Smart 3D Camera Pose Estimation for Complex Augmented Reality Environments Using Deep Learning Techniques

Anonymous 3DV submission

Paper ID ****

Abstract

In recent years, interest in augmented reality (AR) has grown steadily for industrial applications. Today, AR is one of the pillars of Industry 4.0. One of the major problems of AR is the localisation of the display device. Indeed, to obtain an accurate real-virtual worlds registration, it is necessary to efficiently estimate the camera pose at each frame and in real-time. In this paper, we combine a deep learning solution within a geometric approach to estimate the camera poses from RGB images. We designed a Convolutional Neural Network (CNN), called 3DNet, which allows to regress 3D coordinates from a set of RGB patches. In order to extend our approach to unlearned and therefore completely unknown environments, we have implemented a transfer learning approach to regress the 3D point coordinates using a pre-trained model. We evaluated the performance of our approach on two databases: the public 7scenses database from Microsoft, and our own RGB-D database created using the Intel RealSense D435i camera. We compared the results of our approach with those of state-of-the-art. We obtained better results in both 3D points prediction and camera pose estimation.

1. Introduction

The problem of estimating the camera pose in AR is all the more difficult to solve when the working environment is complex (e.g. industrial environment), with strong constraints: highly variable and uncontrollable lighting conditions, presence of reflective materials (metals) and absence of colours or textures, changing environment (movement of objects).

Classical AR methods do not work well in these conditions and do not provide accurate augmentations. Approaches based on visual SLAM partly address these issues [6, 17], but have a significant limitation related to the quality of the detection and matching of points of interest in the images. On the other hand, machine learning is considered as an efficient tool to deal with computer vision problems. Several approaches in the literature propose to formulate the 3D tracking problem as a learning problem where the goal is to learn the relationship between a pair of images and the relative movement of the camera or the object. Kendall *et al*. [15] presented monocular six degree of freedom relocalization system that trains a deep CNN to regress the 6 Degree Of Freedom (6-DOF) camera pose from single RGB images. They achieved efficient real time results using a transfer learning [22] from a large scale classification data by redesigning a classification problem into a pose regression problem. PoseNet [15] consists of a 23 layer deep convnet which outputs a pose vector of 7-dimensions representing position (3) and orientation (4).

Besides, the hybrid approaches consist of combining machine learning and geometric approach to estimate the pose of the camera in the environment. Machine learning part aims to learn the 3D coordinates in world reference system from a set of 2D pixel inputs. The resulting 2D-3D correspondences are used to infer the camera pose using a geometric approach. Duong *et al*. in [7] proposed a light Convolutional Neural Network called *xyzNet* that regresses the 3D world coordinates of detected key-points in an image. The learning data consists of a set of 2D RGB extracted patches in the image and their 3D world coordinates calculated using depth information of these specific key-points. The 2D-3D correspondences are used to calculate the camera pose in the scene, by removing some ambiguous information using the a geometric approach to obtain more accurate camera pose.

In this paper, based on *xyzNet* [7], we propose a hybrid approach to estimate and refine the camera pose using Convolutional Neural Network to regress 3D world coordinates from a set of 2D RGB patches, a geometric approach to estimate the pose of a camera in a scene, and an optimization algorithm to refine the obtained results. Compared to *xyzNet* network we changed the CNN's architecture and parameters in the learning phase using Batch Normalization [12] layers, *Adam* [16] optimizer and the Mean Squared Error regression loss function. We also provide a transfer

learning solution to regress 3D world coordinate system using the *ResNet* [11] convolutional layers and their learned weights. We replace the top layers used for classification by linear regression layers. In the part of camera pose estimation we use the Perspective-n-Point (PnP) and Random sample consensus (RANSAC). The obtained camera pose is then refined using the Levenberg-Marquardt algorithm [20] to obtain more accurate poses. We tested our solution on the 7-scenes dataset [25] and compared our results with the existing approaches. We also created our own RGB-D dataset to test and validate our solution in real time and see its behaviour on less textured scenes. We have demonstrated that our approach can accurately register a 3D model with its image for AR applications, which is another contribution of this paper.In the coming sections, we explain our approach and show in more details its difference from the *xyzNet* [7] solution. We present the experiments carried out as well as the comparative study of the obtained results with those of the state of the art.

The rest of the paper is organized as follows. Section 2 gives the architecture of our solution and describes the proposed data flow. Section 3 details the experimental framework used to evaluate the performance. Section 4 is devoted to the analysis of real-time behavior in an AR scenario. Section 5 describes the proposed transfer learning solution. Finally, Section 6 draws conclusions and the future work.

