Place Cell Inspired Indoor Self-Localisation using Convolutional Neural Networks

Group 1

Department of Bioengineering

Imperial College London

A Project Report Submitted in Partial Fulfilment of the

MEng —– Biomedical Engineering

Supervisor:
Professor Anil Anthony Bharath

Department of Bioengineering
a.bharath@ic.ac.uk

April, 2023

Place Cell Modelling

Keana Aitcheson, Linca Chuk, James Liu, Arinjay Mishra,
Adam Song, Denise Wong

April 13, 2023

Abstract

Spatial memory and navigation in mammals are performed by specialised neurons in the hippocampus called place cells. Their firing rate is proportional to the mammal's proximity to spatial landmarks, resulting in place fields. This work aims to create a model for visual self-localisation inspired by their efficient spatial encoding. The model should be computationally inexpensive in deployment and robust to environmental changes, to be used in tandem with traditional self-localisation techniques in indoor navigation systems of automation and robotics.

The model was designed to output 2-dimensional position and head direction estimates from image data. A SketchUp model was imported into the virtual environment Unreal Engine to generate synthetic image data. Lighting changes and data augmentation techniques including colour jitters and zoom were used to introduce domain randomisation and increase robustness.

A convolutional neural network (CNN) was selected as the basis of the model for its inspiration from biological visual processing systems. The model was trained for 108 epochs, yielding training and validation losses of 0.0337 and 0.0271 respectively. Visualisation of the weights of the convolutional layers showed strong orientation specificity, similar to the receptive fields of simple V1 cells. Analysis of the activations of the dense layers revealed certain neurons with strong location-dependency and direction independence, resembling place cell-like behaviour. The loss curves and biological symmetry demonstrates the model is learning well. However, the Euclidean RMSE of 4.46m indicates that further training, hyperparameter tuning, and an expanded dataset are necessary before the model becomes viable.

1 Introduction

1.1 Project Overview

Self-localisation is the process of determining one's position based on information extracted from the environment. In animals, place cells are specialised neurons in the hippocampus that are responsible for spatial memory and navigation, including self-localisation.

This project aims to create a model for self-localisation inspired by Place Cells using machine learning, specifically Deep Learning (DL), which can take an input image of a familiar environment and accurately regress the camera's position and orientation.

Simultaneous Localisation and Mapping Technology (SLAM) is a similar technology that currently exists, but has some limitations. It is computationally expensive, not robust to changes in the environment such as lighting, and requires detailed information about the location. In industry, systems are built with multiple redundant processes working in parallel such that if one fails, the other processes would correct the system. Ideally, these processes should be independent and have different, complementary strengths so the

system can function under various conditions. This model aims to leverage biological mechanisms to provide a greater degree of flexibility and to potentially overcome these limitations. This has a wide array of applications within indoor navigation and robotics, autonomous vehicles, and assistive devices [1]. Another trend is for engineering designs to have a *digital twin* [2], where a virtual model is created to predict performance and potential issues. The increasing prevalence of these virtual models means that virtual environments for data generation are more available, furthering the viability of a learning-based approach.

Previous work done by Rivera-Rubio et al. [3] reproduced the rate-coding effect in artificial Place Cells, which demonstrated the biological plausibility of decoding artificial place cell responses from tuning curves to regress one-dimensional location information. Thus, this paper seeks to extend this biologically inspired approach to two-dimensional space in an end-to-end fashion. Bloesch et al. [4] has also shown success in DL-approaches in learning implicit map representations of the environment. Together with the evidence of biologically plausible mechanisms for self-localisation, this paper hypothesises that an end-end DL model can be trained to leverage biological mechanisms to perform self-localisation, where Place-Cell-like behaviour may emerge in the encoding layers.

To reduce costs associated with manual data labelling, a virtual environment was proposed for the generation of training, validation and test data. This framework, utilising Unreal Engine, allows for a semi-automatic method to retrieve ground-truth data, so that a larger data set can be used during training. The framework also enables the altering of lighting conditions in order to i) test robustness to lighting changes in a reproducible way, and ii) train the network to be lighting invariant. The increased size and variety of the data aim to ensure a dataset more representative of real-world applications, increasing the resulting model's robustness and generalisation to new environments. This study's proposed approach of separating training and validation paths to obtain images also efficiently addresses the problem associated with testing within seen data, since independent views will be generated even at the same location. Additionally, for the assessment of biological mechanisms observed in the network, suitable visualisation methods may be used, such as the generation of artificial tuning curves to compare with the responses expected from neuroscience literature. For quantitative assessment of the model's ability to self-localise, the root mean squared error (RMSE) was computed for each coordinate, alongside the Euclidean error of each position estimate.

1.2 Main Objective

The specific objectives of the study were to:

- To create a virtual environment for the generation of synthetic image data for training and testing the model
- To train a Deep Convolutional Neural Network to regress spatial position and orientation.
- To extract biological behaviour in the convolution and flat encoding layers, comparing responses to responses reported from the neuroscience literature.

2 Background

2.1 Self-localisation in Mammals

Place cells and grid cells play a crucial role in self-localisation within mammals. Place cells provide the brain with spatial memory capacity, and Grid cells provide an internal co-ordinate system enabling the creation of spatial maps [5]. Together, these two types of cells allow for navigation and the recognition and recollection of surroundings.

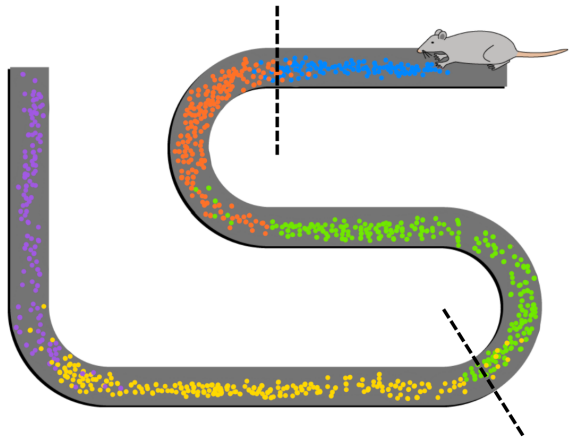
Other associated structures include head direction cells which encode directional information for the orientation of a mammal's head [6].

