Vision Based Control of Unmanned Aerial Vehicles Using Deep Learning

Mohit Kumar Ahuja*,†, Guillaume Allibert† and Cédric Demonceaux*

***Abstract*— The current research on using DNNs (Deep Neural Networks) to autonomously control UAVs (Unmanned Aerial Vehicles) is limited. The present paper details CNN (Convolutional Neural Network) architectures used to predict the movement of a billiard ball on a billiard table. Two different architectures are proposed which will directly process raw visual inputs. The first proposed architecture uses Frame-centric (FC) prediction based on visual glimpse centered on the frame, while the second proposed architecture uses Object-centric (OC) prediction based on visual glimpse centered on the object. Further, the future velocities and angles of the moving ball provided by the CNNs is used to control the UAV. It is demonstrated that the proposed model is able to predict the future velocities and angles of a billiard ball with high accuracy.**

I. INTRODUCTION

Deep learning is becoming popular nowadays as it has significantly improved the state-of-the-art of many problems that Machine Learning and Artificial Intelligence (AI) community faced for a long time. The present work studies the control of a UAV using Deep learning [7], [8], [9], [10]. In the past two years, some works demonstrated autonomous control of UAV using deep learning. To the best of our knowledge, all of them have implemented deep learning algorithms for solving a classification problem, where according to the situation, the UAV is capable of taking decisions like moving right/left/straight/backward/stop. However, very few works can be found which shows the study of regression used for autonomous control of UAV.

After an extensive literature review, it was decided to work on a project in which a UAV, hovering over a billiard ball will have the capability to predict the future velocities and trajectory of that ball if it gets hit. Also, the UAV will autonomously follow the ball even if the ball is hit with a huge force. We have improved the results of the previously designed architectures as well as designed our own Convolution Neural Network (CNN) which predicts the future velocities and trajectories of the ball and let the UAV follow the ball autonomously.

We decided to work on deep learning due to its advantages over traditional machine learning algorithms. Traditional machine learning algorithms give better results with small data-sets. However, machine learning algorithms are not capable of coping with big data sets and give unsatisfactory performance. Whereas deep learning algorithms provides accurate results with big data-sets.

The rest of this paper is organized as follows. Section II explains system descriptions used for training and testing our architecture. Section III explains the previous works and the state of the art. Section IV explains the methodology, in which architecture of FC based CNN is proposed for predicting the future movements of the ball on a billiard table which takes an input of visual glimpse centered on the frame. It also explains a novel model (BRNET-Billiard Network) which is used to control the UAV by processing two consecutive frames (centered on the object) of a camera. The camera is mounted on a quadcopter that uses the object-centric (OC) approach to predict the future velocities and angles of the moving ball. And simulated results are shown in Section V. The conclusion is explained in Section VI.

II. SYSTEM DESCRIPTION

The model was trained on a Tesla K40c GPU with 57,935 simulated images and tested on a Jetson-TX2 with 6400 simulated images. The setup for the quadrotor is ready as shown in Figure 1. After collecting real time data of billiard board, the model will be trained again and tested in real time. The hardware includes a quadcopter with open source Pixhawk autopilot, and NVIDIA Jetson TX2 with orbitty carrier board. The vision processing uses a single down-facing Basler acA640-120gc GigE camera with a gimbal, which run in 720p mode at 30 fps. All processing is done on the Jetson TX2. The quadcopter can also be replaced by a similar 450-550 drone, such as a DJI Flame Wheel, if desired.

The output of the network will provide high-level commands for pixhawk controller, such as velocities to control the platform. Velocity commands are produced by our network running on Jetson-TX2 which is mounted on the quadcopter and uses Mavros ROS package for communication.

III. PREVIOUS WORKS

In the last decade, plenty of research has been carried out for autonomous control of UGV's [11], [12], [13], using DNN but there is limited research on autonomous control of (UAV) using deep learning [14], [15], [16].

As our task is to predict future velocities and angles and use those predictions to control the UAVs, previous works tried to predict the moving object parameters by just using the image sequence. Some others presented a completely new approach to predict the future velocity of the ball by taking previous some images as an input. We will discuss some

This work was supported by CNRS-I3S Lab, Sophia Antipolis, France.

*,†Mohit Kumar Ahuja and *Cédric Demonceaux are with the University of Burgundy, France. Mohitkumarahuja@gmail.com, cedric.demonceaux@ubfc.fr

†Guillaume Allibert is with the Université Nice Sophia Antipolis, Biot, France. allibert@i3s.unice.fr

Fig. 1: The hardware which will be used for real time predictions, which contains quadrotor, jetson-TX2, and Pixhawk autopilot controller.

of the related approaches proposed previously and will also discuss the limitations of those approaches.

In [4], Wu et. al. have proposed a generative model for solving the problems of physical scene understanding from real-world videos or images. But this approach will give us the displacement of any moving object from a set of images. It can be used to predict the current velocity of the ball which might have some error. Due to this fact, this network can not be used for predicting future velocities of the moving object.

In [1], Katerina et. al. proposed an approach that takes input as a group of 4 frames (In FC: centred at frame. In OC: centered at object), velocity of the moving object and the information stored in previous Long Short Term Memory (LSTM) units. Their main objective was to predict the future frames of the ball. Since our work concerns about the applicability of self-governing control of a UAV to follow a billiard ball, their proposed model poses some limitations like manual interventions. Also, their model doesn't have a satisfactory prediction accuracy.

Some architectures proposed to make real time decisions like following a road and taking emergency actions [7], [8], [9], and [10] such as stopping if a dynamic obstacle comes in the frame of quadcopter camera. And other similar architectures are presented and all of them are related to classification problem. Some have also used Model Predictive Control(MPC) to improve the accuracy of the system [14], [15], and [16].

IV. METHODOLOGY

The learning approach aims to predict future velocities and angles of a moving ball on a billiard table. These predicted velocities and angles are later converted into control flying commands which enable a UAV to safely follow the moving ball. Two different CNN architectures are proposed, which improved the state of the art.

A. Frame-Centric Approach

This proposed architecture is based on architectures explained in [1] and [6]. The complete model is shown in Figure 2. The improved architecture takes an input of just one FC image, its current velocity, angle and predicts a norm of future velocities and angles for 5 future frames. It uses the first four hidden layers of the CNN proposed in [1] followed by three fully connected layers. The output of the network provides the velocity and angle of the ball movement for the next five frames.

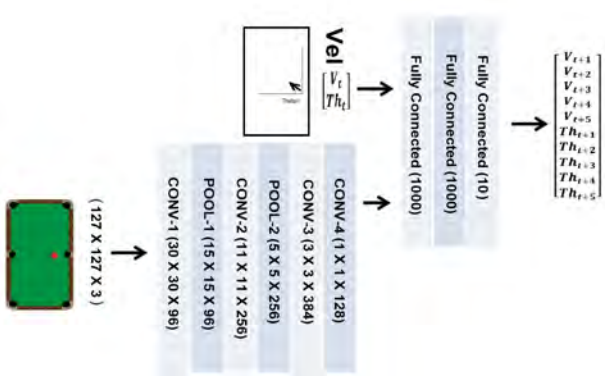


Fig. 2: Proposed CNN for FC images which takes a frame centric image of a billiard ball and current velocity and angle information and predicts future velocities and angles of a moving ball for 5 future frames.

The predicted velocity and angle can be modified to predict for as much future frames as required. But some modification at the output layer is required to provide that particular number of variables. In our case, the future velocity and angle have been predicted till 5 future frames i.e for coming frame at time(t+1), (t+2), (t+3), (t+4), and time(t+5). The input and output ($\hat{Y}$) of the network can be seen in Figure 3. The resulting vector representing the data for future velocities and angle is shown below:

$$\hat{Y} = \left[V_{t+1}, \quad \ldots, \quad V_{t+5}, \quad \Theta_{t+1}, \quad \ldots, \quad \Theta_{t+5}\right]^{\top} \quad (1)$$

Where V_{t+x} and Θ_{t+x} , for x = 1, ..., 5, represents the velocities and the angles, respectively. When training the model, the velocities and the angles were normalized to avoid biasing problem as the values should be in a scaled format. The limitation of this architecture is, it requires velocity and angle of the ball in current frame to be provided manually for each prediction.

B. BRNET-Billiard Network

In order to overcome the limitations in the previously proposed FC architecture, an Object-Centric (OC) based architecture has been designed by considering the fact that it should be capable of following a ball by taking just the images. No external input should be provided to the network. This network is inspired from Siamese network [5]. A number of image sequences were gathered with

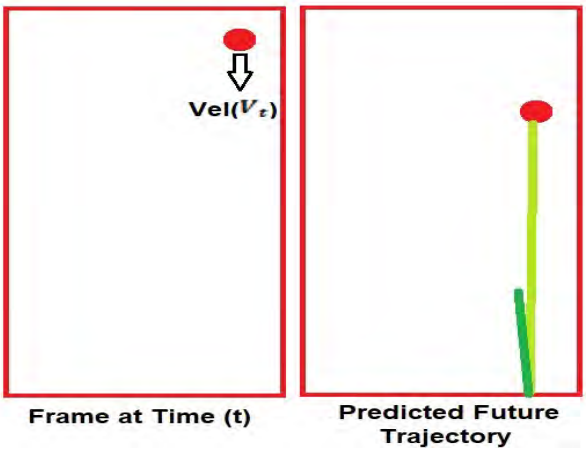


Fig. 3: Input and output of FC-Based architecture.

different locations of the ball on a billiard table. Unlike the previous model, the new designed model (named as Billiard Network(BRNET)) takes an input's of first two images from the camera (1^{st} image at time(t) and 2^{nd} at time(t+1)) and outputs the future velocities and angles of the ball. While training this network, optical flow was used to compute the displacement of ball from one image to the other. The computed displacement and frame rate of the camera was then used to compute the velocity of the ball in current frame. The complete architecture of BRNET is shown in Figure 5.

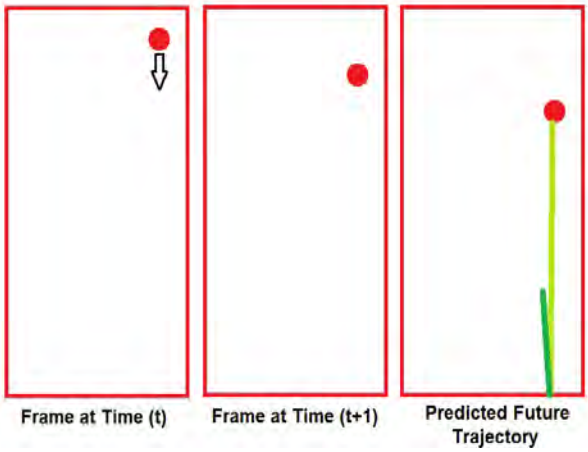


Fig. 4: Image showing input and output of BRNET

As the model inputs images centered at the object and not on the frame, the model became generic to handle environments which were never encountered during training, therefore it can be used for planning actions in novel environments without the requirement of task-specific supervision. Figure 4 shows the input and output of BRNET. It was discussed that the direction of learning to predict the effect of the ball's movement on the world directly from visual inputs is an important direction for enabling robots to act in previously unseen environments.

V. SIMULATION RESULTS

A. FC Approach

1) Dataset: In the present work, Matlab 2016b was used for generating the dataset. As this architecture deals with FC images, the requirement was to generate a sequence of simulated images of the complete billiard table. A box of size 100x100 pixels was created and filled with green color. On the same box, another circle was generated however filled with red colour. The whole image has a resemblance of a billiard table consisting of a red ball and now the task is to move the ball.

Simple laws of physics were used to move the ball and also to decide the angle of rotation after hitting the edge of the table. The velocity of the ball was also reduced after every frame which represented the friction caused by table to the moving ball. The ball was placed on coordinates (10,10) and moved in 24 different direction starting from 0 degree and moving with some velocity. The frames of the board were saved with the corresponding velocity and angle information till the ball stand still. After that, it started again with an increment of 15 degree in the angle and the process was repeated till 24 possible directions were covered. After the ball covered all 24 angles on location (10,10), it was incremented by 10 in x-axis. After completion of different angle movements in 10 different locations in x-axis starting from (10,10) till (90,10), it was increment by 10 in y-axis.

And this continues in y-axis from (10,10) till (10,90), by doing so the images were generated from 81 different locations with 24 angle movements. The total trajectories produced were 1944. On complete generation of the data-set with different angle movements and locations of the ball, 64,365 images were obtained. After data generation, we resized all the images to match with the improved architecture. The original size of each image was 434x344x3 and after resizing it became 127x127x3.

2) Training: Deep neural nets with large number of parameters are very powerful but over-fitting is a serious problem in such networks. To reduce over-fitting, training data along with its labels was shuffled and then it was divided into three parts Training-Validation-Test. After division, training set had 57,935 images, validation set had 3,215 images and test set had 3,215 images.

The whole network was implemented using TensorFlow deep learning library. During training the model on the training dataset, the code was designed in a manner that in a single iteration the network is applied to the validation set as well. On completion of 5 iterations, the result of train accuracy[1] and validation accuracy were printed. If

[1]**accuracy means the % of data which is correctly predicted after training with a threshold of $\pm 1\%$. The threshold of $\pm 1\%$ means the prediction of 344 will be considered as correctly predicted if ground truth label is 345.**

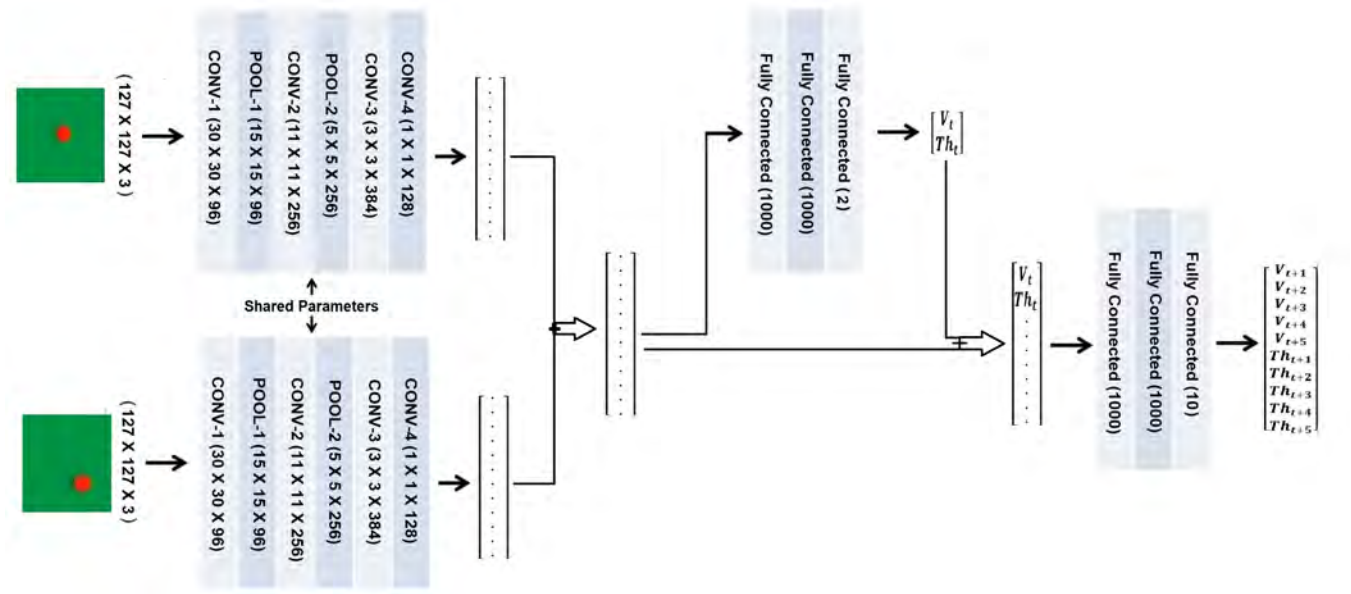


Fig. 5: BRNET Architecture for OC Images which takes two consecutive frames of a camera and predicts future velocities and angles of a moving ball for 5 future frames.

the difference between the training accuracy and validation accuracy was huge, it meant the network was overfitting. And the training was started all over again with different parameters. So the training of the model was stopped after achieving 95% [2] training and validation accuracy. The model was saved after every 5 iterations.