2. Method Overview

Our approach to estimate the 3D camera pose is composed of three main parts. First, extract 2D patches from a single RGB image and calculate their 3D world coordinates, then train our 3DNet network using this set of 2D-3D features. Second, implementing an efficient and robust network model to well regress the 3D world coordinates. Third, efficiently estimate the 3D camera pose in the learned environment using computer vision algorithms. Figure 1 shows the pipeline describing the data flow of our solution.

2.1. Patches extraction and 3D calculation

Several deep learning approaches for camera pose estimation [4, 13, 15, 27] aim to estimate the pose from the whole input RGB image. In this work, inspired by *xyzNet* [7], we use RGB-D patches around detected key-points in the image to regress the 3D world coordinates of the pixels. From the RGB-D image we extract patches with significant information using SURF [2] which is scale- and rotation-invariant detector and descriptor. We reduce the presence of homogeneous patches present on the flat surfaces *e.g.* ground, wall and sky which causes some ambiguous information about the 3D world coordinates. We adjust the hessian threshold to preserve more salient interest points.

From each RGB image we chose a region of interest (ROI) with a fixed size f around the detected key-point p_i which represents our patch $\mathcal{P}_i$. For each RGB-D patch $\mathcal{P}_i$ of the shape $(f, f, 3)$ centered at key-point with pixels $p_i = (p_i^x, p_i^y)$ and the corresponding depth $D_i \neq 0$ we calculate the 3D position in the camera reference frame $P_i^c = (X_i^c, Y_i^c, Z_i^c)$ using the standard pinhole camera model as follows :

$$P_i^c = D_i K^{-1} \begin{bmatrix} p_i \\ 1 \end{bmatrix} \tag{1}$$

Where K is the camera calibration matrix containing the intrinsic parameters. Knowing the rotation matrix R and translation vector t for each frame, the world coordinates $P_i^w = (X_i^w, Y_i^w, Z_i^w)$ of a pixel p_i are calculated using the transformation equation as follows :

$$P_i^w = R^{-1} P_i^c - Rt \tag{2}$$

We note that the images used for this process are obtained from a sequence of images acquired with a previously calibrated RGB-D camera. The 2D-3D points matching thus generated include geometric information of the camera and the scene. Their repetitiveness will allow our *3DNet* to estimate the camera pose parameters on an unseen data of the learned environment.

2.2. Our 3DNet architecture

3DNet is a Convolutional Neural Network to regress 3D points from RGB image patches. Inspired by the work of Duong *et al.* [7] we designed a regression neural network to achieve this 3D points prediction process from 2D pixels. *3DNet* takes RGB-image patches with a size of 50×50 pixels as inputs. The convolutional layers learn the features found in patches using filtering and pooling operations. A flatten vector of learned features serves to fit a Multi-Layer Perceptron (MLP) to regress 3 outputs corresponding to the 3D coordinates given in the world coordinate system. *3DNet* is composed of five convolutional blocks. Unlike the *xyzNet* [7] we add in each block layers that perform convolution with a set of filters of 3×3 kernels, max-pooling and sub-sampling over a 3×3 area and a rectified linear (ReLU) activation function. We replace local response normalization layers (LRN) by batch normalization layers (BN) which are trainable layers. This normalization technique aims to normalize activation in intermediate layers of deep neural networks. According to Singh *et al.* [26] its proclivity to improve precision and speed up training, BN has become a popular DL technique. The learned features by convolutional blocks are flattened to a Multi Layer Perceptron which is composed of two fully connected layers followed by a dropout layer to avoid the over-fitting. The architecture of *3DNet* is illustrated in Figure 2 using Net2Vis [1] visualizer.

We also provide some changes in the model parameters for the training phase. Thus, the weights are learned by

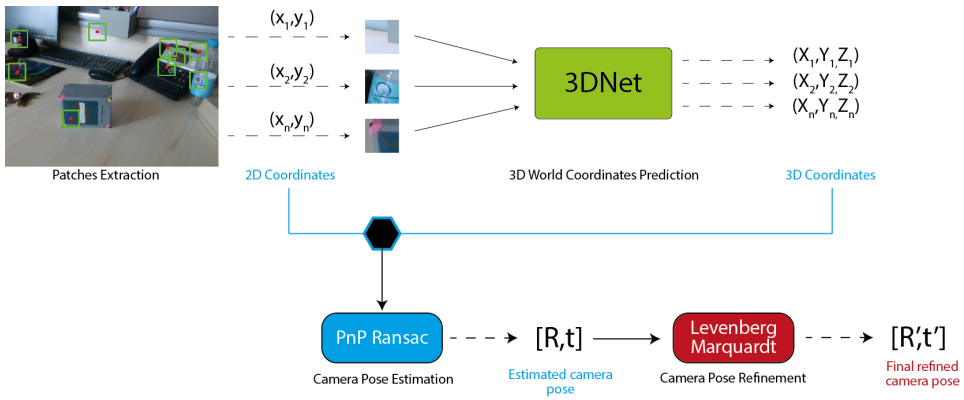


Figure 1: 3DNet's diagram representing the data flow and the different blocks of our solution. From an input RGB image, a set of patches centered around the 2D key-points are extracted, these patches are then passed through the 3DNet model to predict the 3D world coordinates. Next, from the set of 2D-3D correspondences the Pnp Ransac makes a first estimation of the camera pose. Finally the previously estimated camera pose is refined using the Levenber-Marquardt algorithm and we get the final pose of the camera.

