2. As described in subsection 2.1.2.4, Pose Cells have wrap around connections as well. This way the environmental size that can be explored by the robot is not restricted to the amount of pose cells in the network.
Each cell in the network can have an activity range from 0 up to 1. This also represents the probability of that specific cell being the correct pose of the robot. When the network is viewed as a whole, and not concentrating on individual pose cells, it all makes more sense. Areas where the activity is strong, is a more probable estimate of the robots pose, than areas with weak pose activity.

2.2.3 Internal Dynamics

In the RatSLAM model, attractor dynamics control the activity in the pose cell network. As described in subsection 2.1.2.1, we have two stages:

1. The excitatory update
2. The global inhibition update

A three dimensional Gaussian distribution is used to create the excitatory weight matrix. This matrix is used by all cells, to inject energy to their neighbours in the pose cell matrix.

Because of the way the excitation works, there might be multiple activity packets present in the network simultaneously. This is the reason why we use a relatively gentle global inhibition. This excitation and inhibition update, is done continuously, during the 'life' of the pose cell network [14] .
As we mentioned in subsection 2.2.2, the activity of each cell is in the interval [0-1], and this is why we add a third step in our internal dynamics: *Normalisation*. At the end of each update,the sum of activity of all pose cells in the network becomes 1, and the individual activity of each pose cell is normalised.

2.2.4 Local View Cells

Another module of the RatSLAM system, are the Local View Cells. These cells act on the system in two ways. They perform associative learning to map with pose cells, and they inject activity levels into the pose cells for localisation. When using a single camera in the model, the external cues injected to the system, are visual scenes. These visual scenes are represented by local view cells. At each cycle of the model, the local view module, creates a connection between the visual scene, and the pose cell that has the largest activity. These links form the pose-view map, as shown in figure 2.5. When a pose-view map is created, it is used by the local view module, to inject energy to the pose cell network. In each system cycle, active local view cells inject activity to the pose cells they have connections with, based on the pose-view map. The amount of activity that will be injected, depends on the strength of association between local view and pose cell. This is the mechanism that RatSLAM uses to maintain or correct the robots believed pose.
In order for the system to work efficiently, the learning method used to build the pose-view map, should not associate raw camera data with pose cells. This is why raw images are processed in the local view module, in order to be converted in a more usable structure for the system. Since the pose cell network will be represented as a matrix, image data are reduced in dimensionality, and are represented by a sparse feature vector (figure 2.6).

2.2.5 Path Integration

When the RatSLAM model is used with a single camera, we have a small peculiarity. The external cues processed by the Local View cells, are raw image data. The same data are used to feed the path integration module, as the internal self-motion cues. It is worth mentioning, that the path integration module manipulates the input data in a different way than the local view module.

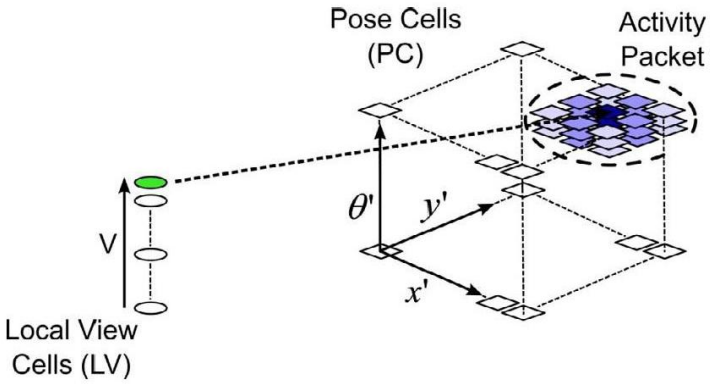

Figure 2.6: The pose cell and local view cell structure

The biological approach in path integration, is injecting a copy of the activity packet forward in time. The RatSLAM system though, takes a different approach. Instead of injecting a copy of the activity packet forwards in time, it shifts the current activity packet, based on the self-motion cues and the position coordinates, (x',y',θ'), of each pose cell. The path integration module, calculates the offset in all three directions,(dx',dy' and dθ') based on the input data, and shifts the activity accordingly. An example can be seen in figure 2.7, where the activity packet shifts over a period of time. That shift, is calculated by the path integration module. The activity packet is initialised at the centre of the pose cell network, just like in figure 2.7.a .At 2.7.b, the path integration module, calculated that based on the image data, the activity packet has to be shifted and performs the shift. Since our network is continuous, the shift takes place on the opposite side of where the activity packet has reached an edge, just like in figure 2.7.d.

2.2.6 Experience Mapping

RatSLAM creates a fine-grained topological map, named the **experience map**, which is composed of individual experiences, *e*, connected by transitions, *t*. Each experience represents a union of **Pose Code** (pattern of activity in pose cells), and **Local View Code** (pattern of activity in local view cells). Each experience is positioned at position **p** in the experience space which is a useful manifold to the real world. So an experience, e_i, can be represented by:

$$e_i = \{PoseCode, LocalViewCode, p^i\} \tag{2.1}$$

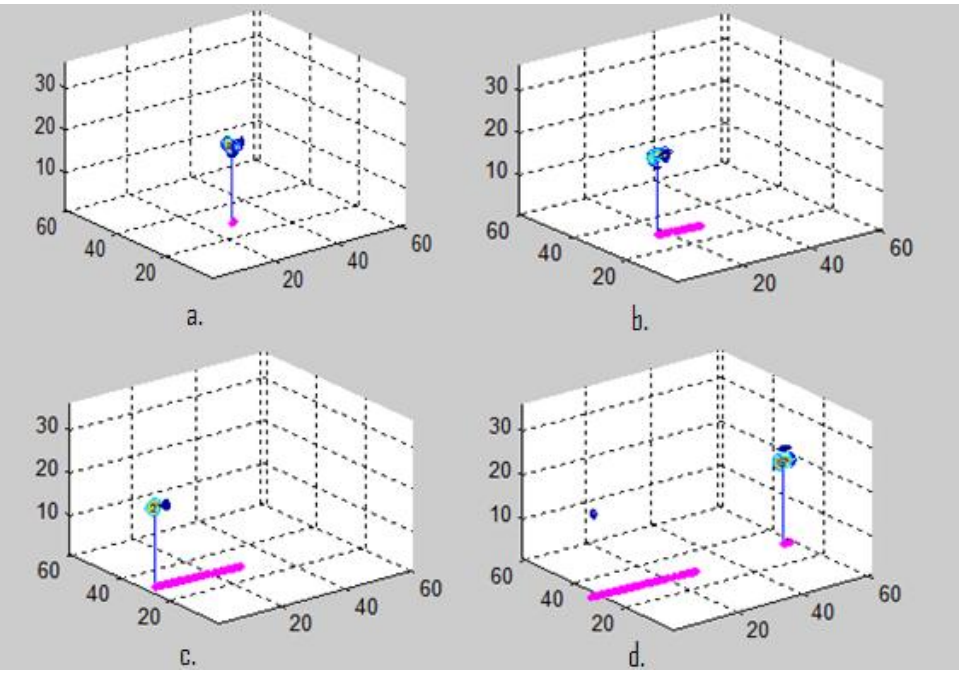

Figure 2.7: A sequence showing how the activity packet is shifted in the pose cell network

In every system cycle, if either Pose Code or Local View Code change sufficiently compared to stored experiences, a new experience is created. Along with the new experience, an associated transition is created,t_{ij} (equation 2.2) [11].

$$t_{ij} = \{\Delta p^{ij}\} \tag{2.2}$$

This transition stores the change in robot position based on self-motion cues. It forms a link between the new experience,e_j, and previous experience,e_i based on equation 2.3.

$$e_j = \{PoseCode, LocalViewCode, p_i + \Delta p^{ij}\} \tag{2.3}$$

The initial experience, is created arbitrarily. This is not a problem, since the experiences that will follow, will be build out from the first experience's position, and the transition links. This is why the experience map is used to create a visual representation of the robots route. As soon as the first experience is

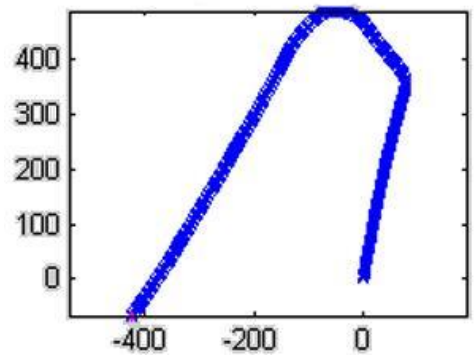

Figure 2.8: Visual representation of an experience map

plotted on the experience map, the rest will be plotted based on the transitional information. An example of an experience map is shown in figure 2.8, where each experience is plotted as an **x** on the graph. The initial experience was assigned the position [0,0].

During a loop closure, there is not a specific algorithm that will make this detection. Instead, in RatSLAM the way to recognise a loop closure is fairly simple. When sufficient change in Pose Code or Local View Code occurs, these changes are compared to existing experiences. If there is a match with an existing experience, it is meant that the robot is situated at a previously explored area, thus a loop closure. If no match is found, then a new experience is created. When a loop closure is detected, it is highly unlikely that the transition used to map the previous experience with the matched one, will point at the same position in the experience map, as the already stored position of the matched experience. This is why a correction algorithm is continuously used to correct the transitions of the experiences. This correction will be mostly effective only when a loop closure occurs [11].

Using the example from figure 2.9, at 2.9a., there is sufficient change in the pose or local view code, but an experience has been detected that matches these new values. It is visually noticeable, that the experience position, is not at a place that has been visited before, thus faulty representing the position of the matched experience-which has already been plotted on the map. This is where the experience correction algorithm starts correcting the map transitions, so that the matched experience, is represented by it's corresponding position on the experience map, like in figure 2.9b. Then in figure 2.9c. and d, we can see how the correction algorithm adjusts the rest of the transitions to make a more accurate representation of the experience map.

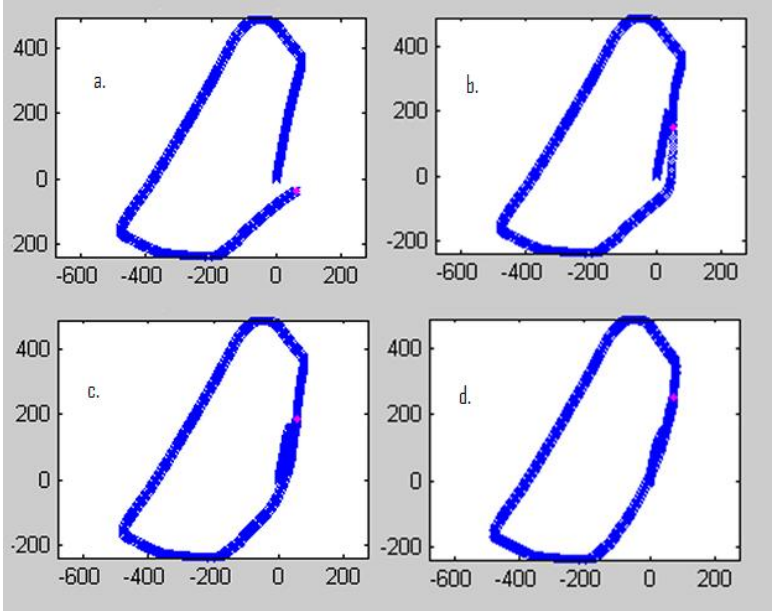

Figure 2.9: Experience map correction algorithm during loop closure

In Chapter 3, we explain how the RatSLAM model was implemented, and how it was extended to adapt in our needs, of personal localisation using a single camera. Then a series of tests where carried out and analysed in Chapter 4.

Chapter 3

The Details

The purpose of the project, is to create an algorithm to aid personal localisation and mapping. The RatSLAM system was used to provide this solution. As explained in Chapter 2.2, the algorithm can be divided in several modules. Each of these modules where designed in a way so that there where no dependencies, apart from using the same global variables. The advantage of modularising the program, is that each module could be designed and implemented several times, without interfering with the rest of the modules.

Figure 3.1 shows how the algorithm was divided. At the beginning of each cycle, the *visual template* module will handle the external cues, which will be camera images, and manage the local view cells. Following this step, the *visual odometry* module will compute an estimation of the persons movement, based on the camera images. It will calculate how much a person moved forward, and whenever the person changed orientation or not. These results will be passed on to the next module, the *pose-cell iteration*. Here the following steps will take place:

1. Add view template energy of the current active local view cell
2. Apply local excitation
3. Apply local and global inhibition
4. Normalise the pose-cell activation levels
5. Do the path integration, by shifting pose-cell activity packets based on the offsets provided from the *visual odometry* module

Finally the cycle ends with the *experience map iteration* module. This is where the experience map is created, updated after each cycle run, and corrected based on the experience correction algorithm.