After training, the predicted output was multiplied with the maximum velocity and angle. The final accuracy was computed by comparing the actual values of velocities and angles not the normalized ones. This has been done to get the network in one scale since, the velocities can range from 20 to 50 and angles value can range from 0 and 345.

3) Results: The latest model was loaded in which train-dev accuracy was more than 95%. The trained model was used to predict outputs for test set and 94% test accuracy[3] was achieved. To be clear, the test set was an unknown dataset (never encountered while training) for the model.

Afterwards, a new dataset was generated with ball located on coordinates (15,25) and many more coordinates which were not used while training. The size of the new dataset was 6400 images of size 127x127x3. And after the generation of completely new dataset, it was tested using our trained model and obtained a test accuracy of 91.3%, which is 31% more than the state-of-the-art[4].

[2]Training and validation accuracy of 95% means 95% of training and validation labels are correctly predicted as the ground truth labels.

[3]Test accuracy of 94% means that 94% of test labels are correctly predicted as the ground truth labels

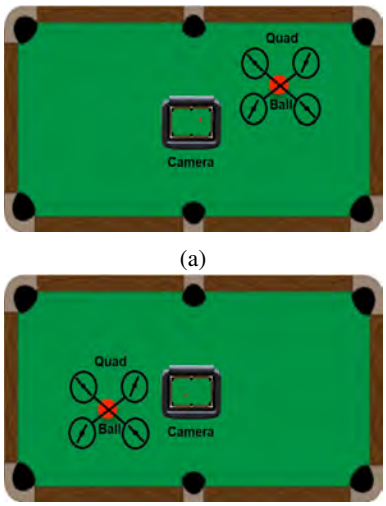
[4]The accuracy mentioned in [1] is 60%. Which is 31% less than our obtained accuracy

4) quadcopter Control: The control of quadcopter using FC images was a daunting task as the camera needs to be fixed on a particular height from the billiard table to get the whole frame of the billiard board. As shown in figure (6a) and in Figure (6b), the camera is fixed in both the figures and the quadcopter is following the ball from a larger height than the camera. Because if the quadcopter will follow the ball with a lesser height than the camera, it will be captured in the frame of the camera and the predictions will be spoiled for next frames. One more drawback of this architecture is that we have to feed the velocity and theta of the moving ball manually.

B. BRNET

1) Dataset: With some minor changes, the data was generated for BRNET as it was generated for FC-approach. Since, it deals with OC images, the requirement was to generate a sequence of simulated images centering billiard ball. The only difference while saving the frames in OC case is, from the whole image, only the patch of 100x100 pixels which was centering the ball will be saved. And the images were saved with their corresponding velocity and angle information till the ball stands still. But while saving the image, it is saved with a size of 434x344x3 due to Matlab's image saving function properties. After that, it started again with an increment of 15 degree in the angle and the process was repeated till it covered 24 possible directions.

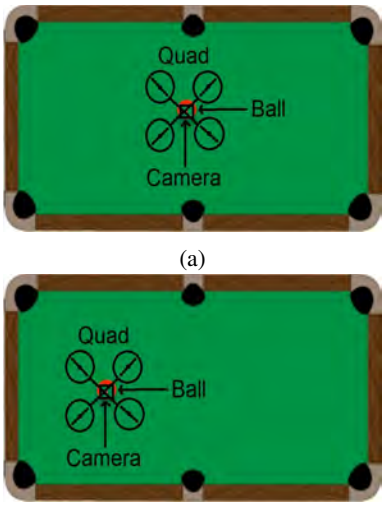
The dataset for this model was generated from very less coordinates. After the ball covers all 24 angles on location (10,90), it was then relocated at coordinates (20,80), (20,90), (10,80), (50,90), (50,80), and (50,50). After completion of



(a)

(b)

Fig. 6: Figure shows: (a) Quadcopter Control in FC approach at time(t), (b) quadcopter Control in FC approach at time(t+1).



(a)

(b)

Fig. 7: Figure shows: (a) Quadcopter Control in BRNET at time(t), (b) Quadcopter Control in BRNET at time(t+1).

different angle movements in 7 different locations, the dataset was ready for augmenting the data. Training data collected from point(10,90) was inverted vertically and the image patch looked like the ball was at (10,10) and the angle value was increment for the same.

Out of all the 7 points, the trajectories generated from the ball being in the center of the table were not augmented. But rest all images collected from 6 points mentioned above were rotated by 90, 180 and 270 degrees. By doing so, the dataset generated from rotation of the images covered all corners and edges of the table. So, images were augmented for 34 different locations with 24 angle movement. The total trajectories produced were 816. On completion of data augmentation, 26,112 images were obtained.

After data generation and augmentation, images were resized to match with the designed architecture. The original size of each image was 434x344x3 which reduced to 127x127x3 on resizing.

2) Training: Training of BRNET architecture is being performed in the same manner as it was for FC-approach. But in BRNET, before training the architecture, Lukas-Kanade optical flow method was used to compute the norm of velocity and angle of the ball between two images to create ground truth velocities of ball. But Lucas-Kanade with multi-resolution method worked better to detect the right motion in right direction. It was coded in such a manner that it will only compute the mean of vectors greater than 1 or less than one. So, it neglects small motions and only the accurate velocities were obtained. The velocity and angle data saved while generating the dataset will not be used for training but it will be used to compare with optical flow output. The purpose of doing so is to check the accuracy of optical flow predictions for ball displacement because in real world, the ground truth velocity are not known. Rest of the training procedure is similar in this architecture as it was followed in previous architecture.

3) Results: The goal is to load the latest model in which the train-dev accuracy will be more than 98%. And test our trained model on the test set to get more than 95% accuracy. To be clear, the test set will be an unknown dataset (never encountered while training) for the model.

Afterwards, a new dataset will be generated with ball located on coordinates (15, 25) and many more coordinates which were not used while training. And after generation of completely new dataset, it will be tested on our trained model.

4) quadcopter Control: The control of quadcopter using OC images will be a very easy task as the camera will be mounted on the quadcopter and it will only capture the view of ball centered in the image as shown in figure (7a) and in Figure (7b). The major advantage of this architecture is no manual feeding of velocity and angle of the moving ball is required. It will be autonomously computed in the architecture.

VI. CONCLUSION

A. Conclusion

This research work demonstrated the applicability of deep learning for accurate prediction of future velocities and trajectories of a billiard ball. We have presented two different CNN architectures to predict the future velocities and angles

of the moving ball using deep learning. An improved frame-centric (FC) architecture based on visual glimpses centered on the frame of the billiard table was proposed. Also, a novel model (BRNET-Billiard Network) was proposed which processes two consecutive frames of a quad-copter camera and uses an object-centric (OC) approach to predict the future velocities and angles of the moving ball using deep learning. Further, it is shown that these predicted outputs can be used to control the UAV to follow the ball. The accomplished studies and experiments performed showed significant improvement in the state-of-the-art.

It is shown through experiments that the proposed model is able to predict the future velocities and angles of a billiard ball with high accuracy.

B. Future Work

As part of the future work, the proposed models will be trained on real data collected from camera mounted on a quadcoptor. The trained model will then be tested on a Jetson-TX2 mounted on a quadcopter to follow the movements of billiard ball in real time. Once it starts following the ball accurately, the same model will be trained on human image data and then tested to see whether the UAV follow humans or not.

REFERENCES

[1] Katerina Fragkiadaki, Pulkit Agarwal, Sergey Levine, and Jitendra Malik. *"Learning Visual Predictive Models of Physics for Playing Billiard"*. arXiv preprint arXiv:1511.07404, Jan 2016.

[2] Pulkit Agarwal, Ashvin Nair, Pieter Abbeel, Sergey Levine, and Jitendra Malik. *"Learning to Poke by Poking: Experiential Learning of Intuitive Physics"*. Advances in Neural Information Processing Systems, 5074-5082, NIPS 2016.

[3] Agrawal Pulkit, Carreira Joao, and Malik Jitendra. *"Learning to see by moving"*. IEEE International Conference on Computer Vision (ICCV),pages = 37 - 45, 2015.

[4] Wu, Jiajun, Yildirim, Ilker, Lim, Joseph J, Freeman, Bill, and Tenenbaum, Josh. *"Galileo: Perceiving physical object properties by integrating a physics engine with deep learning"*. Advances in Neural Information Processing Systems 28, pp. 127135. Curran Associates, Inc., 2015.

[5] G Koch, T EDU, R Zemel, and R Salakhutdinov. *"Siamese Neural Networks for One-shot Image Recognition"*.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *"ImageNet Classification with Deep Convolutional Neural Networks"*. Advances in neural information processing systems, 1097-1105, 2012.

[7] Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza. *"DroNet: Learning to Fly by Driving"*. IEEE ROBOTICS AND AUTOMATION LETTERS. PREPRINT VERSION. ACCEPTED JANUARY, 2018.

[8] A. Giusti, J. Guzzi, D. C. Cirean, F. L. He, J. P. Rodrguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. *"A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots"*. IEEE Robotics and Automation Letters, 2016.

[9] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. *"Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness"*. arXiv preprint arXiv:1705.02550.

[10] Dhiraj Gandhi, Lerrel Pinto and Abhinav Gupta. *"Learning to Fly by Crashing"*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), arXiv preprint arXiv:1704.05588, Sep 2017.

[11] Pulkit Agarwal, Joao Carreira, and Jitendra Malik. *"Learning to See by Moving"*. Computer Vision (ICCV), IEEE International Conference on 37-45, 2015.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Karol Zieba, and Jake Zhao. *"End to End Learning for Self-Driving Cars"*. arXiv preprint arXiv:1604.07316v1.

[13] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. *"Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car"*. arXiv preprint arXiv:1704.07911v1, 2017.

[14] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. *"Vision-Based Autonomous Mapping and Exploration Using a Quadrotor MAV"*. IEEE International Conference on Intelligent Robots and Systems, October 2012.

[15] Ian Lenz, Ross Knepper, Ashutosh Saxena. *"DeepMPC: Learning Latent Nonlinear Dynamics for Real-Time Predictive Control"*.

[16] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. *"Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search"*. arXiv preprint arXiv:1509.06791v2. Feb 2016.

Machine Learning Algorithm for novel microwave sensor systems

Stefanos Athanasiadis

Abstract The spin-out company MicroSense Technologies Ltd (MTL) has invented a new microwave sensor system that enables the detection of concentration of water and ingredients in processed foods, which is an indicator of the quality of the product. The sensor systems relies on the measurement of the dielectric permittivity of the product. For example water has a high permittivity, the microwave signal reflected from the product induces a shift which can be correlated of the content of water in the product. The purpose of the project is to be able to link this change of signal to different products, and for a given product to different concentrations of water, indicative of its quality by constructing a machine learning classifier. In this project the Artificial Neural Networks were selected to perform the classification due to their robustness and accuracy.

***Index Terms*— Artificial Neural Networks – Machine Learning – Microwave Sensor – Principal Component Analysis**

I. Introduction

The quality and composition of food products are of significant importance during the stages of a food production flow line. Marie Ferrie [1] defines quality as "The distinctive trait, characteristic, capacity or virtue of a product that sets it apart from all others". Food manufacturing companies must have great standards regarding the quality of their products because the consumer's awareness and knowledge is high in today's social and economic environment. Hence the microwave sensors is an accurate technology to fill this need. Many researches on microwave techniques were performed throughout the last years, resulting in practical implications in pharmaceutical, oil and food industries. [2], [3], [4], [5]

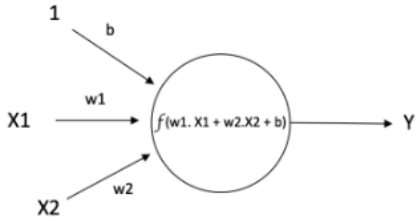
This technology offers real-time monitoring, is non-destructive, contactless and non-invasive, in contradiction to other complex chemical procedures. A statement of the basic principle behind this kind of sensor was given by the authors of this article [6], "Depending on the type and amount of a chemical ingredient in the product under test, a variation of the complex dielectric permittivity is produced. This perturbation of the dielectric permittivity affects the electromagnetic response of the sensor, thus allowing to obtain information about the composition of the solution."

Frequency and time domain measurements are the two main categories of measuring microwaves. The first method determines the permittivity by calculating the transmission or reflection coefficient. The second method that uses time domain frequencies, traces the time domain waveform and then can either compare it with the one before the measurement or apply Fourier-Transform (FT) and follow the frequency domain method. [7]

II. Introduction To Artificial Neural Networks

Development of Neural Networks (NNs) date back to the early 1940s. It experienced an upsurge in popularity in the late 1980s. This was a result of the discovery of new techniques and developments and general advances in computer hardware technology. Some NNs are models of biological neural networks and some are not, but historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, perhaps intelligent, computations similar to those that the human brain routinely performs, and thereby possibly to enhance our understanding of the human brain. [8]

The basic unit of a neural network is the neuron. It receives input from some other neurons, or from an external source such a feature and computes an output. Each input has an associated weight *w*, which is assigned on the basis of its relative importance to other inputs. The node applies a function ***f*** to the weighted sum of its inputs as shown in the below Fig. **1**.



Output of neuron = Y= $f(w1.\,X1 + w2.X2 + b)$

Fig. 1: A single neuron

The above network takes numerical inputs X1 and X2 and has weights w1 and w2 with those inputs. There is another input 1 with weight b (called the Bias) associated

with it. The Bias provides every node with a constant value. The output Y from the neuron is computed as shown. The function *f* is non-linear and is called activation function. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data is non-linear and the neurons need to learn these non-linear representations. [9]

III. Processing of Sensor Raw Data

The first step is to process the raw data gathered from the FoodSense microwave sensor by selecting the correct features to classify the data to their corresponding class / recipe. The number of features for each observation is 1500, that is the frequency readings gathered from the sensor. PCA finds the most important feature, [10], [11] , [12]which is the one with the largest variance. The dimension with the largest variance corresponds to the dimension with the largest entropy and thus encodes the most information. Fig. **2** illustrates the variance of all 1500 features, where a pinpoint on the 8th component has been placed, showing the middle ground. The first 8 to 12 features are the most significant components in descending order. The rest features, the smaller eigenvectors, represent noise components.

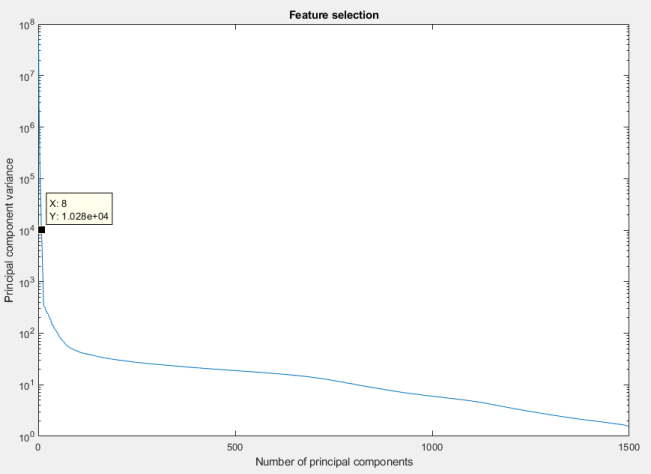


Fig. 2: Feature Extraction based on the variance

IV. Implementation Of Artificial Neural Networks

Following the results of PCA, the data from all the recipes were combined together in one data matrix with rows as many as the different sensor readings called observations and columns the number of features selected in the PCA analysis (8 and 12 Features). Then another matrix with the same rows as the data matrix and 6 columns, one for each class (recipe), was constructed as a label matrix that contains information of the class each observation has. For example, the rows of the matrix that contain the measurements from recipe 26, are labeled at the label matrix as a unique binary combination: 100000. Recipe's 51 unique combination is 010000 and so on. Fig. **3** gives another correlation between recipes and class number, where this number system is used in the following evaluating confusion matrix.