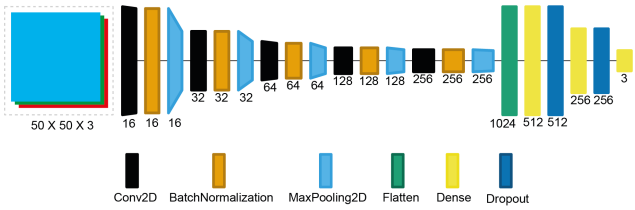


Figure 2: Our CNN : **3DNet** architecture for 3D world coordinates regression.

minimizing the Mean Squared Error (MSE) regression loss function with the Adam [16] optimization algorithm. According to Kingma *et al*. [16], Adam algorithm is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters". The MSE loss function is given as follows :

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(P_i^w - \hat{P}_i^w)^2 \qquad (3)$$

Where P_i^w and $\hat{P}_i^w$ represent the real and the predicted 3D world coordinates respectively, corresponding to m input patches.

2.3. Camera pose estimation

The second part of our solution consists of estimating the camera pose. Once we have a set of detected 2D key-points and their predicted 3D world coordinates using our *3DNet* trained model, we can estimate the camera extrinsic parameters using a geometric approach. Indeed, the estimation of the camera parameters can be efficiently done by resolving the Perspective-n-Points (PnP) problem. The purpose of the PnP problem is to determine the position and orientation of a calibrated camera (known intrinsic parameters) given a set of n correspondences between the 3D and 2D points . Theoretically, the resolution of the PnP problem leads to the resolution of a linear system using 3 control points. Several state-of-the-art research works such as Moreno Noguer *et al*. [21], Quan Long and Zhongdan[23], and Haralick *et al*. [10], uses this approach to estimate the 3D pose parameters of a previously calibrated camera. From a set of input patches, *3DNet* predicts the coordinates of the corresponding 3D points. Some of these points may be noisy, it is then necessary to discard them from the camera pose estimation process. We use PnP and RANSAC (RANdom SAmple Consensus) to remove this noisy information (outliers) and consider only the more correct ones (inliers) to estimate the camera pose. Ransac algorithm, proposed by Fischler et Bolles [9], aims to estimate the parameters of a model from a set of randomly selected points. For each point we calculate the error according to the fitted model. The point that gives an error less than a defined threshold is considered as a good point (inlier). RANSAC ends either if enough inliers are found or if a maximum number of iterations is reached. Finally, the parameters of the model are estimated using all the inliers of the iteration with the highest number of inliers. As mention by Duong *et al*. [7], "The RANSAC generates a set of $\mathcal{T} = \{T_i\}$ poses by performing the PnP on a random set of 2D-3D match points. The best inliers are defined by maximising the number of inliers corresponding to each hypothesis based on the re-projection error".

$$\max_{\forall T_i \in \mathcal{T}} \sum_{p_j \in p} \rho(\alpha_{ij}) \tag{4}$$

Where :

$$\rho(\alpha_{ij}) = \begin{cases} 1, & \text{si } \alpha_{ij} \leq \tau \\ 0, & \text{sinon} \end{cases} \tag{5}$$

and $\alpha_{ij} = ||p_j - KT_i^{-1}\hat{P}_j^w||^2$ and τ is the maximum threshold of the reprojection error which defines the inliers. The pixel j is considered as an inlier of the hypothesis T_i if $\rho(\alpha_{ij}) = 1$. Let $\mathcal{I}$ be the set of indices of inliers associated with the best solution. The final camera pose is performed by running PnP once on all inliers to minimize the re-projection error function (Duong *et al*. [7]) :

$$E(T) = \sum_{i \in \mathcal{I}} ||p_i - KT_i^{-1}\hat{P}_i^w||^2 \tag{6}$$

Our contribution consists of further optimizing the obtained results by PnP-Ransac. We use the Levenberg-Marquardt algorithm [20] to refine the camera pose obtained by the PnP-Ransac algorithm. The method consists of : given a set of 2D-3D matching points and initial parameters (starting solution) the algorithm refines the camera pose by minimizing the re-projection error with respect to the estimated pose, according to a Levenberg-Marquardt iterative minimization process [8, 19].

