Comparative Evaluation of Random Forest and Fern Classifiers for Real-Time Feature Matching

Iñigo Barandiaran[1], Charlote Cottez[1],Céline Paloc[1], Manuel Graña[2]

[1]VICOMTech Paseo Mikeletegi, 57
20009, San Sebastian, Spain
{ibarandiaran, cottez,cpaloc} @vicomtech.org

[2] University of Basque Country
Computer Science School, Pº. Manuel de Lardizabal, 1
20009, San Sebastián, Spain
ccpgrrom@si.ehu.es

ABSTRACT

Feature or keypoint matching is a critical task in many computer vision applications, such as optical 3D reconstruction or optical markerless tracking. These applications demand very accurate and fast matching techniques. We present an evaluation and comparison of two keypoint matching strategies based on supervised classification for markerless tracking of planar surfaces. We have applied these approaches on an augmented reality prototype for indoor and outdoor design review.

Keywords

Feature matching, Tracking by Detection, Augmented Reality.

.

1. INTRODUCTION

The main goal of the Augmented Reality (AR) technology is to add computer-generated information (2D/3D) to a real video sequence in such a manner that real and virtual objects coexist in the same world. In order to get a realistic illusion, the registration problem must be addressed. Real and virtual objects must be properly aligned with respect to each other. In this way, the position-orientation (pose) of the camera relative to a reference frame must be accurately estimated or updated over time. One of the key task for the registration problem to be solved is the keypoint or feature matching. Optimal markerless tracking uses natural features such as edges or corners extracted from images. In this work, we address the registration problem for interactive AR applications, using tracking by detection techniques based on supervised classification techniques (see Fig.2).

Our approach to solve the registration problem is based on tracking of plane surfaces. In indoor or outdoor scenarios, planes are commonly present. The ground, the building facades or walls can be seen as planes. These 3D world planes and its projection in the image are related by homography transformation. By recovering this transformation it is possible to estimate the position and orientation (pose) of the camera.

Keypoint matching is a critical task in many computer vision applications, such as optical 3D reconstruction or optical markerless tracking. As described in [Lep06], we propose to treat wide base-line matching of features points as a classification problem. We have implemented a Random Forest classifier [Bre01] and a semi-Naïve Bayes classifier [Özu07], and carried out an evaluation of both in the context of optical markerless tracking for Augmented Reality applications.

The article is structured as follows. Section 2 gives an overview of current optical tracking techniques and methods in augmented reality applications. Section 3 describes two approaches based on supervised classification. Section 4 gives an overview of the behavior of both approaches. In Section 5 a case study augmented reality application using our implementations is described. Section 6 summarizes some conclusions and future work.

2. RELATED WORK

Although the real-time registration problem using computer vision techniques has received a lot of attention during recent years, it is still far from being solved. Ideally, an augmented reality application should work without the need of landmarks or references for the object or the environment to be tracked. This issue is known as markerless tracking.

We can divide optical markerless tracking technology in two main groups: recursive techniques or model-based techniques. Recursive techniques start the tracking process from an initial guess or a rough estimation, and then refine or update it over time. They are called recursive because they use the previous estimation to propagate or calculate the next estimation. During the estimation process several errors may occur, such as wrong point matching or ill conditioned data that can degenerate the estimation. Due to the recursive nature of this kind of tracking, they are highly prone to error accumulation. The error accumulation over time may induce a tracking failure that requires a re-initialization of the tracking process, which can be cumbersome and not feasible in practical applications.

Other approaches are known as tracking by detection or model-based tracking. In this kind of techniques some information of the environment or the object to be tracked is known a priori. They are also known as model-based tracking because the identification of some features in the images (texture patches or corners) corresponding to a known model are used to recognize such objects. This kind of tracking does not suffer from error accumulation (drift) because, in general, does not rely on the past. Furthermore, they are able to recover from a tracking failure since they are based on a frame by frame detection not depending on the past. They can handle problems such as matching errors or partial occlusion, being able to recover from tracking failure without intervention [Wil07].

Tracking by detection needs information data about the object or objects to be tracked prior to the tracking process itself. This data can be in the form of a list of 3D edges (CAD model) [Vac04], color features, texture patches or point descriptors [Low01]. A good comparison about different point descriptors can be found in [Mik05]. The tracker is trained with a priori data, to be able to recognize the object from different points of view. A good survey about different model-based tracking approaches can be found in [Lep05].

