Neural SLAM

S. Ginargyros, Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr
Version 7.4.1

Artificial Intelligence &
Information Analysis Lab

Deep Learning in Computer Vision

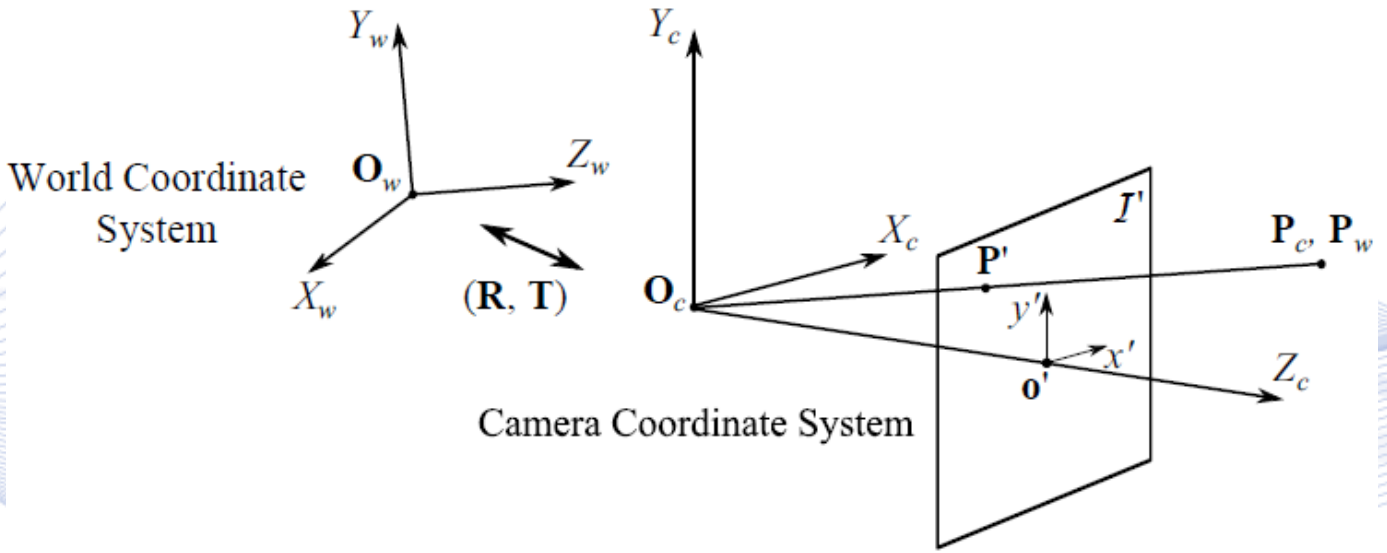
- **Neural Camera Calibration**
- Neural Mapping/Reconstruction
- Neural Localization
- Neural SfM
- Neural SLAM

Neural Camera Calibration

Camera calibration is the process that estimates where the light recorded in every pixel $\mathbf{p} = [x', y']^T$ of a camera comes from in the 3D world (X_w, Y_w, Z_w).

Neural Camera Calibration

The required transformation from the world to the camera coordinate system involves a translation followed by a rotation, based on the ***extrinsic camera parameters***.



Neural Camera Calibration

Extrinsic camera parameters:

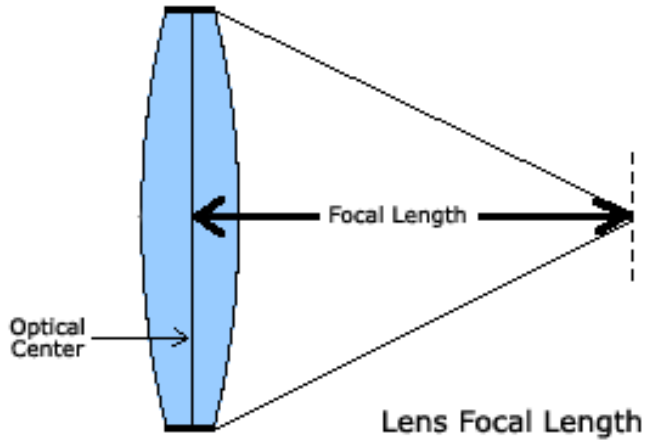
- ***Translation vector*** $\mathbf{T} \in \mathbb{R}^3$
- ***Orthonormal rotation matrix*** $\mathbf{R} \in \mathbb{R}^{3\times3}$ (Consists of $\mathbf{R}_Z, \mathbf{R}_X, \mathbf{R}_Y$ sub-rotations)
- The relationship between a point $\mathbf{P}_w \in \mathbb{R}^3$ in world coordinates and its camera coordinate counterpart $\mathbf{P}_c \in \mathbb{R}^3$ is:

$$\mathbf{P}_c = \mathbf{R}(\mathbf{P}_w - \mathbf{T}).$$

Neural Camera Calibration

Intrinsic camera parameters :

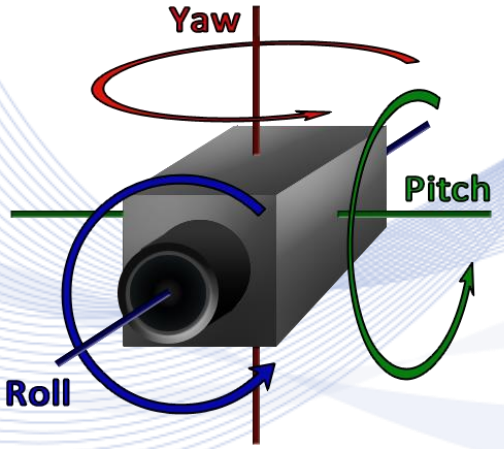
- **Intrinsics**: optical center $\mathbf{O}_c$, focal length f, etc.



Neural Camera Calibration

The angles roll ,pitch and yaw represent the angle of rotations R_x, R_y and R_z accordingly. Rotation matrix can be synthesized from rotations on basic axis:

$$\mathbf{R} = \mathbf{R}_Z(\theta)\mathbf{R}_X(p)\mathbf{R}_Y(e)$$



Roll: θ (around Z axis)

Pitch: p (around X axis)

Yaw: e (around Y axis)

[ITU2017] Rotation Parameters

Neural Camera Calibration

Specifically $\mathbf{R}_Z(\theta)$, $\mathbf{R}_X(p)$, $\mathbf{R}_Y(e)$ can be written as:

- $\mathbf{R}_Z(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$, θ: roll angle
- $\mathbf{R}_X(p) = \begin{bmatrix} \cos(p) & 0 & \sin(p) \\ 0 & 1 & 0 \\ -\sin(\varphi) & 0 & \cos(p) \end{bmatrix}$, p: pitch angle

[ITU2017] Rotation Parameters

Neural Camera Calibration

And:

- $\mathbf{R}_Y(e) = \begin{bmatrix} \cos(e) & -\sin(e) & 0 \\ \sin(e) & \cos(e) & 0 \\ 0 & 0 & 1 \end{bmatrix}$, e: yaw angle

[ITU2017] Rotation Parameters

Neural Camera Calibration

Many researchers [LEE2020] [ITU2017] [LOP2019] have proposed the use of neural networks for the task of camera calibration or camera parameter estimation.

Neural Camera Calibration

In most studies like [LEE2020] and [ITU2017] a **CNN** architecture is getting trained on big datasets of images with labeled Intrinsic calibrated camera parameters.

In most cases the inference for the camera calibration is done with a **single image (monocular).**

Neural Camera Calibration

A recent study [LOP2019] proposes an alternative estimation for the camera parameters that is based on image cues. Cues usually are extracted based on the horizontal line and the roll angle θ, between the vector $\mathbf{a}$ and the horizon.

[LOP2019] Line of Horizon, and angle

Neural Camera Calibration

The network is trained to predict the distorted offset ρ and vertical field of view F_v instead of the roll angle θ and focal length f. The undistorted offset τ is where the horizon would be if there was no radial distortion.

[LOP2019] Camera Parameters based on Image cues

Neural Camera Calibration

Specifically **field of view** which is predicted by the net can be expressed as:

- $F_v = 2 \arctan \frac{h}{2f}$, f : focal length, and h : image height.

[LOP2019] Camera Parameters based on Image cues

Neural Camera Calibration

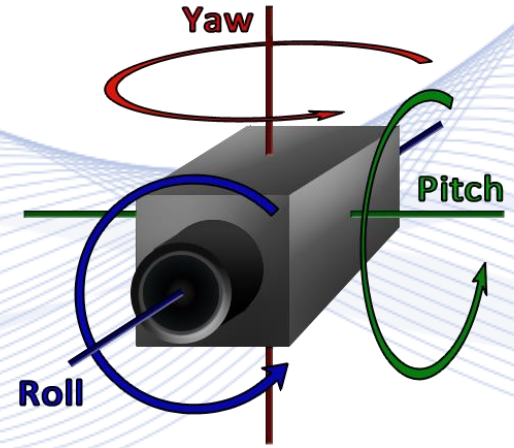
Undistorted offset τ, when there is no radial distortion can be expressed as the distance from the center of the image to the horizon.

$$\tau = f \tan(\theta)$$

Which can adapt in case of radial distortion, to a distorted offset ρ by scaling with distortion coefficients $k1\ and\ k2$

Neural Camera Calibration

Some extrinsic camera parameters can also be extracted implicitly. For example the vanishing point in an image holds information about the pitch p and yaw e, angles of the camera as stated in [ITU2017].