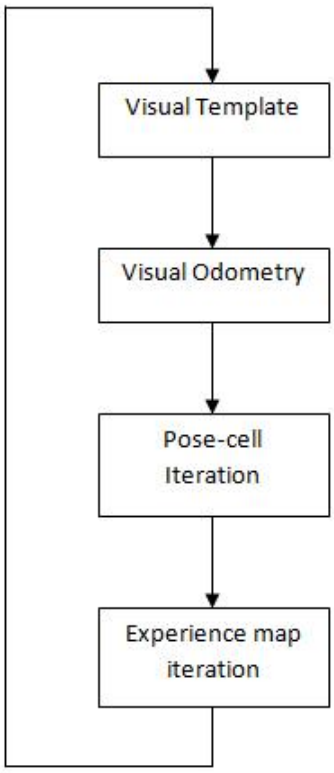

Figure 3.1: Modules that the RatSLAM algorithm consists of

This design was the one chosen to create the RatSLAM system, and it has proven useful. Each module was created and then tested immediately, without the need of the rest of the modules. It has proven useful during the implementation of the system and during testing as well. Errors and bugs where spotted faster, and malfunctioned modules where removed and re-implemented in a mobile way.

The purpose of this project is to localise and map the movement of a person using simple computer vision. This is why it was decided that a single camera should be the only way of providing cues or stimuli to the system. This single camera, will provide raw images to:

1. The visual template module to handle the local view cells (representing environmental cues)

2. The visual odometry module to calculate the persons offset compared to his previous known position (representing self-motion cues).

The raw images are processed within the modules before being used, by being transformed into more usable structures. Each cycle shown in figure 3.1 will be carried out whenever a new camera image(frame) is available.

3.1 Creating the basic RatSLAM algorithm

The following subsections give a detail description of the algorithms used to deliver a complete, working RatSLAM model.

3.1.1 Pose Cells

The activity of the pose cells is described in the tree dimensional activity matrix, *P*. Each item in P, represents a pose cell in the (x',y',θ') coordinates. The activity of these items is updated by the attractor dynamics, path integration, and visual template processes.
The excitatory matrix, $\varepsilon_{a,b,c}$, is created by using a tree dimensional Gaussian distribution. *a*, *b* and *c* represent distances between pose cell units in x', y' and θ' coordinates, and the distribution used is:

$$\varepsilon_{a,b,c} = e^{\frac{-(a^2+b^2)}{k_p}} e^{\frac{-c^2}{k_d}} \tag{3.1}$$

where k_p and k_d are the width constants for place and direction respectively. The change in pose cell activity, because of local excitation, is calculated by:

$$\Delta P_{x',y,\theta'} = \sum_{i=0}^{(n_{x'}-1)} \sum_{j=0}^{(n_{y'}-1)} \sum_{k=0}^{(n_{\theta'}-1)} P_{i,j,k}\varepsilon_{a,b,c} \tag{3.2}$$

where $n_{x'}$, $n_{y'}$, $n_{\theta'}$ are the dimensions of the pose cell matrix in the x', y', θ' space. The excitatory matrix indexes, connect on the opposite side of the activity matrix. This is why a,b and c are calculated like in equation 3.3.

$$\begin{aligned} a &= (x' - i)(mod(n_{x'})) \\ b &= (y' - i)(mod(n_{y'})) \\ c &= (\theta' - i)(mod(n_{\theta'})) \end{aligned} \tag{3.3}$$

In this model, instead of inhibiting cells concurrently while exciting, we will apply local and global inhibition after the excitation takes place. This way, we

can create an inhibitory matrix, that will be applied in the same way on the pose cell activity matrix as the excitatory matrix, but with negative weights. This will make our computation steps easier, with less parameter tuning, and we can combine local and global inhibition in one equation. Global inhibition is constantly applied to all pose cells in each cycle. The new activity matrix will now change its activity level based on the inhibition equation 3.4.

$$\Delta P_{x',y,\theta'} = \sum_{i=0}^{n_{x'}} \sum_{j=0}^{n_{y'}} \sum_{k=0}^{n_{\theta'}} P_{i,j,k} \psi_{a,b,c} - \varphi \tag{3.4}$$

where $\psi_{a,b,c}$ is the inhibitory matrix and φ is the global inhibition energy amount. After this step, all pose cell activity levels are kept positive, by evaluating negative activity amount to zero. We then normalise the activity in the matrix, by updating the activity level of each variable with equation 3.5.

$$P^{t+1}_{x',y,\theta'} = \frac{P^{t}_{x',y,\theta'}}{\sum_{i=0}^{n_{x'}} \sum_{j=0}^{n_{y'}} \sum_{k=0}^{n_{\theta'}} P^{t}_{i,j,k}} \tag{3.5}$$

The path integration process, described in section 2.1.2.2, updates the pose cell activity by shifting the current activity packet, using, equation 3.6.

$$P^{t+1}_{x',y,\theta'} = \sum_{i=\delta x'_0}^{\delta x'_0+1} \sum_{j=\delta y'_0}^{\delta y'_0+1} \sum_{k=\delta \theta'_0}^{\delta \theta'_0+1} P^{t}_{(x'+i),(y'+j),(\theta'+k)} \alpha_{i,j,k} \tag{3.6}$$

where $\delta x'_0$, $\delta y'_0$, $\delta \theta'_0$ are the rounded down integer offsets in the x', y' and θ' directions [14] . The residue component α, is a 2*2*2 cube created from equations 3.7 and 3.8.

$$\alpha_{a,b,c} = g(\delta x'_f, a - \delta x'_0) g(\delta y'_f, a - \delta y'_0) g(\delta \theta'_f, a - \delta \theta'_0) \tag{3.7}$$

$$g(u, v) = \begin{Bmatrix} 1 - u, & if\, v = 0; \\ u, & if\, v = 1 \end{Bmatrix} \tag{3.8}$$

Equations 3.9 and 3.10 show how δ'_0 and δ'_f are computed.

$$\begin{bmatrix} \delta x'_0 \\ \delta y'_0 \\ \delta \theta'_0 \end{bmatrix} = \begin{bmatrix} \lfloor k_{x'}\, v\, \cos\theta' \rfloor \\ \lfloor k_{y'}\, v\, \sin\theta' \rfloor \\ \lfloor k_{\theta'}\, \omega \rfloor \end{bmatrix} \tag{3.9}$$

$$\begin{bmatrix} \delta x'_f \\ \delta y'_f \\ \delta \theta'_f \end{bmatrix} = \begin{bmatrix} \lfloor k_{x'}\, v\, \cos\theta' - \delta x'_0 \rfloor \\ \lfloor k_{y'}\, v\, \sin\theta' - \delta y'_0 \rfloor \\ \lfloor k_{\theta'}\, \omega - \delta \theta'_0 \rfloor \end{bmatrix} \tag{3.10}$$

The errors introduced in path integration, are not corrected like self-motion errors are, in a probabilistic SLAM. This is why we use local view calibration, as described in section 2.1.2.3. We use local view cells, to inject activity in the pose cell network, based on learnt associations. Local view cells are represented by vector *V*, where each element of the vector represents the activity of a local view cell. The associations between local view cells and pose cells, are stored in the association matrix, β. A local view cell becomes active, if the image input provided is sufficiently similar to a previously stored image, associated with a stored local view cell. In this case the local view cell will inject that activity into pose cells that are stored in the association matrix, related to it. The learnt connections in β, follow a learning routine similar to Hebb's law, where the association between a local view cell, V_i, and a pose cell, $P_{x',y',\theta'}$ is given by equation 3.11.

$$\beta^{t+1}_{i,x',y',\theta'} = \max(\beta^{t}_{i,x',y',\theta'}, \lambda V_i P_{x',y',\theta'}) \tag{3.11}$$

This equation is applied in each cycle, on all active local view cells and pose cells. The calibrated pose cells, will now be updated with a correction value, based on the equation 3.12.

$$\Delta P_{x',y',\theta'} = \frac{\delta}{n_{act}} \sum_i \beta_{i,x',y',\theta'} V_i \tag{3.12}$$

where δ determines the strength of visual calibration, and n_{act} is the number of active local view cells [9] . In section 3.2, we describe how activity is injected into the local view cell vector, *V*, via a single camera.

3.1.2 Experience Map

In section 2.2.6 we discussed why an experience map is created, and how to connect experiences via their transition. The transition value, t_{ij} is unlikely going to stay the same during a run. This is because the experience correction algorithm will correct these transitions, following an experience map update [7] . The changes in transitions will be more apparent during a loop closure, where the correction will be higher. The equation for calculating the corrected transitions is shown in equation 3.13.

$$\Delta p^i = \alpha \left[\sum_{j=1}^{N_f} (p^j - p^i - \Delta p^{ij}) + \sum_{k=1}^{N_t} (p^k - p^i - \Delta p^{kj}) \right] \tag{3.13}$$

where α is a correction rate constant, N_f is the number of links from experience e_i to other experiences, and N_t is the number of links from other experiences to experience e_i [6] . The correction rate constant α was set up to 0.5 since larger values resulted in map instability [13].

3.2 Extending the RatSLAM algorithm

RatSLAM has been previously implemented in various experiments, including mapping a large outdoor suburb area [12]. This experiment has been proven successful, but with two critical restrictions:

- The orientation of the camera being used must be forward facing with respect to motion.
- The movement of the camera has little or no translational movement parallel to the camera sensor plane.

Taking into account these restrictions, we conclude that in order for the system to work effectively for our purpose, we must extend the visual odometry module, seen in figure 3.1, so that it works in situations where the above restrictions are not taken into consideration. Also, we have to make sure that the visual template module, correctly recognises already visited areas, so that loop closures are detected successfully. In this system, we will use the same input images, provided to the visual odometry module.

3.2.1 Local View cell calculation

Previous RatSLAM experiments, used three ways for calculating the local view cell vector:

- A cylinder recognition system
- A sum of absolute differences matcher
- A histogram matcher

Since we are using a single, not panoramic camera, we have chosen the sum of absolute differences matcher as the main algorithm for the vision system.
When a new frame is available from the camera, the visual template module, takes a portion of the image, and computes the sum of absolute differences on each column of the sub-image, creating a profile template. This is then compared to all previously stored sub-images. If the difference is small, then it is decided that the image represents a previously visited area. Otherwise a new local view cell is created, representing this profile template. The comparison between the new profile and all stored profiles, is done using equation 3.14.

$$f(s, I^j, I^k) = \frac{i}{w - |s|} \left(\sum_{n=1}^{w-|s|} \left| I^j_{n+max(s,0)} - I^k_{n-min(s,0)} \right| \right) \tag{3.14}$$

where I^j and I^k are the scan-line intensity profiles to be compared, *s* is the profile shift, and *w* is the image width. This comparison is made over a small range of pixel offsets, ψ to provide some generalisation in rotation to the matches. The best match is found by equation 3.15.

$$k_m = argmin\, f(s, I^j, I^k), \quad s \in [-\psi, \psi] \tag{3.15}$$

This match is then evaluated by equation 3.16,

$$d = \min_{s \in [-\psi, \psi]} f(s, I^j, I^{k_m}) \tag{3.16}$$

and compared to a threshold d_m. If the result is below this threshold, then the profile template is considered an already saved template. Otherwise, a new local view cell is created in *V*, representing this profile template following the equation 3.17.

$$V_i = \begin{cases} d_m - d_i, & d_i \leqslant d_m \\ 0, & d_i > d_m \end{cases} \tag{3.17}$$

for all *i*.