2.2 Significance of Place Cells

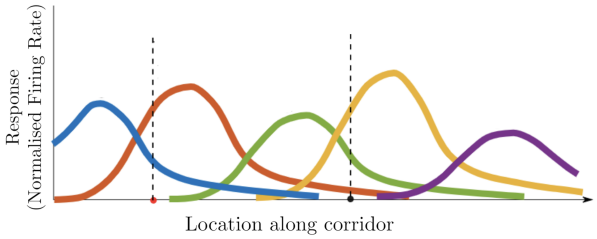
Hippocampal Place Cells are a multimodal class of pyramidal neurons. Rate coding describes the relationship between the intensity of a stimulus and the rate of firing of a neuron. It is hypothesised that information is encoded within neurons by this rate coding effect [7]. In Place Cells, their firing frequency and activation vary depending on environmental cues such as distance to landmarks, prominent edges, distal cues, and proximal cues. The closer the cue, the higher the frequency of firing.

Individual Place Cells have an effective "place field", a well-defined area of the environment to which a place cell responds [8]. Place cells encode the concept of location through their firing rate, which is a function of spatial position, illustrated in Fig. 2.2.1. The integration of multiple place fields gives rise to the concept of a "cognitive map", in which the mammal generates an implicit representation of their environment [5]. Characteristics of place fields

can change when subjected to a new experience or new context via remapping, which can be global (complete) or partial [9]. The plasticity exhibited by Place Cells can be paralleled with Online Learning, where a DL system continuously updates based on new observations [10]. This would allow a DL system to learn new environments automatically, but this remains outside the scope of the project.



(a) Visualisation of Place Fields in a Simple Corridor. Each coloured dot corresponds to the firing of a particular place cell. The density of the dots corresponds to the intensity of place cell firing. [11]



(b) The corresponding tuning curves of colour-coded Place Cells. The plot depicts how the firing rate varies as the rat traverses the corridor, entering and exiting their place fields. Adapted from Rivera-Rubio et al. [3]

Figure 2.2.1: Visualisation of Place Fields of Place Cells (a) and Corresponding Colour-Coded Place Cell Firing Rates (b)

2.3 Classical SLAM approaches

Classical Age SLAM approaches have had widespread success in industry, as outlined previously. Classical SLAM implementations are mainly variants based on Extended Kalman Filters (EKF-SLAM) [12], where a state vector comprising of landmark positions and their associated uncertainties is updated at each time step. The computational complexity for the algorithms, therefore, scales approximately quadratically with the number of landmarks. For complex scenes that require a large number of landmarks, this leads to an unacceptably large number of computations required. Alternative approaches employing sparser map representations have been developed, however, these are still not viable for resource constrained devices [1]. For example, real-time usage would not be possible on mobile devices.

Another pitfall of classical SLAM approaches highlighted in the review [1], is that they lack robustness to different lighting conditions. In particular, approaches using visual-bag-of-words techniques to represent image features have poor performance in dynamic lighting conditions since visual words cannot be recognised. They also suffer from perceptual aliasing, where visually similar locations generate similar image features, causing spurious loop closure, which in turn leads to poor estimations. These both arise because SLAM algorithms require manual tuning, where the handcrafted feature extractions may not be flexible enough to represent the high-level image and scene features. These issues can be addressed through using learned feature extraction layers found in Deep Neural Networks.

2.4 Place cell inspired self-localisation

The novel functionalities of Place Cells provide alternative ways of looking at computational approaches to self-localisation. The main inspiration for this research, however, was the 2015 study by Rivera-Rubio et al. [3] which investigated indoor self-localisation through the use of a Generalised Regression Neural Network (GRNN), that took the activations of Artificial Place Cells (APCs) as input. Their research showed the biological inspiration's validity and the efficacy of a regression-based neural network in achieving self-localisation with better efficiency and competitive accuracy than that of traditional alternatives (benchmarked against LSD-SLAM). However, Rivera-Rubio et al. [3]'s study was conducted in 1-dimension along a corridor, and required *a priori* allocation of APC place fields given knowledge of the environment being tested. Thus, this paper seeks to expand upon Rivera-Rubio et al. [3]'s research by achieving self-localisation in 2-dimensions, which further benefits from information on head-direction, and thus require a different method for ground-truth labelling of image data. Furthermore, handcrafted feature extraction algorithms were used to generate the APC responses, requiring manual tuning. In contrast, our approach seeks to learn these visual features by training a regression-based Convolutional Neural Network (CNN) in an end-to-end fashion.

2.5 Deep Learning

To attempt to tackle some of the issues outlined in Section 2.3, a learning-based approach may be proposed. More specifically, this study elected to utilise Deep Learning (DL), a paradigm where large volumes of data are used to train a neural network with many hidden layers. Kendall et al. [13] demonstrated that a deep CNN architecture was capable of competitive performance, with an indoor localisation accuracy of 0.5m and 10°, whilst running in real-time [1]. This approach heavily utilised transfer learning, by pretraining the network for object classification on the gold standard *ImageNet* and *Places* datasets, with no incentive to retain pose information. This allowed high-level spatial feature extractors to be automatically learned.

It is important to note that CNN architectures are biologically inspired, representing the computational analogue for biological visual systems, with well-learned kernels resembling biological receptive fields. This can be seen in Section 4. After replacing the final softmax layer from the CNN classifier with affine regressors and training on the desired location data, the network was then able to accurately regress pose directly from input images. This highlighted that high-level spatial features learnt were generalisable, and would be applicable to different scenes. Furthermore, as outlined in the survey [14], there is evidence that implicit map representations of the environment can be encoded into DL systems. Bloesch et al. [4] show that scene representations learnt via an autoencoder could be used successfully as a front end to a monocular SLAM system. This further supports our biologically inspired end-to-end approach to self-localisation.

Finally, a common criticism of DL methods is that they are computationally expensive. Though it is true that training large models can require large computational resources, once a model has been trained, it is possible to distill its knowledge into a much smaller model [15]. This means that at inference, the smaller student model can be used to run on far fewer resources, whilst retaining the generalisation power of the original model. As such, knowledge distillation provides a viable option for the efficient implementation of DL systems. Another potential option is Neuromorphic Architectures. These technologies provide a low-power, massively parallel computing paradigm that complements the simulation of neural networks [16]. They also open opportunities to enable online or continual learning, which could potentially enable learning-based self-localisation to extend to unknown environments.

[1] Each pose calculation took 5ms.

3 Methods

In this section, the creation of the virtual environment and convolutional neural network, hyperparameter turning and artificial place cell analysis are discussed.

3.1 Virtual Environment

3.1.1 SketchUp and Model Creation

A virtual environment was created using SketchUp. The environment includes a living room, dining room, pantry, bar, kitchen, two bedrooms, two bathrooms, a walk-in closet, and a patio all of which are furnished using models from the SketchUp 3D warehouse. This was done to ensure the space included a variety of material textures, colours, and shapes to encourage more robust visual features to be learnt. The furniture models additionally serve as complex visual landmarks for the neural network to potentially learn.