[ITU2017] Rotation Parameters

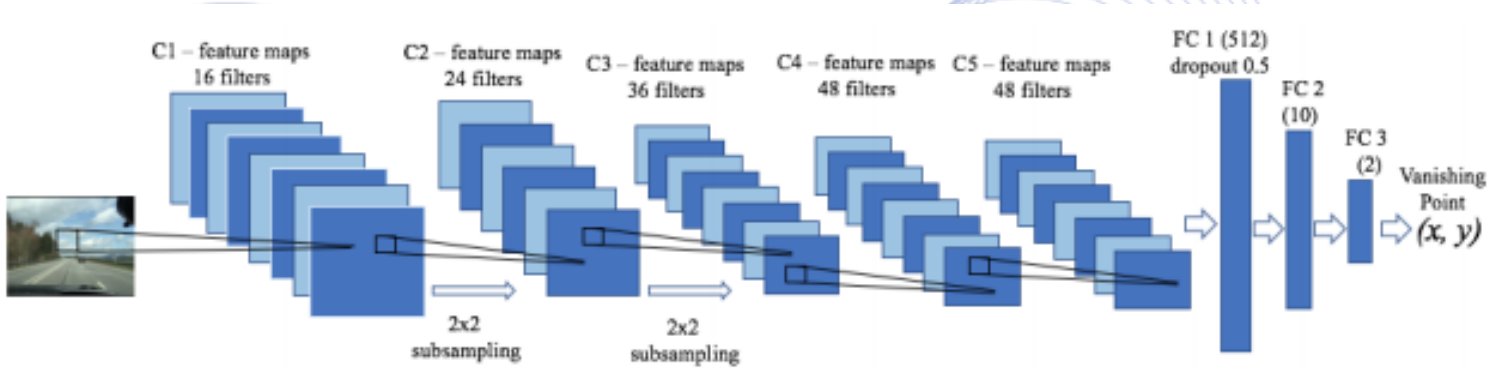
Neural Camera Calibration

This particular approach require a labeled dataset in the format of: I, VP where I: Image and VP: Vanishing Point.

There are various geometric algorithms performing **vanishing point detection** in order to label the dataset automatically such as [BUI2013] [KON2009] but they are not real time, thus it is convenient to utilize Neural Networks for inference.

Neural Camera Calibration

The CNN requires 160x48 resolution images, and it is composed by 8 layers plus a fully connected with just 2 neurons where the vanishing point coordinate $VP(x, y)$ prediction happens.



[ITU2017] Network Architecture

Neural Camera Calibration

From the last layer of the CNN we pull the predicted Vanishing point coordinates $VP\ (x, y)$ and through the following formulations we can also predict the p: pitch and e: yaw.

- Coordinate y_{vp} holds information about the p.
- Coordinate x_{vp} holds information about the e.

Neural Camera Calibration

As a **sidenote** there has been research recently (from Microsoft in 2020) [SCH2009] which proposes that having 10.000 parameters in your camera is much better than twelve. This is due to camera's natural lens distortion that can't be properly modelled in the 12 dimension parameter space.

Deep Learning in Computer Vision

- Neural Camera Calibration
- **Neural Mapping/Reconstruction**
- Neural Localization
- Neural SfM
- Neural SLAM

Neural Mapping/Reconstruction

Definition of the variables/ sizes.

$\mathbf{X}_n$: Position of the agent at discrete time-step n. (world coordinates)

$\mathcal{M}$: The map, gets updated each time-step n with $\mathbf{m}_n$

Neural Mapping/Reconstruction

Scene Reconstruction is the process of **estimating** the map $\mathcal{M}$, of an unknown environment by processing sensor cues (Image, Lidar data, etc..).

3D Reconstruction : $\mathbf{X}_n = (X_\mathrm{w}, Y_\mathrm{w}, Z_\mathrm{w})$

2D Reconstruction : $\mathbf{X}_n = (X_\mathrm{w}, Y_\mathrm{w})$

$\mathcal{M}$:

3D reconstruction

$\mathcal{M}$:

Neural Mapping/Reconstruction

There is a wide variety of applications that can or already use related algorithms.

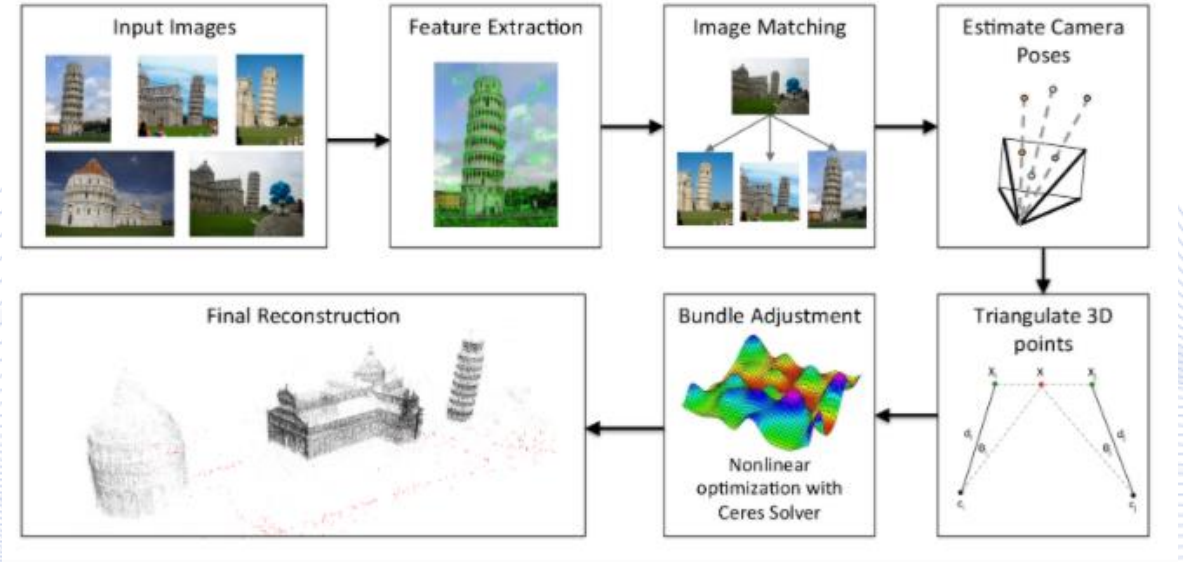
- Autonomous driving
- Robotics
- 3D printing
- Medical
- Architecture etc..

Neural Mapping/Reconstruction

Traditionally scene reconstruction is done with geometrical approaches such as **Structure from Motion**, which in many cases is still fast and reliable.

Neural Mapping/Reconstruction

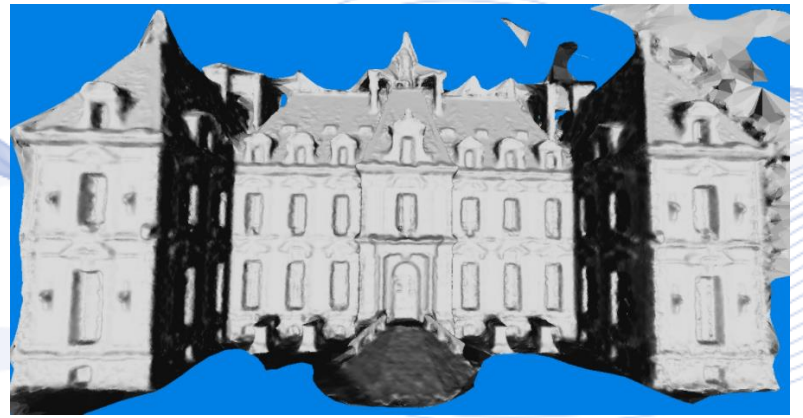
Structure from Motion requires **Multiview** in order to reconstruct a scene in a **point cloud**. A typical algorithmic workflow:



http://www.theia-sfm.org/sfm.html

Neural Mapping/Reconstruction

Since we have collected the point cloud, we proceed with **densification**, **meshing** and **texturing** to produce a photorealistic output. A common open-source library for that purpose: https://github.com/cdcseacave/openMVS



OpenMVS densification and texturing [github]

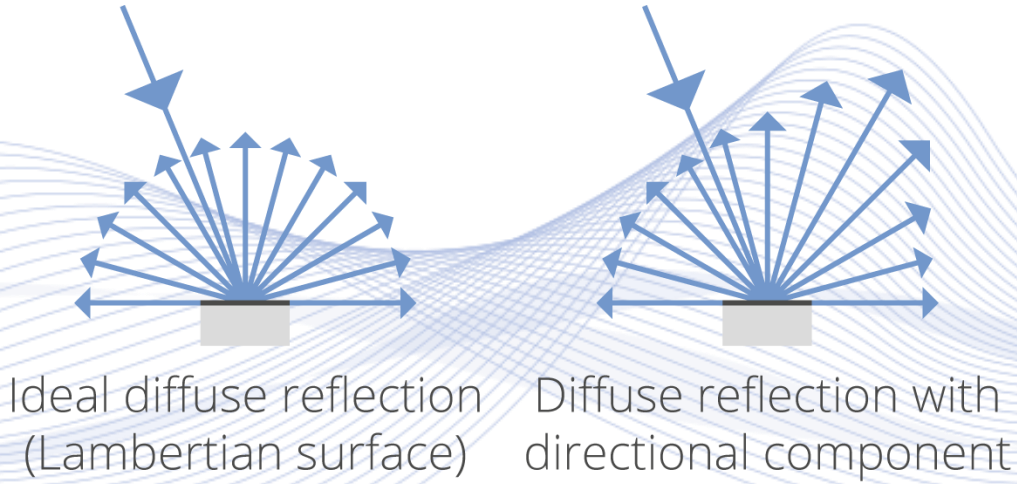
Neural Mapping/Reconstruction

Even though in many cases classical approaches can be enough, there are a bunch of scenarios **where they fail:**

- Not enough images, or enough overlap between them.
- Surfaces that can’t produce features (reflections)
- Thin or repeated structures.

Neural Mapping/Reconstruction

The most common reason for SfM to fail as stated [SCH2016] is non-Lambertian Surfaces.



Reflectance Model of a surface

Neural Mapping/Reconstruction

We can tackle most of these problems with **Deep Learning**.

- Most methods will inference with monocular input.
- More descriptive, robust and problem-wise image representations.
- Make use of prior knowledge.

Neural Mapping/Reconstruction

2D Scenario

When reconstructing a map in 2D we focus on the **top down egocentric** projection of the map as it usually provides the richest geometrical information.

Neural Mapping/Reconstruction

3D Scenario

We reconstruct the scene using 3D point clouds. Usually these techniques can acquire camera poses and trajectories too.

Neural Mapping/Reconstruction

Mapping/Reconstruction total possible scenarios:

2D world + **2D** Reconstruction

3D world + **2D** Reconstruction

3D world + **3D** Reconstruction

We will analyze each one of them.