Some authors propose the use of machine learning techniques to solve the problem of wide baseline keypoint matching [Özu06]. Supervised classification systems require a previous pre-processing, in which a system "is trained" with a determined set of known examples (training set) that present variations in all their independent variables. Once the process is finished, the system is trained and ready to classify new examples. Some of the most widely used supervised classifiers are for example, k-Nearest Neighbors, Support Vector Machine or decision trees.

While k-Nearest Neighbors or Support Vector Machine can achieve good classification results, they are still too slow and therefore not suitable for real-time operation [Lep04]. Recently the approach based on decision trees has been successfully applied on tracking by detection during feature point matching task (see Fig.1), by training the classifier to establish correspondences between detected features in a training image and those in input frames [Özu06]. This approach has been also applied to object recognition in [Moo06].

Figure 1: Feature Points recognition and matching.

Figure 2: Example of an outdoor augmented reality scene (San Telmo square in San Sebastian, Spain.)

Based on this recent progress in the field, we integrate two supervised classifiers in the implementation of a tracking module and carry out some evaluation studies.

3. DESCRIPTION

In this section we briefly describe two approaches for feature point recognition based on supervised classifiers that have been recently applied in some real-time implementations [Bof06, Özu07, Wil07]. In these approaches the problem of wide-baseline matching problem is treated as a classification problem.

Random Forest

Random Forest classifiers were firstly introduced by [Ami97]. Recently, such classifier has been applied in optical tracking for interest point matching and recognition [Lep06]. It is able to detect key-point occurrences even in the presence of image noise, variations in scale, orientation and illumination changes. This classifier is a specific variation of a decision tree[Bre01].

When the tree is constructed and trained, it classifies a given data (example) by pushing it down the tree. While the data is going down the tree, a discriminant criteria at every node is deciding to which child the example has to go.

A Random Tree is called random because instead of doing exhaustive search for the best combination of features to be tested at each node, only some random combinations are used. When the number of classes to be recognized and the size of the class descriptor are high, an exhaustive analysis is not feasible. In addition, the examples that are going to be used during the training process are selected at random from the available ones. The combination of some random trees forms a multi-classifier known as Random Forest. One of the advantages of the Random Forest is their combinational behavior. Even when a random tree is poor with a low recognition rate, the combination with other classifiers can generate an efficient one.

3.1.1 Random Forest Training

In supervised classification each class must be defined and labeled before the training process itself. In order to define the classes, we use a point extractor [Rosten06] to get the candidate points and their surrounding patches. Then the classifier assigns a class number to each patch, and their class descriptor is defined. The descriptor of each class is constructed as the intensity values of the pixels that forms the extracted patch centered at interest point *p* (see Fig.3).

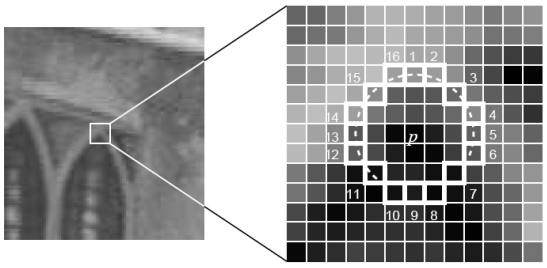

Figure 3: (left) Interest point. (Right) Pixels surrounding interest point *p*. [Ros06].

Once the classes to be recognized by the classifier are defined, the training set must be generated.

As described in [Lepe06], we can exploit the assumption that the patches belong to a planar surface, so we can synthesize many new views of the patches using warping techniques as affine deformations(see Fig.4).These affine transformations are used to allow the classifier to identify or recognize the same class but seen from different points of view and at different scales. This step is particularly important for tracking, where the camera will be freely moving and rotating in space.

Synthetically generated views will allow to build up new training and test data sets, that will be used during classifier training and testing steps.

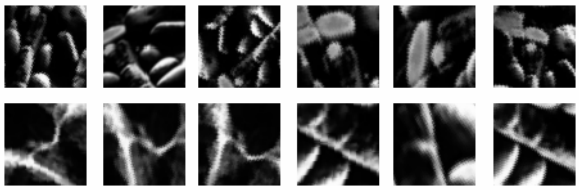

Figure 4: Randomly generated training examples of four classes by applying affine transformations.