3.1.2 Unreal Engine

The completed virtual apartment was imported into Unreal Engine. Seven paths were created, with one path reserved for testing, along which the camera would navigate in both the forward and reverse directions, capturing the images and their corresponding (x, y, θ) coordinates at 30 frames per second. To generate the reverse direction, the camera is rotated 180 degrees and moves along the same path as the forward direction.

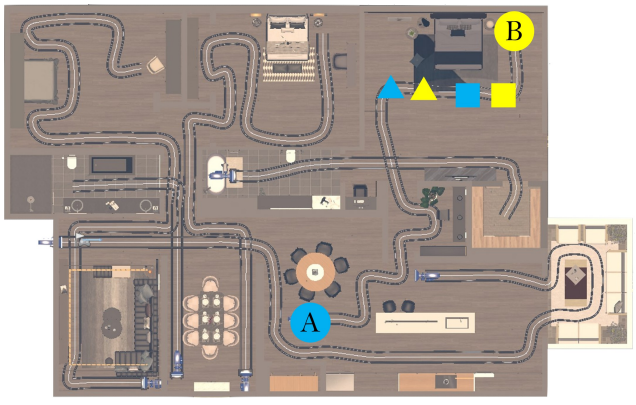


Figure 3.1.1: Floor plan of the virtual environment. Note the coloured markers on the diagram related to Fig. 4.2.2 The blue markers denote the start of a section, and the yellow markers denote the end of a section. The circles mark the limits of "Path 6" which goes from A to B, while the triangles and squares denote the endpoints of the neuron activation described in Fig. 4.2.2

Images were generated under two distinct lighting conditions: i) artificial lighting conditions with no shadows and ii) hyperrealistic lighting conditions using ray-tracing, and they are contrasted in figure 3.1.2. The more realistic lighting conditions were created based on the furniture present in the house. For example, fire-like light was created within the fireplace. The colour temperatures were based on Lighting Design Studio's colour temperature guide [17]. In total, approximately 30,000 images were produced.

3.2 Convolutional Neural Network

3.2.1 Tools for Building the Model

Python was chosen due to its object oriented nature and a wide array of industry standard libraries. In particular, the TensorFlow platform was used since the Keras API allowed for precise data and model manipulation in a high-level fashion.

3.2.2 Pre-processing Images and Data Augmentation

The images were split into training, validation, and testing sets. The validation set was obtained by sampling the training set and removing 1 in 7 images, and the testing set comprised of images from the reserved path as outlined in section 3.1.2 The images were loaded into Pandas dataframes alongside their accompanying ground-truth labels of coordinates. The x, y, and θ coordinates were then normalised between 0 and 1. These dataframes were then passed into the ImageDataGenerator object in Tensorflow due to its efficiency in passing inputs in batches to the neural network without the need to load all images into RAM, as well as in its ability to apply random transformations for data augmentation of the training set. The pixel intensities of all images were normalised, rescaling them by a factor of 255.

For the training set, the data augmentation techniques selected were random rotations, zoom, horizontal flips, variations of image width and height, as well as colour perturbations of brightness, saturation, hue, and contrast. Additionally, training images were shuffled so they no longer occurred along the fixed paths. This helped increase the diversity of the training set to improve robustness and prevent overfitting of the network to the training data. These random transformations were conducted with a set 'seed' of 42 for reproducibility across hyperparameter tuning and trials.

3.2.3 Convolutional Neural Network Architecture

A 3-channel (RGB) 320 by 180 pixels input layer for a colour image was initialised, followed by the first convolutional layer that uses 16 10x10 pixels filters to convolve with the input image. Zero padding and the default stride of (1, 1) ensured that the output of this layer was of the same size as the input image. The Rectified Linear Unit (ReLU) was chosen as the activation function to introduce non-linearity [2]. The advantage of ReLU over other non-linear activation functions such as Sigmoid, is that it avoids the vanishing gradient problem, where gradients approximately equal to zero causing weight updates via backpropagation to virtually halt. The first pooling layer was then added, which downsamples the input image by shifting a 2 by 2 window over the input image and taking the maximum. Subsequently, the first dropout layer was added, which helps prevent overfitting by setting random outputs from the previous layer to 0 with a probability of 0.4 during training. Note that dropout layers are removed during inference.

The architecture then consisted of 3 repeated stacks of a 3x3 convolution layer with ReLU activation, pooling layer and a dropout layer.The first 3 convolutional layers had 16 filters, while the fourth had 32. The architecture as a whole is depicted in Fig. 3.2.1. By reducing the size of the filters, the number of units in each layer (width) and the number of layers (depth) could be increased, thus increasing the representational power of the network [19]. As per the usual CNN formulation, the feature maps were then flattened into a 1-dimensional array, and passed through 6 dense layers with 256, 128, 32, 16, 8 and 3 neurons sequentially. Place cell behaviour was expected to emerge within the third to fifth dense layers. These three dense layers were used to decode the high-level features into (x, y, θ) coordinates, which were outputted by the last dense layer.

3.2.4 Training and Validation of the Model

Model training can be formulated as an optimisation task, aiming to find the weights, $\boldsymbol{\theta}$, that minimise the loss function, L, on a training set, $\boldsymbol{y}$:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(\boldsymbol{\theta}))^2, \tag{1}$$

$$\boldsymbol{\theta}_{opt} = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \tag{2}$$

In this model, the Mean Squared Error (MSE) was used as the loss function, which is standard for regressors. Since analytical solutions may not exist, the gradient

[2] Non-linearities are required for neural networks to act as universal approximators [18]