Neural Mapping/Reconstruction

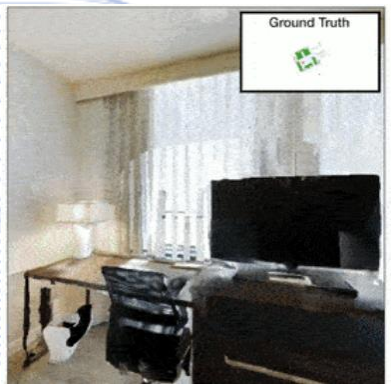
Deep Learning 2D Reconstruction:

2D world to 2D map: where the agent has some kind of environment sensor, deep learning is not much needed, since the observations can almost always reconstruct the environment with a basic transformation of the sensor readings.

Neural Mapping/Reconstruction

Deep Learning 2D Reconstruction:

3D world to 2D map: it's more complex thus researchers tend to utilize deep learning techniques to overcome the complexity. Such as in "Learning to Explore using Active Neural SLAM" [CHA2020], **Mapper** module.



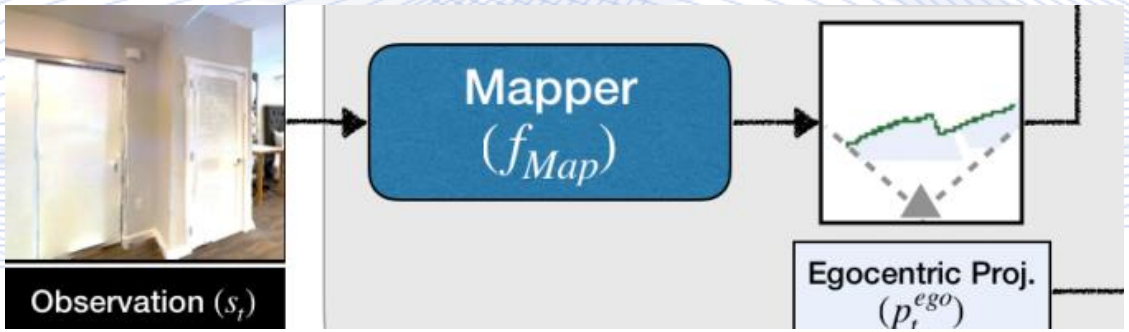
3D world observation [CHA2020]

2D map (in green) CHA[2020]

Neural Mapping/Reconstruction

Deep Learning 2D Reconstruction:

3D world to 2D map: In this case the **Mapper module** consists of ResNet18 convolutional layers to produce an embedding of the 3D observation. This embedding is passed through **2 fully connected** layers, followed by **3 deconvolutional** to get the top-down **2D spatial map** prediction.



[CHA2020] Mapper Module from 'Active Neural SLAM'

Neural Mapping/Reconstruction

Deep Learning 3D Reconstruction:

The most challenging Scene Reconstruction is 3D to 3D, where Deep Learning really shines. In most cases it's sufficient to produce a Depth-Map from an image taken in the 3D world, and then reconstruct the 3D Scene. This is the most dominant approach, as stated in recenter research [GAR2016], [ZHO2017], [GUI2020].

Neural Mapping/Reconstruction

Research has moved towards inferring depth maps from ***monocular*** images/videos.

Unsupervised depth/disparity estimation methods have prevail due to their low cost and efficiency.

(Supervised needs labeling..)

Neural Mapping/Reconstruction

Unsupervised monocular depth map estimation using stereo for training [GAR2016]:

- Pixel $\mathbf{p}$ of $\mathbf{I}_l$ (left image) should appear in $\mathbf{I}_r$ (right image) at position $\mathbf{p}'$:

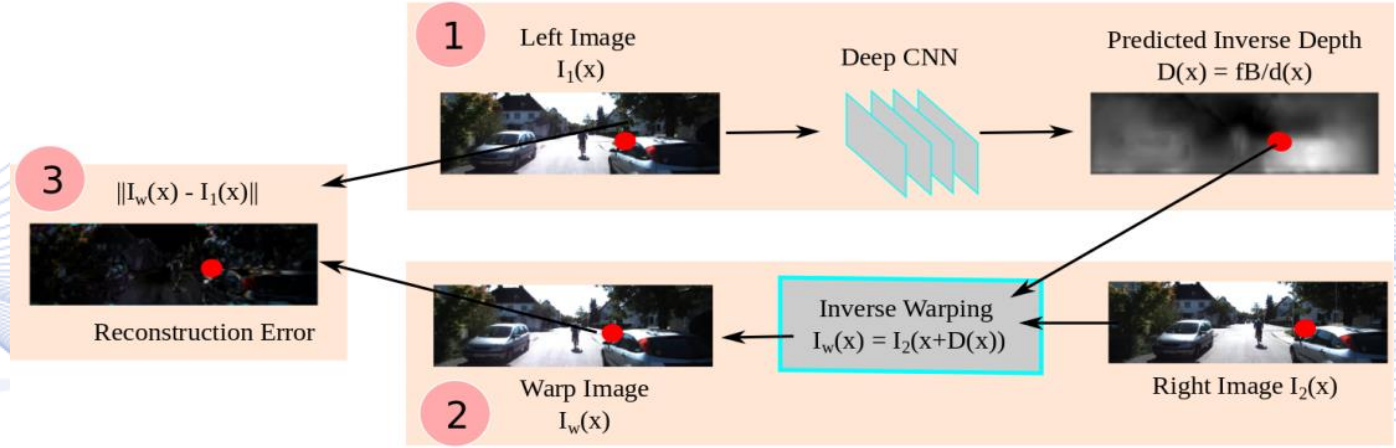
$$\mathbf{p}' = \mathbf{p} + \frac{fT}{D(\mathbf{p})}$$

- f: focal length, T: stereo baseline,
- $D(\mathbf{p})$: depth map entry at point $\mathbf{p}$.

Neural Mapping/Reconstruction

Next, image $\mathbf{I}_r$ is warped to form an approximation $\mathbf{I'}_l$ of $\mathbf{I}_l$, such that:

$$\mathbf{I}_l(\mathbf{p}) \approx \mathbf{I'}_l(\mathbf{p}) = \mathbf{I}_r(\mathbf{p'}).$$



[GAR2016] Algorithmic Pipeline

Neural Mapping/Reconstruction

- Then, the photometric loss function J_p is minimized for optimal depth estimation:

$$J_p = \sum_{\mathbf{p} \in \mathcal{X}} \| \mathbf{I}_l(\mathbf{p}) - \mathbf{I}_r(\mathbf{p}') \|^2.$$

- During DNN **training** using stereo image pairs, DNN learns to estimate $D(\mathbf{p})$, by minimizing J_p.
- During **testing**, a monocular image $\mathbf{I}$ is fed to DNN to produce the desired disparity map $\mathbf{D}$.

Neural Mapping/Reconstruction

Summary:

- Traditionally, scene geometry was directly sampled using 3D sensors, such as LiDARs.
- Recently, Deep Neural Networks (DNNs) enabled accurate scene geometry and semantics estimation, using visual sensors only, such as RGB or RGB-D cameras.

Deep Learning in Computer Vision

- Neural Camera Calibration
- Neural Mapping/Reconstruction
- **Neural Localization**
- Neural SfM
- Neural SLAM

Neural Localization

Robot **localization is essential** for navigation, planning, and autonomous operation. There are vision-based approaches addressing the robot localization problem in various environments both in 2D and 3D.

Example uses : Autonomous cars, factory robots, housekeeping agents, delivery drones etc..

Neural Localization

Definition of the variables/ sizes in **global localization** (initial position is unknown) problem:

$\mathbf{z}_n$: Robot r observation at discrete time-step n.

a_n : Action taken by the agent at discrete time-step n.

$\mathbf{X}_n$: Position of the agent at discrete time-step n.

Can be up to 6DoF when $\mathbf{X}_n = (X_\mathrm{r}, Y_\mathrm{r}, Z_\mathrm{r}, R_\mathrm{X}, R_\mathrm{Y}, R_\mathrm{Z})$, r: robot

$\mathcal{M}$: The map, gets updated each n with $\mathbf{m}_n$.

Neural Localization

Localization : Estimating the **location** $\mathbf{X}_n$ of an autonomous agent **given** an **observation** $\mathbf{z}_n$ (a) and a map of the **environment** $\mathcal{M}$(b).

(a) Agent Observation [CHA2018]

(b) A map of the environment [CHA2018]

Neural Localization

Passive Localization: Traditional localization algorithms, receive a sequence of observations and actions, and use the map information to output a sequence of location predictions.

Localization function : $\mathrm{L}(a_n, \mathbf{z}_n, \boldsymbol{\mathcal{M}}) = \mathbf{X}_n$

$$n = 1\,, \quad \mathrm{L}(a_1, \mathbf{z}_1, \boldsymbol{\mathcal{M}}) = \mathbf{X}_1$$
$$n = 2\,, \quad \mathrm{L}(a_2, \mathbf{z}_2, \boldsymbol{\mathcal{M}}) = \mathbf{X}_2$$
$$\dots$$

Neural Localization

Active Localization: When localizing actively, the agent is capable of predicting the actions to be taken. That's **affecting future observations** and results in faster and more accurate algorithms.

Localization function : $\mathrm{L}(a_n, \mathbf{z}_n, \boldsymbol{\mathcal{M}}) = \mathbf{X}_n, a_{\mathrm{n+1}}$

$$n = 1\,, \quad \mathrm{L}(a_1, \mathbf{z}_1, \boldsymbol{\mathcal{M}}) = \mathbf{X}_1, a_2$$
$$n = 2\,, \quad \mathrm{L}(a_2, \mathbf{z}_2, \boldsymbol{\mathcal{M}}) = \mathbf{X}_2, a_3$$
$$\dots$$

Neural Localization

With this in mind we are interested in the bellow probability of location $\mathbf{X}_n$.

$$\mathrm{B}(\mathbf{X}_n) = \mathrm{P}(\mathbf{X}_n | \mathbf{z}_{1:n}, a_{1:n-1}, \boldsymbol{\mathcal{M}})$$

Where $\mathrm{B}(\mathbf{X}_n)$ is the Belief for $\mathbf{x}$ at discrete time-step n.

Neural Localization

With this in mind we are interested in the bellow probability of location $\mathbf{X}_n$.

$$\mathrm{B}(\mathbf{X}_n) = \mathrm{P}(\mathbf{X}_n | \mathbf{z}_{1:n}, a_{1:n-1}, \mathcal{M})$$

In the framework of Bayesian filtering the **likelihood** can be expressed as.

$$\mathrm{l}(\mathbf{z}_n) = \mathrm{P}(\mathbf{z}_n | \mathbf{X}_n)$$

Neural Localization

Under the Markov assumption we can recursively calculate the B with these two equations.