3.2.2 Visual Odometry

3.2.2.1 Video Stabilisation

The initial idea for the visual odometry module algorithm, was to create a sub-module named *Video stabiliser*. This sub-module, would process the frames coming into the visual odometry module, and providing a sequence of images, where the movement of the person would be stabilised. Since there are numerous image stabilization algorithms available, a program was used, named *Virtual Dub*, to stabilize an off-line video. After this video was stabilised it was sent to the visual odometry module to calculate the offsets. The calculations where based on the sum of absolute differences matcher.
In order to calculate rotation, equation 3.14 is used on consecutive frames. The pixel shift, s_m, is the shift in consecutive frames, I^j and I^k, that minimises *f()* based on the equation 3.18.

$$s_m = arg \min_{s \in [\rho - w, w - \rho]} f(s, I^j, I^k) \tag{3.18}$$

where ρ is the value that ensures that there is sufficient overlap between images. As soon as s_m is found, it is then multiplied by the gain constant, σ, which is a value calculated empirically based on the camera used. You point a camera to an object, knowing the distance from the object, and the width of the object. We then calculate the angle created from the triangle between the camera and the edges of the object. That is the value of the variable σ.
A similar approach is used to calculate the amount of distance travelled by the person forwards, between two consecutive frames. Again, we use equation 3.14, and generate a speed estimate of the persons movement between two frames, based on equation 3.19.

$$u = \min[u_{cal} f(s_m, I^j, I^{j-1}), u_{max}] \tag{3.19}$$

where u_{cal} is again empirically measured based on the speed of movement and camera. In order to compute the value of u_{cal}, we do the following steps:

1. Capture a video of a person moving forwards, at walking speed.
2. At the same time, estimate the average walking speed, *aw*.
3. Run the video sequence through equation 3.14 and find the minimum differences in shift between intensity profiles of consecutive frames.
4. Calculate the average minimum difference shift and multiply by the number of frames per second to get *md*.
5. Finally divide the average speed travelled by the person, by *md* to get u_{cal}.

$$u_{cal} = \frac{aw}{md} \tag{3.20}$$

It was observed, that during the system runs, sometimes there where extreme values calculated for the speed estimation. In order to avoid these erroneous measurements, we use u_{max} as the upper limit for speed calculation, set to a value that approximately represents the top speed of a walking person.
As soon as the visual odometry module was completed, the stabilised video was passed to it for processing. The estimations of speed and rotational offsets were not very accurate, as rotational values where always over-estimated. The method of stabilizing the video prior processing was proven promising for the visual odometry module, although some calibrations had to be made. The was another, major drawback though when using this method. Feeding the stabilized video to the visual template module gave some unexpected results. Even though the video sequence was clearly making a loop closure, the visual template module was not matching new templates with already stored ones, thus not recognizing that a place in the map has been revisited. Indeed, by looking at figure 3.2, we see that throughout the whole video sequence, local view cells are continuously being created, but never identified again.

In a re-run of the same sequence, the threshold value of d_m in equation 3.17 was increased to give a greater chance for a template match. Unfortunately, the results have shown that even though a loop closure has been detected we had plenty of false positive matches as well(figure 3.3). This means that multiple erroneous loop closures would wrongly affect activity in the pose cell matrix, and eventually the creation of a faulty experience map.

Using an image stabilization algorithm imposed some new problems to our system. During the stabilization process, frames are edited, cropped, zoomed and transposed, in order to provide a less "shaky" video stream [15] . The

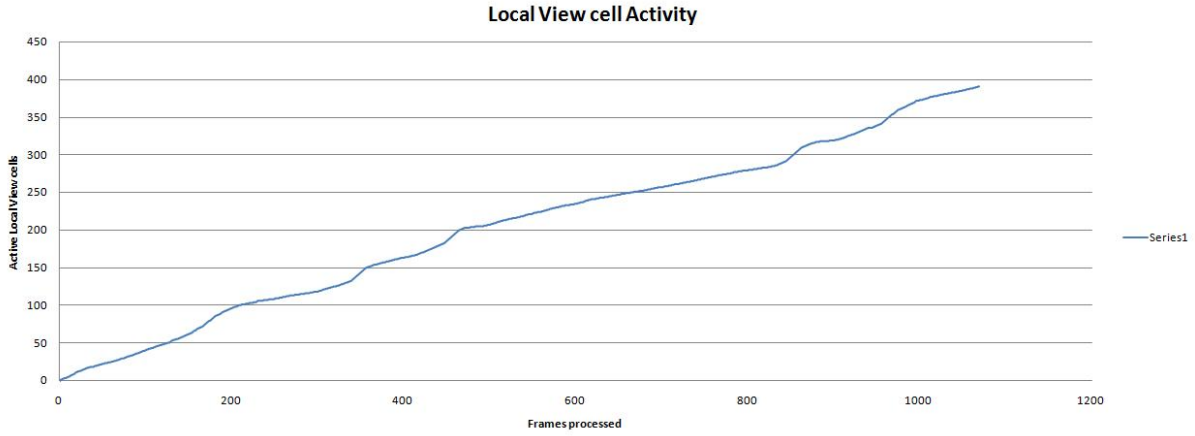

Figure 3.2: Local view cell creation and recognition per frame

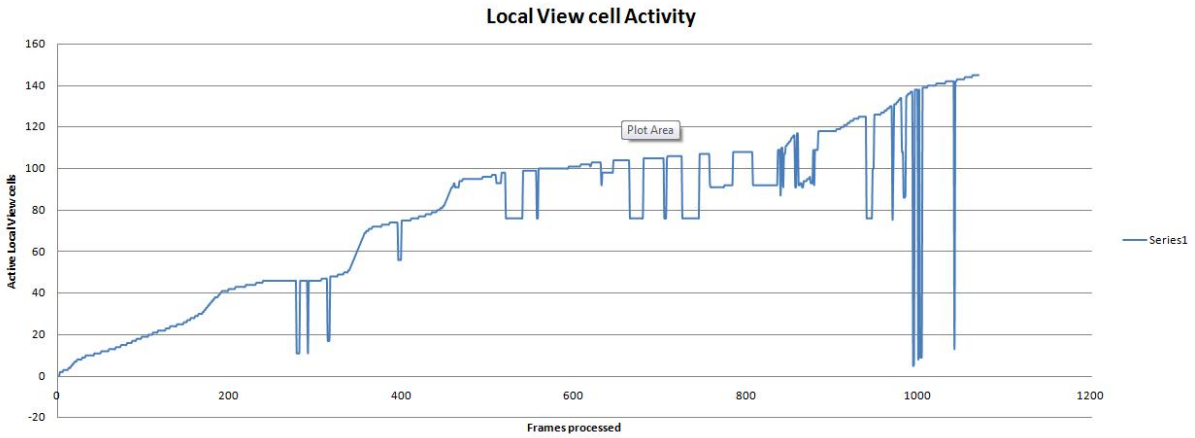

Figure 3.3: Local view cell creation and recognition per frame, with increased d_m threshold

result, is a stabilised video stream, which is not 'clear' enough. The image frames appear blurry and out of focus. Sometimes black borders appear on the edges of image frames, because of excessive camera shake. As a consequence, the visual template module was creating biased image templates, that where not representing the actual intensity profiles of images that were originally captured by the camera.

In figure 3.4a., we see a created experience map, that closely represents the actual path a person has taken. In figure 3.4b. and 3.4c. we see the experience maps created using the stabilized video, with variable's d_m value increased in c. The experience maps created when using the image stabilization algorithm as a solution to the extended RatSLAM system have proven that another method

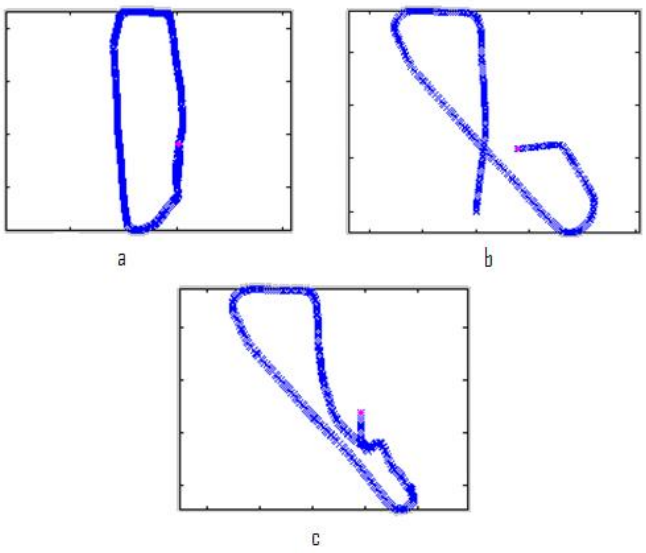

Figure 3.4: Experience maps created during an experimental run, with different d_m values

had to be chosen, in order to make the system more reliable. In the following section, we describe a new method used that made RatSLAM deliver more consistent and correct results.

3.2.2.2 Vertical Image Shift

This idea was inspired while observing the frame-by-frame motion action of an off-line video sequence captured for testing. Motion vectors where applied to the image frames, in order to detect the camera motion when held by a person while walking. The video result, showed a repeated pattern of human motion while walking forwards. If we observe the consecutive frames shown in figure 3.5, we can see that an arc is created by the motion vectors. In fact if more image sequences where shown, the motion vectors follow a pattern similar to the one shown in figure 3.6.

This pattern is the reason why equation 3.17 cannot be used for calculating intensity differences in consecutive frames. For example, let's assume that the camera is not shaking (i.e always parallel to the camera plane) and two consecutive image frames are sent to the visual odometry module. The visual odometry module then takes a subset of this image, and sums the pixel intensities in a column wise direction, like in figure 3.7.

The next step is to shift the second image on top of the first one, in order

Figure 3.5: Motion vectors detect camera movement on a frame-by-frame rate. The sequence begins from the top left image

to find the minimum offset between the two images, where the sum of absolute difference is minimised, as illustrated in figure 3.8. This method does not work correctly though when the camera is not consistently parallel to the camera plane. This is because the camera shake introduces erroneous shifts in the images, both in the horizontal **and** the vertical axis. The idea of the algorithm described below, is again based on the sum of absolute intensity differences,

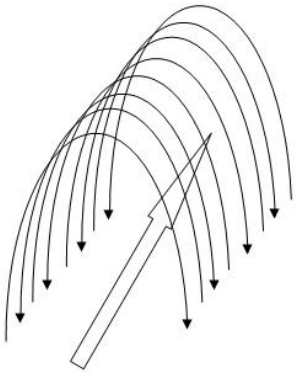

Figure 3.6: A repetitive motion pattern describing the camera movement while a person walks forwards

Figure 3.7: Consecutive frames captured by a stable camera. The inner squares, are the subsections the visual odometry module extracts for calculating rotation. The sum of absolute differences based on pixel intensities is then calculated in a per column pattern

but with additional processing to eliminate the camera shake. Instead of shifting sub-images in the horizontal axis and immediately calculate the minimum possible offset, we introduce an intermediate algorithm:

- We shift the sub-images vertically, until we find the minimum shift, based on sum of absolute differences in **per row** pixel intensities .
- We then extract a new sub-image from the second frame, but this time we add the minimum offset shift.
- Finally we replace the old sub-image with the new one, and exit the algorithm.

Figure 3.8: Shifting Frame t+1 to the left, until the sum of absolute difference in intensities between the two images is minimised

Figure 3.9: Two consecutive image frames, with sub-images selected for the visual odometry module.

The outputs of each step during this algorithm run are presented visually in figures 3.9, 3.10 and 3.11.

As we can see from figure 3.11, the vertical camera shake is eliminated by the algorithm. As stated in section 3.2.2.2, the camera shake was always following a pattern in the shape of an arc. By eliminating the vertical camera shake, we now need to eliminate horizontal camera shake. This could be done by re-applying the algorithm described above, but this time checking if the horizontal shift was below a threshold, $shift_{max}$. Instead,we provide the sub-images to the visual odometry module and expand the rotation calculation algorithm to cope with this horizontal shake. When estimating rotation, we calculate the minimum shift, s_m calculated in equation 3.18. This shift though is biased, since the camera horizontal shake is added to it as well. This is why we extend the algorithm, and the new s_m is decided by equation 3.21,

$$s_m = \begin{cases} 0, & s_m < bs \\ s_m, & otherwise \end{cases} \tag{3.21}$$

Figure 3.10: Sub-image of frame t+1 is shifted vertically until it is matched with frame t in the best possible way

Figure 3.11: Frame t+1 is assigned a new,corrected sub-image, which is then sent to the visual odometry module

where *bs* is the horizontal shift created by the camera shake. This value can be calculated empirically, by processing a video sequence of forward movement while holding a camera, and calculating the average horizontal offset created by camera shake during that sequence.
When running the extended RatSLAM model with the method used in this section, the results where outstanding. The same video stream was used during this experiment, as the one used with the video stabilisation algorithm, in section 3.2.2.2. The results, comparing to the ones we discussed in sections 3.2.2.2 are more accurate and consistent. Using the video stabilisation algorithm, the

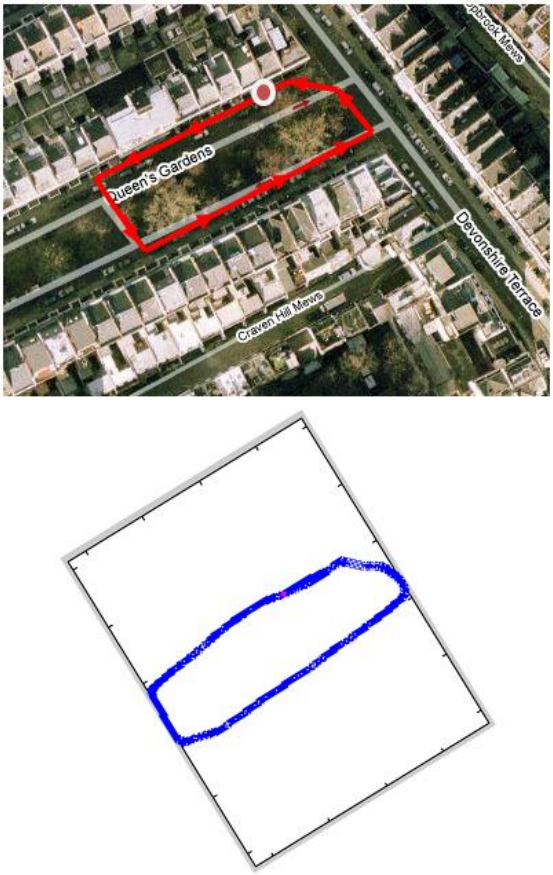

Figure 3.12: Experience map, compared to the actual route taken. The dot on the actual map is where the video sequence started

experience map created (figure 3.4b. and c.) was not a good representation of the actual route the person has travelled during the video recording. On the other hand, using the algorithm explained in this section, the experience map created was an accurate representation of the route travelled. As we can see from figure 3.12, the map is consistent, and almost spot on compared to the actual route followed.