Once the training set is ready, the training task can be performed. During this task, a number of examples are randomly selected from the available ones. These examples are pushed down in the trees. In order to decrease the correlation between trees, and therefore increase the strength of the classifier, different examples from the training set must be pushed down in each tree. This randomness injection favors the minimization of trees correlation.

While building up the tree, each node of the tree is treated as follows:

- N training examples from the training set are in the node.
- S random sets of *n* pixels are selected.
- For each set, its information gain [Brei01] is calculated.
- The variable set with the greatest information gain value is selected.
- The examples are tested with the selected set of pixels. Depending on the result of this test, they are pushed down to their corresponding child node.
- The above process is recursively done for the children nodes, whether until there is only one example, or only one class is represented in the remaining examples or the maximal predefined depth is reached (see Fig.4).

Once the descriptors reach the bottom (maximal depth) of the tree, it is said that they reached a leaf node and the recursion stops. In leaf nodes the class posterior distributions are stored. These distributions represent the number of class examples from the

training set that has reached that node. Once an example of a given class reaches a leaf node, the posterior probability distribution stored in that node must be updated accordingly.

The tests to be performed in each node can be simply binary tests, based on the comparison of the intensity values of two pixels, as:

$$T = \begin{cases} 1 & if\left(v(p_1) - v(p_2)\right) \geq r \\ 0 & otherwise \end{cases}$$

Where $v(p_1)$ and $v(p_2)$ represent the intensity values of two pixels located at positions p_1 and p_2 respectively. The values of p_1, p_2, i.e, the pixel locations to be tested are randomly selected during the training step. The value of *r* represents a threshold that was also randomly selected while training. In our experiments, we select a random value for *r* between 0 and 25. We have also experiment that, given the weakness of the tests, smoothing every patch before training and classification, significantly increases the final reliability of the classification.

Figure 4: Random Tree construction example.

3.1.2 Classification

Once the classifier is built, i.e, the pixels to be tested in each node and the class posterior distributions are calculated, it is ready to classify new examples different from the ones in the training set. During the classification task any new example is dropped down in every tree that constitutes the forest. These examples will reach a leaf node depending on the results of the tests obtained in the previous nodes they visit (see Fig. 3). The posterior distributions stored in leaf nodes are used to assign a class probability value to the examples that reach that node, $P(Y = c \mid T_1 = T_l, n = \eta)$ where T_l is a given tree of the forest and η is the reached node by the example (patch) Y and c is the assigned class label.

As any multi-classifier, the random forest needs to combine the independently generated output by each tree in the forest, in order to assign a final class label to the examples to be classified.

Depending on the number of classes to be trained, the output of the trees during classification can reach a very low value. When combining these outputs among the trees, the final value can be close to zero because of floating point precision (underflow) [Alk00]. Therefore, instead of using the addition or the product rules for classifier combination, we apply the addition of the logarithms of the posteriors to generate the final class label.

Ferns

Random Ferns like Random Forest is also a multi-classifier, compound of a determined number of entities or classifiers. This approach was firstly introduced in [Özu07], and it was also independently proposed in [Wil07]. In opposite to Random Forest, Ferns is a non-hierarchical structure where each entity that constitutes the multi-classifier is basically a set of tests. In Random Forest the test set of each tree is the collection of different tests that are distributed along the nodes that forms the tree. Due to the flat structure of every entity in a Ferns classifier the test set is a simple ordered list of positions of pixels to be evaluated.

3.1.3 Training

As in Random Forest, we use the interest point extractor proposed in [Rosten06] to get the candidate points and their surrounding patches. The extracted patches will be the classes to be recognized by the classifier.

Due to the non-hierarchical structure of the classifier, during the training step not information gain is calculated. Therefore, in contrast to Random Forest where only the data that falls in a child node is taken into account in the test, in Ferns classifiers the test set (the pixels to be evaluated in each example) is applied to the whole training data set.