- $\mathrm{B}_p(\mathbf{X}_n) = \sum_{\mathrm{X}_{t-1}} \mathrm{P}(\mathbf{X}_n|\mathbf{X}_{n-1}, a_{n-1})\ B(\mathbf{X}_{n-1})$
- $\mathrm{B}(\mathbf{X}_n) = \frac{1}{Z}\mathrm{l}(\mathbf{z}_n)\ \mathrm{B}_p(\mathbf{X}_n)$

B_p is the belief prior observing $\mathbf{z}_t$, B is after.

Neural Localization

How we incorporate **Neural Networks** for this task?

- Most common is to use a **perception network** to predict the $\mathbf{l}(\mathbf{z}_n)$ of the agent to be localized in $\mathbf{X}_n$, after feeding the model observations of the environment $\mathbf{z}_n$.

- Then a **policy network** which predicts the next action a_n, after feeding it the map $\mathcal{M}$ and some past actions a_{n-numb}.

Neural Localization

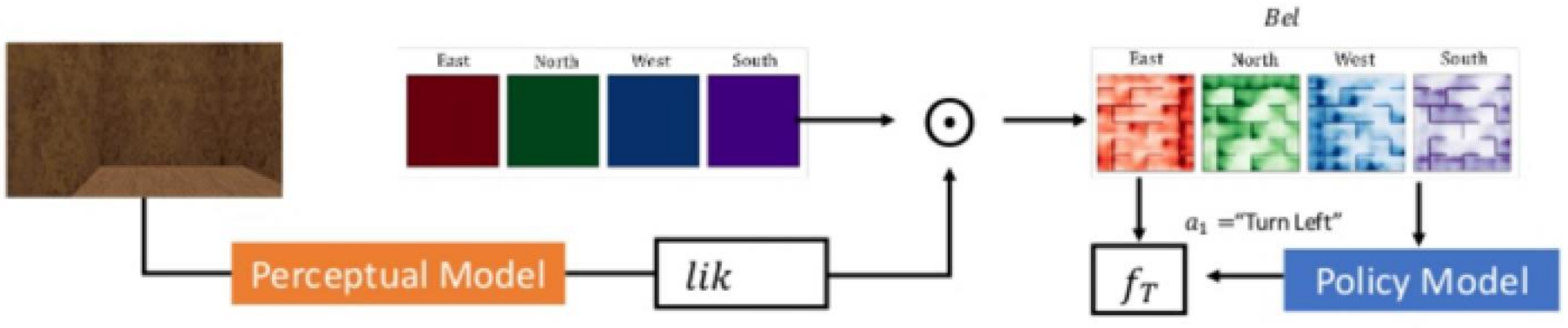
All we need is to formulate this :

$$\mathrm{B}(\mathbf{X}_n) = \frac{1}{Z}\mathrm{l}(\mathbf{z}_n)\,\mathrm{B}_p(\mathbf{X}_n)$$

After the perceptual network predicts the likelihood, we element-wise dot product with the posterior belief of location $\mathbf{X}_n$.

Neural Localization

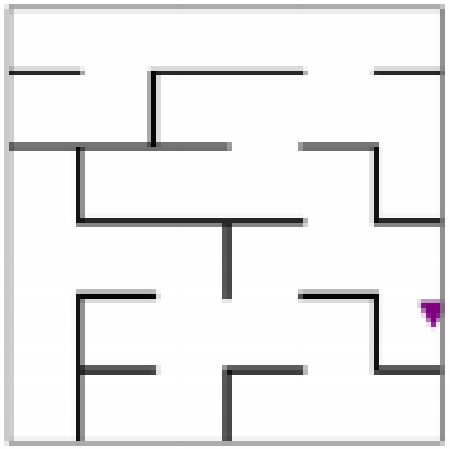
Visually the process looks like this:



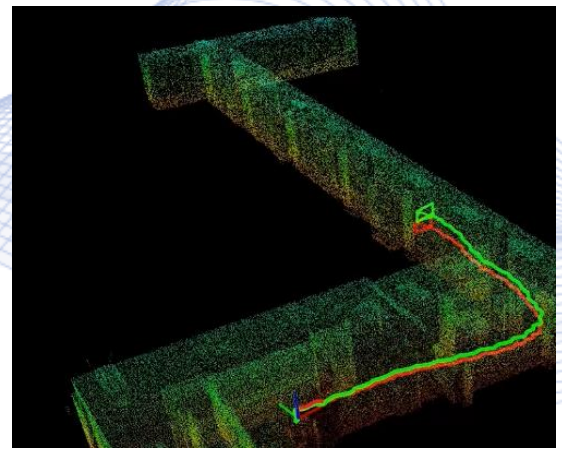
Neural Active Localization Network [CHA2018]

Neural Localization

For the particular 2D localization example scenario with 4 possible agent orientations (East, North, West, South), the output would look like the left image although same principles can be expanded in 3D environments as in the right image.



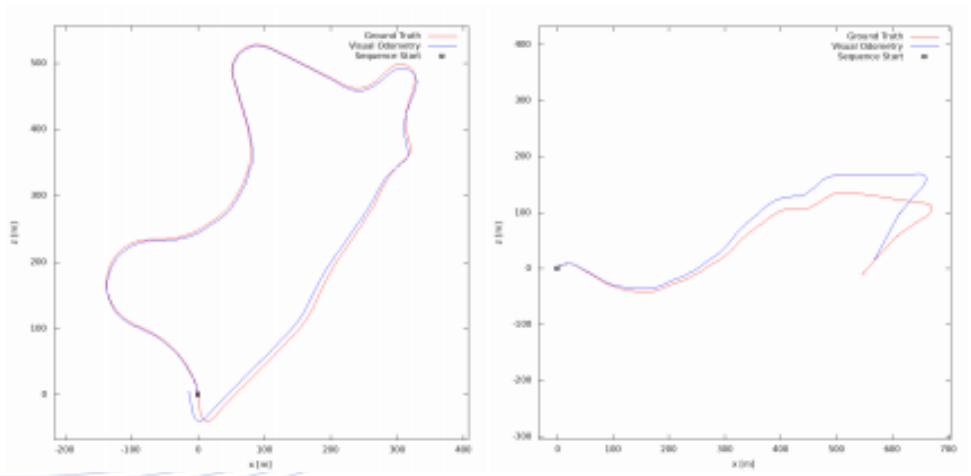
2D Localization [CHA2018]



3D Localization [GUI2020]

Neural Localization

Modern Deep Learning techniques such as **Google's SfM-Learner[7]** and **Toyota's PackNet[8]**, can also estimate camera's trajectories in 6DoF.



Toyota's PackNet Trajectory estimation

Neural Localization

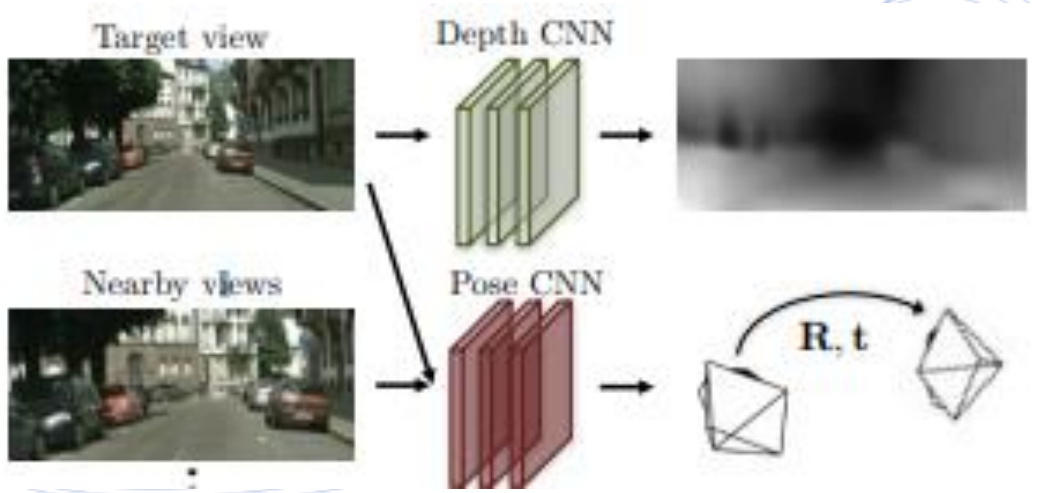
But since these techniques emphasize in **Structure from Motion**, they will be analyzed in depth in the next section!

Deep Learning in Computer Vision

- Neural Camera Calibration
- Neural Localization
- Neural Mapping/Reconstruction
- **Neural SfM**
- Neural SLAM

Neural SfM

Google's approach (2017) [ZHO2017], reaching 1000 citations is now a classic paper in the field. The code is publicly available on github https://github.com/tinghuiz/SfMLearner



[ZHO2017] Depth And Pose Network outputs

Neural SfM

Google's approach (2017) [ZHO2017]

The model **only requires a single image** for inference, and it is additionally providing a camera pose estimation along with the depth-map. The training is performed in an unsupervised manner from unlabeled video sequences.