During the same experiment, we recorded the activity of the local view cells, and how they where manipulated by the visual template module. The results again where accurate. We can see in figure 3.13, that local view cells where created, whenever unknown scenes where encountered. During the end of the video stream, the person returns to the starting point, and follows an already explored route. Again, in figure 3.13 we observe that towards the end of the run, the visual template module has recognised that image frames given to

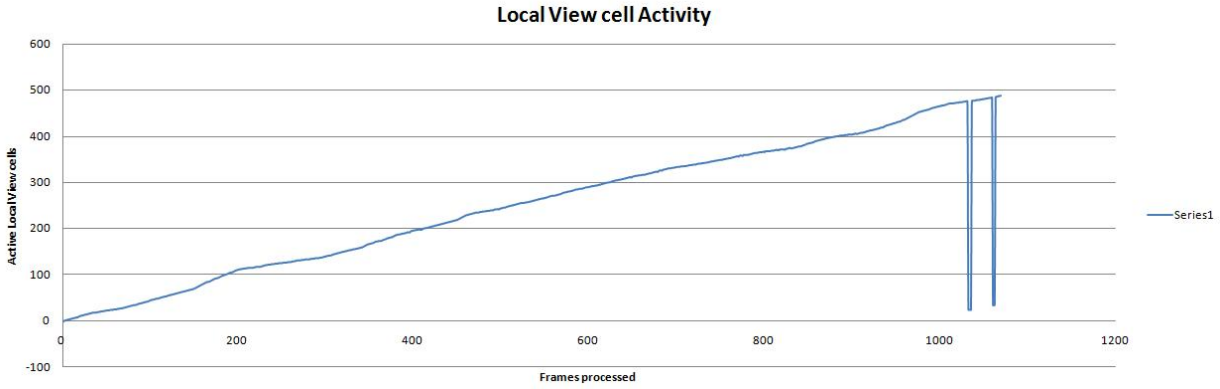

Figure 3.13: Local view cell creation and recognition per frame intervals. We can see that towards the end of the video sequence, familiar scenes are detected, meaning that there is a possible loop closure

the module for processing, have been recognised as already saved images, thus possibly an already explored area. Further testing of the extended RatSLAM system is presented in Chapter 4.

3.3 Implementing the extended RatSLAM algorithm

The programming language that was initially chosen for coding the RatSLAM model, was C++. The only drawback, was the lack of an existing interface between video capture devices and the programming language. This is the reason why it was decided that an other programming language had to be chosen. So after research, the language chosen was Matlab. There where several reasons why MatLab was chosen, the most important being:

1. The ability to read and manipulate multimedia files

2. By adding a specific plug-in you can read live video sequences from any camera device

3. Combining files written in C++ with functions in Matlab, in order to process computationally expensive algorithms faster

4. Using the plotting area to show a graphical visualisation of how RatSLAM works (i.e Experience map, the pose cell 3-D network, the raw video feed.)

The ability to combine C++ files and Matlab functions was really helpful, since it made the whole algorithm run 4 times faster. Profiling results are shown in Chapter 4. Understanding the way of communication between the programming languages was a difficult point during the project development.One example, is that Matlab numbers matrices column-wise, whereas C++ numbers them row-wise, starting from 0, not 1.Also Matlab uses only one type of structure, the mxArray. So sending variables to C++ is simply providing a number of mxArrays. One could be an Integer, another being a Matrix. They had to be handled in a special way so that the correct inputs where given to the program.

Within Matlab, each module shown in figure 3.1 was written in a separate file. This way, each module was developed, tested and debugged individually. A separate file, combined and coordinated all modules, and displayed the result on the plot area. The C++ files, where written within Matlab. They where then compiled, and a .mex file was created, which was the one used by the program during runtime, whenever the C++ function was called.

Chapter 4

Evaluation/Discussion

When the project was being developed, there where two kind of requirements that had to be fulfilled. One was to deliver consistent and accurate results, the other being to compute those results fast enough. Therefore, two types of testing where taking place at the same time. A built in profiler was used to time the program during execution, and maps where created and saved, to view if the algorithm mapped an area accurately. Both types of testing, required the complete working Matlab code, including the compiled C++ files.
The experiments where conducted on a 2.83 GHz Intel Core Duo processor, and 3 GB of RAM. The system was using the Windows Vista 64-bit operating system. The version of Matlab installed was ver. 7.7.0(2008b), and the C++ compiler used was Microsoft Visual C++ 2008 Express. In order to compile C++ files in Matlab, one has to type **"mex -setup"** in the Matlab console, and follow the steps to set up the preferred C++ compiler. Then, C++ files can be compiled, by typing "mex foo.c" in the Matlab console, where "foo" is the name of the C file.
All off-line videos, where captured using two type of cameras:

- Canon Digital IXUS 700
- SONY DCR-SR35 Video Camera

and then converted into .avi files, using the xvid compressor.

4.1 Program Profiling

The system went through profiling tests in 4 rounds. The video sequence used in all rounds was the same, and no program adjustments where made in between.

The output of the RatSLAM algorithm was exactly the same after each round. This way we can focus completely on efficiency, rather than output quality and result accuracy. The built in Matlab profiler, need not any initial configuration when used.
There are three .mex files (compiled .c files) in the project, and each one was written in order to help Matlab functions with computational burden. In the 4 round testing, we initially profile a complete working program, with no help from .mex files and obtain some unsatisfactory results.

Function Name	No. of Calls	Total time (in s)	Self time time (in s)
main	1	750.344	0.31
experience map iteration	1068	281.602	143.666
visual template	1068	171.518	169.343
pose cell iteration	1068	146.07	134.862

Table 4.1: Profiling results after round 1. Total time is the time a function was consuming. Self time represents the time solely spent by the function, excluding external calls

Table 4.1 shows the profiling results after processing a video of length 135 seconds. The complete process was taking too long, with the computational burden appearing in the functions displayed in table 4.1. More functions where used, but here are shown the ones that taken most of the time. Using the Matlab only system, we see that the RatSLAM model was running 5.5 times longer than the length of the video, in order to create the experience map for the whole video sequence. The results for the visual template module, have showed that 99% of the computational burden was within the function, and not because of external Matlab calls.
In Round 2, we profiled the program, which this time included a .mex file, *segments.c*. It is used by the visual template module to improve it's efficiency. When profiling the program again, we got some better results, shown in table 4.2.

The visual template module, has now dropped to around 52 seconds, with self time of 3.625 . The rest of that time, segments.mex was doing the calculations for the module, making it 3.3 times faster than in round 1. Clearly, the experience map iteration function is now the most computationally expensive function, consuming 45% of the total program run time.
In Round 3, the second .mex is added to the program, *vtcalc.mex*, which is used by the experience map iteration function. Profile results after the run of this round are shown in table 4.3.

Function Name	No. of Calls	Total time (in s)	Self time time (in s)
main	1	630.628	0.282
experience map iteration	1068	281.602	143.666
pose cell iteration	1068	146.07	134.862
visual template	1068	51.802	3.625
segments(MEX-function)	271016	48.492	48.492

Table 4.2: Profiling results after round 2. Total time is the time a function was consuming. Self time represents the time solely spent by the function, excluding external calls

Function Name	No. of Calls	Total time (in s)	Self time time (in s)
main	1	240.165	0.274
pose cell iteration	1068	151.22	138.942
visual template	1068	50.068	3.863
segments(MEX-function)	271016	46.437	46.437
experience map iteration	1068	17.091	1.654
vtcalc(MEX-function)	1068	13.911	13.911

Table 4.3: Profiling results after round 3. Total time is the time a function was consuming. Self time represents the time solely spent by the function, excluding external calls

Using vtcalc.mex made the whole program much faster, by making the experience map iteration function 16.5 times faster than round 2. The pose cell iteration function is now the most computationally expensive function consuming 63% of the total program run time.
Finally in Round 4, *posecell-iteration.mex* is used by the pose cell iteration function, giving the profiling results in table 4.4.

Function Name	No. of Calls	Total time (in s)	Self time time (in s)
main	1	147.602	0.044
visual template	1068	47.882	4.146
segments(MEX-function)	271016	44.153	44.153
pose cell iteration	1068	30.679	10.322
experience map iteration	1068	17.091	1.654
vtcalc(MEX-function)	1068	13.911	13.911
posecell-iteration(MEX-function)	2136	6.007	6.007

Table 4.4: Profiling results after round 4. Total time is the time a function was consuming. Self time represents the time solely spent by the function, excluding external calls

The pose cell iteration function is now 4.9 times more efficient than in round 3. As a result of using these three .mex files, the total run time of the program was estimated to be 147.602 seconds. Comparing this to the total run time of

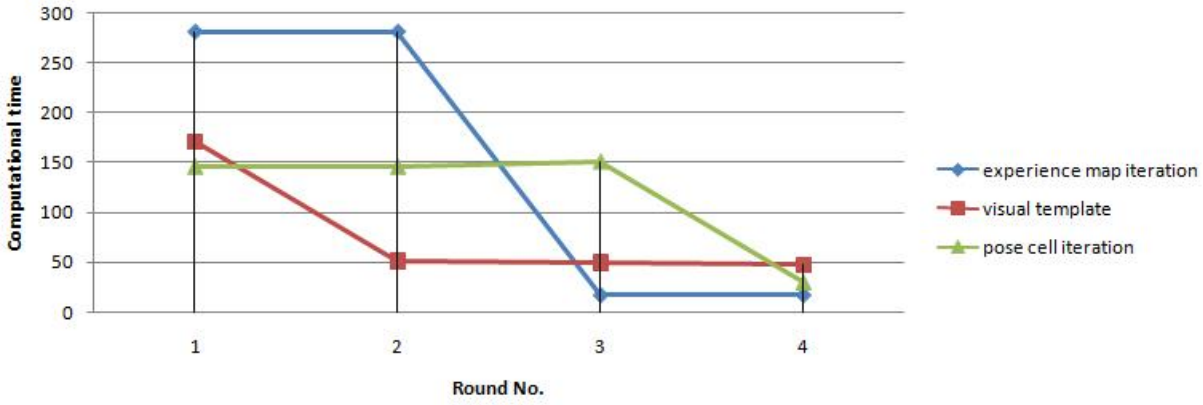

Figure 4.1: How computationally expensive modules are improved, after each round

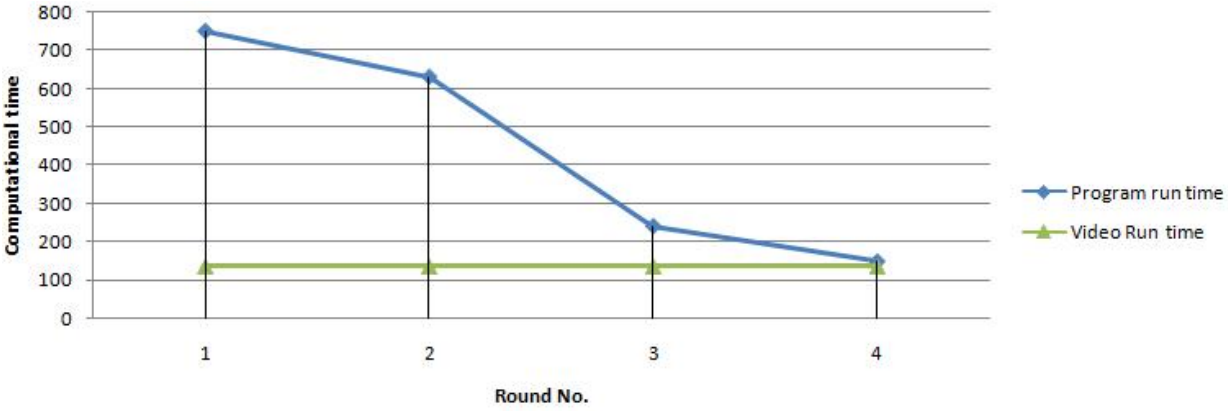

Figure 4.2: Total run time after each round. The aim is to have a total run time, less than or equal to the video run time(video length)

the video sequence, which was 135 seconds, an additional 12.602 seconds of processing time is required. This means that there is a 0.087 seconds of delay in each second of the video sequence. Figure 4.1 shows how the three most computationally heavy modules where improved after each round. As a result the total run time was reduced, almost reaching real-time processing capabilities (figure 4.2).