(a) Artificial Lighting with No Shadows

(b) Ray-Traced Lighting with Shadows

Figure 3.1.2: Comparison of two lighting conditions

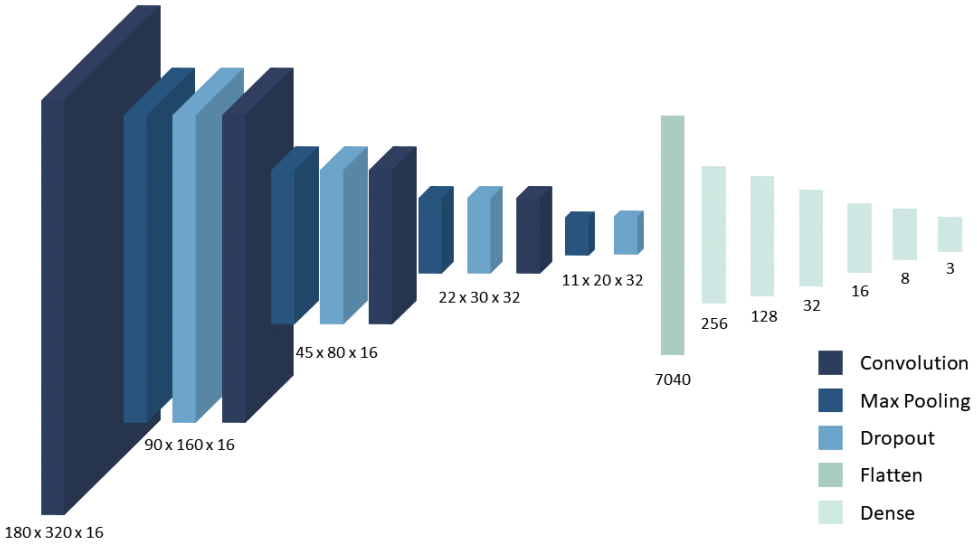


Figure 3.2.1: Architecture of CNN Model Visualised

descent (GD) methods were utilized, where weights are iteratively updated towards a minima. These are of the form

$$\theta_j = \theta_j - \alpha \frac{\partial L}{\partial \theta_j}, \tag{3}$$

where α is the learning rate. Backpropagation is used to calculate these gradients, which exploits the chain rule to prevent unnecessary recalculations. When integrated with the TensorFlow *.fit()* method, the ADAM [20] optimisation algorithm was used to perform stochastic gradient descent (SGD) (See Appendix 7.3). A mini-batch size of 64 was used.

At the end of each training epoch, the MSE was calculated on the training and validation datasets. This was repeated for a maximum of 200 epochs, and with a patience of 20. This meant that if the validation loss did not improve for 20 consecutive epochs, training was halted to prevent overfitting the model.

Lastly, to visualise the learning progress and monitor whether the model was overfitting or underfitting, learning curves depicting training and validation loss against the number of epochs were plotted. Overfitting and underfitting arise from the bias-variance tradeoff (See Appendix 7.4). A large gap between validation and training loss suggests overfitting, whereas if both losses are high but similar in magnitude this may indicate underfitting.

3.3 Hyperparameter tuning

Model hyper-parameters, such as convolutional filter size, dense layer architecture, regularizer values,

dropout layer values, were tuned to optimise model performance. This was carried out by a combination of manual alterations and automated tuning via the KerasTuner API. Optimal models were those that minimised the validation loss after training. The number of dense layers after the convolutional layers, the number of neurons in those dense layers and the mini-batch size were manually varied. This led to the chosen architecture in Section 3.2.3. The optimal maximum learning rate for ADAM optimiser was found to be 0.01 after using the KerasTuner Hyperband algorithm. This algorithm uses a tournament-style bracketing system where models are trained for a few epochs and only the top-performing models are selected to progress to the next round.

3.4 Artificial Place Cell Analysis

To determine whether the model is producing results analogous to human vision, the weights and activations of the CNN convolutional and dense layers were extracted. In the convolutional layers, the visualisation filters should show similar patterns to biological visual receptor fields. In the dense layers, the activation rates should show Place-Cell-like tuning curves, with activation at zero until a place field is entered.

4 Results

4.1 Model Performance

4.1.1 Training and Validation

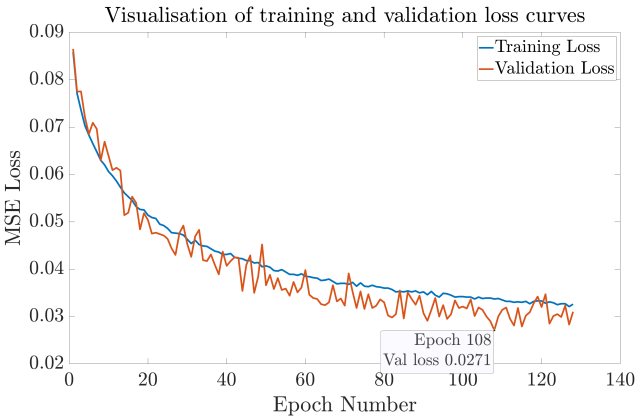


Figure 4.1.1: Training and Validation Losses. Training stopped at 108 epochs after early-stopping with a patience of 20.

Training and validation losses were extracted for each epoch in order to plot learning curves and assess how the model is learning. The training halted at epoch 108 as validation loss did not decrease for the next 20 epochs. At this epoch, the training loss was found to be **0.0337** while the validation loss was **0.0271**. These values are similar in magnitude. This can also be seen in Fig. 4.1.1, where the generalisation gap between training and validation loss is small there is a monotonic decrease in loss values as the number of epochs undergone increases. There is also a degree of stochasticity in the validation loss, which may be expected from the mini-batching.

4.1.2 Evaluation of Performance

This fitted model was then applied to the unseen test set for evaluation. A single pole recursive filter was applied for smoothing position estimates. Plots contrasting the smoothed predictions of x and y against frame number are depicted in Fig. 4.1.2.

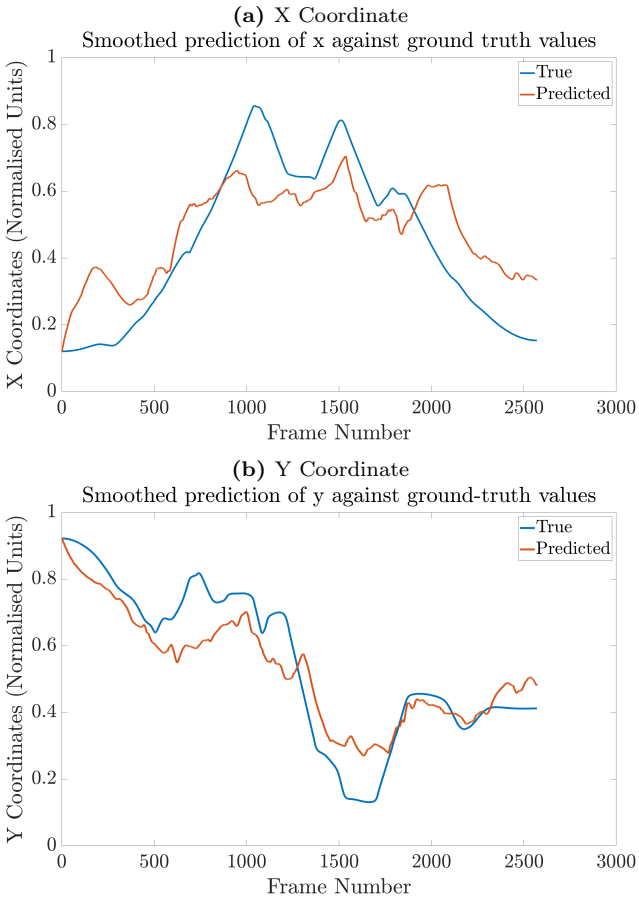


Figure 4.1.2: Prediction and Ground-Truth Value for Each Spatial Coordinate against Frame Number (Time)

The MSE, RMSE and Median Absolute Error (MedAE) values for each coordinate are summarised in table 4.1.1. The difference between the RMSE and MedAE values for each coordinate are indicative of a highly skewed distribution of errors.