Class	1	Recipe 26
	2	Recipe 51
	3	Recipe 39
	4	Recipe 31
	5	Recipe 35
	6	Recipe 37

Fig. 3: Mapping between labels/class and recipes

A. A sample architect of a Neural Network

The following Fig. **4** demonstrate one of the training session of the Neural Network It gives detailed information regarding the architecture of the network, such as the number of neurons in the input, hidden and output layers, the training algorithm, and some parameters that update throughout the session like the gradient.

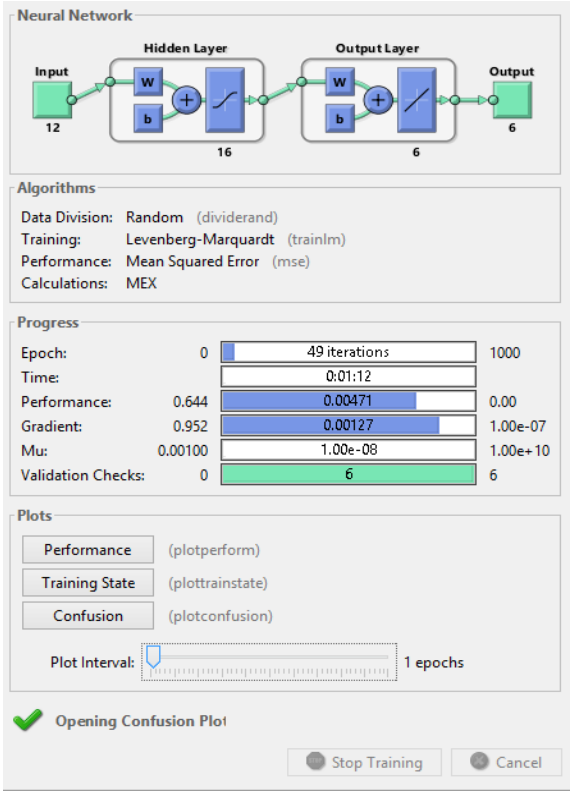


Fig. 4: Neural Networks training session using Matlab's

B. Classification Accuracy - Confusion Matrix

The following Fig. **5**, the confusion matrix, is plotted when the training phase is completed and the model is tested on the test set of the data. In the confusion matrix the rows correspond to the predicted class (Output Class) and the columns correspond to the true class (Target Class). For example cell (3, 6) shows that 122 observations were supposed to be classified as class 6, but instead they were incorrectly classified as 3. The diagonal cells correspond to observations that are correctly classified. The off-diagonal cells correspond to incorrectly classified observations. Both the number of observations and the percentage of the total number of observations are shown in each cell.

The column on the far right of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the precision (or positive predictive value) and false discovery rate, respectively. The row at the bottom of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are often called the recall (or true positive rate) and false negative rate, respectively. The cell in the bottom right of the plot shows the overall accuracy.

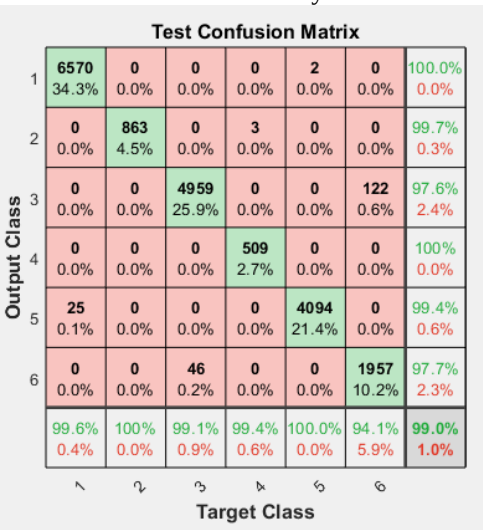


Fig. 5: Confusion Matrix

Observing the results of Fig. 5, the accuracy of the classifier is the most optimal, though some misclassified data were observed between the classes 3 (recipe 39) and 6 (recipe 37). 122 observations were misclassified as class 6 instead of 3 and similarly 46 observations were classified as class 3 instead of 6. In most of the experiment runs the misclassification occurred between these recipes.

V. Results

The total number of test runs that have been conducted are 24. It is the amount of different training and testing sessions. The experiment is divided in two large categories depending the number of neurons in the input layer, meaning the number of features that where selected in the feature selection section mentioned previously. Then each category was subdivided into another 2 categories, depending on the learning rate parameter. The following 4 tables show the 4 aforementioned categories. In every category 6 experiment runs have been executed, 2 of these based on the training method and 3 different number of neurons in the hidden layer, making total of 6 unique combination. Each one of the 6 cells display 3 numbers, the percentage of the correct classification of the test set of data, the number of epochs / iterations and finally the time that the neural network required to train the data.

TABLE 1
1.1 category - 8 Features, Learning rate = 0.05

Percentage of correct classification / No. of iterations / time of training (mm:ss) No. of Features = **8** - Learning rate = **0.05**		Number of Neurons in the hidden layer		
		8	16	20
Training Method	Levenberg - Marquardt Algorithm	95.10% / 72 / 00:30	97.90% / 107 / 02:15	98.20% / 236 / 05:32
	Scaled conjugate gradient algorithm	94.50% / 488 / 02:17	97.00% / 757 / 03:39	97.30% / 1000 / 06:07

TABLE 2
1.1 category - 8 Features, Learning rate = 0.25

Percentage of correct classification / No. of iterations / time of training (mm:ss) No. of Features = **8** - Learning rate = **0.25**		Number of Neurons in the hidden layer		
		8	16	20
Training Method	Levenberg - Marquardt Algorithm	94.50% / 56 / 00:23	98.00% / 89 / 01:52	**98.60% / 96 / 2:21**
	Scaled conjugate gradient algorithm	96.00% / 392 / 00:18	96.80% / 781 / 04:00	98.10% / 934 / 03:27

A. *Comparison of results in category 1.1*

The classification accuracy and execution time give reasonable results, according to the theory that was mentioned earlier. Almost in every execution The Levenberg – Marquardt [13] gave better results than the Scaled conjugate gradient algorithm [14]. The latter algorithm iterates quicker than the former due to the reduction of the number of computations, hence the bigger number of epochs (iterations). The second algorithm is considered to be faster, but it doesn't give better results.

It is observed over the 6 cases that the more neurons the hidden layer has the more accurate results give, but the training time that is required is also getting bigger. Having more neurons the network executes more computations resulting in a more time consuming training. The number of the neurons in the hidden layer that give the biggest accuracy 98.20% in both training methods is 20. Though having more neurons doesn't necessarily means that is the optimal solution. For example the execution time of the first training algorithm using 20 neurons in the hidden layer is 05:32, while in the previous testing it is only 02:15 which gave accuracy of 97.90%. Having a system that requires real time decision makings such as the production line of the Food industry, it is more optimal having a quicker method that would require less computation time.

B. *Comparison of results between categories 1.1 and 1.2*

In this occasion, the results are also reasonable. As mentioned before the learning rate is responsible for the weight adjusting at the end of each iteration. Having a very small learning rate it results in a very slow update thus making the training process very slow, that's why the training time in the category 1.1 is much bigger than in the category 1.2. Though because of bigger value of the learning

rate in the category 1.2 it results in a less accurate classification. In conclusion, the most optimal classification between these 2 categories is the one in the category 1.2 with accuracy 98.60% and execution/training time of 2:21.

TABLE 3
2.1 category - 12 Features, Learning rate = 0.05

Percentage of correct classification / No. of iterations / time of training (mm:ss) No. of Features = **12** - Learning rate = **0.05**		Number of Neurons in the hidden layer		
		8	16	20
Training Method	Levenberg - Marquardt Algorithm	97.50% / 143 / 01:37	**99.00% / 49 / 01:12**	99.00% / 66 / 02:15
	Scaled conjugate gradient algorithm	96.60% / 480 / 02:18	98.10% / 717 / 03:11	98.70% / 1000 / 04:32

TABLE 4
2.2 category - 12 Features, Learning rate = 0.25

Percentage of correct classification / No. of iterations / time of training (mm:ss) No. of Features = **12** - Learning rate = **0.05**		Number of Neurons in the hidden layer		
		8	16	20
Training Method	Levenberg - Marquardt Algorithm	97.20% / 95 / 00:55	98.70% / 84 / 01:54	99.20% / 101/ 03:41
	Scaled conjugate gradient algorithm	96.00% / 761 / 03:17	98.40% / 760 / 02:45	98.80% / 716/ 02:06

C. Comparison of results in category 2.1

It is worth mentioning that the accuracy between 16 and 20 neurons in the first training method didn't change, because having too many neurons resulted in overfitting, also the training process was longer due to more computations as expected.

D. Comparison of results between categories 1 and 2

In category 2 the number of the input of the Neural Network is altered from 8 features to 12, thus resulting better accuracy overall. That means that during the feature selection technique the following 4 features had critical information that could be used for the classification process. Using 12 features as input, the Levenberg – Marquardt Algorithm, 16 neurons in the hidden layer and the learning rate parameter having the value of 0.05 resulted in the best optimal classification accuracy of 99.00% and 1:12 execution time.

VI. CONCLUSION

The quality and composition of food products are of significant importance during all stages of a food production flow line. Food manufacturing companies must have great standards regarding the quality of their products. Thus a technology is required to offer real-time monitoring, non-destructive, contactless and non-invasive, in contradiction to other complex chemical procedures. The technology of microwave sensors is an accurate fit to fill this need. The purpose of this thesis is to implement a machine learning algorithm to the sensor to recognize successfully the productions through the processing line. PCA extracted the correct features was the first step of processing the data that was implemented. PCA has allowed the implementation of the Neural Networks to classify. Following 24 different experiment classification training executions, it is concluded that a unique combination of some parameters such as number of neurons in the input and hidden layer give the optimal classification accuracy of 99.00% with an execution time of 1:12. Ultimately, the selection of an architecture for the neural network came down to trial and error.

ACKNOWLEDGMENT

I would like to thank my supervisors, Prof. Marc Desmulliez, Assistant Prof. Yoann Altmann and Dr. Sumanth Kumar Pavuluri for giving me an opportunity to work under their guidance and providing continual support throughout my thesis. I also want to thank all the professors and teachers in University of Burgundy and the coordinators of VIBOT and MsCV program for making this course possible. I also wish to acknowledge the MsCV students who helped me during the masters program and the love and support of my friends and family.

References

[1] M. Ferree, "What is Food Quality," *Food Distribution Research,* pp. 36-39, February 1973.

[2] O. L. Bo and E. Nyfors, "Application of microwave spectroscopy for the detection of water fraction and water salinity in water/oil/gas pipe flow," *Non-Crystalline Solids,* vol. 305, pp. 345-353, 2002.

[3] K. Saeed, A. .. Guyette, I. C. Hunter and R. D. Pollard, "Microstrip resonator technique for measuring dielectric permittivity of liquid solvents and for solution sensing," in *Proc. IEEE Int. Microw. Theory Tech.Soc. Symp.*, 2007.

[4] K. Joshi and R. Pollard, "Sensitivity analysis and experimental investigation of microstrip resonator technique for the in-process moisture/permittivity measurement of petrochemicals and emulsions of crude oil and water," in *Proc. IEEE Int. Microw. Theory Tech. Soc. Symp. Dig.*, 2006.

[5] K. H. Theisen and T. Diringer, "Microwave concentration measurement for process control in the sugar industry," *Proc. SIT,* vol. 60, pp. 79-92, 2000.

[6] S. Trabelsi and S. O. Nelson, "Microwave sensing of quality attributes of agricultural and food products," *IEEE Instrumentation & Measurement Magazine,* pp. 36 - 41, 21 January 2016.

[7] Z. Meng, Z. Wu and J. Gray, "Microwave Sensor Technologies for Food Evaluation and Analysis – Methods, Challenges and Solutions," *Transactions of the Institute of Measurement and Control,* 20 September 2017 .

[8] B. C. M., "Neural Networks for Pattern Recognition," *Oxford University Press ,* 1995.

[9] L. Fausett, "Fundamentals of Neural Networks: Architectures, Algorithms, and Applications," *Prentice Hall,* no. ISBN 0-13-334186-0, 1994.

[10] K. Potter, "Methods for presenting statistical information: The box plot," 2006 .

[11] B. Xiao, "Principal component analysis for feature extraction of image sequence," *IEEE Xplore*, no. 10.1109/CCTAE.2010.5544358, 2010.

[12] T. S. Ruprah, "Face Recognition Based on PCA Algorithm," *Special Issue of International Journal of Computer Science & Informatics*, vol. 2, no. 1, p. 2231–5292.

[13] S. S. D.Pham, "Training multilayered perceptrons for pattern recognition: a comparative study of four training algorithms," *International Journal of Machine Tools and Manufacture*, vol. 41, p. 419–430, 2001.

[14] Moller, "Neural Networks," vol. 6, p. 525–533, 1993.

Human hand gesture recognition and tracking to control a robotic arm and a collimator

Muhammad Zain Bashir and Gert Behiels

***Abstract*— This study is conducted to investigate the feasibility of using hand gestures to provide touch-less control for a digital radiography modality with special focus on two parameters; x-ray tube positioning and collimator (a device that narrows down the beam of x-rays) dimensions. For safety and testing purposes the task at hand is carried out on a robotic arm acting as the x-ray tube and a square box drawn on the screen acting as the actual collimator. Data filters have been applied to minimize the affect of unwanted hand movements resulting in a smooth robot trajectory because it allows the user to perform a more relaxed hand movement (i.e. the human hand will likely have slight movements in two axes while trying to move only in one.). The results observed how high promise for using such control in a real x-ray room setting.**

I. INTRODUCTION

Healthcare equipment manufacturers these days are trying to provide modality controls with minimum human interaction (MHI). One possibility to achieve this is by making use of Human computer interaction (HCI) and a Digital radiography modality control is one potential area of HCI application. To test its feasibility, a Universal robots UR3 robotic arm and a virtual collimator (box drawn on the screen) was used to test the quality and precision of control that hand gesture recognition and tracking could offer. This involved designing control gestures, filtering of hand tracking data and calculation of robot control commands. To recognize human hand gestures and track human hands in a video stream, an Intel depth camera, SR300 was used with its tracking and gesture recognition module.

II. METHODOLOGY

This section is divided into three subsections; State machine for gesture design, Filtering of tracking data and Robot and collimation control, each describing the major tasks involved in successful completion of this study.

A. State machine for gesture design

For robot control a *Grab* gesture was proposed which involves the user using two gestures, in a sequence, provided by the Intel software development kit (SDK); *Spreadfingers* and *Fist* gestures. This sequence is shown in Fig. 1. To realize this sequential behaviour, a state machine was used.

Fig. 2 shows that the state machine always starts in a *no hand* state and as soon as it receives a hand detected signal from the video stream, it enters the *hand detected state*. From here, the state can go back to the *no hand* state on receiving a no hand signal or can go to the *spreadfingers* state on receiving the spread fingers gesture signal. From the *spreadfingers* state the state can again move to the *no hand* state if the hand moves out of the field of view (FOV) of the camera or move forward to the *fist* state if the camera detects a fist gesture. This completes the *Grab* gesture and hand tracking data starts to be queried for the filtering stage after which control commands are calculated and sent to the robot controller to make the robot follow the user's hand. At any point in time the user can choose to stop the robot control by going back to the *spreadfingers* state by opening their hand.