4.2 Test Cases and Experiments

This section describes experiments run in indoor and outdoor environments.The experiments where carried out on a desktop computer, whose specifications

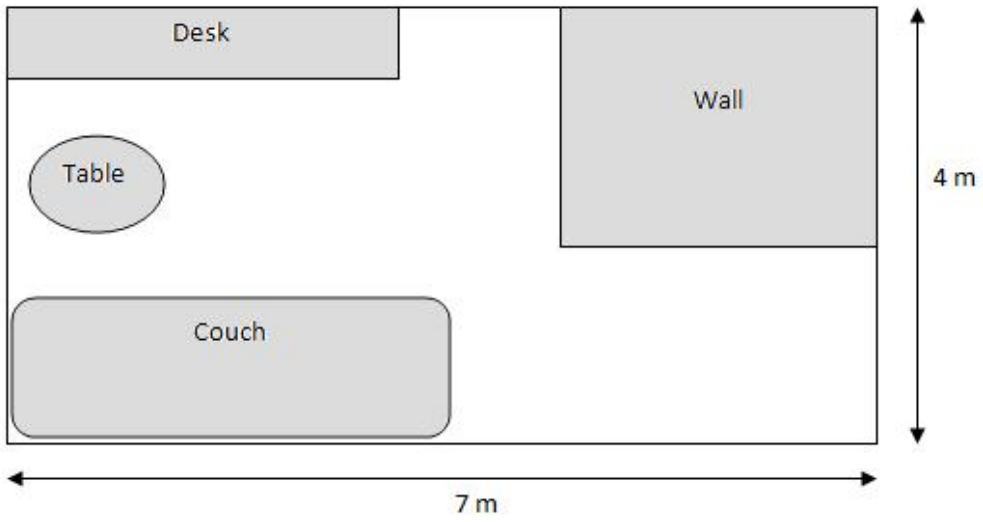

Figure 4.3: Floor plan of indoor test area

where described at the beginning of this chapter.

4.2.1 SLAM in indoor environments

This set of experiments where carried out in an indoor environment, with dimensions of approximately 7 m $\times$ 4 m. No modifications where made to the environment. No information was given to the program prior execution. Learning, recall and map calibration where all made during the experiment at regular intervals. A floor plan of the indoor testing area, is shown in figure 4.3.

In the following tables, we see how variables, stated in Chapter 3, are set in order for the experiment to be accurate. For example, the average walking speed in this indoor environment, is 0.2 m/s -u_{cal}-(table 4.8). It is important to state that d_m, in table 4.6 was set to a low value, comparing to the value given in outdoor experiments. The performance of the algorithm was dependent on how this value is set.

Pose cell Variables	
$n_{x'}$, $n_{y'}$,$n_{\theta'}$	61 $\times$ 61 $\times$ 36
Pose cell size	0.2 m $\times$ 0.2 m $\times$ 10 deg
k_p	7 cells(70 m)
k_d	7 cells(70deg)

Table 4.5: Variable value assignments for indoor environments

Local View cell Variables	
d_m	0.06

Table 4.6: Variable value assignments for indoor environments

Experience map Variables	
a	0.5

Table 4.7: Variable value assignments for indoor environments

Empirically calculated Variables	
σ	0.0625 degrees per pixel
u_{cal}	0.2 m/s
u_{max}	0.35 m/s

Table 4.8: Variable value assignments for indoor environments

Indoor Test 1

In this experiment, a video sequence was captured during a route shown in figure 4.4 .The duration of the test was 4:11 minutes. The path was repeated three times, in two RatSLAM configurations. The first configuration included no local view cell calibration, thus the experience map was displaying pure odometric measurements [7] . The second configuration of RatSLAM, included local view cell calibration and the results where completely different.

Figure 4.5 shows the experience map delivered at the end of the video stream, while using the 1st configuration. We can see that there are odometric errors in the created map, and no detected loop closures. The reason why the errors seem to be so big, is because of image frames that had no features (i.e a white, featureless wall), thus not accurately computing camera movement in between frames. A rigid example, is a sequence of frames, where during rotation, a white wall is encountered, and rotation is calculated inaccurately (figure 4.6).

When the second configuration was used, local view cells calibrated the experience map, making a huge difference. The experience map is presented in figure 4.7. The path was calibrated every single time a loop closure was detected. It's obvious, that the visual template module, is a must during a SLAM test, since external cues are important for calibrating a purely odometric system. A problem encountered during this test, was recognising the initial

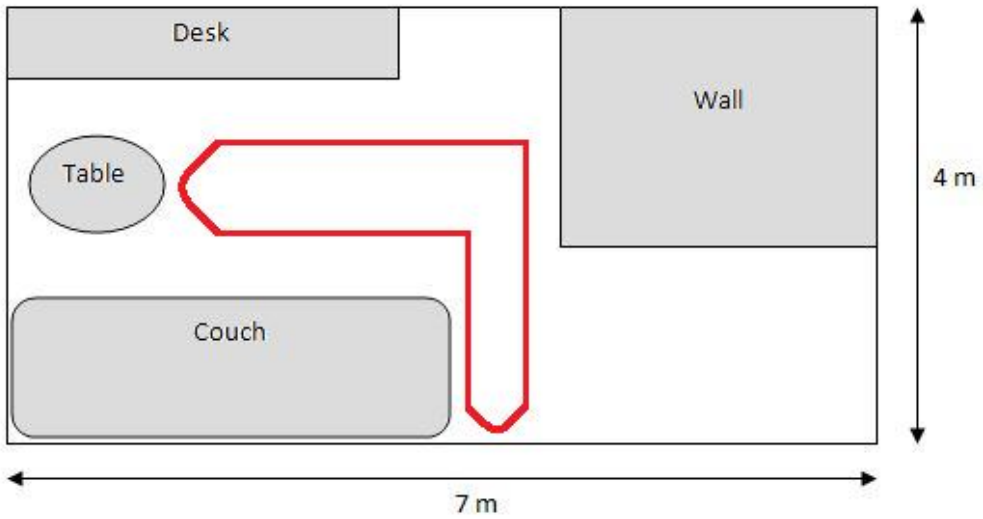

Figure 4.4: Path taken during Test 1

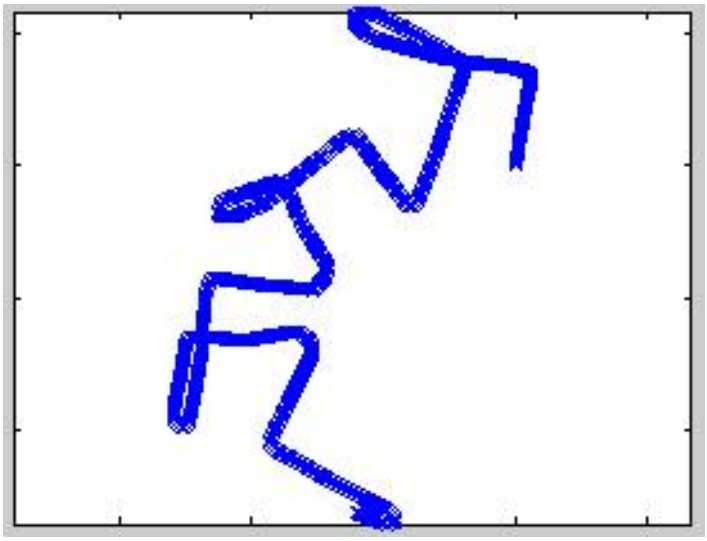

Figure 4.5: Path taken during Test 1

stored images during loop closure. This is the reason why an extra line appears on the bottom right corner of the experience map in figure 4.7.

There where 443 visual templates created, over a total of 2517 image frames. As shown in figure 4.8, all three loop closures where recognised by the visual template module, by identifying familiar scenes that have been visited before. An important observation, is that the system does not recognise already visited paths if travelled in opposite directions. In order to tackle this problem, an omnidirectional camera can be used instead, providing 360 degrees images.

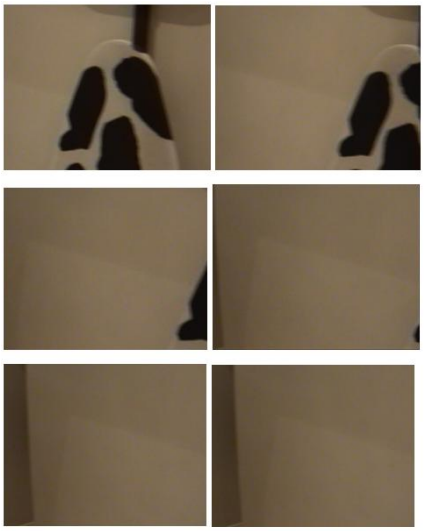

Figure 4.6: A sequence of images where rotation and forward movement are calculated inaccurately

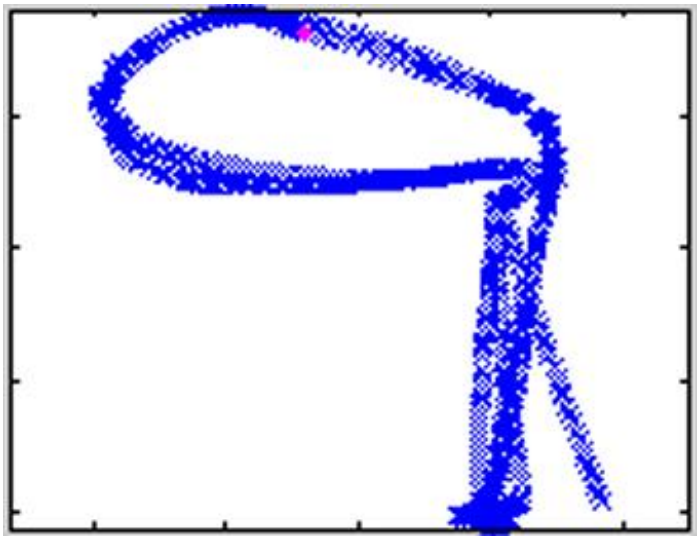

Figure 4.7: Experience map delivered after Test 1, using local view calibration

Indoor Test 2

Under the same conditions, another experiment took place in the same indoor environment, but with a different route, as shown in figure 4.9. The total length of the video sequence was 1:57 minutes. The path was repeated 1.5 times.

