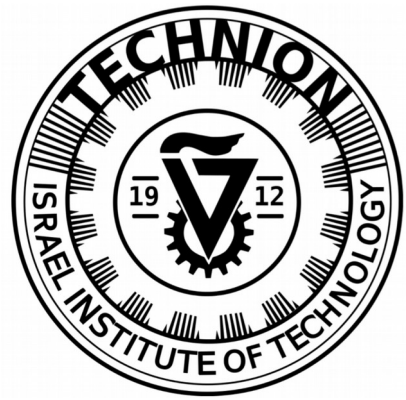
# Vision Aided Navigation (086761) - Final Report

**Itai Zilberman, Daniel Engelsman**




## Bayesian Information Recovery from CNN for Probabilistic Inference

*1. Introduction and overview*

*(i) Introduce the paper(s) topic*

Bayesian Inference (for system identification) is a key method in robotic applications such as autonomous navigation and SLAM. The different landmarks across the trajectory are being tracked in order to localize the robot's pose via triangulation and mutli-view geometry. Typical inference approaches that work with high-dimensional visual measurements (images), use hand-engineered image features (e.g. **SIFT**). However, this method is prone to several weaknesses that shortly will be elaborated.

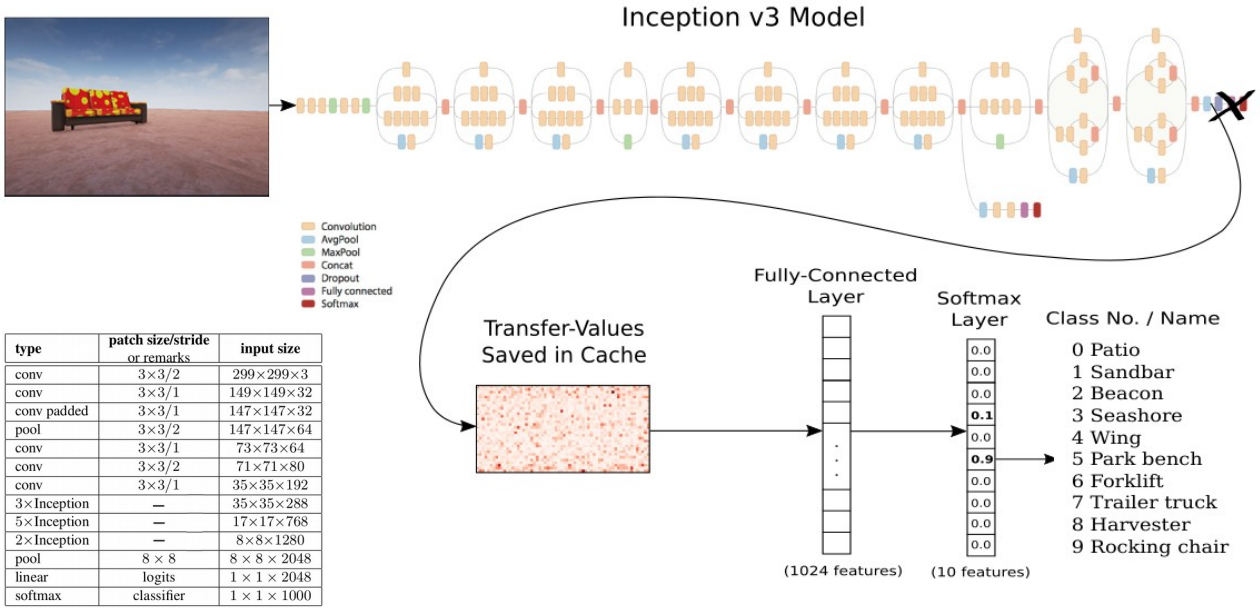
In this paper, the authors developed a **novel** approach to infer system hidden state from visual observations via **CNN features (Classifiers)**, which stands in contrast to classic approaches, where camera pose is inferred out of the given image features.