Neural SfM

Google's approach (2017) [ZHO2017]

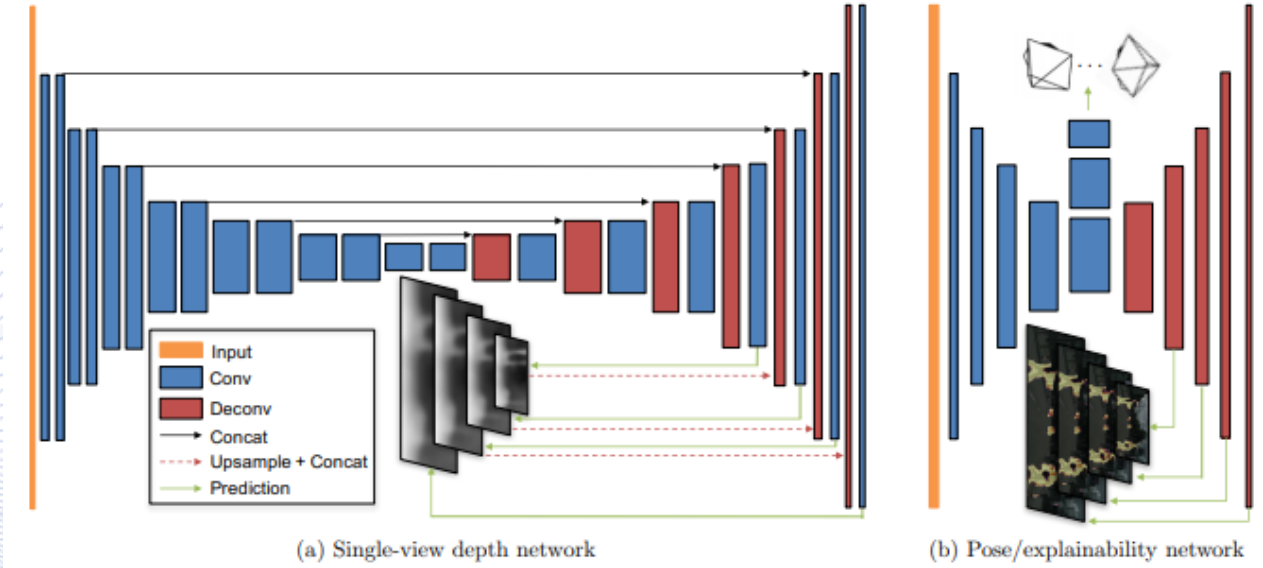
SfM-learner consist of 2 major networks

- Single-view depth network (estimates the depth-map)
- Pose network (estimates the camera pose)

Even though these networks are trained jointly, they run separately on testing.

Neural SfM

Google's approach (2017) [ZHO2017]



[ZHO2017] Network Architecture

Neural SfM

Google's approach (2017) [ZHO2017]

As you can see benchmarked on the KITTI odometry, it outperforms the classic geometrical reconstruction approaches such as ORB-SLAM based on trajectory loss (Pose-Net). Similar results achieves the Depth-Net for depth-tasks and its even more accurate than supervised methods.

ORB-SLAM (short)	0.064 ± 0.141	0.064 ± 0.130
Mean Odom.	0.032 ± 0.026	0.028 ± 0.023
Ours	$\mathbf{0.021 \pm 0.017}$	$\mathbf{0.020 \pm 0.015}$

[ZHO2017] Trajectory loss, Lower is Better

Neural SfM

TOYOTA's approach (2020) [GUI2020]

Even though this particular research is recent, tends to become a classic due to extremely accurate results and real-time inference. Besides the powerful model itself, Toyota released a high quality related dataset both of them open source licensed. https://github.com/TRI-ML/packnet-sfm

input and Depth-Map [GUI2020]

Scene Reconstruction [GUI2020]

Neural SfM

TOYOTA's approach (2020) [GUI2020]

[GUI2020] Real-Time Inference on unknown samples

Neural SfM

TOYOTA's approach (2020) [GUI2020]

The Pack-Net only **needs one image for inference** and predicts a depth-map and the camera trajectory, also it is trained in an unsupervised manner. Unlike Google's approach Pack-Net is able to inference in **real-time**.

Neural SfM

TOYOTA's approach (2020) [GUI2020]

Pack-Net implementation:

- Depth Network (estimates the depth-map)
- Pose Network (estimates the camera-pose)

Neural SfM

TOYOTA's approach (2020) [GUI2020]

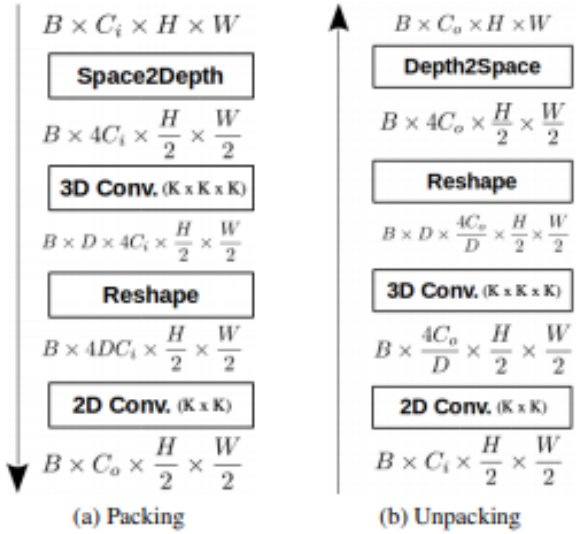
Pack-Net implementation: Depth-Net (training)

Depth-net aim to learn a monocular depth model $f_D: \mathbf{I} \rightarrow D$ that predicts the depth $D = f_D(\mathbf{I}(\mathbf{p}))$ for every pixel $\mathbf{p}$ in the target image $\mathbf{I}$.

Neural SfM

TOYOTA's approach (2020) [GUI2020]

Depth-net does 3D packing-unpacking (downsample - upsample).



[GUI2020] Decoder-Encoder type-of network

Neural SfM

TOYOTA's approach (2020) [GUI2020]

Depth-net takes a stereo image, transforms the right image plus a predicted depth of the left image into a synthesized left image. A **fully differentiable loss** is then defined between the left and the synthesized left image.

Neural SfM

TOYOTA's approach (2020) [GUI2020]

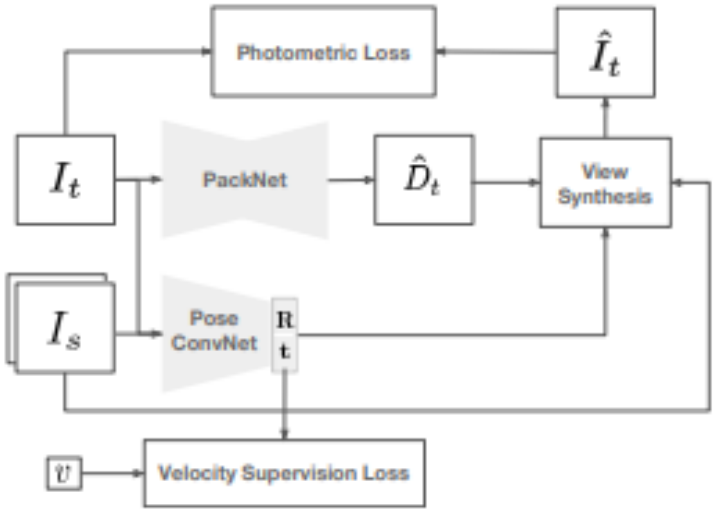
Pack-Net implementation: Pose-Net (training)

Pose-Net aim to learn a monocular ego-motion estimator $f_X: (\mathbf{I}_t, \mathbf{I}_S) \rightarrow \mathbf{X}_{t \rightarrow S}$ that predicts the 6-DoF rigid transformations between the target image $\mathbf{I}_t$ and the set of source images $\mathbf{I}_S$. The transformations to be found are translation in 3D space and rotation respectively.

Neural SfM

TOYOTA's approach (2020) [GUI2020]

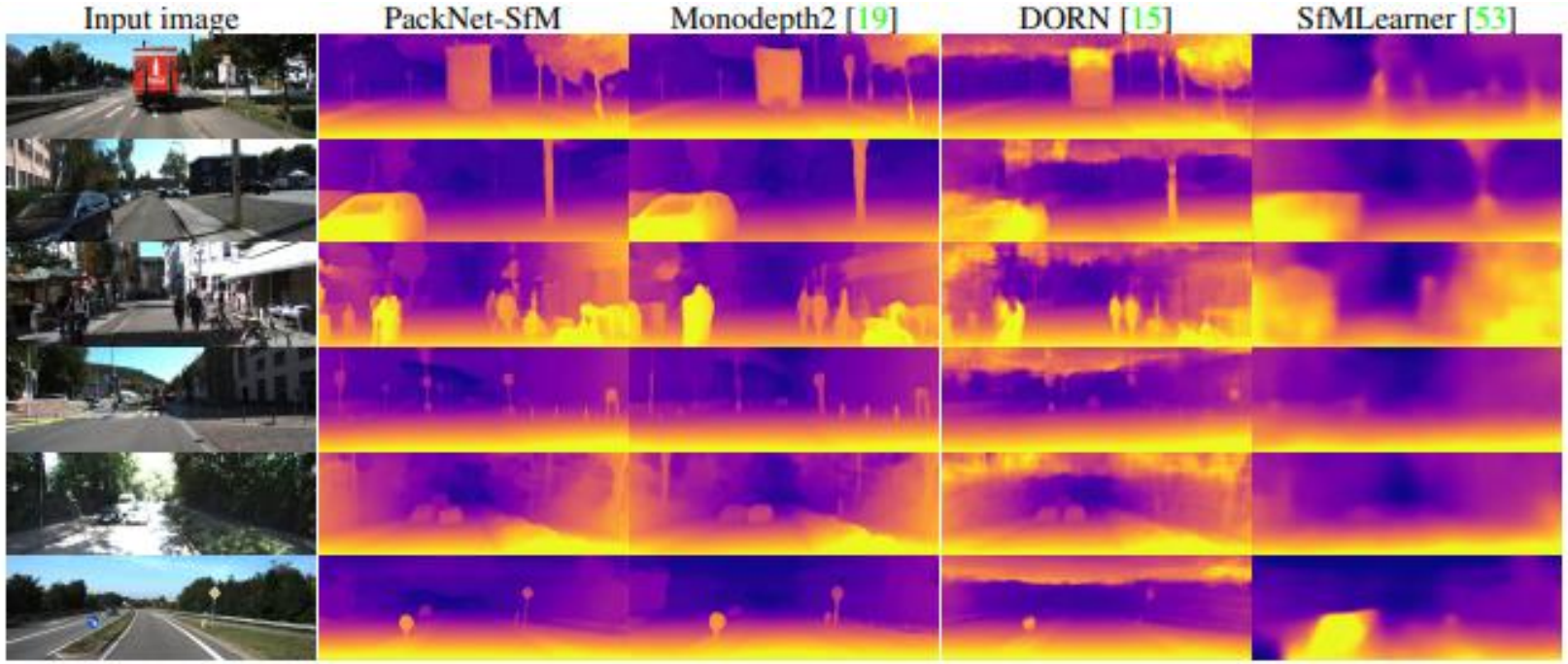
These two networks are trained simultaneously in the following manner:



[GUI2020] Algorithmic Pipeline

Neural SfM

TOYOTA's approach (2020) [GUI2020] – PERFORMANCE



[GUI2020] Comparison with [ZHO2017] and other Depth-Map methods

Neural SfM

SLAM algorithms utilize certain features that differentiate them from SfM.