3. Experiments

In this section, we present the computational environment used to train our *3DNet* model, the databases used for its implementation and evaluation and the results obtained in comparison with existing approaches.

3.1. Experimental environment

Our *3DNet* model is trained on the the High Performance Computing Center Cassiopee platform (HPCassiopee)[1] using 16 Intel(R) Xeon(R) CPUs, and implemented with Keras [5]. All the evaluation tests and results are made on the MacBook Pro with Intel Core i5 2GHz Quad core, 16GB RAM and the Intel Iris Plus Graphics 1536 Mb. We tested different configurations for the learning parameters of our model on the *7scenes* [25] Dataset. We trained our model over 500 epochs with a batch size of 2048, learning rate of 10^{-4}, a dropout of 0.5 using Adam optimizer and a weight decay of 10^{-5}.

3.2. Datasets

In order to train and test our model, we used the *7scenes* [25] and our own created dataset. Each dataset provides a set of RGB images, the Depth information of the scene, the intrinsic and extrinsic camera parameters. The frames of the *7scenes* dataset were recorded from a handheld Kinect RGB-D camera at 640×480 resolution and divided over training and testing data. The RGB and Depth information of our **Box-on-Office** dataset were recorded using the *Intel Realsense D435i* RGB-D camera. The scene consists on a simple objects placed over the office area, as illustrated in Figure 3. The proposed approach for camera pose registration consists of using the KLT algorithm to track the 2D pixels of 4 corners of the front of the box. Knowing their 3D coordinates according to the predefined 3D world reference axis, we use AP3P algorithm [18] to compute the poses of the camera through all frames of the scene. For each frame we register RGB image, Depth image, Pose Matrix, Rotation Vector, Translation Vector, and Depth information in meters. The dataset is divided into 1162 data for training and 587 for testing.

(a) RGB image (b) Depth image

Figure 3: An example of our **box-on-office** dataset's frames information recorded by the intel *D435i* RGB-D camera. On the left, the 640×480 RGB image of the scene containing box, monitor, keyboard, phone, bottle, ... , placed over the office area. On the right, the Depth information of the frame.

3.3. 3DNet points prediction

The learning step of our model consists of passing the extracted RGB patches from the training data and their corresponding 3D labels through the *3DNet* network which makes features extraction by the convolutional blocks and adapt its weights to regress the correct 3D coordinates for reach corresponding RGB patch. The patches extracted from the images are filtered according two criteria, we avoid patches with a wrong depth (zero depth : no-depth detection), and wrong shape (key-points situated near to the image limits). Based on these criteria, we take all the available extracted patches on the image. We note that the number of key-points differs on each RGB image. We will show that the number of key-points can affect the final results, its choice is crucial in order to speed up the pipeline while keeping a good accuracy. The *3DNet* is the most important element of our solution. Indeed, a good prediction of 3D points would allow a better estimation of the camera pose, and vice versa. The accuracy of the *3DNet* model is

[1]The authors acknowledge the Arts et Métiers High Performance Computing Center Cassiopee made available for conducting the research reported in this paper.

	Scene	Chess	Fire	Heads	Office	Pumpkin	Redkitchen	Stairs
xyzNet[7]	Err_P	**0.25m**	**0.19m**	0.14m	0.65m	**0.27m**	0.44m	**0.34m**
	Err_I	**0.13m**	**0.11m**	0.06m	0.26m	**0.11m**	**0.14m**	**0.13m**
Our 3DNet	Err_P	0.39m	0.41m	**0.13m**	**0.30m**	0.46m	**0.34m**	0.49m
	Err_I	0.21m	0.19m	**0.06m**	**0.17m**	0.21m	0.18m	0.22m

Table 1: Comparison of our model predictions to *xyzNet*[7], by evaluating the mean of distance between predictions and ground truths on the set of all predictions (Err_P) and the set of inliers (Err_I).

the key to obtain good results. We use the same parameters to fit the model over all the scenes of our datasets. We found that the repetition of patches containing similar RGB information with different 3D coordinates can create ambiguity in the model behavior. Figure 4 shows loss evolution on both training and testing data according to model learning epochs. The loss final value differs on the different scenes, this is due to the quality of the information given to the *3DNet* during the learning process. Scenes with high presence of homogeneous patches have greater loss value. The number of extracted key-points is not only considered as a key parameter in the model performance, the quality of 2D-3D information influences the quality of the learning process, more good information conducts to better learning accuracy.