The paper makes use of 2 different **ANN (Artificial Neural Networks)** :

**1. CNN** network – The image processing system

**2. NN / FC** network – The localization system

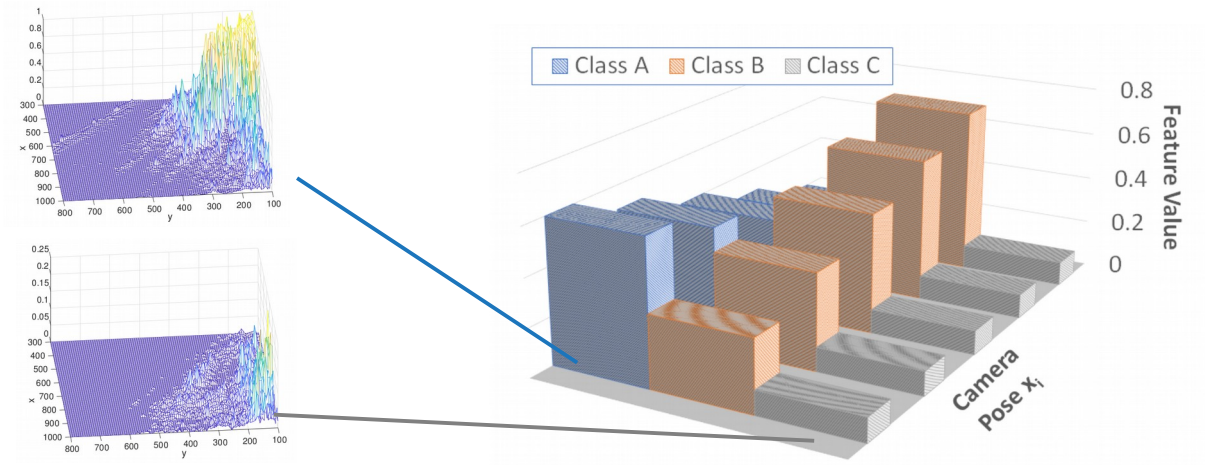
**1.** The CNN model was taken from Wojna et al. work namely Inception-v3, and was already pre-trained successfully, such that no further weights tuning were made. An image input ($\boldsymbol{I_i}$) is being processed across the convolutional layers, and eventually comes out as a feature vector ($\boldsymbol{f_i}$), denoting the highest prediction class (out of 10) :




| type | patch size/stride or remarks | input size |
|---|---|---|
| conv | 3×3/2 | 299×299×3 |
| conv | 3×3/1 | 149×149×32 |
| conv padded | 3×3/1 | 147×147×32 |
| pool | 3×3/2 | 147×147×64 |
| conv | 3×3/1 | 73×73×64 |
| conv | 3×3/2 | 71×71×80 |
| conv | 3×3/1 | 35×35×192 |
| 3×Inception | — | 35×35×288 |
| 5×Inception | — | 17×17×768 |
| 2×Inception | — | 8×8×1280 |
| pool | 8 × 8 | 8 × 8 × 2048 |
| linear | logits | 1 × 1 × 2048 |
| softmax | classifier | 1 × 1 × 1000 |

Top : the Inception-v3 CNN model & specs. On bottom right is the classifiers set.

Below : the corresponding classification for any image taken at $x_i \rightarrow \boldsymbol{f_i = classifier(I_i)}$.



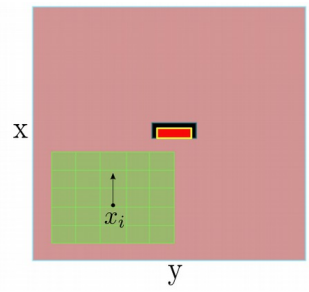

Given 10 different objects, we get 10 different probabilistic maps, showing expectedly no odds far from the sofas, and higher ones at the vicinity of the sofa. Note that objects that do not really appear in the environment still get odds, due to perceptual aliasing.