- SLAM **must** operate **online** (real-time) while SfM usually operates with **batch-processing** (post-process)
- SLAM uses information from past-states, with a **memory** component. (Can be $\mathcal{M}$**:**map, past $\boldsymbol{X}_n$ estimations etc.)
- Complete SLAM systems utilize a Navigation Policy $\pi(.)$ (**active**), while SfM doesn't need to navigate (**passive**).

Deep Learning in Computer Vision

- Neural Camera Calibration
- Neural Localization
- Neural Mapping/Reconstruction
- Neural SfM
- **Neural SLAM**

Neural SLAM

A complete Neural SLAM system [CHA2020] is **a multi-module pipeline**. The different modules synergize to solve different tasks:

- Reconstruction $\mathcal{M}$
- Localization $\boldsymbol{X}_n$
- Memory (Can be $\mathcal{M}$**:** map, past $\boldsymbol{X}_n$ estimations, past $\boldsymbol{X'}_n$ sensor readings)
- Navigation policy $\pi(.)$ (because its **active**)

Neural SLAM

Definition of the variables/ sizes in **SLAM** problem:

$\mathbf{z}_n$: Robot r observation at discrete time-step n.

a_n : Action taken by the agent at discrete time-step n.

$\mathbf{X}_n$: Position of the agent at discrete time-step n.

Can be up to 6DoF when $\mathbf{X}_n = (X_\mathrm{r}, Y_\mathrm{r}, Z_\mathrm{r}, R_\mathrm{X}, R_\mathrm{Y}, R_\mathrm{z})$, r: robot

$\mathcal{M}$: The map, gets updated each n with $\mathbf{m}_n$.

Neural SLAM

SLAM refers to the problem of Simultaneous localization and mapping, where the agent needs to **reconstruct** a representation of an unknown environment $\mathcal{M}$ (a map) and simultaneously **localize** $\boldsymbol{X}_n$ itself in it, using **observations** $\mathbf{z}_n$ for each **discrete** time-frame n**.**

Neural SLAM

Research has shown [CHA2020] that in most exploration tasks Neural SLAM is more efficient when used **Actively**. As stated before, **active** navigation means that the agent is capable of having control over its future observations.

An example run of a SLAM algorithm, starting at $n = 1$:

$$n = 1\,, \quad \mathrm{S}(\mathbf{z}_1) \rightarrow \boldsymbol{\mathcal{M}}_1 + \mathbf{m}_1, \mathrm{L}(\mathbf{z}_1, a_1) = \boldsymbol{\mathcal{M}}_2, \mathbf{X}_2$$
$$n = 2\,, \quad \mathrm{S}(\mathbf{z}_2) \rightarrow \boldsymbol{\mathcal{M}}_2 + \mathbf{m}_2, \mathrm{L}(\mathbf{z}_2, a_2) = \boldsymbol{\mathcal{M}}_3, \mathbf{X}_3$$

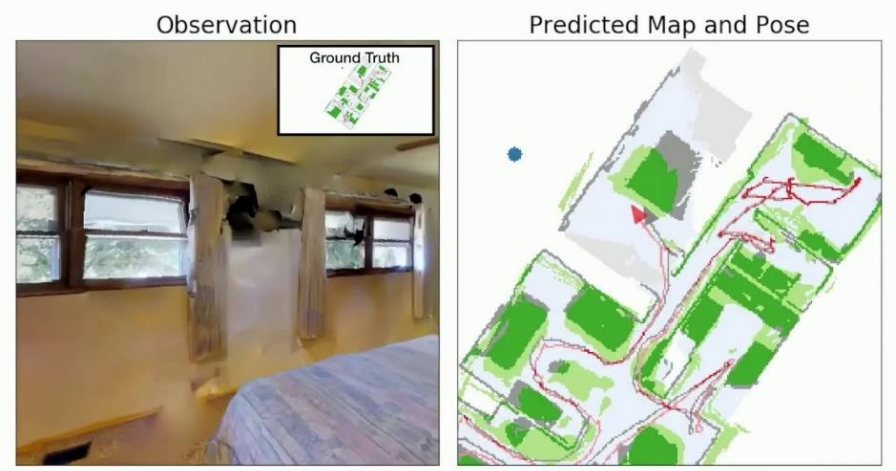
Neural SLAM

In traditional geometric SLAM there is a big variety of sensors we can use to perceive the environment including LIDAR, RADAR, CAMERAS etc.

However in the case of **Neural SLAM** we will focus on **visual** passive sensors such as camera's, since they are the most cost efficient and effective, paired with **Deep Learning**.

Neural SLAM

This is an output of experiments on [CHA2020] ‘Neural Active Slam 2020’, an agent trained on domestic environments, **Reconstructs** the environment (with green) and **Localize** itself + past states (red traces).



My experiments on [CHA2020]

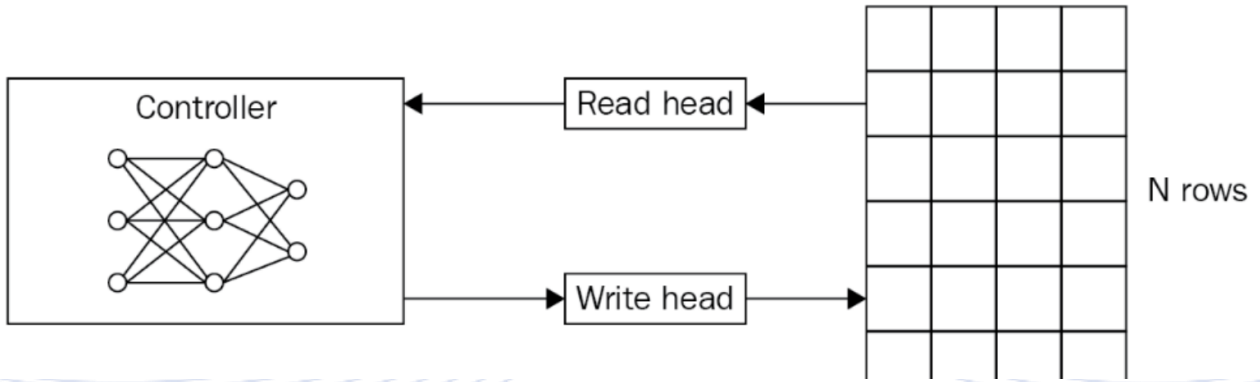
Neural SLAM

In October of 2016, DeepMind introduced the term DNC in their paper “Hybrid Computing using A Neural Network with dynamic external memory” [GRA2016].

DNC is an acronym for “**Differential Neural Computer**” and it solves the problem of external memory on neural networks.

Neural SLAM

DNC's and **Neural Turing Machines** [GRA2014] have the ability to store their weights in external matrices, and do read's and write's between the entries and the actual weights using a neural controller (another neural network).

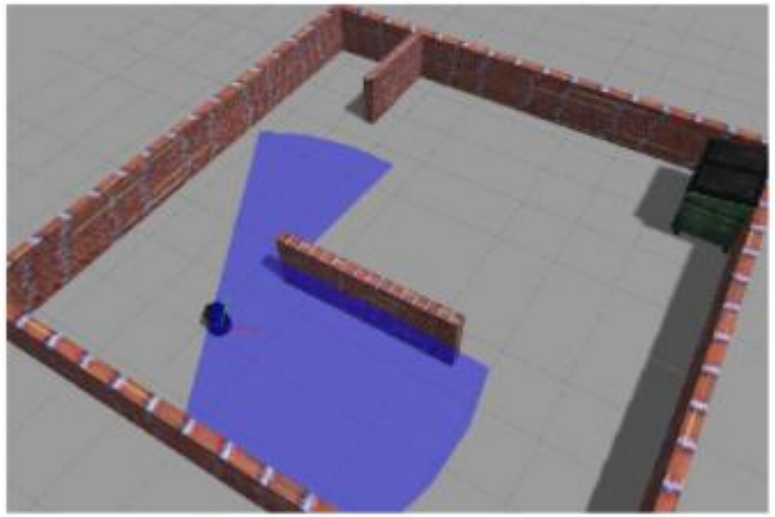


[GRA2014] How external memory works

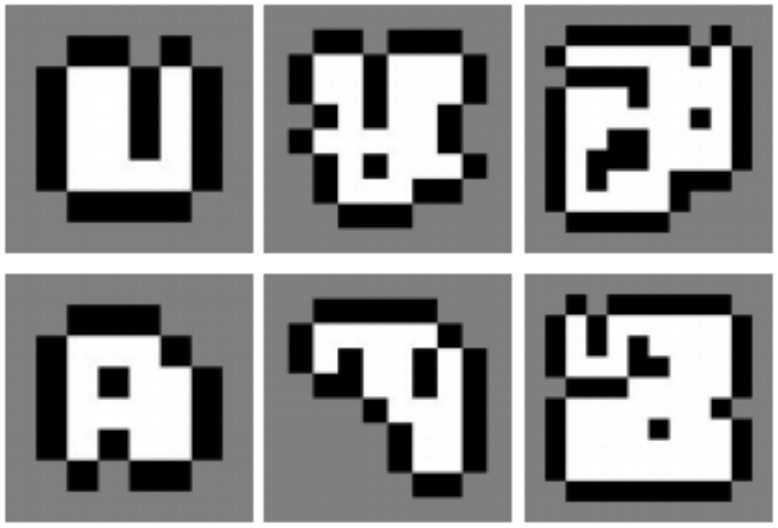
Neural SLAM

All of the above motivated the publication of [ZHA2017] :

Neural SLAM-Learning to Explore with External Memory



[ZHA2017] Gazebo



[ZHA2017] 2D Generated Map

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

- The agent's can make long term decision based on an internal representation of a global map.
- Slam Model and Path Planning modules are deeply integrated as a whole, taking each other into account benefiting from learning alongside each other.

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

At timestep $n \in [0, N]$ the agent is at state $\mathbf{S}_n,\ \boldsymbol{s}_n \in \boldsymbol{S}$
Then selects an action $\mathbf{A}_n,\ \boldsymbol{a}_n \in \mathbf{A}$ based on the policy $\pi(.|\boldsymbol{s}_n)$ which corresponds to a motion command for the agent.