We compare first our *3DNet* to *xyzNet* [7]. In table 1 We evaluate the model prediction performance by calculating the distance error between predicted labels and ground truth and compare with the results reported on [7]. We precise that we did not evaluate on all available *7scenes* [25] for both training and testing data, we choose a number of testing frames between 100 and 200 frames from each scene, and we choose only the *seq01* frames for training the model for every scene. Comparing to the reference method we conclude that our model makes good predictions on the testing data. The less textured scenes with less homogeneous information help the model to learn the most important features on the environment and make good predictions. This is the case with *Heads* and *Office* where we achieved good model's predictions.

3.4. Camera pose estimation accuracy and time execution

It is important to get an efficient camera estimation with a reduced execution. This will allow us to use the solution on a real time situation. We evaluate our method in term of camera localization accuracy and run-time. We compare the results to existing methods cited in xyzNet [7] paper. DSAC [3] learns the parameters of a convolutional neural network (CNN) so that models fit robustly to predictions using RANSAC and minimize a task-specific loss function. PoseNet2 [14] improves the performance of [15] with geometrically formed loss functions. Using the training images, [24] creates a 3D point cloud with associated descriptors in the off-line phase and employs a visual vocabulary to speed up feature matching.

7 scenes [25]	Active Search [24]	PoseNet2 [14]	DSAC [3]	xyzNet [7]	Our 3DNet
Chess	0.04m,2.0°	0.13m,4.5°	**0.02m,1.2°**	**0.06m,2.4°**	0.96m,2.7°
Fire	**0.03m,1.5°**	0.27m,11.3°	0.04m,1.5°	**0.06m,2.2°**	0.95m,3.4°
Heads	**0.02m,1.5°**	0.17m,13.0°	0.03m,2.7°	0.08m,4.8°	**0.08m,2.5°**
Office	0.09m,3.6°	0.19m,5.6°	**0.04m,1.6°**	**0.13m**,5.5°	1.39m,**2.7°**
Pumpkin	0.08m,3.1°	0.26m,4.8°	**0.05m,2.0°**	**0.06m,2.0°**	0.83m,3.1°
Kitchen	0.07m,3.4°	0.23m,5.4°	0.05m,2.0°	**0.05m,1.8°**	0.19m,1.8°
Stairs	**0.03m,2.2°**	0.35m,12.4°	1.17m,33.1°	0.21m,6.0°	**0.24m,1.9°**
Average	**0.05m,2.5°**	0.23m,8.1°	0.20m,6.3°	**0.09m**,3.5°	0.67m,**2.6°**

Table 2: Comparison of the median pose errors with the state-of-the-art methods on 7scenes dataset.

7 scenes	**Key points per frame** (N_{kp})	**Inliers per frame** (N_I)	**Surf Patch extraction** (S_t)	**3D Prediction** (P_t)	**Pose Estimation** (PE_t)
Chess	752	70	53ms	1356ms	383ms
Fire	584	41	48ms	1119ms	344ms
Heads	450	33	30ms	800ms	240ms
Office	362	42	35ms	689ms	247ms
Pumpkin	334	46	33ms	609ms	264ms
Kitchen	481	56	40ms	935ms	313ms
Stairs	512	81	37ms	963ms	332ms
Average	482	55	38ms	892ms	296ms

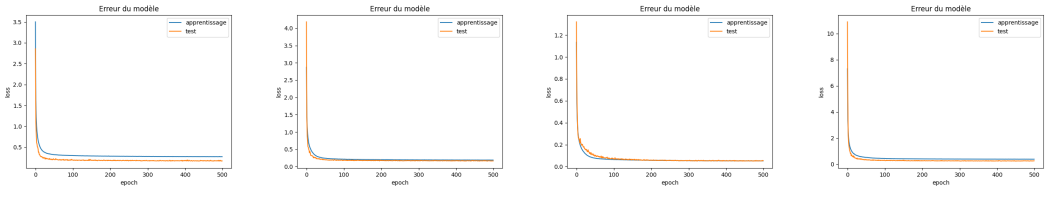
Table 3: Evaluation of the approach in term of key-point and inliers mean number, and the execution time of several modules starting from Surf patches extraction, to model predictions, until the pose estimation.

In table 2 we show the accuracy of pose estimation using our approach. Comparing with the xyzNet [7], we obtain favourable results on the *Heads* and *Stairs* scenes, specially for the rotation pose parameters. Our method has better results over all the scenes in term of rotation comparing to PoseNet2 [14], we outperform the method on both rotation and translation on the *Heads*, *Kitchen*, and *Stairs* scenes. For the DSAC [3], we only obtain better results on the *Stairs* scene in both rotation and translation. The Active Search [24] gives the best results regarding the pose accuracy. Our approach refining the camera pose using RANSAC and Levenberg-Marquardt ranks second in term of rotation with an error average of 2.6°.