**2.** The NN model can be break down into 2 main steps :

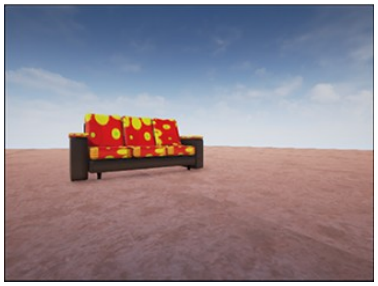
## A. Pre-deployment

### **A.1** Data labelling phase – (CNN training) pairing, input, expected data

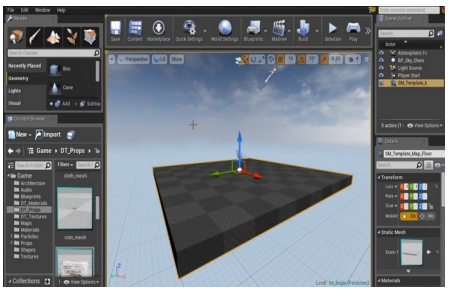
The very first step in a deep learning process is to label a group of samples with a meaningful tags, hence creating the ***train_dataset***. Using Unreal Engine (UE) simulation platform, the authors formed a virtual environment with a single object at its origin (=sofa). The labelling was made using TensorFlow library, linking each robot pose at time step i ($\mathbf{x}_i$), with a CNN's outputs ($\mathbf{f}^{\wedge}_i$). The training dataset is denoted as $\mathbf{D = \{x_i, f^{\wedge}_i\}}$, and contains 200,000 uniformly sampled robot poses ($\boldsymbol{x}_i$) across the environment, capturing corresponding images ($\boldsymbol{I}_i$) and their CNN output $\rightarrow \mathbf{f}^{\wedge}_i$.




Training field



Sofa observed from robot position



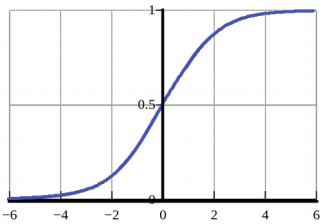
UE simulation platform

### **A.2** Training phase : tuning the NN model's weights

The **NN** model is used to learn a **viewpoint-dependent** measurement model of CNN features ($f_i$) given the robot pose ($x_i$). It's then approximating the model by a spatially-varying Gaussian distribution. Both mean and covariance networks are being trained on the same NN architecture, where the latter is done only after the mean network is trained.

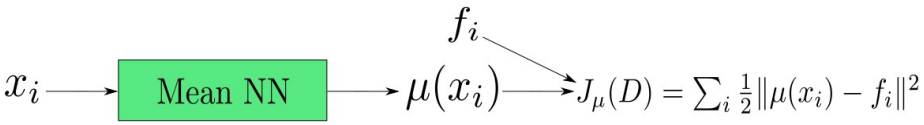
**A.2.1** Mean network training

The final layer's output is denoted as $\boldsymbol{o_i}$ and its mean is :




$$\mu(x_i) = \text{Sigmoid}(o_i) = (1 + e^{-Oi})^{-1} \in [0, 1] , \qquad \boldsymbol{\mu_i, o_i \in \mathbf{R}^{10}}$$

The reason for using that function is its stabilization properties as it nullifies negative inputs and enhance positive ones, and thus improves the classification process. The network's loss criterion is L2 regression (= Least Squares Error).




At last, the optimal network that produces the minimal loss on the testing loss is picked.

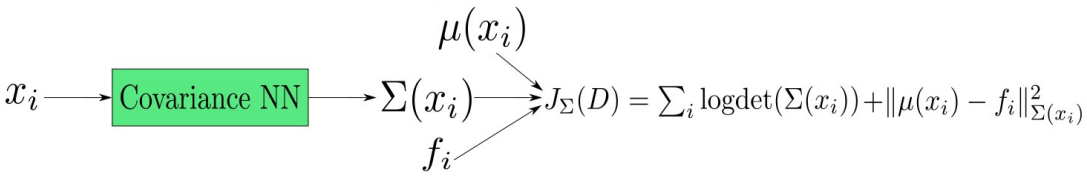
**A.2.2** Covariance network training

Based on that mean network, now comes the covariance network training, on the **same architecture** (found empirical more accurate). Its final layer's output (given $\boldsymbol{m \in \mathbf{R}^{10}}$) is of dimension $\boldsymbol{o_i \in \mathbf{R}^{10\times 11/2=55}}$, and the upper triangular matrix ($\boldsymbol{R}$) is filled such that :

$$\begin{aligned} \text{diag}(R(x_i)) &= \exp(o_i(1:m)) \\ \text{utri}(R(x_i)) &= o_i(m+1:end) \\ \text{ltri}(R(x_i)) &= 0 \end{aligned}$$