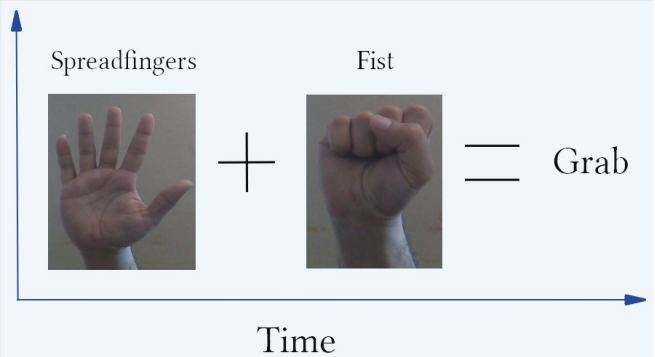


Fig. 1: Grab gesture sequence

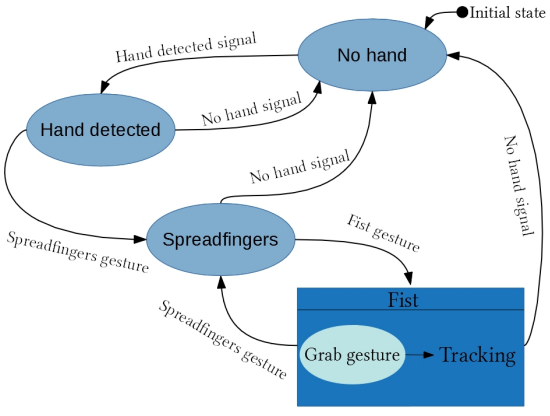


Fig. 2: State machine for Grab gesture

For virtual collimator control, a *Two hands pinch* gesture was proposed which makes use of, in sequence, the *Spreadfingers* and the *Two fingers pinch* gesture provided by the Intel SDK. This sequence is shown in Fig. 3.

Fig. 3 shows that the state machine for *Two hands pinch* gesture also starts with the *no hand* state and enters its child state *two hands* on detecting a two hands signal from the

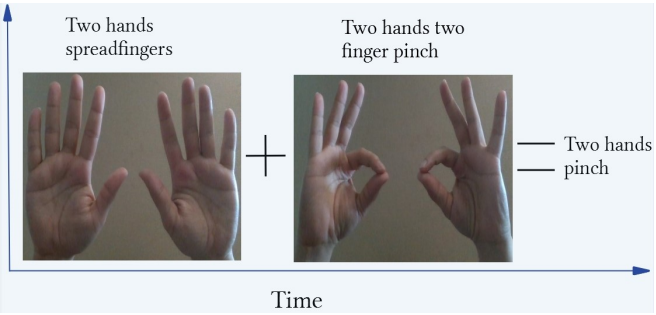


Fig. 3: Two hands pinch gesture

video stream. With entry in the *two hands* state the system immediately enters it's two parallel child states; *left hand* and *right hand*. State transitions within these two child states take place independently of each other. Each hand starts in a *not pinched* state and moves to the *pinched* state if a pinch is detected by the hand. If both hands enter the *pinched* state, the *Two hands pinch* gesture is completed and hand tracking data starts being queried so that dimension change control can commence. Once the user is satisfied with the collimator dimensions, they can stop this control gesture by opening one or both their hands. This takes the hand state to a *transition* state where it stays for $500ms$ before moving to the *not pinched state* again. The *two hands pinch* gesture is valid in the *transition* state as well. If in transition the state the system receives a pinched signal, it moves back to the *pinched* state.

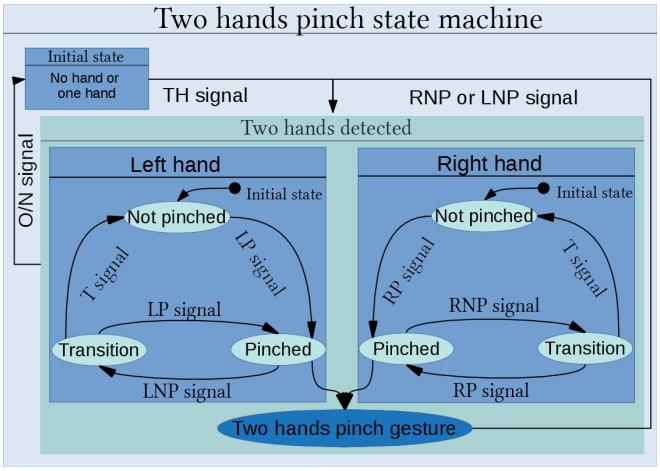


Fig. 4: State machine for two hands pinch gesture

B. Filtering the tracking data

For noise removal from tracking data, thresholding and Kalman filter [1] were used for the *Grab* gesture while and averaging filter was used for the *Two hands pinch* gesture.

1) Averaging: To change the collimator dimensions the positions of the center of masses of both hands were used to calculate a difference. This difference was then added to to the collimator dimensions and the collimator dimensions updated every iteration. Five difference values were stored in a FIFO memory and their average calculated to add to the collimator dimensions.

2) Thresholding: Different threshold values were tried and a value of 1 cm was to chosen to threshold the hand center of mass position in all three axes; x,y and z. This means that if the hand center of mass position is within 1 cm of the previous position, the previous position is kept otherwise it is updated with the new position received by the camera.

3) Kalman: A simple Kalman filter was chosen with the position and velocity of the hand as the state variables of the state $s(t)$:

$$s(t) = \begin{bmatrix} P_x(t) & P_y(t) & P_z(t) & V_x(t) & V_y(t) & V_z(t) \end{bmatrix}^T$$

The process model governing hand motion was modelled using equations of motion [3]:

$$x = x_0 + vt + \frac{1}{2}at^2 \tag{1}$$

where x is the current hand position, x_0 is the initial hand position, v is the hand velocity and, a is the hand acceleration and t is the time taken to reach x. A constant velocity was assumed so any accelerations that occur were assumed to be process noise.

The state transition matrix $\mathbf{F}_t$ [3] was chosen as:

$$\mathbf{F}_t = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where Δt is the time between two frames.

The measurement matrix $\mathbf{H}$ which maps the true state vector parameters onto the measurement domain was chosen as follows:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$\mathbf{Q}_t$ and $\mathbf{R}_t$ which are the process and the measurement noise covariance matrices [2] respectively were defined as follows:

$$\mathbf{Q}_t = \begin{bmatrix} E_{P_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & E_{P_y} & 0 & 0 & 0 & 0 \\ 0 & 0 & E_{P_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & E_{V_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & E_{V_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & E_{V_z} \end{bmatrix} \tag{4}$$

$$\mathbf{R}_t = \begin{bmatrix} M_{P_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & M_{P_y} & 0 & 0 & 0 & 0 \\ 0 & 0 & M_{P_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & M_{V_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & M_{V_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & M_{V_z} \end{bmatrix} \tag{5}$$

where E_p and E_v are variances in the position and velocity values of the process model respectively while M_p and M_v are variances in the measured position and velocity values.

Fig. 5 shows the result of filtering with $E_p = E_v = 5$ and $M_p = M_v = 0$ and then changing the noise values to $E_p = E_v = 0.001$ and $M_p = M_v = 0.05$ after 10 frames when trying to move along the x-axis. This *noise update* approach is used to prevent the filter from predicting values close to zero in the beginning because the filter is initialized with a a position and velocity values of zero. Starting with zero measurement and a high process noise ensures that the filter calculates its predictions based only on sensor values. The noise values are then adjusted to use both the noise and the measurement model. Fig. 6 shows, in 3D, the result of *noise update* approach when trying to move in 3D.

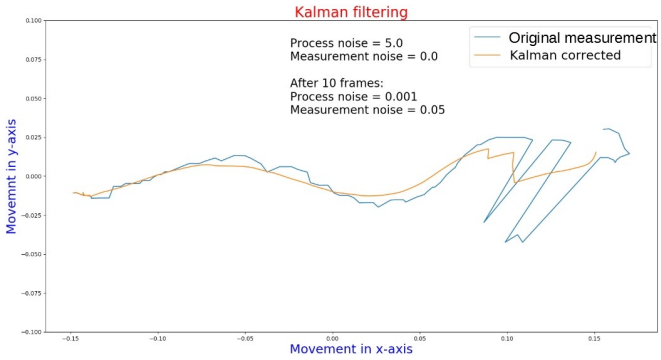


Fig. 5: Kalman filtering with noise update approach

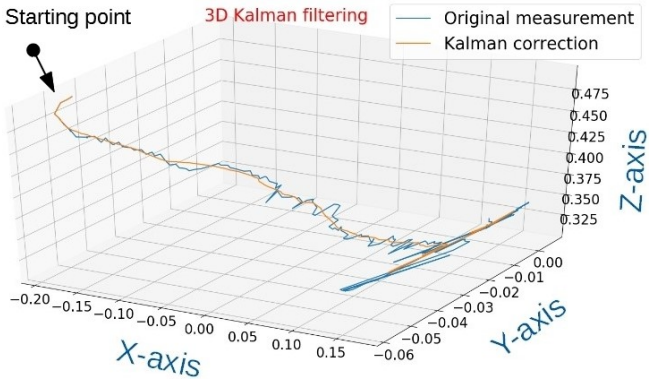


Fig. 6: Kalman filtering in 3D

C. Robot and collimator control

Once the data has been filtered, it is used to compute robot and collimation commands to be sent to their respective controllers.

1) Robot control: As soon as the user initiates a *Grab* gesture, the position of the current hand center of mass and the current robot position are stored, denoted as the position vectors $\mathbf{P}_i$ and $\mathbf{R}_i$ respectively. Each time a new filtered position data point is received, denoted as $\mathbf{P}_c$, a difference vector $\mathbf{d}$ is calculated using (6).

$$\mathbf{d} = \mathbf{P}_c - \mathbf{P}_i \tag{6}$$

The robot frame (right hand coordinate system) and the camera frame (left hand coordinate system) are not superimposed on each other so this difference can not be directly added to the initial robot position. Therefore, difference values are added to the corresponding axes of the robot to calculate new position vector, $\mathbf{R}_n$ of the robot end-effector according to (7).

$$\mathbf{R}_n = \begin{bmatrix} \mathbf{R}_{ix} \\ \mathbf{R}_{iy} \\ \mathbf{R}_{iz} \end{bmatrix} + \begin{bmatrix} -\mathbf{d}_z \\ \mathbf{d}_x \\ \mathbf{d}_y \end{bmatrix} \tag{7}$$

where $\mathbf{R}_i$ is the initial/current robot position. Once the new position command has been sent to the robot, the initial/current value is updated with the new position.

2) Collimator control: Since a virtual collimation (square box drawn on the screen) was dealt with for collimator control, all values are in pixel coordinates. As soon as the *two hands pinch* gesture is made, an initial distance value $\mathbf{D}_i$ is calculated as using (8)

$$\begin{bmatrix} \mathbf{D}_{ix} \\ \mathbf{D}_{iy} \end{bmatrix} = \begin{bmatrix} |H_{rix} - H_{lix}| \\ |H_{riy} - H_{liy}| \end{bmatrix} \tag{8}$$

where H_{rix} and H_{riy} are the initial (at gesture initiation) x and y coordinates of the right hand center of mass in pixels and H_{lix} and H_{liy} are the initial (at gesture initiation) x and y coordinates of the left hand center of mass in pixels.

When the two hands are brought together or pulled further apart, a current distance value is also calculated using (9) denoted by $\mathbf{D}_c$.

$$\begin{bmatrix} \mathbf{D}_{cx} \\ \mathbf{D}_{cy} \end{bmatrix} = \begin{bmatrix} |H_{rcx} - H_{lcx}| \\ |H_{rcy} - H_{lcy}| \end{bmatrix} \tag{9}$$

where H_{rcx} and H_{rcy} are the current x and y coordinates of the right hand center of mass in pixels and H_{lcx} and H_{lcy} are the current x and y coordinates of the left hand center of mass in pixels.

Once $\mathbf{D}_i$ and $\mathbf{D_c}$ have been calculated, their difference is taken to calculated using (10) to calculate the mount by which the collimator dimensions should change.

$$\mathbf{C} = \mathbf{D}_c - \mathbf{D}_i \tag{10}$$

As part of filtering mentioned in section II-B.1, this difference value is computed 5 times and an average taken before sending it to the nex stage. This average difference vector is then added to the initial collimator dimensions $\mathbf{N}_i$ to calculate the new collimator dimensions $\mathbf{N}_n$ using (11).

$$\begin{bmatrix} \mathbf{N}_{nx} \\ \mathbf{N}_{ny} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{ix} \\ \mathbf{N}_{iy} \end{bmatrix} + \begin{bmatrix} \mathbf{C}_x \\ \mathbf{C}_y \end{bmatrix} \tag{11}$$

Just before the cycle ends, initial collimator dimensions $\mathbf{N}_i$ are updated with the new collimation dimensions $\mathbf{N}_n$.

III. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

This study was conducted to check the feasibility of using hand gestures in a real x-ray room setting and the results of this show high promise for this possibility. The use of state machine [5] to use gestures provided by Intel SDK in a sequence has also been shown. This means that these gestures can be used in combination to come up with several new gestures each for a different task

B. Future Works

Since going from one position to another, the robot makes a series of acceleration, constant velocity and deceleration cycles, it introduces a lot of braking and consequently a jerky trajectory for the robot. The most important future work is thus smoothing out hte robot trajectory by planning its trajectory. There are some limitations of the SDK for example not detecting hands in certain orientations so another future investigation would be to use deep learning frameworks for hand detection. The use of other libraries such as OpenPose for hand pose and gesture recognition [6] should also be considered. An inverse kinematic analysis [4] of the robot must also be made to work out the singular positions of the robot so that move commands for those positions are not published.

IV. ACKNOWLEDGMENTS

We gratefully acknowledge the contribution of Agfa HealthCare Nv for the provision of all the necessary facilities to conduct this study.

REFERENCES

[1] Kalman, Rudolph Emil," A new approach to linear filtering and prediction problems", Journal of basic Engineering 82.1 (1960): 35-45.
[2] Bishop, Gary, and Greg Welch. "An introduction to the Kalman filter." Proc of SIGGRAPH, Course 8.27599-23175 (2001): 41.
[3] Miners, B. W. "Kalman filtering and prediction for hand tracking." An Advanced DSP Project for Dr. RD Dony (2001).
[4] Buss, Samuel R. "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods." IEEE Journal of Robotics and Automation 17.1-19 (2004): 16.
[5] Gill, Arthur. "Introduction to the theory of finite-state machines." (1962).
[6] Suryanarayan, Poonam, Anbumani Subramanian, and Dinesh Mandalapu. "Dynamic hand pose recognition using depth data." Pattern Recognition (ICPR), 2010 20th International Conference on. IEEE, 2010

Deep Into Water: Reflective Areas Detection using Deep Learning

Marc Blanchon

Abstract— **Scene segmentation and understanding is an open problem in computer vision and Artificial Intelligence (AI). Indeed with the recent advances in AI, the need for more accurate and efficient scene understanding models is more evident than before. This necessity and being able to benefit from strong segmentation power of Deep Convolutional Neural Networks(DCNN), has motivated the research community to propose numerous algorithms in the past years. Most of the recent algorithms uses a single modality as a source of information, and lack of multi-modal system is noticeable. In addition, since RGB images being the natural and most general choice of information, many existing algorithms of scene understanding are not able to provide robust results on detecting reflective areas.**

Therefore in this paper, we propose (i) a multimodal acquision system and dataset using RGB, NIR (Near Infra-Red) and polarimetric cameras and (ii) a model that using a discriminative source of information and DCNN power is able to segment reflective areas. Testing the proposed algorithm using different modalities illustrates the robustness and effectiveness of a multi-modal system, specially polarimetric, for reflective areas.