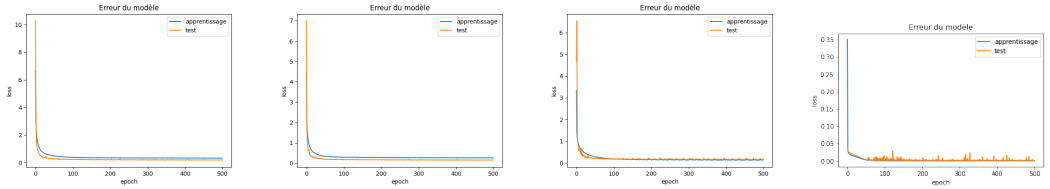
In table 3 we evaluate, on each scene, the mean number of Key-points, the number of inliers per frame, the runtime according to SURF patch extraction, *3DNet* predictions, and Pose estimation time. The results show the effect of the key-points and inliers on the execution time. Indeed, the presence of a large number of key-points includes slower execution time of the model. The pose estimation contains both PnP-RANSAC and LM with 500 iterations. We achieve an average of $1226ms$ for all the pipeline on our environment testing CPU. We show on the next section how we can variate the key-points number and balance between computational time and accuracy.

3.5. Key-points variation and pipeline measures

In this section we evaluate the mean key-points number variation and its effect on the pipeline accuracy. We adjust the hessian threshold in SURF to change the number of detected key-points in the frame. Greater hessian threshold



(a) Chess (b) Fire (c) Heads (d) Office



(e) Pumpkin (f) Redkitchen (g) Stairs (h) Box-on-Office

Figure 4: Loss training and testing evolution on different scenes according to training epochs.

includes less number of key-points. We made the experience on the *Heads* and *Stairs* scenes to see the changing on the pipeline accuracy and run-time. In table 4 by varying the hessian thresh value from 500 to 5000 we see that run-time of all modules (SURF detection, model predictions, and pose estimation) became faster by increasing the thresh value. This is due of the less number of detected key-points and inliers. In terms of accuracy, for the predictions error, we see that the minimum error is $0.12m$ obtained with a hessian thresh between 1500 and 5000. For the inliers error the optimal value is obtained for a hessian thresh value of 500 and 3500 with an error of $0.7m$. Concerning the pose estimation accuracy, the minimal translation error is obtained with a hessian thresh value of $500, 1000, 2500, 4000$ and equals to $0.8m$. The rotation error starts at an error of $2.56°$ with a hessian value of 500 and increases throughout the range $[500, 2000]$ then decreases to an error of $2.78°$ with a hessian thresh of 3000, and increases again through the rest of the hessian values. From these observation, we can balance between execution time and accuracy, to set the value of the Hessian threshold to 3000 to have an execution time of $0.42s$, a translation and rotation error of $0.09m$ and $2.78°$ respectively.

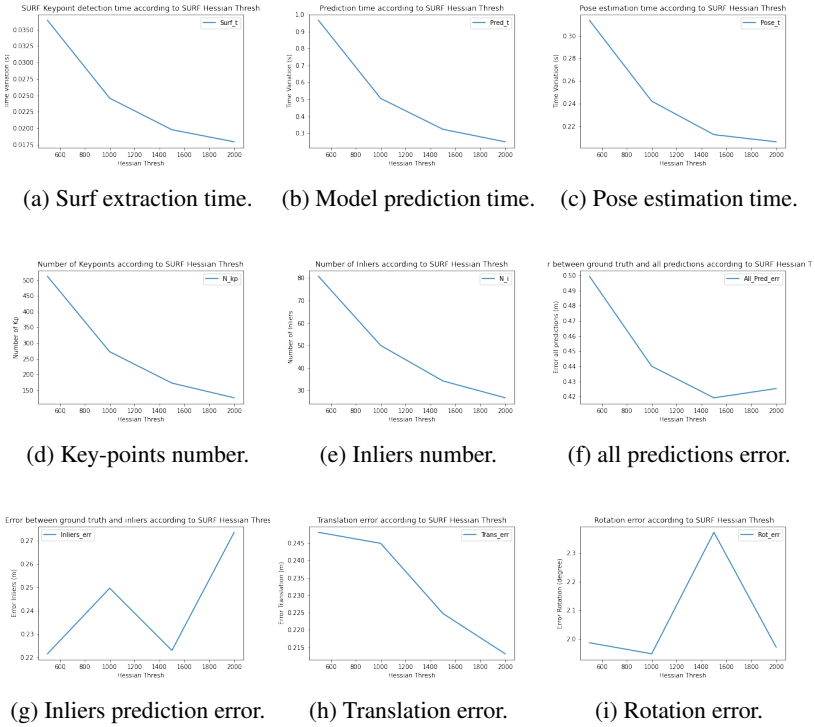
Figure 5 shows the results obtained on the *Stairs* scene. The hessian threshold maximal value is set to 2000. Once this value is exceeded, fewer key points are detected. Therefore, RANSAC cannot find an adequate number of inliers that satisfy the re-projection error criterion and consequently cannot estimate the pose. this is the case on less textures scenes. From these curves, we deduce an optimal threshold for the Hessian that gives a compromise between execution time and accuracy. This optimal threshold is fixed at 2000, it allows to obtain an execution time of $0.48s$, a translation error of $0.21m$ and a rotation error of $1.97°$.