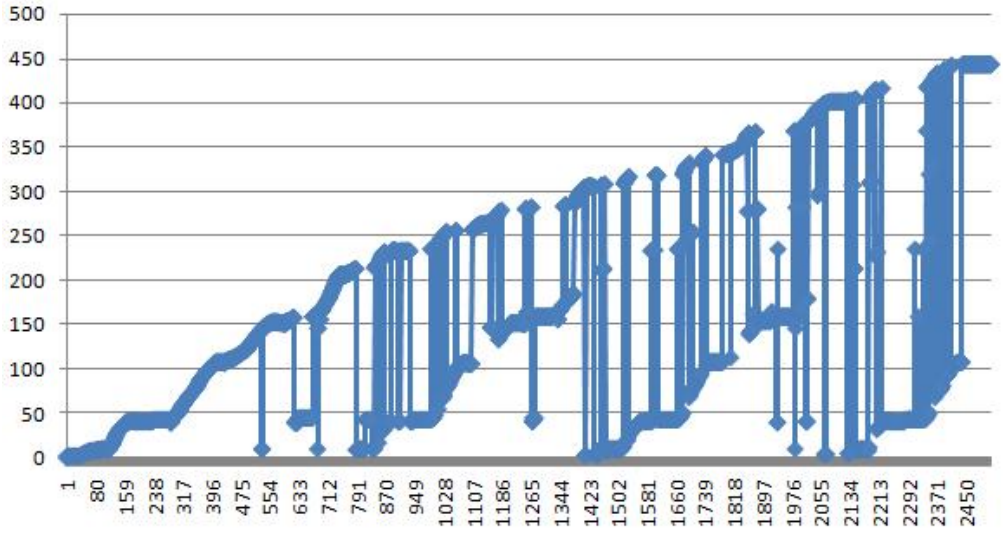

Figure 4.8: Creation and recognition of Visual templates

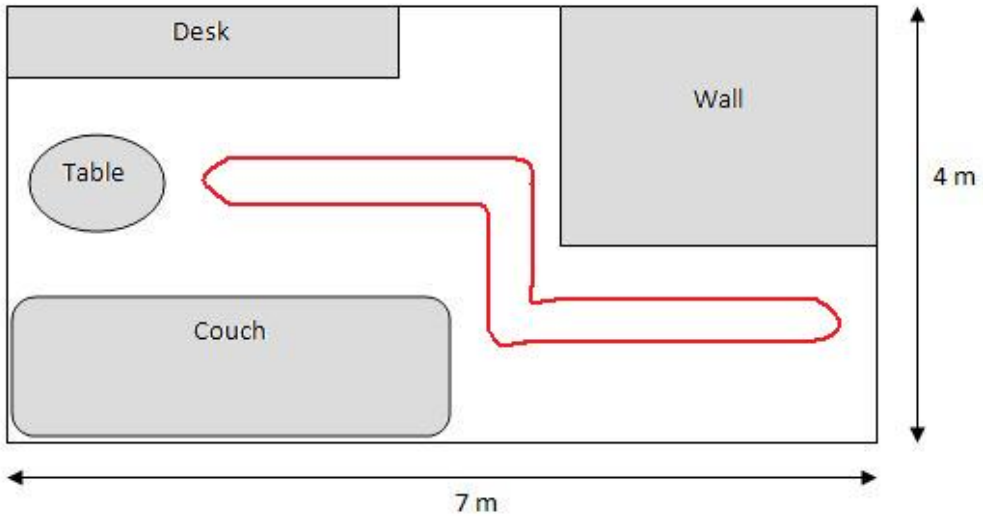

Figure 4.9: The route for Indoor test 2

Again we run the video sequence twice, one with a path integration only performance, and one with the complete program, including local view calibration. The results where similar to the ones in Test 1. When local view calibration was disabled, the experience map created did not show any consistency or accuracy comparing to the actual route taken by the person (figure 4.10). It is worth mentioning though, that even if the experience map is not accurate, each individual line on the map is a good representation of the corresponding path line travelled in the real world. Thus the repeated line patterns occurring in the

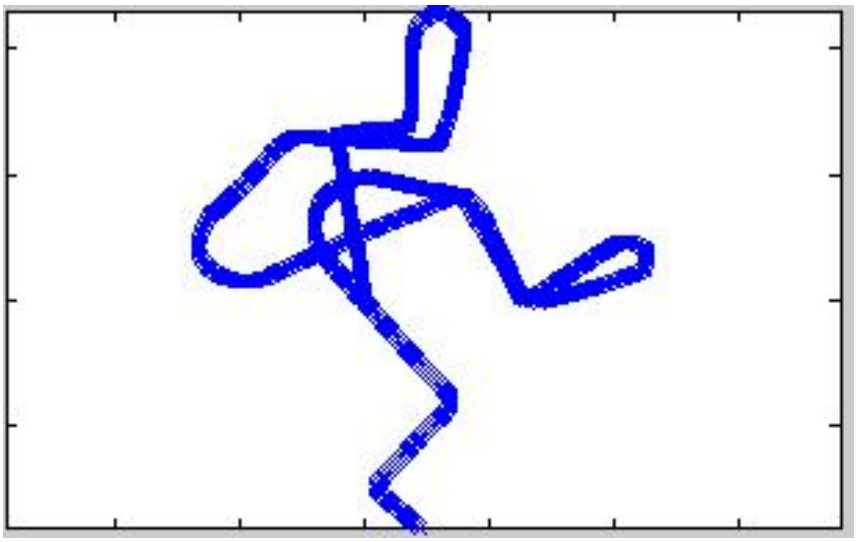

Figure 4.10: Experience map created while in a Path Integration only performance. The pattern of the path that was travelled twice, is erroneously plotted twice, the second time being rotated comparing to the first pattern.

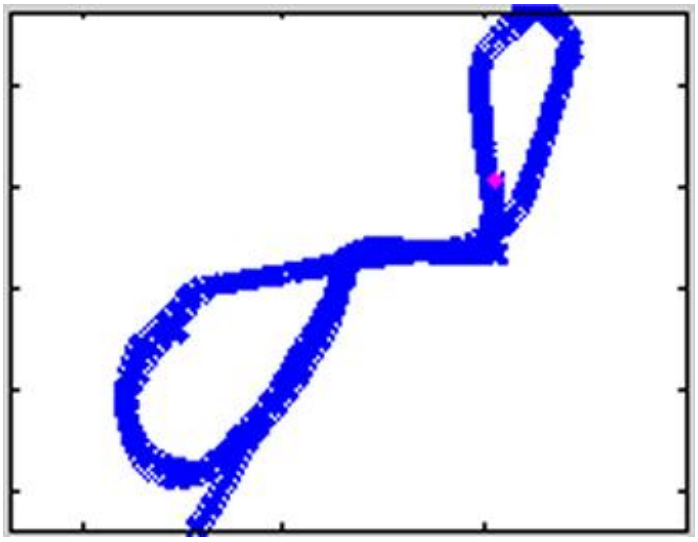

Figure 4.11: Experience map creation during a complete program run

map, which in normal conditions, should have been just a single pattern.

When local view calibration was activated in the 2nd run, more accurate results where plotted on the experience map, by calibrating odometric errors. Loop closures where detected, already visited routes where identified, and the final result was much more accurate than the first run. Figure 4.11 shows what the experience map looked like at the end of the run.

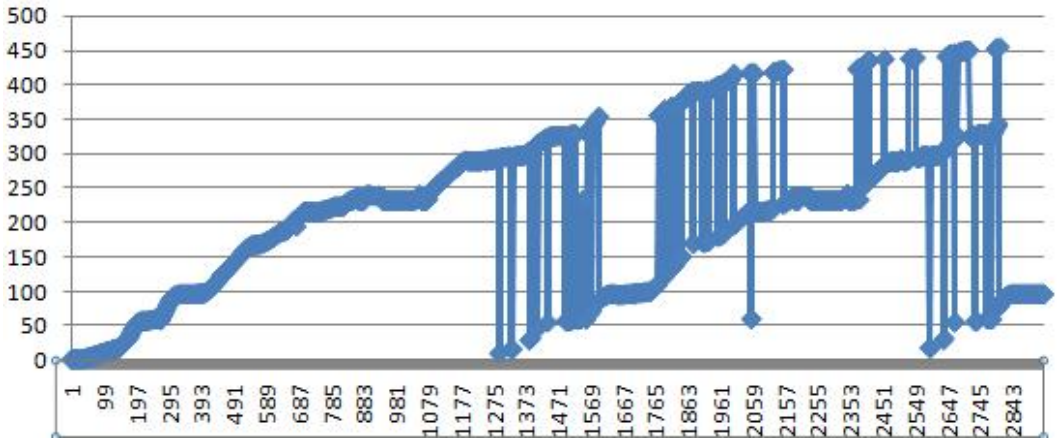

Figure 4.12: Creation and Recognition of Visual Templates

In this test run, 455 local view templates where created, in a total of 2937 frames (figure 4.12).We can clearly see the image identification, indicating a possible loop closure between at frame 1275, and then a continuous identification of images, giving a possible sign that the path has been explored before. The same pattern starts repeating itself from frame 2590.

4.2.2 SLAM in outdoor environments

This set of experiments, focuses on SLAM in outdoor environments.No modifications where made to the environment. No information was given to the program prior execution. Again, learning, recall and map calibration where all made during the experiment at regular intervals. No extra,off-line processing time was given for any calculation. Some system variables had to be changed in order for the program to work well in outdoor environments, shown in the following tables.

Pose cell Variables	
$n_{x'}$, $n_{y'}$,$n_{\theta'}$	$61 \times 61 \times 36$
Pose cell size	2 m $\times$ 2 m $\times$ 10 deg
k_p	7 cells(70 m)
k_d	7 cells(70deg)

Table 4.9: Variable value assignments for outdoor environments

We can see that the local view template threshold, d_m, is set to 0.1 (table 4.10), comparing to 0.06 used in the indoor environment. This is because while doing a sample run, it was noticed that visual scenes which have already been

Local View cell Variables	
d_m	0.1

Table 4.10: Variable value assignments for outdoor environments

Empirically calculated Variables	
σ **for Sony DCR-SR35**	0.089 degrees per pixel
σ **for Canon Ixus 400**	0.056 degrees per pixel
u_{cal}	1.7 m/s
u_{max}	2.35 m/s

Table 4.11: Variable value assignments for outdoor environments

Figure 4.13: Actual path taken during Outdoor Test 1

visited, where not identified [9]. Thus the threshold went through a tuning period, where it's value was incrementally raised, until familiar scenes where recognised. While incrementing the value above 0.1, many false positives appeared in the results. Thus it has been decided that this value was the ideal for balancing visual discrimination and generalization.

Outdoor Test 1

During this experiment, a single 60 m $\times$ 20 m square was travelled, creating a single loop closure in under 5:52 minutes (figure 4.13). The starting point was A, and following the square in an anti-clockwise direction, end up at point B.

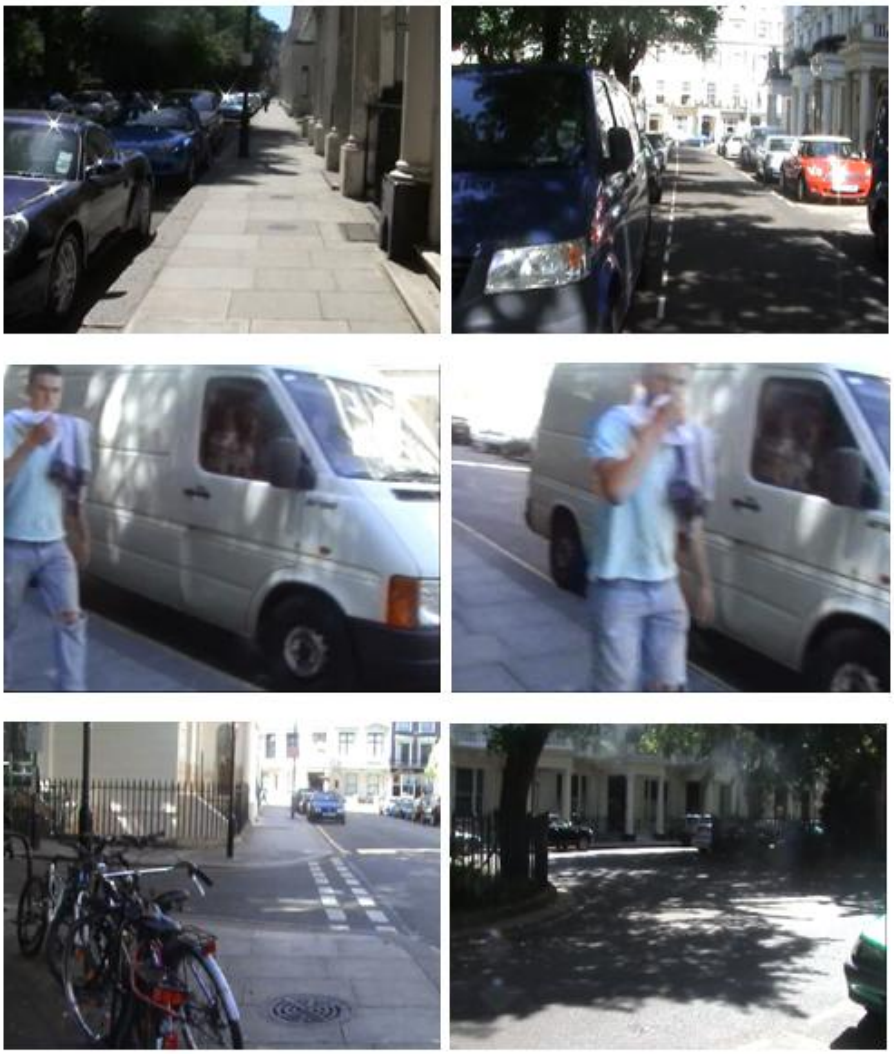

Figure 4.14: Frames from the video sequence for Outdoor Test 1

The video was captured during mid-day on a normal working day. There was not any preparation prior the capture of the video. As shown in figure 4.14, there where parked cars of different colours, paths where intense light reflections where reducing the picture quality, and areas where other people are present, and are captured in the video sequence.