I. INTRODUCTION

Scene segmentation and understanding has been a popular topic in the field of robotics, AI (Artificial Intelligence) and vision. This task is specially challenging due to visual search difficulty, image complexity, object/scene recognition, and so on... With a special interests for robotics, it is obvious that multi-sensors can be used as source of information for scene segmentation. For that reason :

- The first motivation was to implement a multi-modal framework using ROS (Robot Operating System), where different sensors are integrated and a multi-modal dataset can be gathered. Surprisingly, the majority of multi-modal open datasets, consist of the RGB, depth, and/or 3D point-clouds and the use of different cameras such as NIR (Near Infra-Red), and PolarCam (Polarimetric Camera) were very rare or non-existent.
- To this extend a second motivation was set to create a public multi-modal dataset with and without reflective areas.
- Finally, due to the recent advances of DL (Deep Learning) in pixel segmentation approaches, the final target was set to create an innovating method based on convolutional networks.

In the reminder of this paper, Sect. II covers a brief background on DL algorithms (limited to scene segmentation) and polarimetric cameras. Sect. III explains the acquisition setup and gathered dataset. Sect. IV the proposed method is explained, and finally in Sect. V illustraites our reslults and finnaly Sect. VI conclude the presented work.

This work was supported by ANR VIPeR

M. Blanchon is with LE2I, Laboratoire d'Electronique, Informatique et Image, VIBOT ERL-CNRS 6000, Universite de Bourgogne France Comte fr.marc.blanchon@gmail.com

II. BACKGROUND

A. Deep Learning

Deep learning, a predominant concept in the context of research, is a sub-domain of Machine Learning (ML) and AI (Artificial Intelligence), that allows the learning of features directly from input data. Therefore eliminating the feature extraction and its subsequent steps within classical ML methods, up to, the final step, prediction/classification. It is notable that this method is neither more nor less than a model optimizer to achieve a generic form of description of the input. To speak briefly about the history of deep learning and its usefulness, it is possible to refer to its creation or at least to the creation of the first working algorithm, the MLP (Multi Layer Perceptron) by Alexey Ivakhnenko and Lapa in 1965 [7]. After many years of refining, the biggest advance remains the AlexNet classification network [9], obtaining more convincing results on the ImageNet Challenge [5]. This solution promoted the use of CNN (Convolution Neural Network) in terms of image processing and image recognition tasks. As far as the limits of basic image processing are concerned, deep learning allows a noticeable progress in image annotation.

The first remarkable DL approach to segmentation is from Long et al. [11], allowing a segmentation of image of any sizes without fully connected layers. Previously, the noticeable problem of Deep Learning Network approach was the size reduction. Indeed, the max pooling layers of the network directly induce the reduction of information and by extension the loss of information.

As the years and the evolution of power increased, multiple networks, each with better performance, have been released. Multiple networks have been benchmarked on VOC2012 evaluation server[(*)] and, a positive evolution in precision every year is perceptible, starting with the Fully Convolutional Network of Long et al. [11] to 67.2% in 2014. Then the SegNet [1] by Vijay Badrinarayanan et al. in 2015 with approximately 60%. Then a big step in the precision has been remarked with a score of 75% in 2015 with the dilated convolutions of Fisher et al. [20]. In 2016, Deep Lab v2 [3] achieved a score of 80% (Liang-Chieh et al.). At the end of 2016, a significant improvement called RefineNet [10]

[(*)]VOC 2012 evaluation server : host.robots.ox.ac.uk/pascal/VOC/voc2012/

was reported by Guosheng et al. with 84.2%. To end with an evolution observing a convergence trend, DeepLab v3 [4] obtained 85.7% accuracy in mid 2017.

To conclude on the state of the art about Deep Learning, the networks across the years tends to reach perfect cognition in terms of semantic scene segmentation (understanding). It is very encouraging to see a field of research so motivating and getting such good results in a short time (compared to other areas).

B. Polarimetry

The polarimetry (Fig.1) is the science of measuring the polarized state of the light. In consequence, a polarimetric camera gives the experience of recovering the light changes in the captured environment. Because of this behaviour and the main goal of the thesis, the information from this camera could be the perfect candidate as a discriminant factor aiming the complex scene semantic understanding. To the views of the general idea of reflection zone detection, the light changes is the perfect descriptor of reflective regions.

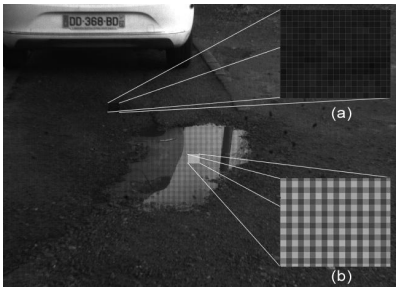


Fig. 1. Reflection Influence on Polarimetry. (a) is a zoom on the non-polarized area, (b) is on a polarized area. Clearly, on a polarized surface, the micro-grid appear and reveal an intensity change according to the polarizer affected.

Despite the useful and informative aspects of polarimetric system the use of such cameras have been very limited, due to the limitation of hardware and automatic integration, up to recent years. Using the DoFP (Division of Focal Plane) technique the new PolarCam (Polarimetric Camera) are introduced in the market, which makes their integration into acquisition systems much more feasible [15], [12], [14]. DoFP technique allows to have the polarized filters in an array directly on the detector, as it is shown in Fig. 1. In this design four polarized filters, each with unique angle, are used to capture four different measurements instantly in one shot.

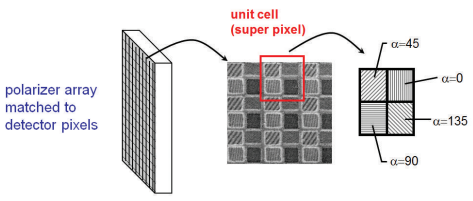


Fig. 2. Polarimetric Camera Explanation. Image source: Trioptics.

In the images captured with the PolarCam, every four pixels in image space correspond to one pixel in the final scene. Although this technique allows for instant and fast acquisition of polarized measurements, due to angle pattern of the filter array, pre-processing and interpolation is required to get the final polarized images.

Many image processing and computer vision applications can benefit from recent DoFP-PolarCam (Fig.2). For example, Kai Berger et al. [2] proposed a method for depth recovering from polarimetric images in urban environment, treating the modality as a common RGB camera. Also, despite its complexity, Ratliff et al. [19] found a strategy for interpolation of such a specific data.

Using other polarized setups, within this research scope, few methods were proposed for water detection using polarized information. Nguyen et al. [13] proposed a method for water tracking with a polarized stereo system achieving an approximate accuracy of 65% exceeding the previous state of the art method accuracy of $\approx 45\%$. Another application in recognition of mud for autonomous robotics, Rankin et al. [18] proposed a method and offered a full benchmark for its segmentation processes.

III. DATA ACQUISITION

(a) RGB Kinect Image.

(b) RGB UCam Image.

(c) NIR Image.

(d) Polarimetric Image.

Fig. 3. Polar-VIBOT Images Example of all modalities for the same snapshot.

The acquisition setup is composed of 4 cameras, as near as possible: PolarCam, Kinect 2, IDS UCam and NIR. Figure 4 shows the acquisition setup. The idea behind this camera layout is to have an optimized field of view. This optimization consists of having the maximum common scene in the 4 cameras spaces. By bringing the sensors closer, the respective field of view for each camera will cross each other. At the end, the most common field of view is the biggest common image (Fig.3).

The environment used to monitor the cameras and to acquire the image is ROS [17], [6] indigo. This framework facilitates the communication between multi-camera and computer and allows a more efficient synchronization. In addition to this, the monitoring therefore admits a common area of work.

Fig. 4. Camera Setup

A. Synchronization

The synchronization is one of the critical point concerning the dataset acquisition. Since PolarCam returns the data at higher frequency compared to the other two cameras, there was two possibilities to be able to have a consistent dataset and the chosen one leads to a synchronization on the lowest frequency camera.

The frequencies from the different cameras are all different and not synchronized. In the acquisition of a multi-modal dataset, the key principle remain in the synchronization of the images.

Because of these unsynchronized signals, a strategy of recovery has been implemented(**). Using the ROS Framework, the images follow a pipe and are passing through different topics. Because synchronization wasn't possible, the first previous image is taken after triggering the image storage on the lowest frequency signal. It is considerable that some problem can occur such as image shift, motion blur or displacement which can results in image displacement. All this noise is then reduced at each acquisition meaning that if the acquisition is not properly done, another snapshot trigger can be repeated till the acquisition in the four modalities is corresponding to the desired data chart.

Following the same principle, the acquisition can be performed in term of sequences which results at the end in a multitude of images grabbed one by one and separated by a specified artificial frame rate. In addition to synchronized acquisition, a bag file storing all four modalities using ROS framework is stored as well, which is used specifically for calibration purposes, explained in the next section.

B. Calibration

As previously explained, one of the functionality and key advantage of ROS environment is the possibility of creating bags, saving the selected data stream. In order to know extrinsics and intrinsics parameters of the cameras as well as preparing the field for the multi-modality, the calibration was performed using a free library called Kalibr [16].

As commonly used, a calibration pattern is disposed and moved in front of the camera while recording. The Kalibr toolbox then match features from all the views to deduce intrinsic and extrinsic parameters of the cameras. An abstraction model for the cameras is used, for every cameras, they are considered as pinhole with a radial-tangential distortion model [8].

(**)The code can be found on Github: `https://github.com/BlanchonMarc/Ros_AcquisitionFromTopics`

IV. METHOD

Deep Learning allows to have a generic method, therefore it is possible to use a standard segmentation network from the state-of-the-art for similar applications. In this research SegNet [1] was selected as the primary and base segmentation network. SegNet [1] is a segmentation network, which allows semantic understanding of the scene. This network was selected, due to its architecture (encoder-decoder), relatively fast training time, its original use (scene understanding) and finally since its architecture facilitates the late fusion of different modalities.

This section presents the Polar-VIBOT dataset, the necessary pre-processing steps, the network training, and the evaluation process of the developed pipeline.

A. Polar-VIBOT Dataset

The dataset is a major factor of effectiveness. The decision to make the acquisition of a dataset was taken to achieve three objectives: (i) to have a representative dataset in term of water and reflective area detection, (ii) to have a more balance dataset in terms of classes, and (iii) to have a multimodal dataset (NIR, PolarCam, and RGB) suited for autonomous navigation and vision applications beside scene segmentation.

Polar-VIBOT contains 178 images (for each modalities) and has been dispatched in 8 classes: unlabeled, sky, water, windows, road, car, buildings and other. These 8 classes are considered important in the field of autonomous navigation which also leads to an increased usefulness of the dataset in multiple tasks. For the purpose of scene segmentation application, hand-made ground-truth are created on a set of selected modalities (Polarimetry and RGB from UCam), which are explained in the following.

B. Towards the Reflective Area Detection

Concerning the main objectives, segmentation of the reflective surfaces, some constraints are put in place:

- Define the best modality to perform the task.
- Transform the chosen modality to become discriminant (if necessary).
- Pre-process the modality to adapt to the image size of the dataset.

Being aware of the usefulness of a discriminative data in term of DL is very important, leading to increased overall accuracy in the desired task and allowing by extension a better definition of the model. Having a discriminative data in DL can increase the overall accuracy and allows by extension a better definition of the model.

1) Modality Choice & Transformation: The dataset being composed of 4 different modalities (5 points of view), one of the major task was to define the proper modality. In this case, the polarimetry and RGB have been chosen. Polarimetry, because of its ability of recovering the light influence in the scene. Reflection is defined as the change of direction of the light on a surface. Polarimetric camera has the ability to recover the Degree of Polarization (DoP) and Angle of Polarization (AoP).

In order to perform a segmentation using polarimetric images, many factors have to be taken into account. One of the major problem is that, as a direct consequence of the size of the dataset (204 images), the usage of raw image from the camera is not possible. Since the amount of data being considerably low, the augmentation is a mandatory step that cannot be easily performed on raw data. The grid created by the 4 pixel of polarization cannot be straightly rotated in an usual way. In addition, no interpolation are feasible because the creation of polarization data is a complex task or even not permitted.

As a consequence of these previous facts, the usage of Ratliff et al. [19] allowing a recovery of AoP, DoP and intensity measures for each images is necessary. These three parametric images can then be combined in order to mimic the HSL color space.

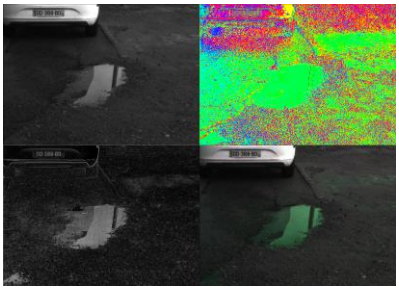


Fig. 5. Ratliff1 transformation and HSL creation results. Top left : intensity, top right: AoP, bottom left: DoP, bottom right: HSL (H = AoP , S = DoP, L = intensity).

2) Ratliff Interpolation Method: Conventional interpolations are process-able on simple modalities but when addressing a problem inducing polarimetry, the physical meaning of the data oblige to use a specific methods. Ratliff et al. [19] provides an interpolating solutions to recover multiple information from a raw polarimetric image. Key advantages of this extraction is that any of these interpolated images can then be processed using conventional interpolation process.

The first step to use this method is to describe the polarized light in the form of Stokes vector. Stokes vector are by definition a set of four values that describe the polarization state of an electromagnetic wave. To compute AoP (Angle of Polarization) and DoP (Degree of Polarization)::

$$\mathrm{DoP} = \sqrt{\bar{s_1}^2 + \bar{s_2}^2} \tag{1}$$

$$\mathrm{AoP} = \frac{1}{2}\tan^{-1}\left(\frac{s_1}{s_2}\right) \tag{2}$$

And the intensity being the combination of all polarized states intensities:

$$\mathrm{I} = \frac{P_0 + P_{45} + P_{90} + P_{135}}{2} \tag{3}$$

At the end, any image can be computed using the three descriptive images. A constraint come from the AoP, because these descriptive values are periodic π, in consequence, for an assignation, AoP have to be multiplied by 2. For an example of combination of previously computed parameters, the HSL image contains:

$$\mathrm{HSL} \Longrightarrow \quad H \longrightarrow 2 * \mathrm{AoP}, \quad S \longrightarrow \mathrm{DoP}, \quad L \longrightarrow I \tag{4}$$

3) Augmentation: After applying Ratliff [19] interpolation, it is now possible to apply different transformation for data augmentation. The considered dataset being insufficient in size, the augmentation of data is a commonly used method allowing a proper training, validation and testing set. Many other advantages leads to an usage of this procedure, the main one being the avoidance of over-fitting.

As transformation methods to create new images, three processes have been selected:

- Rotation
- Flipping (in all direction)
- Random distortion

These three procedures are applied using augmentator Library that allows a random combination of randomly distributed transformation in order to recover a correct amount of images. The transformation is exactly the same for the image and the ground truth image. Thanks to the augmentation procedure, from 204 images (177 validated after deletion of incoherent images), the dataset obtained contains 2124 images.

The next procedure consists of randomizing the repartition of images in 3 different sets (6 considering the ground truth sets) in order to have the training, the testing and the validation containing each 708 images, all different. This randomization allow a complete control over the visibility of the network. Visibility of the network in this case being, restricting the network to see only new images and never see the same images in all the different sets.

4) Network Training and Estimation: Now that the input of the network is fixed and adapted, it is possible to move on to the parametric estimation in term of deep learning process. The training of the network and its estimation also includes their own set of constraints and objectives.

- Choosing the architecture
- Estimating the correct parameters and methods allowing an optimal training
- Defining the Metrics that will allow a correct appreciation of the results

Based on the results obtained (influence of the parameters), Learning Rate (LR) will be set to 10^{-4} and the number of epoch to 500. The used loss function will be Cross Entropy Loss and the Optimizer Adam.