Instead of saving the posterior distributions over the classes in leaf nodes, Ferns classifiers build up a look-up table where those distributions are stored [Özu07]. This look-up table is a *N*x*M* matrix where *N* represents the number of different classes that are going to be classified by the classifier and *M* the number of possible outputs given a test set. The number of columns, *M*, is calculated as 2^p where *p* is the number of tests that forms a test set. The result of each individual test and the ordering on the set defines a binary code that, interpreted as a decimal number, can be used as an index for the look-up table to get the corresponding posterior probability distribution of the class. Access to a position in the look-up table is similar to reach a leave node in the Random Forest approach. The class posterior distributions are computed as in Random Forest.

The process for building up the training data set remains the same as the one proposed for Random Forest.

3.1.4 Classification

During classification, the examples are tested by the whole entities that form the classifier. In every entity the complete test set is evaluated given an example. This test generates the binary code that allows the classifier to access a position in the table (column) and recover the posterior probability stored for each class (row).

As Random Forest does, the outputs of each entity that forms the Ferns classifier must be combined to obtain the final class label for the example. During combination numerical errors (underflow) may occur, so we use the addition of logarithms, instead of addition or product combination rules.

4. EVALUATION

We have implemented our own API to evaluate the influence of different factors on the behavior of Random Forest and Ferns classifiers during training period as well as during execution period. Depending on different factors such as, number of different classes, the number of classifiers, or the size of the training set, the point classification rate may vary.

Due to the random selection of features, all tests were carried out ten times and the results were averaged. The entire evaluation was conducted by using feature points extracted from well-textured images. Due to the weakness of tests performed in each step of both classifiers, well textured and non-symmetric textured images are needed to obtain accurate results (see Fig.5).

Figure 5: Image used for the evaluation

We have evaluated the performance of Random Forest and Ferns classifiers by using the same number of structures in both approaches. The same number of trees is tested against the same number of entities that constitutes a Ferns classifier. Moreover, the number of tests to be evaluated in each entity in the Ferns classifier is equal to the maximal allowable depth of every tree in the forest.

The following tests were all carried out by using the same training data set for both classifiers, in order to obtain consistent and reliable results.

Rotation Range

We are interested in the evaluation of the behavior of both approaches depending on the allowable rotation range to be applied to the patch examples during the training phase. During the generation of the training set each patch representing a class is transformed by applying it an affine transformation that approximates the homography. This affine transformation can be decomposed as a rotation matrix R and a scaling matrix S with parameters $[\lambda_1, \lambda_2]$. These affine transformations are generated by extracting values at random from uniformly distributed intervals for rotation and scale parameters.

Obtaining robustness against variation in rotation and scale is particularly important for tracking, where the camera will be freely moving around the object. In this way, the classifier must be trained to be able to recognize the classes (points) from different orientations and scales.

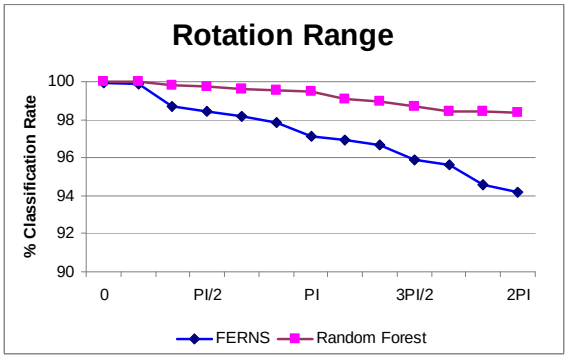

Figure 6: Rotation Range evaluation results.

Figure 6 shows the classification rate of Random Forest and Ferns classifiers trained with 225 different classes, no variation in scale and 325 examples per class. Both classifiers show similar behavior. In our test Random Forest classifier performs slightly better.

As expected, by increasing allowable rotation range with a fixed trained set size, the classification rate decreases.

Scale Range

As the rotation range test, the scale range test in intended to evaluate the robustness of the classifier against changes in scale. In this test we trained 225 different classes, with no variation in orientation and 325 examples per class. We define the scale range as follows: a value v of 1 means no variations in scale, a value (v <1) means a maximum possible reduction of scale of (1-v) percent, while value (v>1) means a maximum increase of scale of (1+v) percent. The following results were obtained by applying an

isotropic scaling to the examples in the training set, where the scaling matrix S of the affine transformation matrix is $\lambda_1 = \lambda_2$, with a randomly generated value, given a predefine range.