Agent receives reward signal $\mathbf{R}_{n+1} \in \mathbb{R}$ and goes to the next state $\boldsymbol{S}_{n+1}$.

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

The goal for the agent is to maximize the expected return .

This research utilized the **A3C** algorithm as a backbone Deep RL method, and the **GAE** to learn optimal policies.

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

They added an external memory chunk **M** of size $H\ x\ W\ x\ C$ containing $H\ x\ W$ memory slots (basically an array) with C channels.

This can be accessed by the Deep RL Network via a write head and a read head as stated in DNC's and NTM's.

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

Compared to the A3C without any memory, Neural Slam had significantly greater success ratio over random generated 2D map exploration.

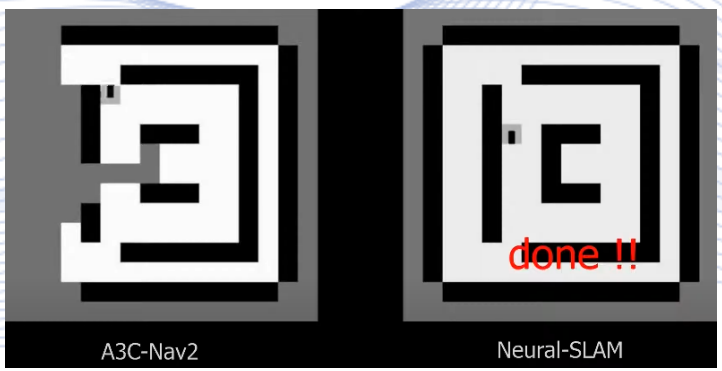
	Steps	Reward	Success Ratio
Random	5531.600 ± 4299.554	-596.644 ± 505.436	-
A3C	333.780 ± 300.098	4.373 ± 16.880	33/50
A3C-Nav1	290.500 ± 275.228	6.938 ± 15.639	37/50
A3C-Nav2	283.480 ± 279.098	7.196 ± 15.566	37/50
A3C-Ext	569.640 ± 272.931	-8.127 ± 15.408	18/50
Neural-SLAM	174.920 ± 174.976	13.732 ± 9.839	46/50

[ZHA2017] Results

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

Except the explored area success ratio, bellow you can see that Neural SLAM is **faster** too, due to the fact that “it remembers” its past states and decisions.

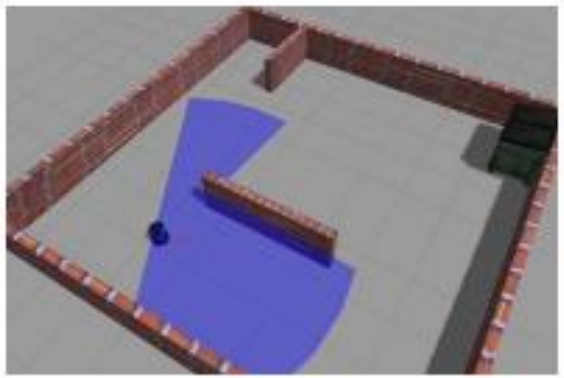


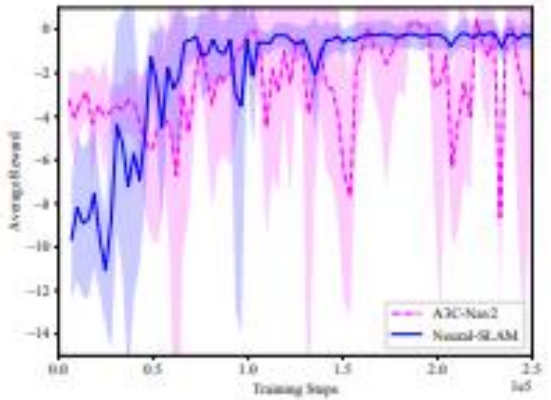
[ZHA2017] Results

Neural SLAM

Neural SLAM - Learning to Explore with External Memory:

Identical results occurred in a Gazebo 3D modelled environment where Neural Slam navigated more efficiently.





[ZHA2017] Results

Neural SLAM

SLAM and SfM algorithms as you have seen by now can be evaluated by a number of metrics.

The last technique emphasized on the **exploration efficiency** and the **time** of the algorithm, which relates with more **theoretical** SLAM scenarios.

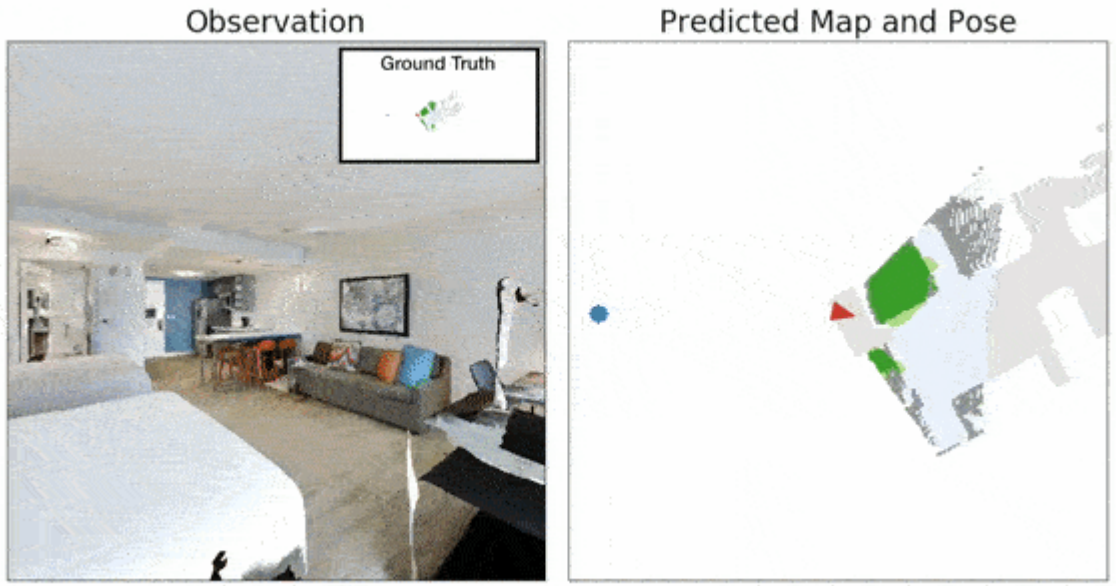
Neural SLAM

Other modern Deep Learning SLAM techniques emphasize on Reconstruction/Localization accuracy, sensor noise models etc., which is more practical and applicable in real world scenarios.

A major example is : **Learning to Explore using Active Neural SLAM** 2019 [CHA2020]

Neural SLAM

Learning to Explore using Active Neural SLAM



[CHA2020]

Neural SLAM

Learning to Explore using Active Neural SLAM

- This particular implementation can be utilized in real world scenarios.
- Can be robust to sensor input noises.
- Won the CVPR 2019 Navigation Challenge https://aihabitat.org/challenge/2019//

Neural SLAM

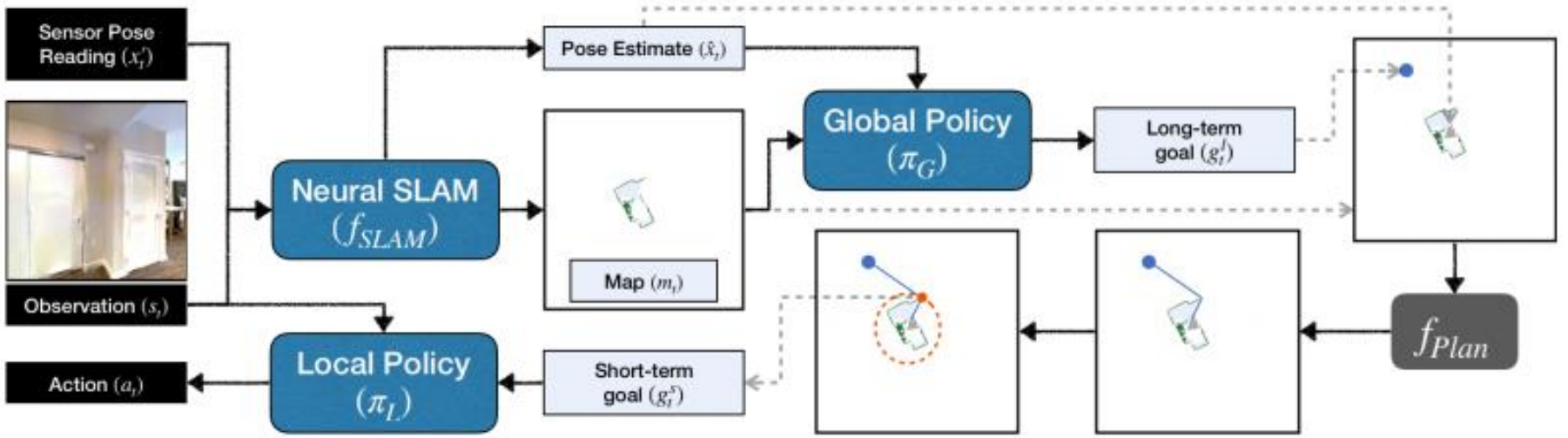
Learning to Explore using Active Neural SLAM

The navigation model has 3 components:

- Neural Slam Module (predicts map and pose of agent)
- Global Policy (long term goal)
- Local Policy (each of short term goals)

Neural SLAM

Learning to Explore using Active Neural SLAM



[CHA2020] Pipeline

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

Takes as input:

- The current RGB observation $\mathbf{z}_n$,
- The current and the last **sensor reading** of agent pose $\mathbf{X}'_{n-1:n}$
- Last agent pose and map **estimates** $\mathbf{X}_{n-1}$, $\mathbf{m}_{n-1}$

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

And outputs:

- An **updated map** $\mathcal{M}_n$ with the addition of $\mathbf{m}_{n-1}$
- The current agent **pose estimate** $\mathbf{X}_n$

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

All together mathematically formulated:

$$\mathbf{m}_n, \mathbf{X}_n = f_{SLAM}(\mathbf{z}_n, \mathbf{X}'_{n-1:n}, \mathbf{X}_{n-1}, \mathbf{m}_{n-1} | \boldsymbol{\theta}_\mathbf{Z})$$