Hessian Threshold	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
SURF time (s)	0.04	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Pose time (s)	0.29	0.25	0.23	0.21	0.20	0.19	0.19	0.18	0.17	0.17
Prediction time (s)	0.78	0.52	0.39	0.30	0.26	0.21	0.17	0.15	0.13	0.11
Key-points number	450	303	226	177	143	118	99	83	71	61
Inliers number	33	26	22	19	16	15	14	12	11	10
All prediction error (m)	0.14	0.13	0.12	0.12	0.12	0.12	0.12	0.12	0.12	0.12
Inliers prediction error (m)	0.07	0.08	0.08	0.08	0.08	0.09	0.07	0.08	0.09	0.10
Translation error (m)	0.08	0.08	0.09	0.10	0.08	0.09	0.10	0.08	0.10	0.10
Rotation error $(°)$	2.56	2.71	2.79	2.99	2.96	2.78	3.04	3.21	3.79	4.14

Table 4: Heads : Pipeline measures variation according to Hessian threshold.

4. 3DNet performances on real-time AR experiment

In this section we made a real-time augmented reality experience. We evaluate our solution on our own created dataset. So, we used our trained model *3DNet* to predict the 3D points of the scene from the SURF points extracted from the current image. Then we estimated the camera pose from the 2D-3D point matching, and finally we generated the augmented reality scene by projecting the 3D model of the box into the current frame. We performed this process on the video stream of the RGBD camera, which allows to obtain a real time tracking. We use the intel *D435-i* camera to get the RGB frames. We tested several scenarios to analyze the behavior of our solution for this case study. First, We evaluated the model on a similar learned environment with similar objects on the scene. Then, we made some modifications on the environment by replacing existing objects, adding new objects to the scene, or removing objects from the scene. Figure 6 shows the results of this experiment. We notice that the registration of the 3D model of the box is well done, whatever the situation. Nevertheless, in



(a) Surf extraction time. (b) Model prediction time. (c) Pose estimation time.

(d) Key-points number. (e) Inliers number. (f) all predictions error.

(g) Inliers prediction error. (h) Translation error. (i) Rotation error.

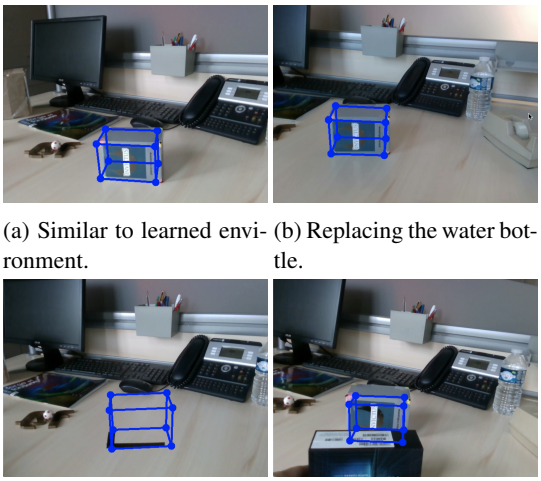
Figure 5: Stairs : Pipeline measures variation curves according to Hessian threshold values.

the case where a new object is added in the scene, we see that the registration is disturbed. This is due to the fact that the scene contains new information on which the model has not been trained during the learning phase.

In table 5 we present the model evaluation on our dataset testing data. We get better results than *7scenes*, in terms of both run-time and model accuracy. This demonstrates the good performance of our approach when used in real time on poorly textured images. We also confirm that reducing the number of key points allows to obtain the same accuracy while decreasing the execution time. Thus, for the scene *Data Test 2* we observe that for a higher Hessian threshold (2000), we obtain the same rotation and translation errors as with a Hessian threshold of 500 but with a prediction time 2 times faster.

5. Transfer learning approach

In this section we present a transfer learning approach for estimating the camera pose. We use a model trained on a classification problem and transfer its learning information to resolve the problem of 3D world coordinates regression. We use the ResNet50 [11] model to extract feature weights from the 2D patches. This pretrained model allows us to use weights that are already adjusted to make predictions. This technique has the advantage of accelerating the learning process, we freeze the weights learned in



(a) Similar to learned environment. (b) Replacing the water bottle.

(c) Removing the box from the office. (d) Adding a second box to the scene.