V. RESULTS & DISCUSSION

This section presents the obtained results. The experiments have been conducted in the same way for two modalities of RGB and polarimetry from the Polar-VIBOT , only the polarimetry will be shown and RGB will serve as a comparison factor.

In terms of accuracy per classes, those computation have been performed on test set with the same constraints as the validation (randomness). Despite of the part of randomness in the estimation, the estimated characteristics are similarly reflecting the performances of the proposed method. As a reminder, training, validation and test sets are all composed of 708 images. To finish, the experiments was performed using a dedicated server composed of an Nvidia Tesla K40c (12GB Memory) GPU, 128GB of RAM and two CPU accumulating a total of 24 physical cores (48 threads). As explained in section IV-B.4, the learning rate $= 10^{-4}$ and maximum number of epochs 500.

Concerning the display of the results, all images and graphs respond to a color chart (Fig.6). Each color will have an assigned class present in Polar-VIBOT .

Fig. 6. Color Chart.

A. Polarimetry

The evolution curve of metrics showing overall accuracy, F1 Score, IoU and mean accuracy can be computed.

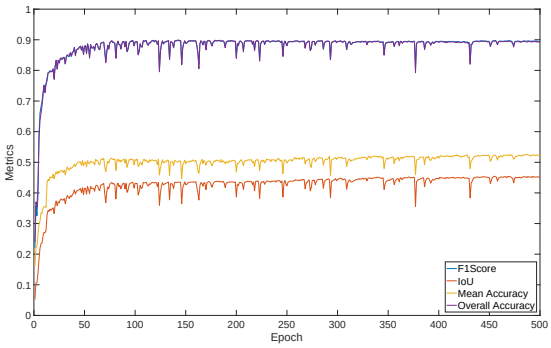


Fig. 7. Polarimetry Results - Metrics Estimation.

As it is visible, F1 score and Overall Accuracy are mingled. Another remark on the curve remains on its shape. Indeed after 500 epochs, the metrics continues to increase. This suggests that the training could have been longer and that the accuracy of the model has not reached its optimum.

B. Test Set Model Estimation

From the test set output of the trained model, the accuracy can be deduced visually.

Also from the test set can be deduced the precision per classes.

As the Fig.9 show, the 8 classes are convincingly detected. Some classes need clarification. ”None” class stands for ”not useful for the task”, this is the scene areas that are not necessary for the thesis application. On the other hand the ”Unlabeled” class mainly comes from hand-made segmentation mistakes. As the pixel-wise segmentation needs a label per pixel, some small areas have been missed when performing annotation which leads to an 8^{th} class representing more or less the human error.

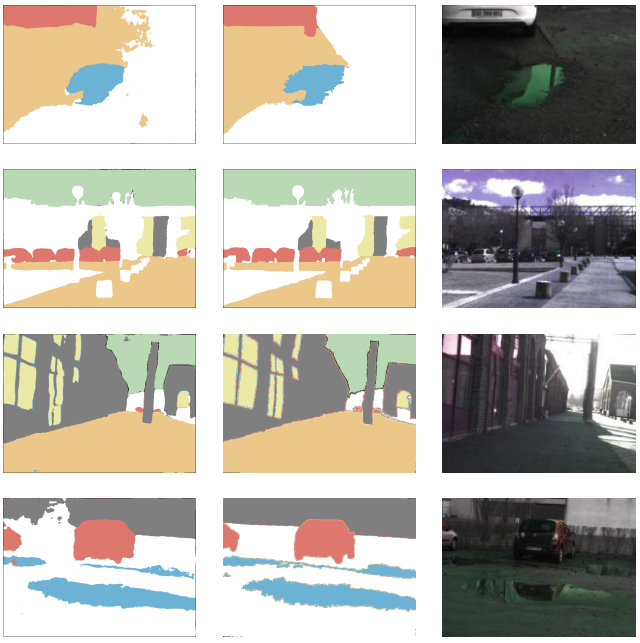


Fig. 8. Polarimetry Results - Test Set Output. Left is Prediction, Center is Ground Truth and Right is Input Image (Polarimetry HSL).

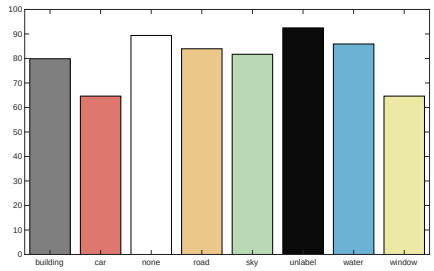


Fig. 9. Polarimetry Results - Accuracy per classes.

Now for the rest of the classes, the lowest accuracy per classes corresponds to the windows and cars which is one of the biggest challenge composing the dataset. This results remains remarkable, as well as the other classes obtaining high precisions. Still, it is considerable that the model hasn't reached optimum accuracy, so the model may be improved by training for more epochs.

C. Comparison

In this section a discussion comparing polarimetry and RGB modalities performances are presented. The first difference between the two models is about the reached states. Indeed, despite of the same parameters initialization, the two trainings have reached different state, while one is still

TABLE I
ACCURACY DIFFERENCE.

	Unlabeled	Sky	Water	Windows	Road	Cars	Building	None
Polarimetry	92.49 %	81.71 %	85.91 %	64.35 %	83.95 %	64.63 %	79.89 %	89.38 %
RGB	80.14 %	89.57 %	78.61 %	44.50 %	78.45 %	48.48 %	67.84 %	83.4 %
Difference	12.34 %	-7.86%	7.29 %	19.85 %	5.5 %	16.25 %	12.05 %	5.98 %

optimizable, the other one has already reached its optimal state. Knowing this fact, an assumption about reaching even better results on polarimetry segmentation (with SegNet and same parameters) is possible.

As the table I show, only one class is in deficit when taking polarimetry performances as reference against RGB. In order to show the accuracy improvements, it is possible to make a difference between the results of each process.

$$\text{Accuracy Difference} = \text{Polarimetry Accuracy} - \text{RGB Accuracy} \quad (5)$$

As previously found in table I, only one class perform less by 7.86% compared to RGB, the Sky class. There is an explanation to this behaviour, RGB is strictly referring and segmenting according to color intensity. Despite the effort made to have a generic dataset, the sky remains on the same tone (blue) which leads to a significant advantage on RGB modality.

Surprisingly, even for the classes not designed to be segmented easily by polarimetry (Building, None), the model trained still perform better than RGB.

VI. CONCLUSION AND FUTURE WORK

In conclusion, polarimetry trained model exceeds in every respect the RGB trained model as shown in Section V-C. The method proposed as well as the created dataset proved their usefulness in the scope of semantic pixel-wise segmentation. The initial idea of using PolarCam as a discriminant factor was a great choice and has proved its consistency applied on such tasks. Nevertheless, the exploitation of data must be considered as a potential disadvantage. Indeed, the proposals to use polarimetry are effective but require a time of implementation and constraints. In addition to this, there is no use of raw images which can cause problems in some applications. Being limited by the possible interpolations, the intermediate passage by a Ratliff interpolation is necessary in order to operate the proposed method. Considering multi-modality, clearly the single modality exploitation exceeded all expectations in terms of performance. This is encouraging to tackle multi-modality, which is supposed to make the process even more precise.

REFERENCES

[1] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *CoRR*, abs/1505.07293, 2015.

[2] Kai Berger, Randolph Voorhies, and Larry H Matthies. Depth from stereo polarization in specular scenes for urban robotics. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1966–1973. IEEE, 2017.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] Willow Garage. ROS: Robot Operating System, 2007.

[7] Valentin Grigorévich Ivakhnenko, Alekseĭ Grigorevich Lapa. Cybernetic predicting devices. Technical report, PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, 1966.

[8] Juho Kannala and Sami S Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1335–1340, 2006.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *CoRR*, abs/1611.06612, 2016.

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[12] James Millerd, Neal Brock, John Hayes, Michael North-Morris, Brad Kimbrough, and James Wyant. Pixelated phase-mask dynamic interferometers. In *Fringe 2005*, pages 640–647. Springer, 2006.

[13] Chuong V Nguyen, Michael Milford, and Robert Mahony. 3d tracking of water hazards with polarized stereo cameras. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5251–5257. IEEE, 2017.

[14] Gregory P Nordin, Jeffrey T Meier, Panfilo C Deguzman, and Michael W Jones. Diffractive optical element for stokes vector measurement with a focal plane array. In *Polarization: Measurement, Analysis, and Remote Sensing II*, volume 3754, pages 169–178. International Society for Optics and Photonics, 1999.

[15] Gregory P Nordin, Jeffrey T Meier, Panfilo C Deguzman, and Michael W Jones. Micropolarizer array for infrared imaging polarimetry. *JOSA A*, 16(5):1168–1174, 1999.

[16] Paul Furgale Jrme Maye Jrn Rehder Thomas Schneider Thomas Schneider . Kalibr, 2007.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[18] Arturo L Rankin and Larry H Matthies. Passive sensor evaluation for unmanned ground vehicle mud detection. *Journal of Field Robotics*, 27(4):473–490, 2010.

[19] Bradley M Ratliff, Charles F LaCasse, and J Scott Tyo. Interpolation strategies for reducing ifov artifacts in microgrid polarimeter imagery. *Optics express*, 17(11):9112–9125, 2009.

[20] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.

Human Localization in Robotized Warehouse Using Artificial Landmarks on ground

Dousai Nayee Muddin Khan and Gaël Écorchard

***Abstract*— Human Localization inside the robotized warehouse is defined as the problem to find the exact position and orientation of camera with the presence of continuous movement of robots and humans. To navigate reliably inside the warehouse, human should be localized its location precisely. Localization can be explained in two ways, one is to localize the position of the camera with respect to the environment and the second case can be described as to localize with respect to mobile robots within the warehouse. This paper states the problems and issues from the first case i.e.., to localize the human wearing camera using a safety vest in a robotized warehouse. The localization of human in our warehouse can be achieved by detecting the tagged stickers on the ground.**

I. INTRODUCTION

In present days, most of the industries are fully automated their warehouses with autonomous robots which can increase the productivity and decrease the human labor. But most of the warehouses have the challenge's to interfere their warehouses with humans to increase more reliable and flexible environment. In the present working warehouse, there are many mobile robots which are continuously moving inside the warehouse with the help of artificial landmarks as Data Matrix Codes, simultaneously they are mapping, localizing and navigating with respect to other mobile robots inside the warehouse. All the communication in between them is maintained by robot manager. These robots are working basically to move the racks and bring to the goal position or delivery position. The vital role to interfere human in this warehouse is when there is any object fallen from the rack and it has to be picked by human. By this interference of humans inside the warehouse, the humans position has to be localized and should be communicated with robot manager to give the present location, so that the robots can know the position of the human and plan there path accordingly. The problem of human localization can be sensed and known by using many localizing sensors available in the market like Cameras, LIDAR (Light Detection and Ranging) [1], GPS [2] etc .., Due to many constrains with lack of indoor application for GPS and complexity over LIDAR we preferred to choose cameras for this problem. Cameras are cheaper and easy to assemble and carry on our safety vest. Based on previously designed and implemented algorithms like SLAM (Simultaneous Localization And Mapping) [3], is used when we aren't aware of the surrounding i.e.., unknown map. But in our case we are dealing to recognize the markers and get the position of camera respectively known as Image-based camera localization. Image-based camera localization can be explained as to localize the camera inside the respective environment from the captured images or videos. This localization can be implemented in known or unknown environment. The paper [5] explain about a perspective-n-point (PnP) algorithm which is defined as the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The fig.1 shows the simple flow chart of localization with respective to PnP problem in the environment.

Authors are with the Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), Czech Technical University in Prague, Czech Republic, nayeem.khan@stu.upes.ac.in

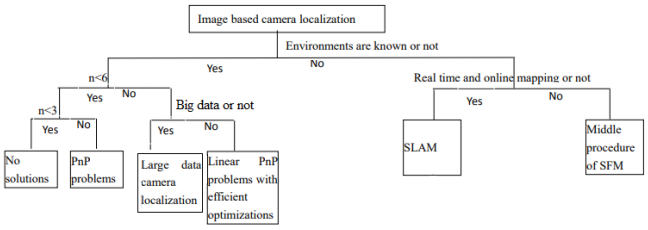


Fig. 1. Image based Camera Localization(Image Source : [4])

II. DATASET COLLECTION

Localization of human being in warehouse with live on board can be a problematic and hard to carry the whole system all the way moving around the warehouse. To make it easier for testing purpose we can make a dataset with simple artificial landmarks on the ground. Artificial landmarks are chosen as the items to detect for the developed algorithms. So to make it easier and flexible for our thesis we create an environment with required sensors and by choosing data matrices of 10 centimeter wide square placed on the ground. For the collection of data in the warehouse, the localization of the ground stickers plays a vital role. We have made our work space as about $4m$ X $4m$. The important precautions taken care to build the dataset are:

- To get the better result, the ground should be bright in colour without any reflective objects around
- Ensure to have a sharp image from Basler camera by opening the shutter to the maximum to reduce the depth of field
- Distance to a camera from the ground should be approximately around 1-1.5metre
- Enough amount of light of about 500-700lux

By taking all the precautions and cautions for the collection of dataset, we are ready to record our dataset by using the

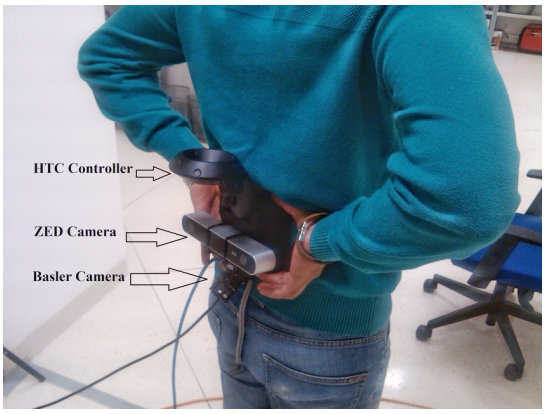


Fig. 2. Setup used for localization

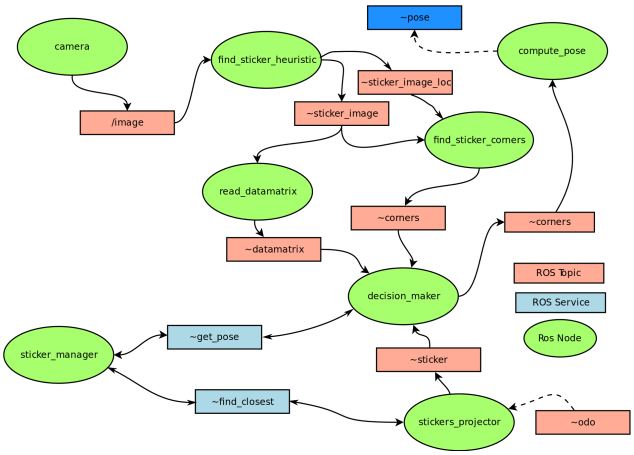


Fig. 3. Algorithms developed

setup shown in fig.2. The recorded dataset output will be used as a reference to test for the explained algorithms. Data is collected in different parameters based on human walking, light and distance to the ground. Their cases can be explained as: Normal and steady walk with a distance of about one metre from the ground with $550lux$ illuminance with an angle of 45 degree. The other case is fast walking with the same distance and angle and about $650lux$ illuminance. Data acquisition was recorded in many other cases also to test the flexibility of algorithms.