Where θ_S is all the trainable parameters of the f_{SLAM}

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

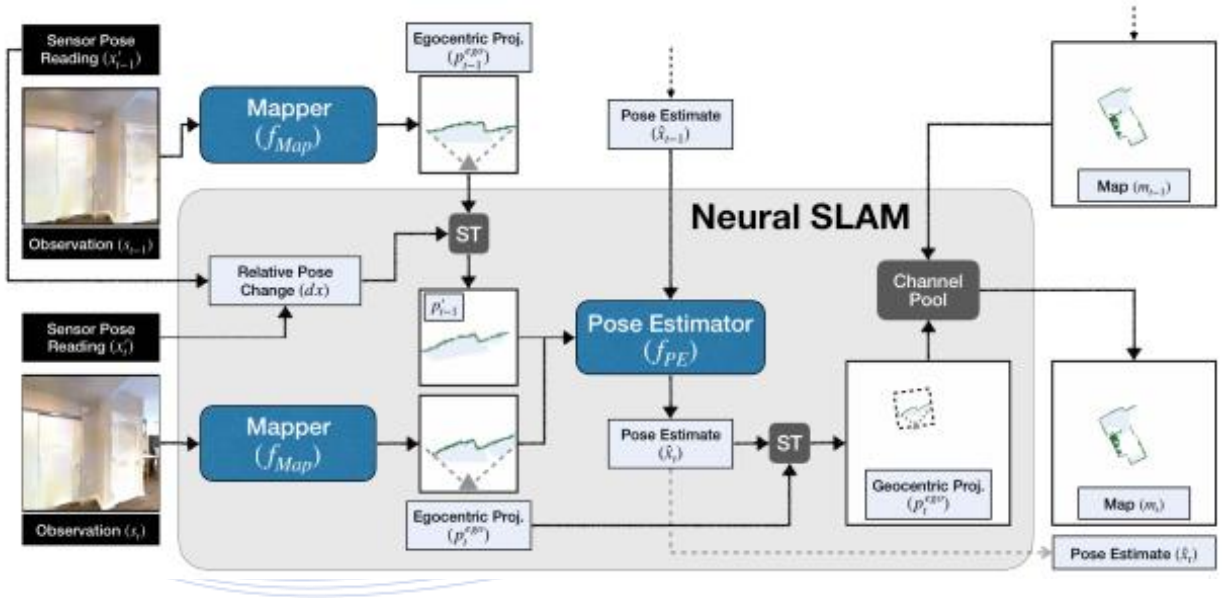
The subcomponents which are ‘learned’ are the:

- Mapper (produces top-down egocentric 2D representation)
- Pose Estimator (predicts the pose)

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:



[CHA2020] Neural Slam Module

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

The **mapper** $f_{MAP}(\mathbf{z}_n|\boldsymbol{\theta}_\mathbf{M})$ consists of:

- Resnet18 Convolutional Layers
- 2 fully connected
- 3 deconvolutional layers

Neural SLAM

Learning to Explore using Active Neural SLAM

Neural SLAM Module:

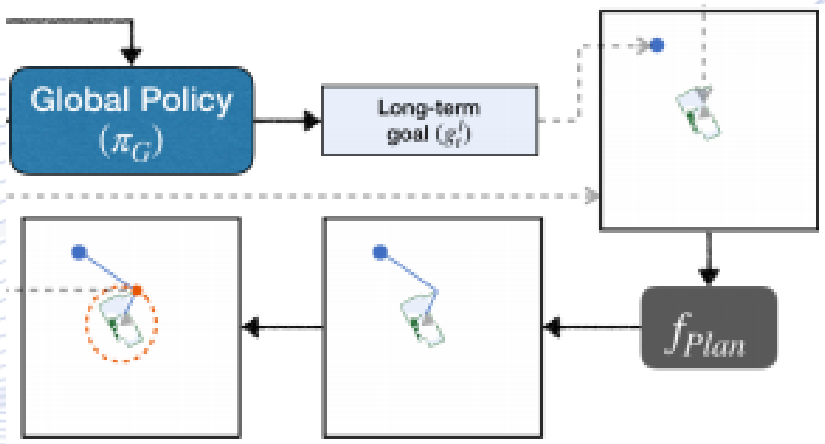
The **pose estimator** $f_{PE}(p'_{t-1}|f_{MAP}(\mathbf{z}_t|\boldsymbol{\theta}_{\mathrm{M}}))$ consists of:

- 3 Convolutional layers
- 3 fully connected layers

Neural SLAM

Learning to Explore using Active Neural SLAM

Global Policy: is the long term goal (blue dot)



[CHA2020]

Neural SLAM

Learning to Explore using Active Neural SLAM

Global Policy:

Takes as input :

- $\mathbf{m}_n$: The predicted map
- $\mathbf{x}_n$: The predicted location of agent

Neural SLAM

Learning to Explore using Active Neural SLAM

Global Policy:

Also takes as input the **visited locations** which are saved, in order to plan the next long-term visit point and **output** it.

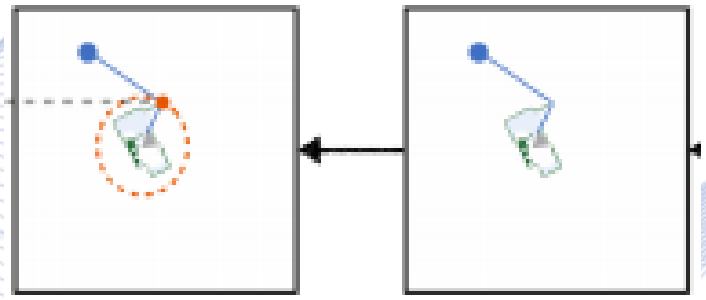
Neural SLAM

Learning to Explore using Active Neural SLAM

Local Policy: (action to be taken)

Takes as input:

- The current RGB observation $\mathbf{z}_n$
- The short term goal $\mathbf{g}_n$ (vector/point)



[CHA2020]

Neural SLAM

Learning to Explore using Active Neural SLAM

Local Policy: (action to be taken)

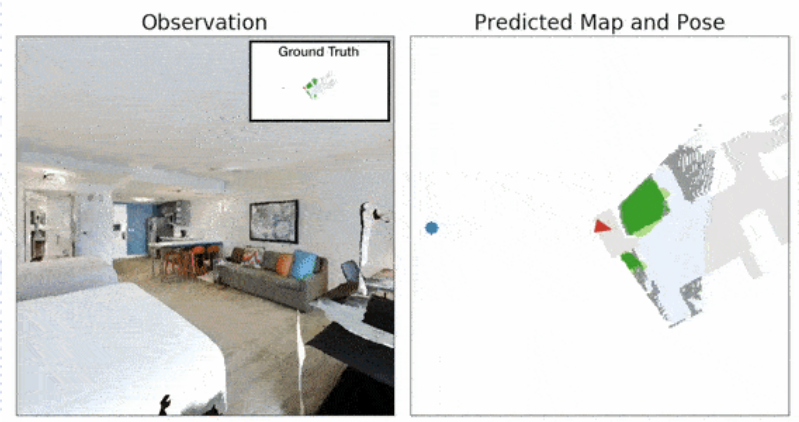
Outputs:

- The action to be taken $a_n = \pi_L(\mathbf{z}_n, \mathbf{g}_n|\boldsymbol{\theta}_L)$, where $\boldsymbol{\theta}_L$ are the parameters of the Local Policy.

Neural SLAM

Learning to Explore using Active Neural SLAM

Is the state-of-the art right now in habitat environment exploration in terms of accuracy and speed.



[CHA2020]

Neural SLAM

Learning to Explore using Active Neural SLAM

The **code** and the best **pre-trained models** are publicly available in their GitHub page:

https://github.com/devendrachaplot/Neural-SLAM/

Bibliography

[PIT2021] I. Pitas, “Computer vision”, Createspace/Amazon, in press.
[PIT2017] I. Pitas, “Digital video processing and analysis” , China Machine Press, 2017 (in Chinese).
[PIT2013] I. Pitas, “Digital Video and Television” , Createspace/Amazon, 2013.
[NIK2000] N. Nikolaidis and I. Pitas, “3D Image Processing Algorithms”, J. Wiley, 2000.
[PIT2000] I. Pitas, “Digital Image Processing Algorithms and Applications”, J. Wiley, 2000.

Bibliography

[LEE2020] Lee, Jinwoo, et al. "Neural Geometric Parser for Single Image Camera Calibration." *European Conference on Computer Vision*. Springer, Cham, 2020

[ITU2017] Itu, Razvan, Diana Borza, and Radu Danescu. "Automatic extrinsic camera parameters calibration using convolutional neural networks." *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2017.

[LOP2019] Lopez, Manuel, et al. "Deep single image camera calibration with radial distortion." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019

[BUI2013] Bui, Trung Hieu, Eitaku Nobuyama, and Takeshi Saitoh. "A texture-based local soft voting method for vanishing point detection from a single road image." *IEICE TRANSACTIONS on Information and Systems* 96.3 (2013): 690-698.

Bibliography

[KON2009] Kong, Hui, Jean-Yves Audibert, and Jean Ponce. "Vanishing point detection for road detection." *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[SCH2020] Schops, Thomas, et al. "Why having 10,000 parameters in your camera model is better than twelve." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

[ZHO2017] Zhou, Tinghui, et al. "Unsupervised learning of depth and ego-motion from video." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017

[GUI2020] Guizilini, Vitor, et al. "3D Packing for Self-Supervised Monocular Depth Estimation." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020.

Bibliography

[SCH2016] Schonberger, Johannes L., and Jan-Michael Frahm. "Structure-from-motion revisited." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016

[CHA2020] Chaplot, Devendra Singh, et al. "Learning to explore using active neural slam." *arXiv preprint arXiv:2004.05155* (2020).

[GRA2016] Graves, Alex, et al. "Hybrid computing using a neural network with dynamic external memory." *Nature* 538.7626 (2016): 471-476.

[GRA2014] Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." *arXiv preprint arXiv:1410.5401* (2014).

[ZHA2017] Zhang, Jingwei, et al. "Neural slam: Learning to explore with external memory." *arXiv preprint arXiv:1706.09520* (2017).

Bibliography

[CHA2018] Chaplot, Devendra Singh, Emilio Parisotto, and Ruslan Salakhutdinov. "Active neural localization." *arXiv preprint arXiv:1801.08214* (2018).

.

Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

Contact: Prof. I. Pitas
pitas@csd.auth.gr