Figure 6: Performance of our approach on several real situations and environment changes.

the convolutional blocks and we replace the *softmax* layers of the ResNet50 [11] by *linear-regression* layers. The learning process is at the level of the final top layers, we set the same top layers configurations as the *3DNet*. The second advantage of transfer learning consists of using less amount

Scenes	Hessian Thresh	All predictions error (Err_P)	Inliers predictions error (Err_I)	Pose error (P_e)	Key points per frame (N_{kp})	Inliers per frame (N_I)	Surf Patch extraction (S_t)	3D Prediction (P_t)	Pose Estimation (PE_t)
Data Test 1	500	0.027m	0.018m	0.02m,0.03°	263	98	0.039s	0.492s	0.076s
Data Test 2	500	0.04m	0.031m	0.02m,0.08°	368	159	0.045s	0.77s	0.098s
Data Test 2	2000	0.04m	0.030m	0.02m,0.08°	137	66	0.034s	0.29s	0.074s

Table 5: Performance result of our approach on Box-on-Office dataset

of data to train the model.

In this work we fit the ResNet50 [11] on our *box-on-office* dataset using the same learning parameters as the *3DNet*. We achieve with this technique a learning loss of 0.0028 and a validation loss of 0.0143, the model took 2 days of training less than the *3DNet*. We get primary results as follows : $Err_P = 0.11m$, $Err_I = 0.03m$, $R_{err} = 1.05°$, $T_{err} = 0.5m$.

Currently we don't get better results than the proposed *3DNet* model, we plan to pursue our research on the transfer learning approach, by testing the different available models and tune the fitting parameters to get better results. We also plan to make transfer learning research on 3D world coordinates regression models, taking a model that learned from an environment to regress the 3D world information and transfer its weights to another environment and see its behaviour. This will reduce the ambiguities caused from transferring classification weights to regression weights.

6. Conclusion

In this paper we propose a deep learning and geometric approach to efficiently estimate the pose of the camera in complex environment for augmented reality applications. We present a regression convolution neural network that predicts the 3D world coordinates based from RGB input data. We made a real time experiment starting by creating our own RGB-D dataset acquired using an RGB-D camera. We set up a geometric approach to estimate and refine the camera pose using the PnP Ransac and the Levenberg Marquardt algorithm. We tested our solution on real time situation and environement changes, our method gave us good results and performs well. We also initiate another approach to regress the 3D world coordinates using a transfer learning to speed up the learning process. We evaluate and compare our method to state of art methods and show the obtained results in terms of accuracy and run-time.

In future work we aim to use our work on more complex environments in terms of various scenes and angle of views. We also target to improve the transfer learning approach in terms of accuracy and run-time on real time situation.

References

[1] Alex Bäuerle, Christian Van Onzenoodt, and Timo Ropinski. Net2vis–a visual grammar for automatically generating publication-tailored cnn architecture visualizations. *IEEE transactions on visualization and computer graphics*, 27(6):2980–2991, 2021. 2

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008. 2

[3] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. Dsac-differentiable ransac for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6684–6692, 2017. 5

[4] Mai Bui, Christoph Baur, Nassir Navab, Slobodan Ilic, and Shadi Albarqouni. Adversarial networks for camera pose regression and refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 2

[5] François Chollet et al. keras, 2015. 4

[6] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007. 1

[7] Nam-Duong Duong, Amine Kacete, Catherine Sodalie, Pierre-Yves Richard, and Jérôme Royan. xyznet: towards machine learning camera relocalization by using a scene coordinate prediction network. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 258–263. IEEE, 2018. 1, 2, 3, 4, 5

[8] Ethan Eade. Gauss-newton/levenberg-marquardt optimization. *Tech. Rep.*, 2013. 4

[9] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 3

[10] Bert M Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International journal of computer vision*, 13(3):331–356, 1994. 3

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 7, 8

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 1

[13] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE international conference on Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016. 2

[14] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5974–5983, 2017. 5

[15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof cam-

era relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 1, 2, 5

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1, 3

[17] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007. 1

[18] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR 2011*, pages 2969–2976. IEEE, 2011. 4

[19] Yi Ma, Stefano Soatto, Jana Kosecka, and S Shankar Sastry. *An invitation to 3-d vision: from images to geometric models*, volume 26. Springer Science & Business Media, 2012. 4

[20] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978. 2, 4

[21] Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Accurate non-iterative $o(n)$ solution to the pnp problem. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007. 3

[22] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014. 1

[23] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Transactions on pattern analysis and machine intelligence*, 21(8):774–780, 1999. 3

[24] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1744–1756, 2016. 5

[25] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013. 2, 4, 5

[26] Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11237–11246, 2020. 2

[27] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017. 2