III. Implementation

As we detect the stickers on the ground then it's possible to get the position of the camera and then get the position of a human being located inside the warehouse. The proposed methods to detect the landmarks on the ground can be explained and developed as shown in fig.3

A. Detecting Stickers

The first thing we need to do from the video stream is to detect the stickers from the ground based on feature matching. Feature matching is basically described as to match features between two images. So for one image we will give our reference landmark image and the other image is from

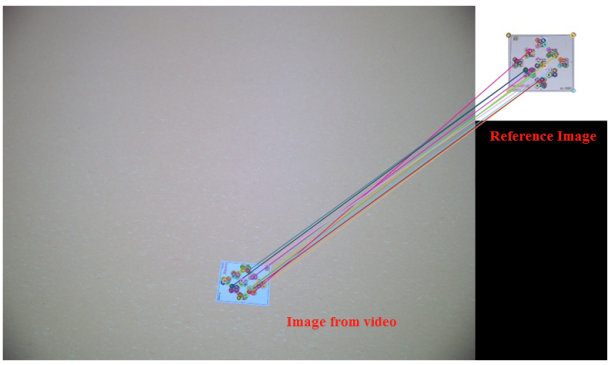


Fig. 4. ORB Feature Matching Algorithm for sticker detection

the video stream, which has to use some feature detection algorithms like SIFT(Scale-Invariant Feature Transform) [6], SURF(Speeded Up Robust Features) [7] or ORB (Oriented FAST and Rotated BRIEF) Feature Detectors [8]. we are using ORB features because they have faster computational time, rotation invariant and resistant to noise. The reference and the image from the video sequence should be of the same size. If there are many features detected from the video sequence, ORB will find the closest neighbour with the lowest distance and more true matches. The results are shown in fig.4

B. Extraction of Region of Interest

The main aim to extract a region of interest is to minimize the computational time. The DataMatrix stickers are very sensitive with respect to the size of the image. Once we detect the sticker from the video sequence as from the above section, we must get the ROI of this respective image. There are many ways to get ROI of the image like based on K- means clustering algorithm [9] or based on blob detection [10]. By calculating the mean we can find the nearest neighbour, which acts as an input for the k-means algorithm. The K-means clustering algorithm will divide the cluster into three regions in both x and y-axis directions and calculate the mean of x and y areas respectively. While blob (binary large object) detection is basically to find the area of touching pixels with the same logical state. Blob extraction, also known as region detection or labelling, is an image segmentation technique that categorizes the pixels in an image as belonging to one of many discrete regions. After implementing and testing the algorithms we have noted and observed our results.

C. Data Matrix Detection

Data Matrix recognition can be done in two different ways, one is to find the corners of the data matrix while the other is using *libdmtx* library [11]. By considering *libdmtx library*, the extracted ROI from the previous region should be a clear and sharp image. If the image is blurred there is very less probability to find the data matrix by using this library. While we tested this library on video sequence by grabbing the images we have faced many issued with blurred images. If the images are blurred then the chances

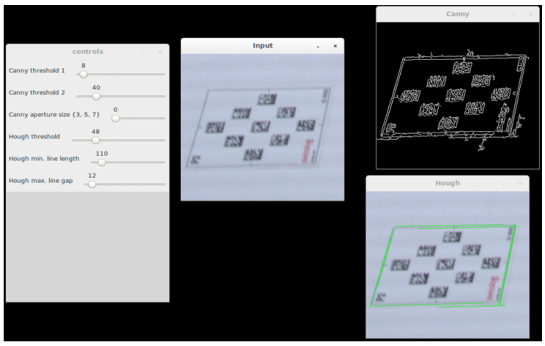


Fig. 5. Results from Hough Line transform

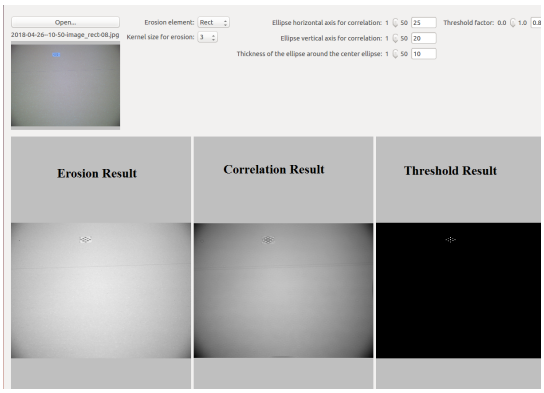


Fig. 6. Developed GUI for detection of stickers

of decoding the image is highly impossible, so the other methods can be explained as to find the corners or lines from the ROI image. There are many algorithms explained and implemented for a corner and line detection, among them Harris corner detection [12] and Hough line transform [13] is considered as to get the better results. The implementation of the result is shown in fig.5.

D. Acquisition of Position and Orientation

To get the position and orientation of the stickers camera with respect to the world frame, we have implemented compute_pose node which will get the pinhole model matrices from world coordinate to the camera frame. This frames are observed by developing a TF listener. The position of the camera can be calculated by using Pnp function called as pinhole model. By using pnp function we can find the object pose from 3D-2D point correspondences. Pnp function is basically known by using the intrinsic parameters of a Basler camera. It uses four corners from the above section to get the object and image points. Output vector rotation vector is solved by using Rodrigues notion from the rotational matrix, it transforms 3D world coordinate to 3D coordinates relative to camera centre. If we have a three dimensional vector $r = [r_x, r_y, r_z]$ with an angle θ and the length of the magnitude of the rotation as r then rotation matrix R is defined as:

$$R = \cos(\theta)\cdot I + (1-\cos(\theta))\cdot rr^{\mathrm{T}} + \sin(\theta)\cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ r_y & r_x & 0 \end{bmatrix}$$

E. Correlation with different Kernel

As depending on the previous results, we want to find more accurate and better algorithms to get more stability. We come up with a solution to apply correlation with different kernel sizes from the resulted erosion images. Erosion is used to get the local minimum over the predefined kernel size. As the kernel is scanned over the image, we compute the minimal pixel value overlapped and replace the image pixel under the anchor point with that minimal value. The resulted image is applied with correlation by using $fiter2D$ function from OpenCV. Correlation is used to find the best and similar matches between two images, this has been implemented for finding artificial landmarks as stated in [14]. This function will get the best matches from the input image and defined kernel. This resulted images is checked with different threshold values from 0 to 1. By this algorithm we have developed and observed the results as in fig.6

IV. Results and Discussions

For the evaluation of our results, we have use for collected dataset from the warehouse and saved them to test for the future references in the previous chapter. We will implement our developed algorithms on this dataset and also for the static images. The results are explained in different scenarios based with respect to the position of the sticker, Obstacles and number of sticker. They are stated as:

- Location of the image is on top, middle and bottom
- Images with lot of obstacles in the environment
- Images with more than one image
- Video stream from the collected dataset

By considering the above three cases, we will first test our results on static images and then to the collected dataset. The kernel size and axis of ellipses can be explained as: If the location of the image is on top, the horizontal and vertical axis for the ellipse are considering to be low between 20-30 range with low kernel size in 3-5. As the image is located on the top and looks small considering to the position of camera we will need small kernel size which can give a better output for 0.7 threshold as shown in fig.7. For the image which is located close to the camera or on the bottom of the image, we will need higher ellipse axis between 40-45 and also higher kernel size between 7-11 with 0.8 threshold as shown in fig.7. If we increase the kernel for the same image then the resultant threshold has to be decreased to detect nine blobs of the DataMatrix stickers. In fig.8 we have tested and observed our output on images with obstacles and another half sticker. In this cases with more disturbances from the environment, the kernel size is set to be low of 3 with 0.7 threshold. As from our collected dataset, we will test with respect to real environment with lot of obstacles and many stickers in one frame as shown in fig.9

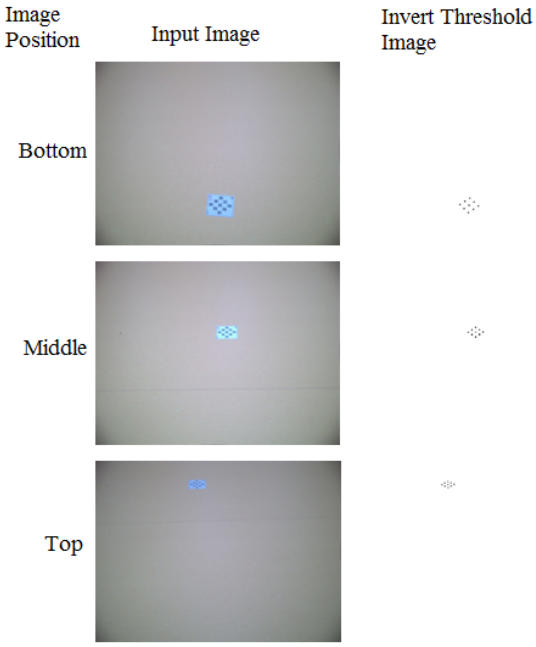


Fig. 7. Results for the Static Images

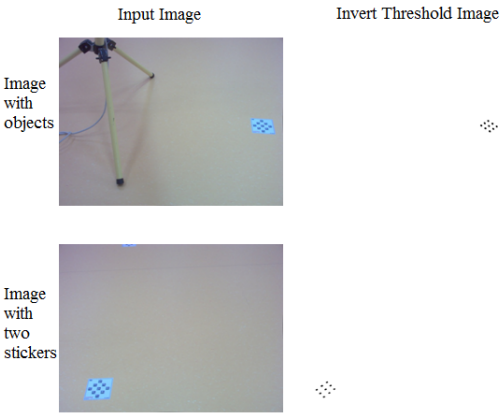


Fig. 8. Results for the Static Images with other Objects

V. CONCLUSION

This paper explains about the various methods for the localization of human in the robotized warehouse. we have explained to localize the human inside the robotized warehouse by using Basler camera, which is assembled behind the safety vest of a human. This work makes to explore and implement different algorithms and test them.

VI. ACKNOWLEDGMENTS

The work was carried at Czech Institute of Informatics, Robotics and Cybernetics (CIIRC) and developed within the SafeLog project funded by the European Unions Horizon 2020 research and innovation programme under grant agreement No 688117.

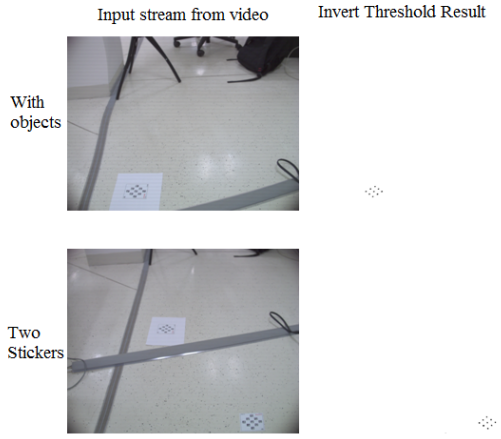


Fig. 9. Results from dataset with other Objects

REFERENCES

[1] Claus Brenner, Vehicle Localization using Landmarks obtained by a Lidar Mobile Mapping System, *IAPRS*, 2010
[2] Xia Yuan, Chun-Xia Zhao and Zhen- Min Tang, Lidar Scan-Matching for Mobile Robot Localization, *Information Technology Journal,* 9 (1), Pages 27-33, 2010
[3] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J. Kelly and Andrew J. Davison, SLAM++: Simultaneous Localisation and Mapping at the Level of Objects, *IEEE Conference on Computer Vision and Pattern Recognition,* 2013
[4] https://arxiv.org/ftp/arxiv/papers/1610/1610.03660.pdf
[5] Xu Wenli and Zhang Lihua. Pose estimation problem in computer vision. *IEEE Conference on Computer, Communication, Control and Power Engineering.* 1993
[6] D.G.Lowe. Distinctive image features from scale-invariant key points. *International Journal of Computer Vision,* 60(2), Pages 91110, 2004
[7] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *In European Conference on Computer Vision,*2006
[8] Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary Bradski. ORB: an efficient alternative to SIFT or SURF, *IEEE International Conference on Computer Vision (ICCV),* 2011
[9] J. MacQueen. Some methods for classification and analysis of multivariate observations. *In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability,* volume 1, pages 281297, 1967
[10] E. Jauregi, E. Lazkano and B. Sierra, Object recognition using region detection and feature extraction. http://www.sc.ehu.es/ccwrobot/papers/jauregi09object.pdf
[11] https://github.com/dmtx/libdmtx
[12] Chris Harris and Mike Stephens. A Combined Corner and Edge Detector. *In Proc. of Alvey Vision Conference* 1988
[13] Line Detection by Hough transformation
[14] M. Mata, J. M. Armingol, A. de la Escalera and M. A. Salichs, A Visual Landmark Recognition System for Topological Navigation of Mobile Robots. *IEEE International Conference on Robotics and Automation, ICRA*. 2001

Deformable organ registration using tactile gestures for minimally invasive surgery guidance

Yamid Espinel, Erol Ozgur, Lilian Calvet, Adrien Bartoli
TGI - EnCoV, Institut Pascal
Clermont-Ferrand, France
yamid@msn.com, erolozgur@gmail.com, lilian.calvet@gmail.com, adrien.bartoli@gmail.com

***Abstract*— Augmented reality is nowadays a very useful technology in laparoscopic surgery thanks to the possibility it gives to the surgeons of making efficient incisions, thus causing less damage to the organ. In this work we present a novel method to perform deformable registration of a preoperative 3D model (usually reconstructed from MR or CT volume), so that it fits into the corresponding organ's shape as shown in the camera image. Like this, it's possible to accurately locate the internal structures of an organ, such as tumours and veins, giving a clear idea to the surgeon of where to cut and how to extract these tumours. To help themselves locate them, surgeons make use of ultrasound imaging in the operating room. With this in mind, we also propose an initial strategy to register an ultrasound image in a semi-automatic way, according to how the probe appears in the laparoscopic image.**

I. INTRODUCTION

Minimally invasive surgery is today in demand thanks to its advantages such as reduced post-operative pain, hospitalisation time and parietal scars. This approach introduces fine surgical instruments and a laparoscope via small incisions, inside the human body and the laparoscope allows the surgeons to view the area which they are operating on.

One of the difficulties surgeons have to face is the subsurface anatomy localization. This is because the surgeon perceives the scene from 2D laparoscopy video showing an opaque organ, which may lead the surgeon to incise in places that could damage delicate anatomies. Most of the time surgeons make use of ultrasound imaging to accurately locate the tumour from the surface, helping them to decide where to perform the incisions that will lead to its complete extraction.

When the size of the tumours are small enough to not appear clearly in the ultrasound image, there is an increased risk of performing incisions that may cause a considerable damage to the organ. In this cases, the resections can be made by taking a big part of the liver in order to increase the probability of doing an effective extraction of the tumour. However, there is no guarantee that the tumour is indeed in the resected part of the organ. To address this issue, augmented reality can be applied on the live stream coming from the laparoscope. It works by augmenting information of the organ's inner anatomy onto the laparoscopy video.

In order to show such 3D figures on top of the laparoscope's 2D images, a 3D model of the organ is built from the segmented preoperative scan data such as MRI or CT. The time duration between acquisition of scan data and the surgery is usually a few months. During the surgery, the 3D model has to be registered to the organ, in such a way that the surgeon can locate the inner anatomy and consequently incise more efficiently.

In this work we present a new method to perform 3D-2D registration of human organs by means of an application that receives tactile gestures to perform deformable transformations. Model registration through tactile gestures brings comfort and reduces the process time, as this has to be done in the operating room quickly. Shape matching between this preoperative model and the organ as shown in the laparoscopic image gives the surgeons an accurate view of the internal structures, letting them to perform more accurate and efficient incisions (see Figure 1).

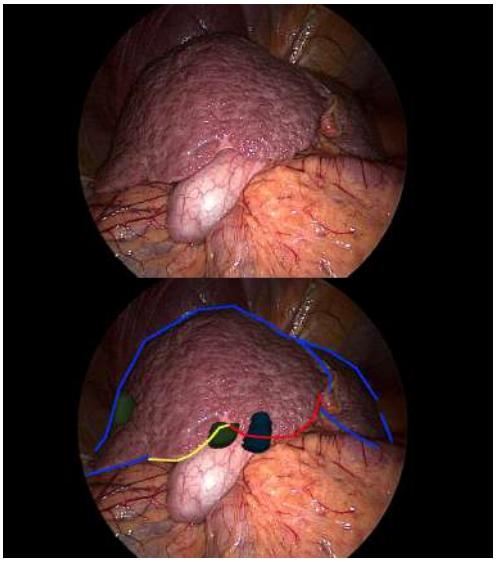


Fig. 1: Augmented reality in liver

We also make a proposal to perform initial registration of an ultrasound plane, in which the user tells the program the location of the probe in the input image by means of a set of markers; then, the program will automatically locate the plane under the tool, leaving only two parameters to correct: the depth of the plane and the roll angle. This can be used as a groundtruth to validate the accuracy of registration done previously, as the preoperative internal structures should now

match the locations of the real ones.