The conditional covariance of $f_i$ is expressed as : $\Sigma(x_i) = [R(x_i)^T R(x_i)]^{-1} \in \mathbf{R}^{10\times 10}$

And the network's loss criterion is ML (= Maximum Likelihood) :




At last, after training both networks, the 5 best ( = minimal loss) *mean and covariance networks* are picked, combined into of best 5×5 (mean × cov.) mixes, representing 25 sub-optimal learned likelihoods = $\boldsymbol{\{P^j(f_i|x_i)\}^{25}}$.

### **A.2.3** Validation phase

The test/training datasets ratio was chosen to be 25%-75% respectively such that the ***test_dataset*** contains $(2\cdot10^5)/3$ samples. In order to achieve the best performance, the system architecture was trained with a range of different parameters :

$N_F$ - # FC layers [2, 8]

$N_H$ - # hidden units [100, 1000]

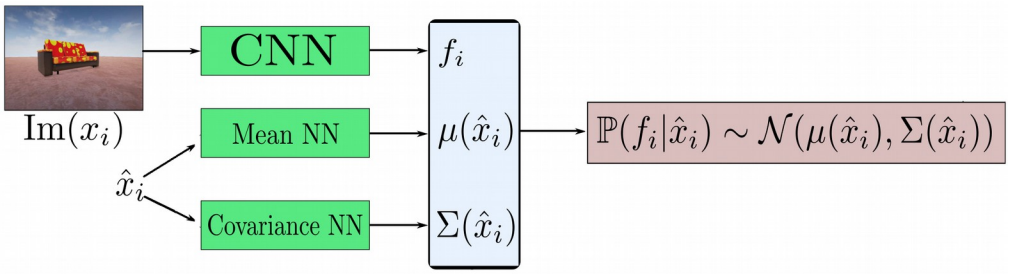
$P_D$ - # Probability [0.7, 1]

And eventually chose to be the one that exhibited the minimal evaluated loss.

## B. Deployment

### **B.** Application phase : online trajectory and state inference

After having trained the networks (*NN, CNN*) and obtaining all the necessary arguments ($\boldsymbol{\mu(x_i)}$, $\boldsymbol{\Sigma(x_i)}$, $\boldsymbol{f_i^{CNN}}$) for the probability likelihood ( $\boldsymbol{P(f_i \mid \hat{x}_i)}$ , $\hat{x}_i$ == estimated ) such that :



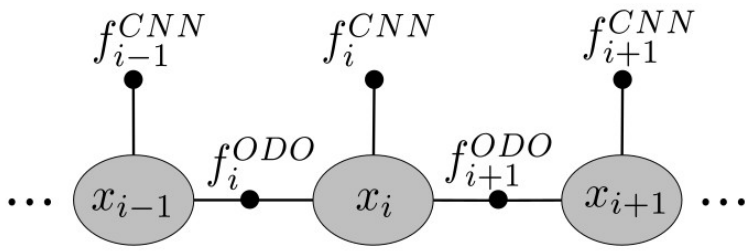

At each time step we get 3 types of data messages, to be later use for localization :

(i) **GT** - UE Ground Truth location (*control group*) @ $\mathbf{x_i}$

(ii) $\boldsymbol{f_i^{CNN}}$ - UE Image → CNN feature vector @ $\mathbf{x_i}$

(iii) $\boldsymbol{f_i^{ODO}}$ - UE Odometry measurement (un-estimated) @ $\{\mathbf{x_{i-1}}, \mathbf{x_i}\}$