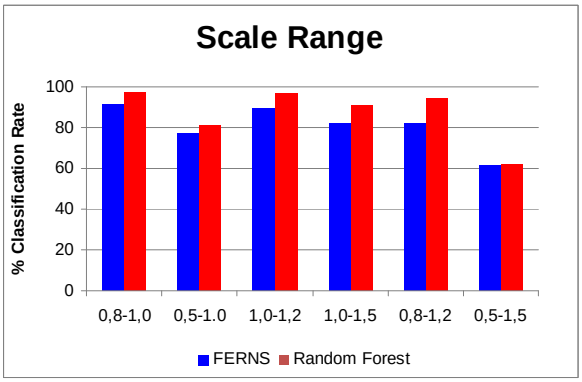

Figure 7: Scale range evaluation results.

We can conclude that the accuracy of the classifiers is drastically affected when severe changes in scale occur (see Fig.7). Also, while the range of scales is increased, the possible variations in appearance of every class also increase, spanning bigger subspaces. In order to be able to handle such subspaces the training data set must be increased accordingly.

Training Set

The training set size is a key factor during the training step of a supervised classifier. These classifiers are very dependent of the quality and size of the training set. The training set has a real influence in the final accuracy of the classifier. We have evaluated the behavior of Random Forest and Fern classifier by varying the size of the training set, i.e, the number of examples, while the number of classes remains constant.

The training sets are built with 225 different classes, where the examples were generated by applying random affine transformations with $[0,2\pi]$ allowable rotation range, and [0.5,1.5] scale range. We have evaluated the behavior of both classifiers by modifying the number of different examples per class.

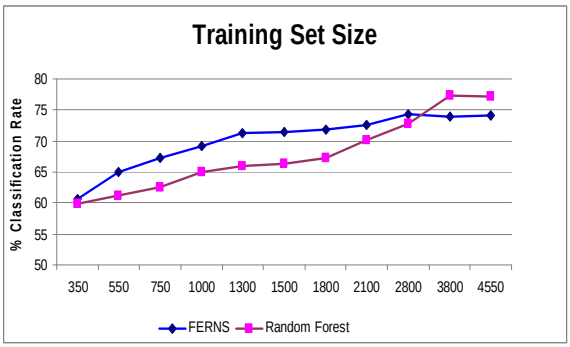

Figure 8: Training set size tests result.

Both classifiers get convergence when a number of examples per class are about 3500. At this value Random Forest classifier gets slightly better results reaching 78% of well recognized examples (see Fig.8).

Number of Classes

Feature or keypoint matching is a critical task in many computer vision applications, such as optical 3D reconstruction or optical markerless tracking. Related to tracking by detection techniques, is important that the tracker can cope with a large number of different points in order to get robustness against factors such as partial object occlusion or noise. In this way, we measured how the classifiers handle different number of classes, while the size of the training set remains fixed. More precisely, the following results were obtained by using 1000 examples per class, $[0,2\pi]$ allowable rotation range and [0.8,1.2] scale range.

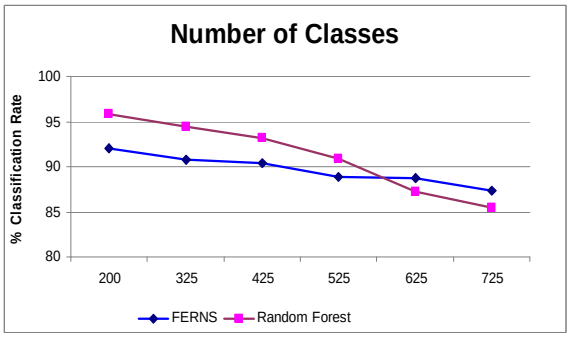

Figure 9: Number of different classes test result.

As expected, (see Fig.9) as the number of different classes increases the accuracy of both classifiers decrease. Our results show that Ferns classifier performs better than Random Trees being able to cope with a bigger number of different classes.

Discussion

Both classifiers perform well when the range of scale values is about [0.7,1.4]. We estimate that this range is sufficient to handle many views when tracking operation. When this range increases, the classification rate starts to decrease rapidly. Both approaches are robust against rotation changes when the training set size is large enough according to the number of supported classes.