This did not affect the overall performance of the algorithm though. We got some good results, even when local view calibration was not enabled. As we can see from figure 4.15, the final map is accurate enough. What RatSLAM has not done, is detect the loop closure towards the end of the video sequence, and fix the experience map. So when the video sequence reached point A after a whole

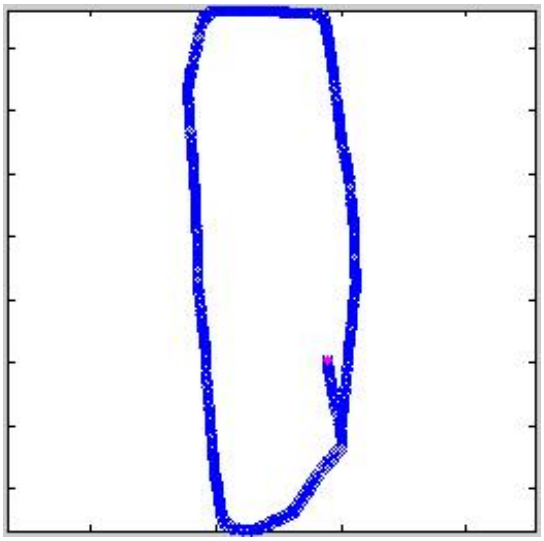

Figure 4.15: Experience map after a Path Integration only performance

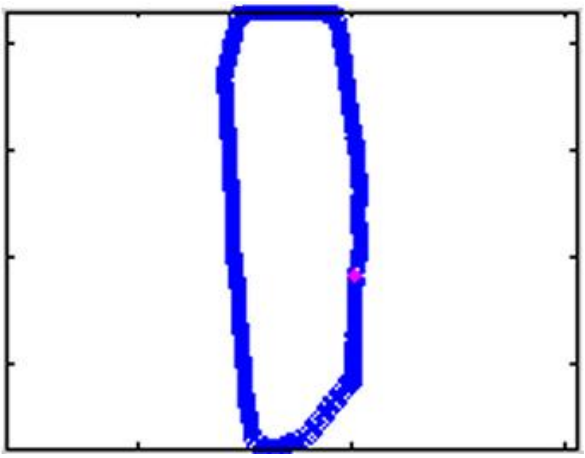

Figure 4.16: Experience map after a Complete System run, including Local View Calibration

loop, an odometric error caused a slight shift to the left. As a consequence, two paths on the experience map represent the single route between A and B.

During another run, the local view calibration was enabled, and the result can be seen in figure 4.16. The loop of the square was detected, and the part of the route that was travelled twice (between A and B) was recognised with the help of the visual template module. As a result, the SLAM algorithm has created a map, that is very accurate, with no additional erroneous paths.

It is important to recall that the experience map is corrected continuously throughout an experiment. In order to cause a big correction in the map, a loop

closure should occur. Using figure 4.17 as an example, we see on the left column the experience map and on the right column the local view cells. The first pair shows an indication that there might be a possible loop closure because familiar scenes are recognised on the local view graph. By viewing the next pair, we can say that the scenes recognised where actually an already travelled path, since the local view graph keeps showing that more familiar scenes are recognised. This is where the gap, between the Starting Point of the route and the current position of the person, is filled. The third pair indicates that further correction was done on the map after the loop closure, and the current position of the person holding the camera is on a path already travelled. This is also confirmed by the visual template graph, where familiar scenes continue to get recognised.

Outdoor Test 2

This test run involves an outdoor environment of 160 m length and 25 m width. The conditions where the same as with Outdoor test 1. The video sequence was 23 minutes long. It includes 3 loop closures, as shown in figure 4.18.

At the end of the run, the final experience map was the one showing in figure 4.19h. The map is consistent concerning the path taken throughout the route. What the program has not been able to do, is to generate a straight line while walking a 160 m straight street. Because of this, the rest of the map becomes slightly bend when the loop closures occur, and a peculiar S bend at a part of straight path. Some of the key moments of experience map correction, during the experiment, are displayed in figure 4.19

In figure 4.19.a ,the first loop closure is detected, thus the result in 4.19.b. In 4.19.c, we see that the 160 m straight line was recorded by the visual odometry as slightly bend. 4.19.d shows the second loop closure. The most important figure, is 4.19.e. At this point, because of the small curve created at 4.19.c, the current path is very inaccurate in comparison to the actual route taken, which should be a straight line connecting the 1st with the second loop closures. At 4.19.f, the visual template injects activity to the pose cell network, since familiar scenes are recognised. This results in the 3rd loop closure. It is obvious though, that the current experience map does not represent the actual route, since an approximately 315 degree turn is introduced into the path, which is non existent. What we see in 4.19.g and 4.19.h, is a further correction of the experience map, caused by the loop closure in 4.19.f. The result of this experiment is not entirely accurate, but it is consistent enough. There are two ways that could improve the output of this test stated below:

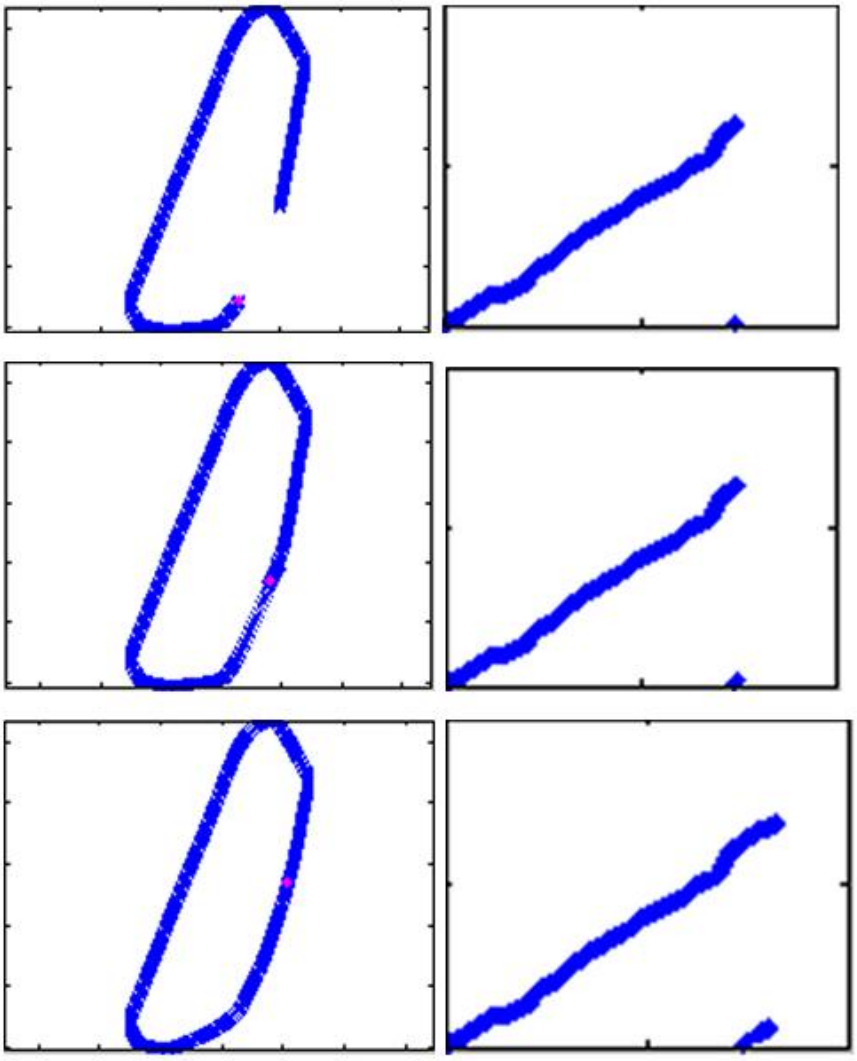

Figure 4.17: Loop Closure and Experience Map correction

1. Increase the number of experience correction loops made after each cycle. The problem is that as you increase the number of correction loops, the less efficient the program becomes.

2. The final map was fairly corrected after 3 loop closures. If the video sequence was extended, and more loop closures where added to the sequence, then the experience map would be further corrected, thus becoming more accurate.

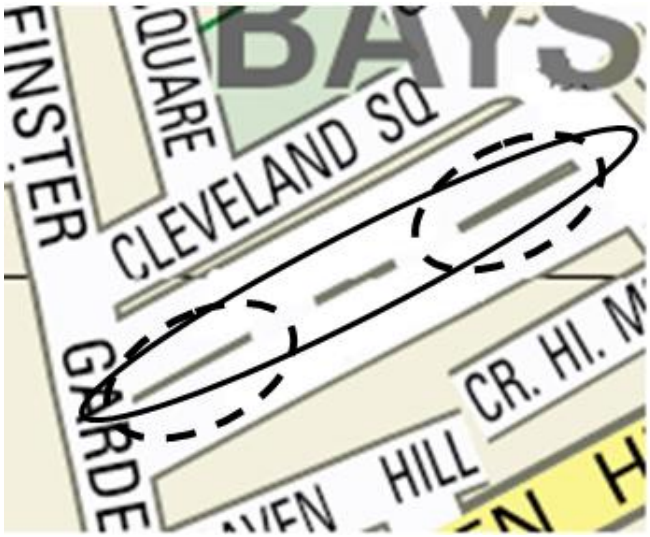

Figure 4.18: The three loop closures during the experiment

Outdoor Test 3

This test was carried out in order to confirm that the Extended RatSLAM model described in this report, alongside the new visual odometry algorithm, delivers accurate results when run on test cases used in the standard RatSLAM algorithm. The video sequence used, is a route around a suburb of St Lucia, in Brisbane, Australia and has a total length of 35 minutes. The video was captured from a laptop's built in camera, mounted on the roof of a car. This way it was ensured that the camera was always facing forward with respect to motion, and the movement of the camera had almost no transational movement parallel to the camera sensor plane [13].

It is important to say that some variables used in the Extended RatSLAM algorithm had to be changed. Most important changes, are the Pose cell size (table 4.12), and the empirically calculated variables σ, u_{cal} and u_{max} (table 4.14). The reason these variables have changed, is because during this video sequence, the speed of movement is larger than the one we previously set (table 4.11), since now the way of travelling through the route was via a car, and not on foot. For this reason, the Pose cell Size has increased as well, each cell now representing a 10 m $\times$ 10 m area in the pose plane.

After the completion of the test, we acquired some test data to evaluate the results. In figure 4.20, we show the real map of the St Lucia suburb, with the route taken shown in colour red.

During the test, 6807 Visual Templates where created, and 7406 experiences

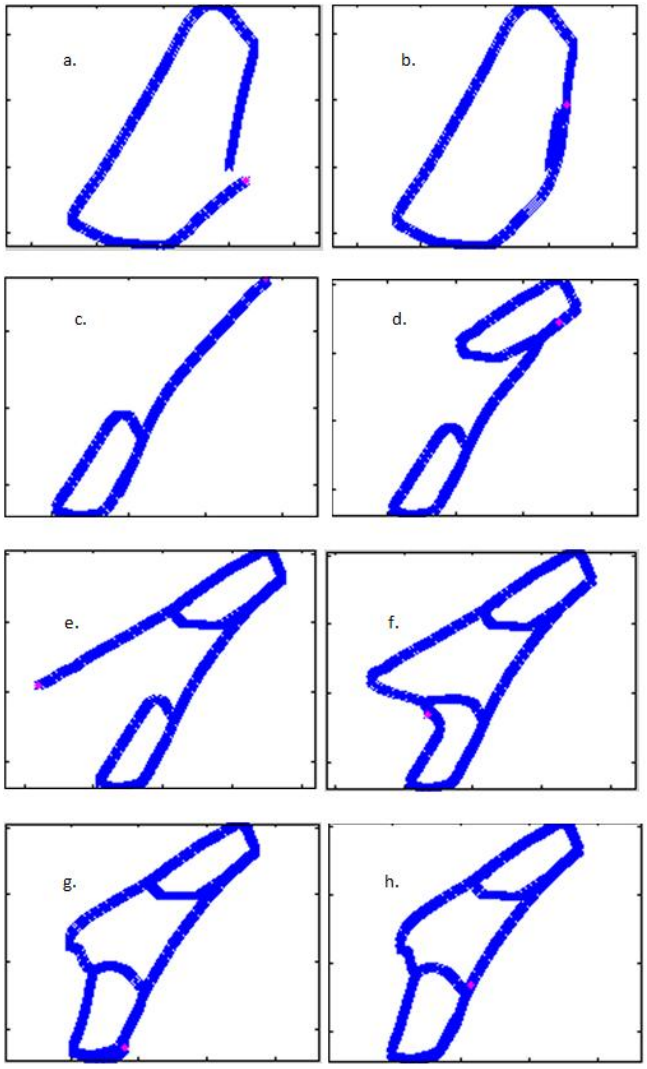

Figure 4.19: Experience map Creation and Correction during Outdoor Test 2

Pose cell Variables	
Pose cell size	10 m × 10 m × 10 deg

Table 4.12: Variable value assignments for the St Lucia experiment

alongside the corresponding transitions. All 12 loop closures where detected, and the map was corrected accordingly. The final experience map is shown in figure 4.21.