Using appropriate likelihoods for each measurement, the trajectory inference problem can be represented as a factor graph ( inference optimization via **GTSAM** ) :



### **B.1** formulate CNN factors

A factor graph defines the factorization of function $f(\Theta) = \prod_i f_i(\Theta_i)$ where $\Theta_i$ is the set of variables $\theta_i$ adjacent to the factor $f_i$ . Here,

$$f^{CNN}(x_i) \doteq \mathbb{P}(f_i|x_i) = \det(2\pi\Sigma(x_i))^{-\frac{1}{2}} \cdot \exp(-\frac{1}{2}\|\mu(x_i) - f_i\|^2_{\Sigma(x_i)})$$

### **B.2** MAP inference

That factor introduces two terms into the MAP inference, and is calculated from NN at $t_i$ :

$$X^*_k = \arg\min_{X_k} \left[\ldots + \text{logdet}(\Sigma(x_i)) + \|\mu(x_i) - f_i\|^2_{\Sigma(x_i)}\right]$$

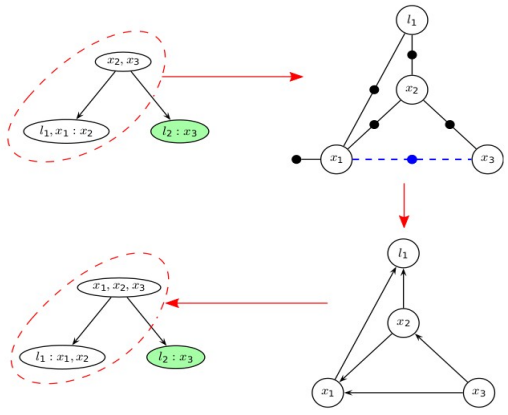
### **B.3** Incremental Gauss-Newton

Solving a non-linear estimation problem on ***real-time***, is done by iSAM2 approach, where the an updated estimate is obtained based on the current measurement at step i.

**Alg. 4** Updating the Bayes tree with new factors $\mathscr{F}'$.

In: Bayes tree $\mathscr{T}$, new linear factors $\mathscr{F}'$
Out: modified Bayes tree $\mathscr{T}$'

1. Remove top of Bayes tree and re-interpret it as a factor graph:
   (a) For each variable affected by new factors, remove the corresponding clique and all parents up to the root.
   (b) Store orphaned sub-trees $\mathscr{T}_{orph}$ of removed cliques.
2. Add the new factors $\mathscr{F}'$ into the resulting factor graph.
3. Re-order variables of factor graph.
4. Eliminate the factor graph (Alg. 2) and create a new Bayes tree (Alg. 3).
5. Insert the orphans $\mathscr{T}_{orph}$ back into the new Bayes tree.




**Left** : Bayes tree evolution algorithm

**Right** : Update tree with a new factor

*(ii) Describe how the paper fits in with the contents of this course* -

The paper is based on almost all topics that were covered in the course's homeworks :

- Bayesian inference, Autonomous ground vehicle, measurements noise - **Hw_1**
- Image feature extraction by CNN instead of SIFT - **Hw_2**
- Measurement model used is image observation - **Hw_3**
- The inference optimization in this paper, was solved using GTSAM library – **Hw_4**
- The paper's trajectory inference problem is illustrated by a factor graph – **Hw_5**

*(iii) why the problem is important -*

State information that's obtained from high-dimensional measurement (images), requires combinatorial data association, and predict only hidden state mean, without considering its uncertainty and **multi-modality** aspects. The paper shows empirically that CNN features likelihood is **uni-modal**, which simplifies the inference task.