When the classification rate is low, the number of miss-matched points (outliers) increases. Once the population of outliers is high, posterior processes related with tracking such as outlier filtering with RANSAC or non-linear minimization methods like Levenberg-Marquardt are seriously afected due to the number of iterations they need to reach convergence.

5. APPLICATION

The approaches described previously were applied within a prototype using head mounted displays (HMD) for collaborative mobile mixed reality design reviews. Figure 11 shows the markerless tracking module by using a camera attached to a laptop working outodoors, while figure 12 shows the same module but working indoors.

Our tracking module uses natural features to estimate the position and orientation of the camera, mounted on the HMD. Once this transformation is computed, the virtual object can be registered and viewed through the HMD as part of the real world. During the tracking process, the transformation must be updated over time.

By using automatically extracted natural features (see Fig.10), the use of artifacts such as reflective markers is avoided, allowing the system to be more flexible and being able to work in non-well controlled conditions, such as outdoor environments.

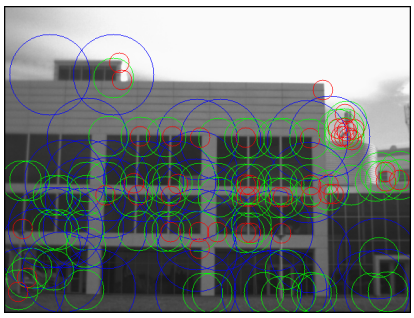

Figure 10: Feature or interest point extracted from a building facade to train a random forest classifier.

The internal camera parameters estimation task is performed only once, when the camera is to be used for the first time.

As described earlier, tracking by detection techniques requires an off-line process where the classifier is trained. During this period, one image of a highly textured plane, such as a building facade or a picture over a table, must be acquired. After the acquisition, some features points and their surrounding texture patches are extracted from the image [Ros06], and synthetic views of the plane are generated.

Based on the results described previously, the integrated classifier in the tracking module of the prototype is trained to be able to recognize about 200-250 different classes (interest points). The forest or Fern classifiers are constructed with 20 entities and a training set compound of 1000 synthetically generated examples for each class in less than 10 minutes. This size of the training set is a good compromise between training time and final accuracy of the classifier. Training time is a very important factor in practical situations such as outdoor setup preparation time. Once the training set is ready, the system is ready for tracking.

The obtained frame rate is about 20-25 frames per second (near real-time) on a 1.6Ghz dual core CPU. This frame rate may vary depending on the accuracy of the tracker, i.e, depending on the number of different points to be recognized. The drift and jitter are well controlled, so no severe movements of the objects occur. On a lower CPU (1Ghz) the obtained frame rate is only 5 frames per second.

Independently on the robustness of the classifier, the wrong classified points (outliers) are removed by using RANSAC. The final estimation is refined by using Levenberg-Marquardt non-linear minimization method using all the inlier points, starting from the estimation obtained by RANSAC. This final minimization is very useful to avoid virtual object jittering, what is an uncomfortable behavior during augmented reality scene visualization.

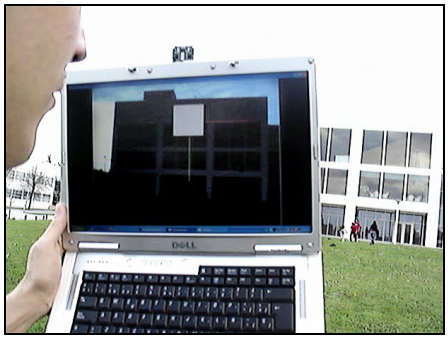

Figure 11: Outdoor Tracking of a building facade.

The tracking by detection approach based on feature point classification and matching allows the tracker to be robust against partial plane occlusion, or fast camera movement. The tracker can run indefinitely without requiring re-initialization.

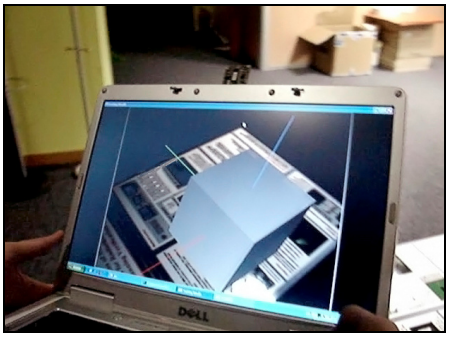

Figure 12: Indoor tracking of a textured floor.