Coordinate	x	y	θ
MSE (units^2)	0.0196	0.0091	0.1420
RMSE	3.44m	2.82m	135.67°
Median Absolute Error	2.56m	2.64m	81.96°

Table 4.1.1: Table summarising errors of the model evaluated on the test set

To visualise the distribution of errors, histograms of the squared error for each coordinate were plotted and they all yielded highly similar shapes. The histogram for x is depicted in Fig. 4.1.3. With the squared errors not exceeding 0.2 and the majority of the errors being concentrated towards the smaller values, the histogram clearly outlines the skewed distribution of errors in coordinate estimates.

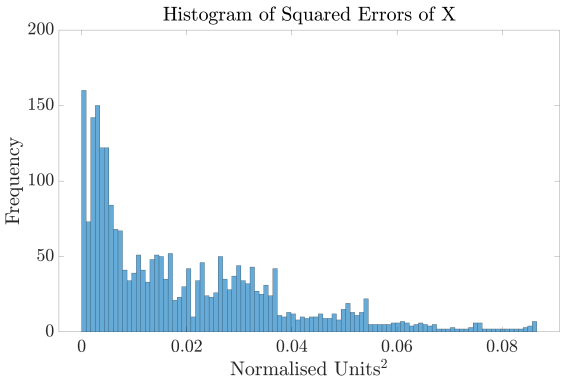
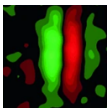


Figure 4.1.3: Histogram of Squared Errors of X Coordinate in Normalised Units^2

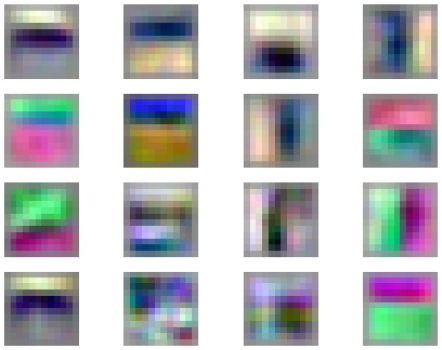
Finally, the squared errors of x and y were summed to calculate the Euclidean error of the position estimates, and the RMSE of the Euclidean Error was found to be **4.46m**, while the MedAE of the Euclidean error was **3.44m**.

4.2 Investigating Biological Symmetry

4.2.1 Visualisation of learned weights and activations



(a) Biological V1 Receptive Field, with red regions being excitatory and green regions being inhibitory [21].



(b) Layer 1 Convolutional Weights

Figure 4.2.1: Visualisation comparing a biological receptive field and the first convolutional layer weights.

4.2.2 Artificial Place Cell Behaviour in Network

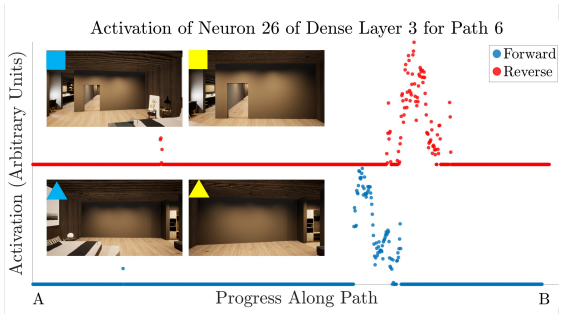


Figure 4.2.2: Activation of Dense Layer 3 Neuron 26 along path 6 in the forward and reverse directions. The marked shapes correspond to images taken at positions denoted by the same shapes found in Fig. 3.1.1. Larger versions of the images can be found in Fig. 7.5.1 in the Appendix. the forward and reverse paths move along the same tracks, with the reverse path's camera rotated 180 degrees. In the forward direction, the neuron activates at approximately 580 to 630 frames, and along the reverse path at approximately 660 to 720 frames. This suggests neuron 26 is responding to some stimuli present in the area represented by the frames. Visually the frames are quite different. This suggests the cell is not responding to specific geometry, texture, or lighting within the frames, but instead responding to some spatial mapping encoded within the neural network.

5 Discussion

The three main aims of this project were to i) create a virtual environment for training and testing sets ii) train a CNN to accurately regress camera pose, and iii) extract evidence for biological mechanisms governing the model's behaviour. There was strong evidence to suggest biological mechanisms were present in the trained network and limited success was also achieved apropos the model's accuracy.

5.1 Contrasting with Biological Analogues

5.1.1 Receptive Fields and Convolutional Filters

The filter masks of the first convolutional layer were visualised and compared with their biological analogue: the receptive fields of the simple V1 cells of the Primary Visual Cortex.

Visualising the filter weights shows promising similarities to the visual receptive fields of biological systems seen in neuroscience literature. This is highlighted in the types of feature extractors being learnt, with some filters showing clear orientation specificity while others demonstrate corner detection masks. This suggests that the colour jittering and data augmentation techniques used have succeeded in producing general spatial features. The resemblance of the learned weights to biological receptive fields also demonstrates that the proposed CNN architecture was successful in mimicking the organisation of the primary visual cortex. These spatial features should then generalise well to new domains and could be used for transfer learning to learn new environments [13].

5.1.2 Place Cell Behaviour in Network

The activations of the neurons of the model's dense layers were plotted along each of the paths, and Place-Cell-like behaviour was observed in the third dense layer. An example of cell activations of this layer along path 3 in the reverse direction was shown in Fig. 7.5.3 which can be found in Appendix section 7.4. Fig. 7.5.3 does not correspond exactly to the ideal tuning curves seen in neuroscience literature. From a biological perspective, this is to be expected as cells in the primary visual context do not activate in the presence of all stimuli. Furthermore, due to the higher order visual processing functionality of the Hippocampus, Hippocampal neurons have highly selective firing patterns, and not all neurons would exhibit Place-Cell-like activity. From a deep learning perspective, this is also to be expected since there are no explicit constraints on the network to learn only Place Cell activation patterns.