*2. Preliminary material and problem formulation*

*(i) Present a description of relevant notations and definitions -*

| ***Symbol*** | ***Meaning*** | | ***Symbol*** | ***Meaning*** |
|---|---|---|---|---|
| | | *Neural Networks* | | |
| **NN** | Neural Network | | **CNN** | Convolutional Neural Network |
| **FC** | Fully connected (==**NN**) | | $\mathbf{P_D}$ | Probability of dropout function |
| $\mathbf{N_H}$ | Number of hidden units | | $\mathbf{N_F}$ | Number of FC layers |
| **A** | Jacobian | | $\mathbf{o_i}$ | Output of NN final layer |
| **CD** | Cholesky Decomposition | | **R** | Jacobian Matrix after CD |
| **D** | Training dataset = $\{\mathbf{x_i}, \mathbf{\hat{f}_i}\}$ | | $\mathbf{I_i}$ | Image($x_i$) |
| $\mathbf{x_i}$ | Robot Pose @ pre-deployment | | $\mathbf{\hat{x}_i}$ | Robot Pose @ deployment |
| $\mathbf{\hat{f}_i}$ | classifier($I_i$) @ pre-deployment | | $\mathbf{f_i}$ | classifier($I_i$) @ deployment |
| | | *System Model* | | |
| $\boldsymbol{h^j}$ | Nonlinear function | | $\boldsymbol{v^j}$ | Gaussian white noise |
| $\boldsymbol{\mu(x_i)}$ | Mean @ step i | | $\boldsymbol{\Sigma(x_i)}$ | Covariance @ step i |
| **LSE / L2** | Least Squares Error | | **ML** | Maximum Likelihood |
| $\mathbf{J_\mu(D)}$ | Mean cost (L2 Loss) | | $\mathbf{J_\theta(D)}$ | Covariance cost (ML Loss) |
| $\boldsymbol{\theta_\mu}$ | Mean network weights | | $\boldsymbol{\theta_\Sigma}$ | Covariance network weights |
| | | *Inference Optimization* | | |
| $\mathbf{P(f_i|x_i)}$ | Conditional pdf / likelihood | | $\mathbf{\{P^j(f_i|x_i)\}^{25}}$ | 5×5 prob. Comb. of mean × Cov. |
| $\boldsymbol{f_i^{CNN}}$ | CNN factors | | $\boldsymbol{f_i^{ODO}}$ | Odometry factors |
| $\boldsymbol{X_k^* / MAP}$ | Maximum a posteriori | | $\boldsymbol{X^j}$ | 2 subsequent poses = $\mathbf{\{x_{i-1}, x_i\}}$ |

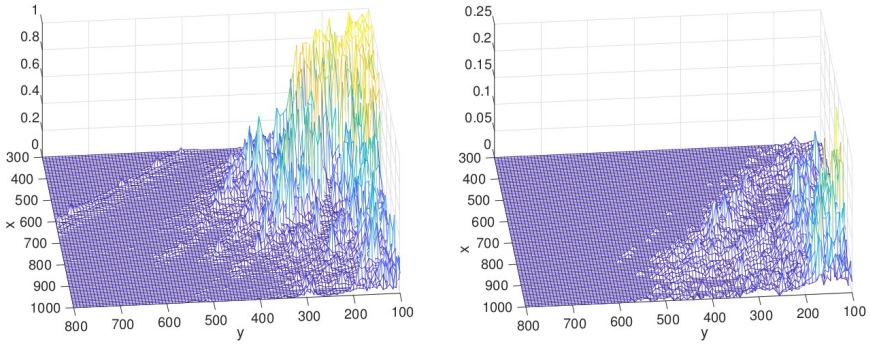
(ii) define mathematically the problem addressed by the paper(s) -

All mathematical expression were introduced along ***section 1***, for clarification reasons.