6. CONCLUSION AND FUTURE WORK

In this work we have presented an approach of tracking by detection for plane homography estimation using the Random Forest based classifier

and Ferns classifier for interest point matching. A comparative evaluation of both approaches was carried out by analyzing their behavior while modifying important parameters, such as number of classes or training set size. Our results show that both approaches are very similar with no clear advantage of one approach over the other.

Also, an augmented reality prototype using these classifiers was described. The prototype is able to robustly track a plane even if partial plane occlusion occurs, at real-time frame rate.

We want to extend our work to support on-line training classification [Wil07]. On-line training allows the tracking to update the model with new feature points not present in the original training set. On-line training can be exploited in several frameworks such as Simultaneous Localization and Mapping (SLAM).

7. ACKNOWLEDGMENTS

This work has been partially funded under the 6th Framework Programme of the European Union within the IST project "IMPROVE" (IST FP6-004785, http://www.improve-eu.info/).

8. REFERENCES

[Alk00] Alkoot,F.M.,and Kittler, J. Improving the Performance of the Product Fusion Strategy. *In Proc. International Conference on Pattern Recognition*. Vol.(2), pages: 164-167, 2000.

[Ami97] Amit, Y. and Geman, D. Shape quantiation and recognition with randomized trees. *Neural Computation*, Vol.(9) pages: 1545-1588, 1997.

[Bof06] Boffy, Aurélien, Tsin, Yanghai and Genc, Yakup. Real-Time Feature Matching using Adaptative and Spatially Distributed Classification Trees. *In Proc. British Machine vision Conference*, Vol.(2) pages: 529-539, 2006.

[Bre01] Breiman, L. Random Forests. *Machine Learning Journal*, Vol. (45), pages: 5-32. ISSN 0885-6125, 2001.

[Lep04] Lepetit, V., Pilet, and J., Fua, P. Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation. *In Proc. Conference on Computer Vision and Pattern Recognition*. ISBN: 0-7695-2158-4, 2004.

[Lep05] Lepetit, V., and Fua, P. Monocular model-based 3D object tracking of rigid objects: A survey. Foundations and Trends® in Computer Graphics and Vision., Vol.(1), pages 1–89, 2005.

[Lep06] Lepetit, V., and Fua, P. Keypoint Recognition Using Randomized Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28(9), pages: 1465-1479. ISSN: 0162-8828, 2006.

[Low04] Lowe, D. Distinctive Image Features from Scale Invariants Keypoints. *International Journal of Computer Vision*. Vol. 20(2), pages: 91-110, 2004.

[Mik05] Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., and Gool, L. V. A Comparison of Affine Region Detectors. *Int. Journal of Computer Vision*. Vol. 65(1-2), pages: 43-72. ISSN:0920-5691, 2005.

[Moo06] Moosmann, F., Triggs, B., and Jurie, F. Fast discriminative visual codebooks using randomized clustering forest, *NIPS*, 2006.

[Ros06] Rosten, E., and Drummond, T. Machine Learning for High-Speed Corner Detection. *In Proc. European Conference on Computer Vision*. Pages: 430- 443. ISBN 3540338322, 2006.

[Özu06] Özuysal, M., Fua, P., and Lepetit, V. Feature Harvesting for Tracking-By-Detection. In *Proc. European Conference on Computer Vision*, pages 592-605. ISBN:3-540-33836-5, 2006.

[Özu07] Özuylan, M, Fua. P, and Lepetit, V. Fast keypoint recognition in ten lines of code. *Computer Vision and Pattern Recognition*, pages: 1-8, ISBN-1-4244-1180-7, 2007.

[Vac04] Vacchetti, L., Lepetit, V., Fua, P. Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking. In *Proc. IEEE and AM International Symposium on Mixed and Augmented Reality*. Vol. (4), pages: 48-57. ISBN:0-7695-2191-6, 2004.

[Wil07] Williams, B., Klein, G. and Reid, I. Real-time SLAM Relocalisation. In *Proc. IEEE International Conference on Computer Vision*, 2007.