However, there are neurons that display clear Place-Cell-like behaviour. For example, Neuron 26 in the third dense layer only activates in paths 2 and 6 in both the forward and backward directions. The graphs of the neuron activation along path 6 in the forward and reverse direction are shown in the Fig. 4.2.2 with the positions in the floor plan marked in Fig. 3.1.1. The activations are shown to be in front of the bed, with the individual frames being visually distinct. This suggests the artificial neuron is not simply responding to some geometric, material, or lighting stimulus but instead activating based on encoded spatial stimulus. While not all the neurons are behaving like artificial Place Cells, there are neurons exhibiting the hypothesised behaviour demonstrating the viability of this training method in learning biological behaviour.

5.2 Evaluation of Accuracy

The shape and monotonic decrease in the loss curves in Fig. 4.1.1 with increasing epoch number, alongside the clear biological behaviour of the network, is indicative that the model is learning the desired behaviour. Additionally, the validation loss of 0.0271 on the unseen validation set is indicative that the model is able to generalise well, and is able to perform well on distinct lighting conditions.

However, the validation loss being non-negligibly lower than the training loss suggested that the model was underfitting. Underfitting was also suggested in the comparison between smoothed predictions of x and y against their ground-truth values in 4.1.2; the peaks of the predicted values were unable to reach the full range of the ground-truth values. The model would therefore benefit from a higher complexity, with more dense layers and neurons in those layers, since the visual features were already well learned. Increasing complexity would thus provide a solution to the current bottleneck present in the architecture, with the dense layer of 256 neurons unable to encode the representations parsed from the previous flattened layer of 8192 neurons. This study hypothesised that this could provide a means for more Place-Cell-like neurons to be present, and less constantly active neurons. It is however important to note that previous attempts utilising DL for self-localisation had huge networks, 22 layers deep [13], and took advantage of huge datasets for pretraining to attain reasonable performance. This would require significantly stronger GPUs and more time to train, making it unfeasible to attempt within the constraints of this project.

When evaluated on the test set the RMSE values for

each coordinate as shown in Table 4.1.1 and the Euclidean RMSE of 4.46m indicate that the model was significantly underperforming in accuracy metrics. It is thus apparent the model requires further training and hyperparameter tuning before it is suitable for deployment. The aforementioned need to increase model complexity is also likely to improve the model's performance on the test set.

Contrasting the RMSE values with the significantly smaller Median Absolute Error of 3.44m, it was clear that the spread of errors were non-uniform. The histogram of squared errors, Fig. 4.1.3, was effective in depicting a clear skew of the distribution of errors, suggesting that high-error outliers contributed to the bulk of the MSE value. Further analysis of the high-error outliers is essential in ascertaining rectifications to be made to the training methodology.

Finally, the histograms of errors and RMSE comparisons demonstrated that the predictions of the head-direction θ were not as accurate as the position estimates of x and y. Although, this may in part be explained by how small variations in head direction yield minute changes in perception of one's surroundings and thus the corresponding images created, further architecture alterations need to be made to improve orientation accuracy. This paper hypothesises that investigating Head-Direction-Cell-like activity may improve the model's ability to predict θ.

6 Conclusion

6.1 Summary

This paper aimed to develop a model inspired by the biological mechanism of place cells to achieve computationally efficient indoor self-localisation robust to lighting changes and random artefacts, to work in tandem with current traditional algorithms. A model of a household was designed on SketchUp and imported into the Unreal Engine 5 virtual environment for generation of synthetic image data. This study was successful in creating a novel environment with 2 distinct and realistic lighting conditions, in addition to constructing the framework for the automatic export of ground-truth labelled image data along the desired pathways. Approximately 30,000 images were extracted.

The model selected was a regression-based CNN, that was fitted and evaluated using the synthetic image datasets. Data augmentation techniques such as colour jitters were further utilised to increase the robustness of the model to environmental changes and reduce overfitting. Hyperparameter tuning of convolutional filter size, pooling size and stride were conducted alongside multiple architectures and dropout layers. The ADAM optimiser was selected for its adaptive learning rate. Plots of the training and validation loss displayed a monotonal decrease with epochs and a small generalisation gap.

Visualisation of the weights of the first convolutional layer showed strong orientation specificity, similar to the simple V1 cells of biological systems. Analysis of the dense layers of the CNN and revealed certain neurons displaying prominent location dependency in their activations, which were further determined to be direction independent.

Although analysis of the loss curves and compelling biological behaviour is indicative that the model was training in an appropriate manner, the high Euclidean RMSE of 4.46m indicates that the model requires further training and hyperparameter tuning prior to deployment. Comparisons of MedAE and RMSE were further emblematic of a skewed error distribution, and analysis of the high-error outliers is necessary in ascertaining improvements in training methodology.

6.2 Limitations and Future Work

The biggest limitation of this paper was the protracted durations that training required. As outlined earlier, although the proposed method of self-localisation is computationally efficient in deployment, the training process of the CNN is highly intensive. Given a longer timeframe for training, hyperparameter tuning could be much more thoroughly conducted, finding the ideal architecture for our use case and allowing for more accurate predictive capacity. This could significantly improve the accuracy of the model. L2 regularizers and batch normalisation, coupled with more epochs and greater patience, could reduce increase the predictive capacity of the model [22]. Learning rate scheduling could also be applied to dynamically vary learning rate as the loss function plateaus over time [23].

Furthermore, rendering and exporting images from Unreal Engine 5 was computationally intensive due to the ray-tracing technologies necessary to create realistic environments that are able to bridge the simulation-to-reality (sim2real) gap. Moreover, with more time, more domain randomisation techniques could be implemented such as more variation of the lighting conditions, furniture placement, and material textures, to further close the sim2real gap [24]. Additionally, this paper was limited to training with a single virtual environment, limiting discussion on its performance in learning and adapting to new environments, where transfer learning would be used to 'freeze' the weights of the convolutional layers whilst the flat layers would be retrained to the new

environment.

Finally, timeframe constraints disallowed for testing of the deployment scenario of the CNN as an embedded onboard device, wherein compaction of the CNN is necessary for the low-power use-case. Compaction can be achieved by reducing the precision of the floating point operations per interface (FLOPs), and reducing model parameters using a method known as pruning, wherein neurons with activations consistently below a threshold value are considered "dead" and removed [25].