*3. <u>Main contribution</u>*

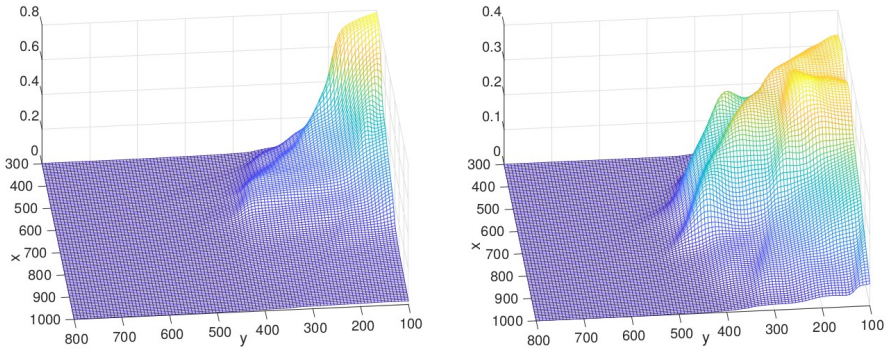
*(i) A detailed discussion of the main results of the paper(s)* :

Let us first look at the probability likelihoods of 2 of the CNN's output classifiers ($f_i$) :



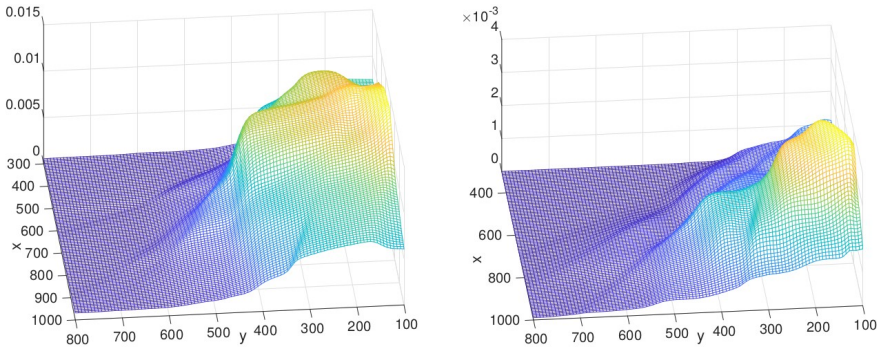

**Above :** The plots above demonstrate the probabilistic map of classifying a certain object ("park bench" and "rocking chair") from an X-Y coordinates on the training field.

However, the same tendency holds true when plotting the mean and covariance of the Learned Likelihood that's obtained. The upper right corner represents the closest position of the robot get w.r.t the sofa. It's therefore no surprise that a "park bench", that resembles "sofa" most, exhibits the highest probabilities of observing / identifying it.

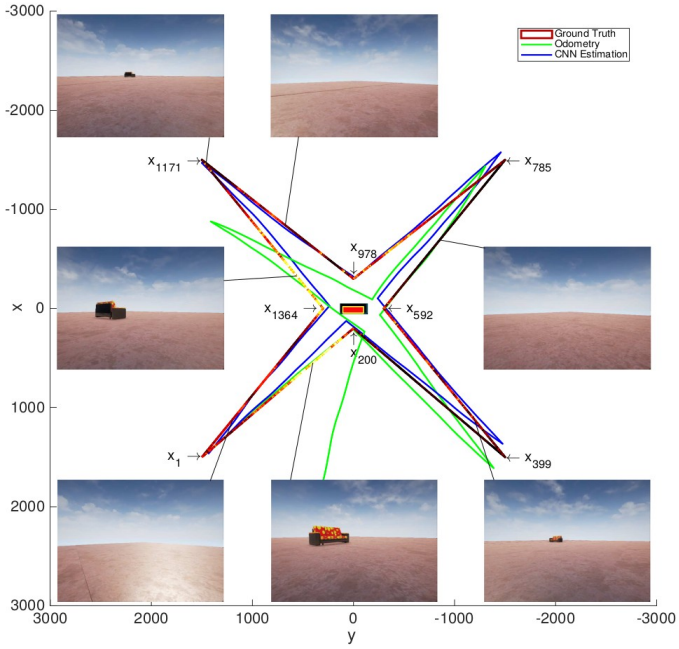



mean and covariance **P( f | x)** respectively of "<u>park bench</u>"