Using the experience map as a visual form of evidence, we can conclude that the algorithm explained in Chapter 3 can be successfully used as an extension

Local View cell Variables	
d_m	0.09

Table 4.13: Variable value assignments for the St Lucia experiment

Empirically calculated Variables	
σ	0.0828 degrees per pixel
u_{cal}	13.2 m/s
u_{max}	18.5 m/s

Table 4.14: Variable value assignments for the St Lucia experiment

Figure 4.20: Actual map of a St Lucia suburb, with the route taken shown in red

of the existing RatSLAM model, and still deliver accurate results on runs tested with the existing RatSLAM model.

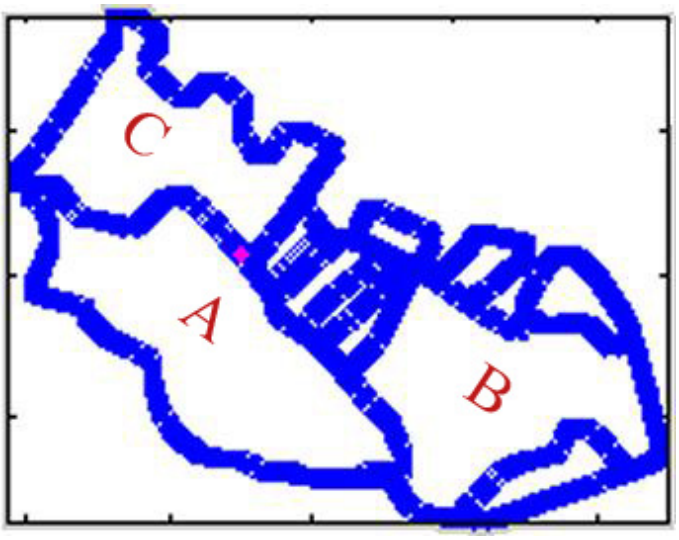

Figure 4.21: Final Experience map after the completion of the test run

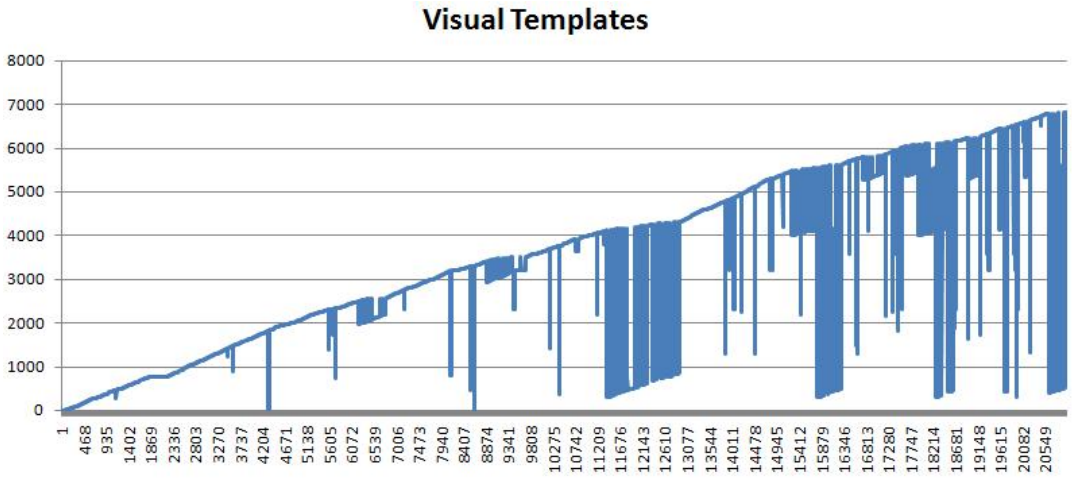

Figure 4.22: Visual template creation and recognition during all 20999 frames

Chapter 5

Conclusions and Further work

Developing a SLAM algorithm is one of the most challenging problems in mobile robotics. Existing algorithms, such as Extended Kalman Filters or Particle filtering using Monte Carlo localisation are purely mathematical implementations which heavily rely on probabilities. As a result a huge computational burden makes mapping large outdoor environments a hard problem. Expensive sensor receivers have also been used in combination with a probabilistic SLAM algorithm, to tackle this problem. Although sensory inputs provide very accurate data of the environment, running the algorithm in real time is impossible. Offline computations are made in order to provide reliable results.

A new approach for solving a SLAM problem has been recently introduced, where the algorithm does not make any use of statistics. It is brain inspired, and tries to simulate the way a brain navigates in space. It relies more on the concept of creating a mental map, like humans do, and recognise familiar sights by going back in time and remembering if a sight has been visited before.

RatSLAM is an algorithm developed by Michael Milford and Gordon Wyeth. It is a biologically inspired SLAM algorithm, that simulates an extended model of the rodents hippocampus. It has the strengths of both the probabilistic and biologically inspired systems. The algorithm was successfully used in large scale outdoor environments.It is proven that it can solve these demanding experiments by using simple vision sensors, like a web-cam, instead of expensive, high fidelity range sensors.

The purpose of this project was to implement an extended version of the existing RatSLAM algorithm, in order to provide personal localisation using a hand-held video camera. Existing work, has presented results where RatSLAM has accu-

rately localised robots, and created detailed maps of unknown environments. During the experiments though, it was clearly stated, that the platform holding the camera was fixed on the robot in a rigid position, so that the camera would always point at a fixed parallel plane to the ground (sensor plane), with minimum to none transitional movement compared to that plane, and facing forwards in respect to movement.

This project has implemented the core RatSLAM algorithm, and extended it's visual odometry algorithms, in order to eliminate the restrictions stated in previous implementations. Two methods have been used to solve this problem. An image stabilisation algorithm, and an algorithm based on pixel intensities. The first method has proven to be erroneous, the main reason being that the video sequence provided to the algorithm was pre-processed, thus reducing the quality of the video. The second algorithm, was basically doing a sort of image stabilisation, but within the RatSLAM model, without altering the video input frames.

Test results have shown that the extended RatSLAM algorithm worked well during experiments, by detecting all loop closures, providing an accurate map representation of the navigated area, and localising the person holding the camera in all times. In some experiments, loop closures where not detected immediately like they would in a probabilistic SLAM algorithm, the reason being that some times images where not recognised as already seen areas. Eventually the loop closures where detected, and correction algorithms eliminated any erroneous map representations of the navigated areas.

Although the algorithm is implementing SLAM, it was not tested in a real time environment. The lack of equipment and the required program, made it impossible to use the algorithm in real time. The purpose of this project though, was to provide an algorithm that implements personal SLAM using simple vision sensors. Although the videos where off-line, and stored on the computer harddisk, there was not any preprocessing made on those videos, and they where given to the algorithm the same way they where captured by the video camera. By having the right equipment, and installing the Image Acquisition Toolbox plug-in in Matlab, converting the existing code into a program that runs in real time is as easy as changing 10 lines of code.

5.1 Further work

Test results have shown that the extended RatSLAM algorithm used in this project is very promising. It can now simultaneously create maps of unknown environments, and localise both mobile robots, where camera shake is barely present , and a person walking, where camera shake is apparent throughout a video sequence.

Using this extended algorithm, possible future research and work can be:

- The whole project could be converted into a real-time algorithm, and be used in experiments with mobile robots by providing on-board SLAM capabilities.
- If used with a mobile robot, which has self-motion sensors, the feedback from those sensors can be used to provide an additional calibration on path integration, thus making the program even more reliable, and less prone to errors.
- An interesting idea, would be to extend this program, and develop algorithms, that would intelligently recognise if a robot is either driving, or walking. This might sound as science fiction at present, but imagine in future years, where humanoid robots have the ability to choose either driving a vehicle or exploring an area on foot. Using an intelligently designed RatSLAM algorithm, the robot would have SLAM capabilities whenever using either of it's two possible ways of transportation, each having a completely different motion pattern, thus feeding the algorithm with a different type of video sequences. Large outdoor environments, and indoor areas could be mapped by a single robot, using a more advanced RatSLAM algorithm.

Bibliography

[1] Probabilistic robotics. *robots.stanford.edu/probabilistic-robotics/ppt/slam.ppt*. [cited at p. 1]

[2] D.Redish A.Elga and D.Touretzky. A coupled attractor model of the rodent head direction system. *Netw.: Comput. Neural Syst.*, 7:671–685, 1996. [cited at p. 1]

[3] A. Arleo. Spatial learning and navigation in neuro-mimetic systems: Modeling the rat hippocampus. *Department of Computer Science. Lausanne: Swiss Federal Institute of Technology*, page 198, 2000. [cited at p. 7]

[4] B. Browning. Biologically plausible spatial navigation for a mobile robot. *Computer Science and Electrical Engineering. Brisbane: University of Queensland*, page 277, 2000. [cited at p. 7]

[5] A. Elga D. Redish and D. Touretzky. A coupled attractor model of the rodent head direction system. *Netw.: Comput. Neural Syst.*, 7:671–685, 1996. [cited at p. 4]

[6] M.J.Milford D.Prasser and G.Wyeth. Experience mapping: Producing spatially continuous environment representations using ratslam. *Australasian Conference on Robotics and Automation, Sydney, Australia*, 2005. [cited at p. 7, 19]

[7] M.J.Milford D.Prasser and G.Wyeth. Effect of representation size and visual ambiguity on ratslam system performance. *Australasian Conference on Robotics and Automation, Auckland, New Zealand*, 2006. [cited at p. 19, 37]

[8] M.J.Milford G.F.Wyeth and D.P.Prasser. Ratslam on the edge: Revealing a coherent representation from an overloaded rat brain. 2006. [cited at p. 2]

[9] D.Prasser G.Wyeth and M.J.Milford. Experiments in outdoor operation of ratslam. *Australasian Conference on Robotics and Automation, Canberra Australia*, 2004. [cited at p. 18, 43]

[10] M.J.Milford G.Wyeth and D.Prasser. Ratslam: A hippocampal model for simultaneous localization and mapping. *International Conference on Robotics and Automation*, 2004. [cited at p. 6]

[11] R.Schulz M.J.Milford D.Prasser G.Wyeth and J.Wiles. Learning spatial concepts from ratslam representations. *International Conference on Intelligent Robots and Systems, Beijing, China*, 2006. [cited at p. 11, 12]

[12] M. J. Milford. Featureless vehicle-based visual slam with a consumer camera. *Australasian Conference on Robotics and Automation, Brisbane, Australia*, 2007. [cited at p. 19]

[13] Michael J. Milford and Gordon F. Wyeth. Mapping a suburb with a single camera using a biologically inspired slam system. 2008. [cited at p. 2, 19, 48]

[14] Michael John Milford. *Robot Navigation from Nature*, volume 41, chapter 8. Springer, 2008. [cited at p. 9, 17]

[15] Yisheng Zhu Ping Shi and Shanbao Tong. Video stabilization in visual prosthetics. *Complex Medical Engineering, 2007. CME 2007*, 2007. [cited at p. 22]

[16] E. T. Rolls S. M. Stringer, T. P. Trappenberg and I. E. T. de Araujo. Self-organizing continuous attractor networks and path integration: One-dimensional models of head direction cells. *Netw.: Comput. Neural Syst.*, 13:217–242, 2002. [cited at p. 4]

[17] T. P. Trappenberg S. M. Stringer, E. T. Rolls and I. E. T. de Araujo. Self-organizing continuous attractor networks and path integration: two-dimensional models of place cells. *Netw.: Comput. Neural Syst.*, 13:429–446, 2002. [cited at p. 4]

[18] A. Samsonovich and B. L. McNaughton. Path integration and cognitive mapping in a continuous attractor neural network model. *J. Neurosci.*, 17:5900–5920, 1997. [cited at p. 7]

[19] K. Zhang. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: A theory. *J. Neurosci.*, 16:2112–2126, 1996. [cited at p. 5]

Appendices

Appendix A

Downloads

The following web-page includes a list of available downloads, of all tests used in Chapter 4. The video sequences show how the experience map is created and corrected during the test runs.
All videos are encoded with the *Xvid MPEG-4 codec*, an open source software (GNU GPL licence), which is available for download from this web-page: *http://www.xvid.org/Downloads.15.0.html*.

Web-page for available downloads:

- http://www.doc.ic.ac.uk/∼sk205/