7 Appendix

7.1 Project Management Summary

After completing initial preliminary research, the workload comprised two main tasks: building the virtual environment and developing the CNN architecture, necessitating a split into two teams. Each team worked independently but had weekly meetings to provide progress updates, reallocate manpower as necessary, and reassess the aims of the following week. Minutes were taken consistently and individual responsibilities were outlined to maintain accountability throughout. The next phase of model evaluation and hyperparameter tuning saw the teams recombine, with each member responsible for investigating a specific parameter. We further maintained a Gantt Chart (Fig. 7.5.2)to track our project schedule.

However, we were forced to depart from our project plan due to local hardware constraints from the high GPU requirements of rendering high-quality images in UE5. This could only be conducted on one laptop and could not be done remotely, making it difficult to collaboratively extract images within the team.

Additionally, due to the large size of our network and datasets, training durations were long and although Google Colab Plus allowed greater runtime, disconnections were a frequent issue, making training for consecutive days unfeasible. This was solved by using a Juypter notebook locally on GPU instead, making real-time collaboration and version control difficult.

7.2 Project Management Lessons Learned

1. Effective Communication

To adhere to deadlines both internal and external, we held weekly meetings and kept minutes to establish clear dissemination of information and ensure minimal miscommunication. Ideas were consolidated on OneNote, coding notebooks were shared for suggestions, and relevant papers were compiled on Mendeley. We created an open and collaborative team culture that encouraged members to speak up freely and proactively for discussion, be it in person or online. Voicing of alternate viewpoints was encouraged to facilitate opportunities for independent thought and a greater depth of understanding for all. This atmosphere of cooperation veritably shone through in this computationally heavy project, where a fresh set of eyes could find errors and opportunities for increased efficiency otherwise missed. Our supervisor similarly championed the tenets of candid discourse, always willing to passionately share his expertise while engendering a desire to conduct our own research.

2. Importance of Contingency Planning

This project demonstrated clearly the value of having well-defined and clearly laid out contingency plans when undertaking complex tasks. We had comprehensive contingency plans in place for both the virtual environment and the CNN model. Had Unreal Engine been unable to run effectively on our hardware, its support pulled, or collaboration became too difficult, we would switch to Unity, having already researched packages suitable for our needs. Had the front-end of our model not exhibited biological behaviour and effective convolutional masks, we would pivot to using the pre-trained AlexNet network, still being able to utilise the highly limited time we had available for training our model.

Similarly, if Google Colab proved incapable of handling the protracted training durations, we had sourced hardware able to locally train using a Jupyter Notebook. This contingency plan was implemented into our project, allowing us to complete all planned iterations of the model punctually.

3. Strategic Time Management

Our project was complex and involved multiple steps that needed to be completed within a limited time frame. To ensure that we made the most efficient use of our time, we prioritised our focus on completing tasks critical for the next steps. Additionally, we split into two teams to work simultaneously on different parts of the project to better tackle the many tasks. We also kept in mind the deadlines for the several assignments that were part of the project and allocated sufficient time to complete them. By prioritising tasks and managing our time efficiently, we were able to somewhat stick to our schedule and achieve our objectives within the project's time constraints.

7.3 Stochastic Optimisation Methods

SGD is a variant of vanilla GD, where instead of using the entire training set to compute gradients, only one training example is used at each update. This can lead to faster convergence as parameters are updated more frequently, but the updates will be noisier. However, over sufficient iterations, SGD will converge by minimising the expected loss function, and the noisy updates may improve generalisation by avoiding local minima. Often in practice, instead of using only single training examples, *mini-batches* are used. By using multiple samples from the training set, the computational efficiency is further increased by taking advantage of vectorisation.

The ADAM algorithm updates differ from Eq. 3 in that they also incorporate the concept of *momentum*. This allows previous updates to influence future updates, and provides the mechanism for the adaptive learning rate. For example, repeated updates in a direction in parameter space will compound and exhibit inertia, increasing step sizes and accelerating updates towards the minima, hence, convergence is attained more quickly. Therefore, ADAM has better and faster convergence than other stochastic optimisation methods [20].

7.4 Bias-Variance Tradeoff

When using MSE as the benchmark for performance, the following can be shown,

$$MSE = Bias^2 + Variance, \tag{4}$$

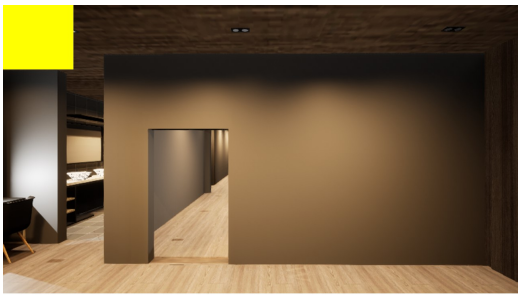
where the error can be decomposed into the bias and variance of the estimated parameters. When minimising the MSE, it is difficult to simultaneously minimise both bias and variance. As a result, some models may have high bias, where the model lacks sufficient complexity to represent the data. This would show in learning curves with training and validation loss having similar but large magnitudes, indicating underfitting. Models with high variance may fit the training data well, but would actually be fitting to noise, and therefore fail to generalise well to unseen or validation data, corresponding to overfitting. This would result in a higher loss on the validation curve than the training curve.

7.5 Additional Figures and Context

(a) First frame of Dense Layer 3 Neuron 26 activation along path 6 in the forward direction. The blue triangle marking corresponds to the marks found in Fig. 3.1.1 and Fig. 4.2.2

(b) Final frame of Dense Layer 3 Neuron 26 activation along path 6 in the forward direction. The yellow triangle marking corresponds to the marks found in Fig. 3.1.1 and Fig. 4.2.2

(c) First frame of Dense Layer 3 Neuron 26 activation along path 6 in the reverse direction. The blue square marking corresponds to the marks found in Fig. 3.1.1 and Fig. 4.2.2



(d) Final frame of Dense Layer 3 Neuron 26 activation along path 6 in the reverse direction. The yellow square marking corresponds to the marks found in Fig. 3.1.1 and Fig. 4.2.2

Figure 7.5.1: Larger images of the first and final frames of corresponding to the activation of Dense Layer 3 Neuron 26