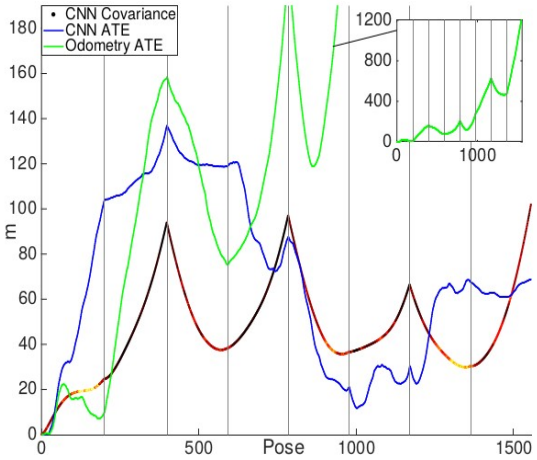
In contrast, we can see much lower likelihood when the object is physically and significantly different from the "sofa". Although the values concentration spawn around the <u>same area</u>, the difference between the both classes **P( f | x)'s**, is dramatic.



mean and covariance **P( f | x)** respectively of "rocking chair"




Here can be seen the "star" pattern trajectory, where at the center the sofa is located. Starting from $x_1$ towards the sofa, and encompassing it back and forth, we can see the differences between the measurements type (**Section B**). The **CNN** approach manages to stick rather close to the robot's **ground truth**, while the **odometry** shows a growing drift along trajectory, that quickly lose track. The **GT** path is colored-coded in order to asses the *learned model quality*. Not surprisingly, at the vicinity of the sofa, where object can be clearly seen, higher score is obtained. In contrast, when reaching far apart from the sofa or when line of sight is not pointing the sofa, we get lower scores.




Analyzing the absolute trajectory error (ATE) shows a clear correlation between the ATE evolution and the robot's location. Each of the error's *local maxima and minima* are obtained when reaching the of the star's base and arm end, respectively. That is to say that either the GT or the CNN use a truncation tool to decrease the Cartesian error, where the odometry shows divergence when without it.

At every time step along trajectory, the robot acquires a set of $\{\hat{\boldsymbol{x}}_i, \boldsymbol{f}_i\}$ for the probability likelihood and updates the Bayes tree, and re-evaluates the error.

*4. Implementation*

*(i) Demonstrate the main results of the paper by simulation or real-world experiments -*

>> Consider on-line demonstration of CNN (maybe from Github, or otherwise Cifar-10)

*5. Discussion and Conclusions*

*(i) Summarize the report :*

● The model learns a spatial measurement likelihood of CNN features

● It exploits the viewpoint-dependency of CNN features for camera pose estimation

● It solves Bayesian inference with spatially varying measurement likelihood, in particular using TensorFlow / computation graphs.

*(ii) provide some criticism*

● The training dataset created is rather big in terms of common NN's, expresses an unrealistic scenario that de-facto might be impractical.

● Some properties at the paper are not fully elaborated and remain inexplicit or parametric. We found it somewhat hard to understand the model's specifications.

*(ii) identify strong points*

● The paper's method does not involve data association for the inference, and provides uncertainty covariance of the final estimation, while showing empirically uni-modality.

● It analyzes CNN features' robustness and suggests set of future improvements.

*(iii) identify weak points*

● As the author himself admits, the model suffers from low robustness to changing light conditions. During the training stage, the images were trained under **a certain set-up** : light magnitude (position of the sun), sky brightness (cloudiness), terrain color, etc.
The image processing (feature detection), is largely influenced by there factors.

When darker illumination was applied, the ATE (absolute trajectory error) increased dramatically since the predictability between real/approximated images, decreased.
That is to say, that the system would drift quickly under these conditions, and localization task would fail.

*(iv) identify unrealistic assumptions or aspects that could be improved and suggest possible directions (or extensions) for future research.*

>> To be defined ...

###
● The word *empirically* appeared whenever ‘trial and error’ won over analytical explain :
a. The probabilistic combination $\mathbf{\{P^j(f_i|x_i)\}^{25}}$ of mean (first) and then covariance (second)
b. The MAP estimation assumed negligence of *logdet* terms, and then whitened the residuals in order to achieve an ordinary least-squares
###