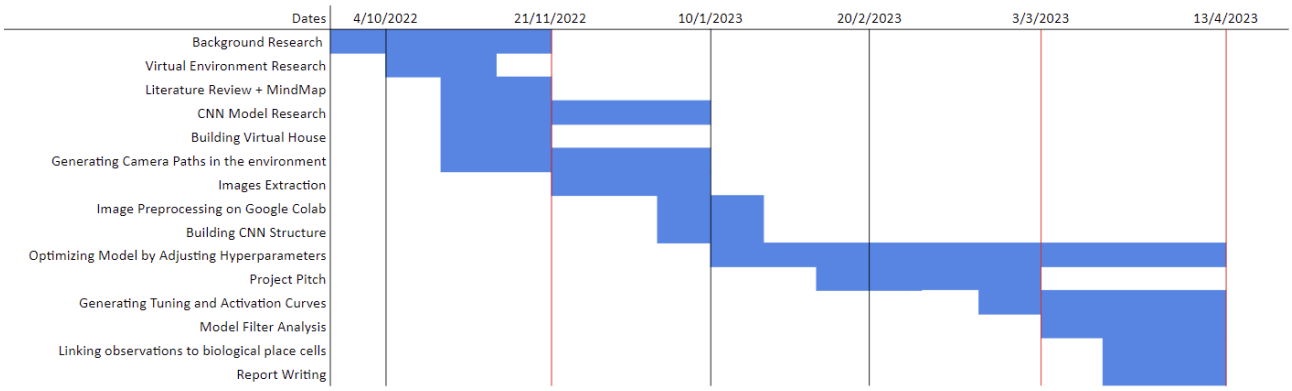


Figure 7.5.2: Gantt Chart

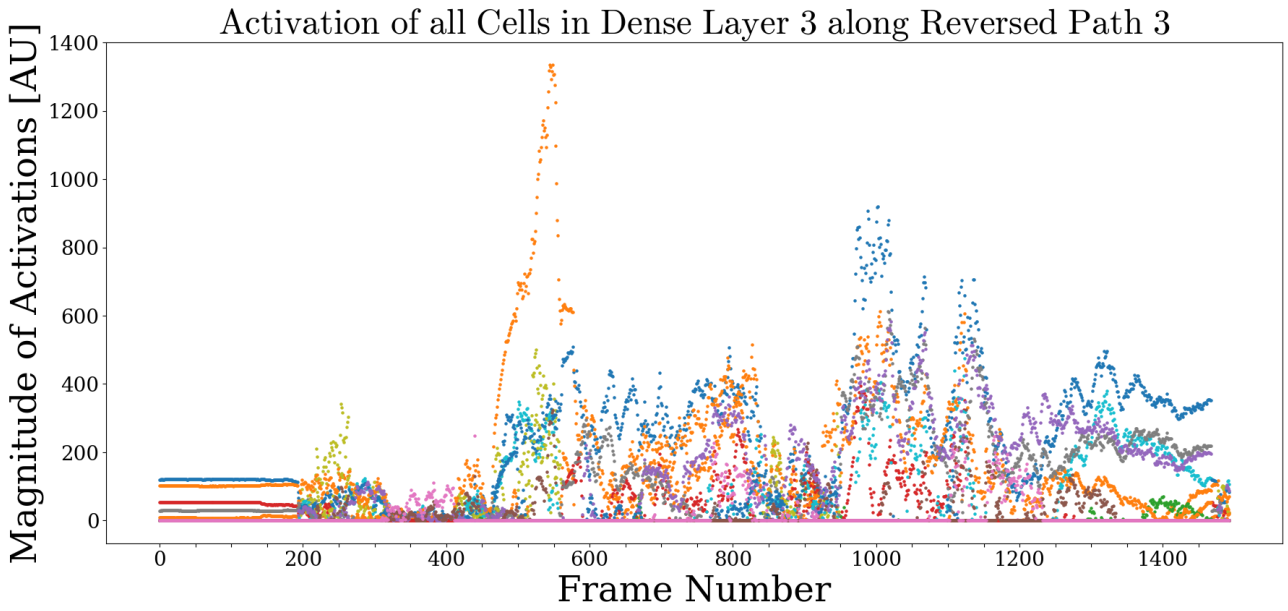


Figure 7.5.3: This graphic depicts the activation of all neurons in dense layer 2 along the reversed path 3. Clearly this layer has non-place cell behaviour. Note the overlapping activations showing Receptor Field-like response of all neurons from dense layer 3 along path 3 in the reverse direction. Note between frames 0 and 180, the camera faces a wall resulting in constant activation values. Other cells follow this pattern, suggesting that the lack of visual features impaired the model's ability to self-localise.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] "Digital twins for the built environment." [Online]. Available: https://www.iesve.com/digital-twins

[3] J. Rivera-Rubio, I. Alexiou, and A. Bharath, "Indoor localisation with regression networks and place cell models," in *Proceedings of the British Machine Vision Conference (BMVC)*, X. Xie, M. W. Jones, and G. K. L. Tam, Eds. BMVA Press, September 2015, pp. 147.1–147.12. [Online]. Available: https://dx.doi.org/10.5244/C.29.147

[4] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2560–2568.

[5] J. O'Keefe and J. Dostrovsky, "The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat." *Brain research*, 1971.

[6] T. Strösslin, D. Sheynikhovich, R. Chavarriaga, and W. Gerstner, "Robust self-localisation and navigation based on hippocampal place cells," *Neural networks*, vol. 18, no. 9, pp. 1125–1140, 2005.

[7] E. D. Adrian, "The basis of sensation." 1928.

[8] R. M. Grieves, S. Jedidi-Ayoub, K. Mishchanchuk, A. Liu, S. Renaudineau, and K. J. Jeffery, "The place-cell representation of volumetric space in rats," *Nature communications*, vol. 11, no. 1, p. 789, 2020.

[9] K. J. Jeffery, "Integration of the sensory inputs to place cells: what, where, why, and how?" *Hippocampus*, vol. 17, no. 9, pp. 775–785, 2007.

[10] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information Fusion*, vol. 58, pp. 52–68, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519307377

[11] "Place cell - Wikipedia — en.wikipedia.org," https://en.wikipedia.org/wiki/Place_cell, [Accessed 11-Apr-2023].

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[13] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[14] C. Chen, B. Wang, C. X. Lu, N. Trigoni, and A. Markham, "A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence," *arXiv preprint arXiv:2006.12567*, 2020.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[16] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[17] L. D. Studio, "Color Temperature," https://lightingdesignstudio.co.uk/color-temperature, 2021, accessed 11-Apr-2023.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] T. Lindeberg, "A computational theory of visual receptive fields," *Biological cybernetics*, vol. 107, pp. 589–635, 2013.

[22] T. van Laarhoven, "L2 regularization versus batch and weight normalization," 2017.

[23] Z. Xu, A. M. Dai, J. Kemp, and L. Metz, "Learning an adaptive learning rate schedule,"

CoRR, vol. abs/1909.09712, 2019. [Online]. Available: http://arxiv.org/abs/1909.09712

[24] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[25] W. Ahmed, A. Zunino, P. Morerio, and V. Murino, "Compact cnn structure learning by knowledge distillation," 2021.