II. BACKGROUND

A. Augmented reality in laparoscopic surgery

In the past decades, minimally invasive surgery has became more popular than open surgery thanks to its greater clinical benefits. However, this kind of intervention introduces a loss of direct vision and tactile feedback of the scene. Augmented Reality (AR) has been then trying to alleviate this drawback by providing an intraoperative guidance with identification of subsurface targets (like tumours and infections), along with critical structures (vessels, nerves, etc.)

Thanks to this guidance, there is a reduced risk of blood loss because of accidental vessel perforations, as well as a shorten intervention time due to a quick localization of the target structures. It's also possible to make more accurate resections, which lead to a complete removal of the cancerous cells, while leaving more healthy tissue intact and thus causing less damage to the organ.

The main field where AR has been used is in neurosurgery. One of the first interventions assisted by AR is described in [1]. Several interventions of this type followed, including [2] and [3]. One of the advantages here is the presence of skull, which is used as a rigid common reference between acquisitions. While the brain is not rigid, its deformation is limited and thus assumed to be rigid for AR purposes.

There are also other surgical fields where usage of AR has been attempted, such as maxilofacial surgery [4] and orthopedics [5]. As for brain surgery, the rapid development in these fields is due to the rigidity of the structures caused by their proximity to bones, limiting deformations between the preoperative acquisition and the intraoperative stage.

In contrast, AR has not been so much applied to digestive surgery, mainly because there is no constant spatial relationship between the organs and rigid structures, letting deformations to occur in a significant way. One of the first attempts reports the visualization of ultrasound images using a head-mounted display (HMD) for an abdominal surgery [6]. Then, a general surgery assisted by AR was an adrenalectomy, in which several organs (like liver, kidneys and lungs) were augmented and identified [7]. Other applications have been made to do augmentation in kidney [8], prostate [9], and in treatment of renal vessels [10].

B. The Hepataug software and registration of the liver

The EnCoV team at the Institut Pascal has developed *Hepataug* [11] [12], an application capable of doing rigid and deformable registration of a liver's preoperative data into a laparoscopic image (see Figure 2). It has tools to load the preoperative data, change their color and opacity, and modify their pose in 3D (translation and rotation in 6DoF) with the help of mouse and keyboard. It can also perform deformation of the data by the use of several types of contours, which constrain the deformation spatially.

This application is developed using C++ and the Qt libraries. OpenGL is used to render the 3D models, while the deformation algorithms are made under CUDA, so that these

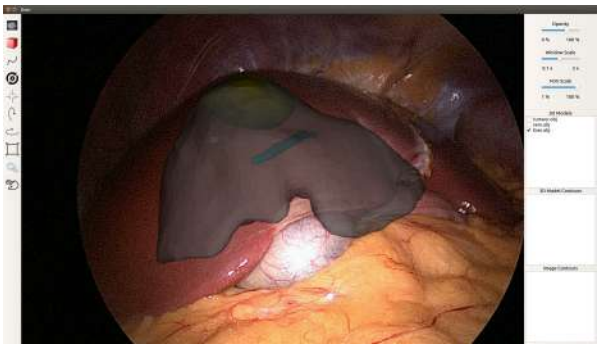


Fig. 2: Hepataug software

calculations are done directly on the GPU. Currently, the application runs under Ubuntu but there are plans to make it portable, so that it can be run even on Android-based devices.

The rendering of the 3D model is done by projecting the model points from the model frame to the camera frame, as seen in Figure 3. To project these points we need the camera matrix, which in OpenGL is expressed as:

$$\mathbf{K_{openGL}} = \begin{bmatrix} \alpha & s & -u_0 & 0 \\ 0 & \beta & -v_0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where α and β are the camera's focal length in pixels, s is the skewing factor, u_0 and v_0 are the principal point coordinates of the image in pixels, and A and B are two coefficients containing the minimal and maximal distances in meters where the model should be displayed, being $A = n + f$ and $B = n * f$. In OpenGL, and n, f are the "near" and "far" planes that mark the boundaries of the scene that will be rendered, as shown in Figure 4. In our software $n = 0.001$ and $f = 1000$.

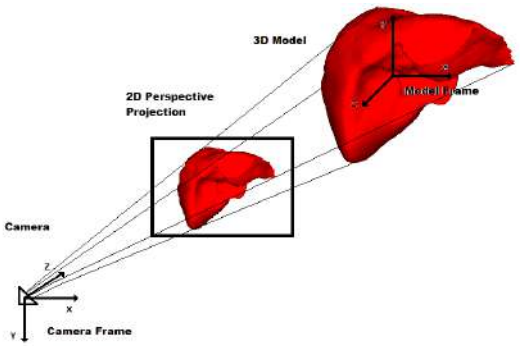


Fig. 3: 3D scene modelling in Hepataug

Once we have this camera matrix, we compute the projection by multiplying K with the transformation matrix M_c^m between the model frame and the camera frame. M_c^m is computed by the program automatically according to how the user interacts with it to change the position and orientation

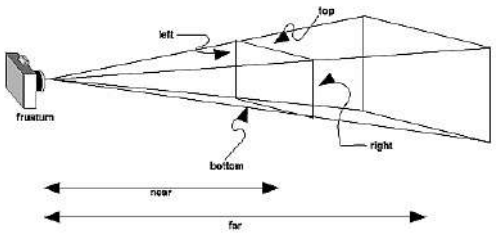


Fig. 4: Perspective viewing volume in OpenGL

of the model.

There are several ways to register the 3D preoperative organ into the laparoscopic image. In general, there are 2 types of registrations that can be made: rigid and deformable. Both of them can be made using mouse and keyboard and/or with tactile gestures. These registrations are considered to be valid when the shape of the model has fitted the real organ in the image, and when the all the inner structures are approximately at the expected locations.

In the case of rigid registration, it is possible to translate and rotate the model in the x, y and z directions. When using mouse and keyboard, the translation along x and y is done using the direction keys, in z using the wheel's button of the mouse, while rotation in every axis is controlled by clicking on the model and dragging towards the desired direction. In the Figure 5 an example of translation and rotation is illustrated.

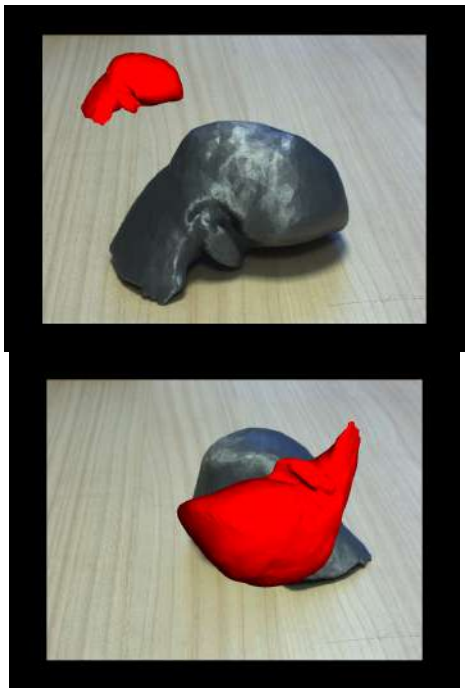


Fig. 5: 3D liver translation and rotation using Hepataug

For a deformable registration, the standard approach right now makes use of contours that constrain the deformation process. These contours must be marked both in the laparoscopic image and the preoperative model, at the corresponding positions. There exist 3 types of contours: *Ridge* contours, *silhouette*, *ligament*. The *ridge* contours are the most restrictive of all, having a direct vertex-to-pixel correspondence and making those vertices to stay at the same positions as their corresponding pixels in the 2D image; they are useful for stable regions of the liver such as the area on top of the gallbladder, where there is little deformation. The *silhouette* contours, contrary to the *ridge* contours, don't have any correspondence to vertices in the 3D model: they just don't let those vertices to cross that boundary, letting the model to slip around the boundary marked by the contour without any kind of linking. The *ligament* contours take all the corresponding ligament vertices in the model as a single region that will be aligned along the contour axis; being specially useful to align the falciform ligament in the model so that it matches the location of the ligament in the 2D image. An example of liver deformation using contours is illustrated in Figure 6, where we make use of all the three: *ridge* in red, *silhouette* in yellow, and *ligament* in blue.

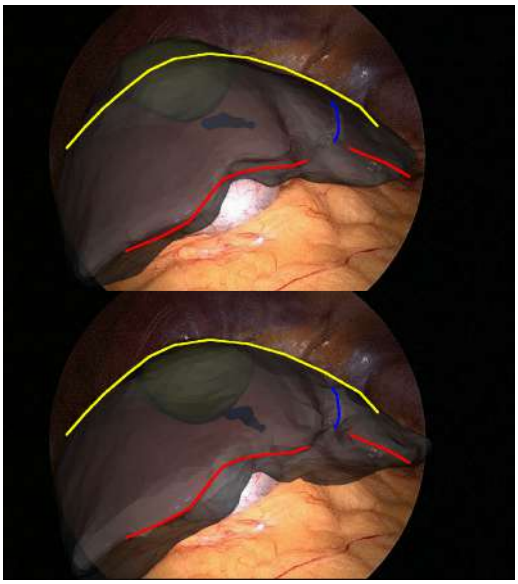


Fig. 6: Liver deformable registration using contours

III. Deformable registration of the liver using tactile gestures

As explained before, one of the main objectives of this project is to achieve fast model deformation by means of tactile gestures. For our case, we pretend to select several vertices or regions of vertices so that when one of them is selected and translated, the model will adapt its shape to these new locations by also taking into account the position of the fixed vertices, which in the end will make the liver to stretch/shrink. This deformation process will always occur by respecting the biomechanical behavior of a real liver (thanks to the position-based simulation algorithm and the Neo-Hookean model), as explained in [14].

A. Marking of a vertex or region on the model

Before starting the deformation process, the user can mark either a single vertex or a set of them. To be able to select/mark, the Edition mode has to be activated by pressing the "E" key. Then, to select a single vertex the user will click or press in the screen position that corresponds to the desired vertex, as shown in the Figure 7.

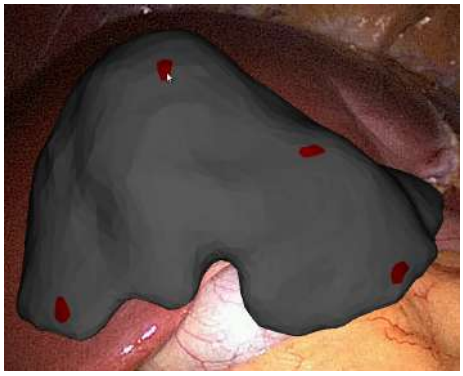


Fig. 7: Marking vertex on the model's surface

In order to find the closest vertex to the selected screen position, we sweep all the 3D vertices in the model and project them onto 2D. To achieve this, the 3D point in the model has to be projected in the camera frame by doing:

$$\mathbf{v_c} = \mathbf{M}_c^m \mathbf{v}_m \tag{1}$$

where v_c is the projected vertex in the camera image, M_c^m is the transformation matrix between the model and camera frames, and v_m is the vertex expressed in the model frame. It must be noted that these locations are expressed in homogeneous coordinates, such that:

$$\mathbf{v_c} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{M}_c^m \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

where u, v are the image pixel locations of the projected vertex and x, y, z are the vertex 3D coordinates in the model frame.

In OpenGL, the transformation matrix M_c^m that projects the 3D points into 2D image coordinates (also called projection matrix) is expressed as:

$$\mathbf{M_c^m} = \mathbf{M}_{pers}\mathbf{M}_{ortho} \tag{3}$$

where M_{pers} is a perspective projection matrix and M_{ortho} is an orthographic projection matrix.

The obtained coordinates for $\mathbf{v_c}$ are then homogenized and, because these values are in the range of [-1,1] (being expressed in normalized device coordinates, as explained in [13]), they can be normalized again in such a way that they are in the range [0,1] by doing:

$$\mathbf{v_c'} = \mathbf{v}_c \times 0.5 + 0.5 \tag{4}$$

Now these coordinates have to be mapped to the corresponding locations in the image frame, which can be done by multiplying them by the size of the viewport:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} u * v_w + v_{ix} \\ v * v_h + v_{iy} \end{bmatrix} \tag{5}$$

where v_w, v_h are the viewport's width and height, and v_{ix}, v_{iy} are the origin coordinates of the viewport, which for our case are (0,0).

As we want to find the closest vertex to the chosen point in the screen, we have to take into account only those who are visible to the camera. This can be achieved by reading the depths of the drawn pixels the screen, with the help of the OpenGL's `glReadPixels()` function. The values returned by this function are stored in an array of size (v_w * v_h) and we can quickly find the corresponding depth index of a rendered vertex using its 2D projection like:

$$i_d = u' + v' v_w \tag{6}$$

By comparing the position in z of every 3D vertex and the depth of its projected 2D pixel, we can decide whether if it's visible by the camera or not. If the difference between z and the pixel's depth is greater than certain value (normally a very small value, like 1×10^{-5}), then it's an indication that the vertex is covered by another one closer to the camera.

If this visibility condition is fulfilled, the coordinates of the projected 2D pixel are compared with the chosen screen location by means of an Euclidean distance. If the distance is smaller than a previously computed one, then it will be kept in memory and the index of this vertex will also be stored. The index of the closest vertex will then be returned after sweeping all of them, so this vertex will be set as marked. For this selected vertex to be able to move independently from the others, we add it to a vector of selected regions.

Regarding the case of a region marking, the user should start by pressing outside the model and then dragging towards the area of interest. A marking rectangle will appear, changing its size according to how the user moves the finger in the screen. When the finger is retired, the vertices inside the rectangle will be set as marked, as illustrated in the Figure

8.

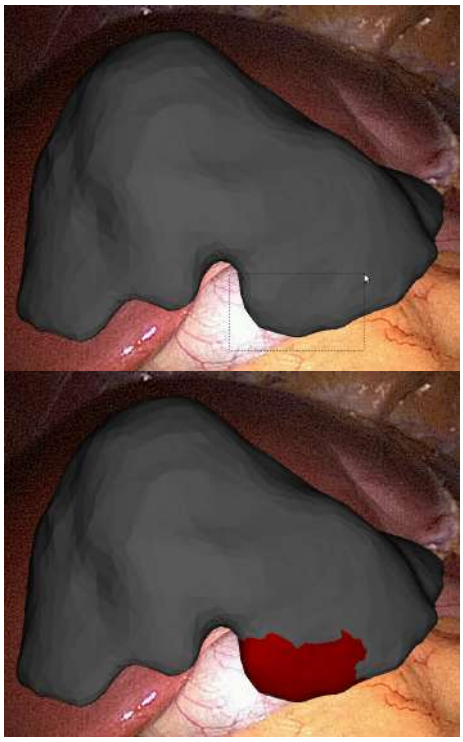


Fig. 8: Marking a set of vertices in the model's surface

A vertex is outside the rectangle if its projected 2D pixel meets any the following conditions:

$$u' < r_x - \frac{r_w}{2}\ ,\ u' > r_x + \frac{r_w}{2}\ ,\ v' < r_y - \frac{r_h}{2}\ ,\ v' < r_y + \frac{r_h}{2} \quad (7)$$

where r_x, r_y are the center coordinates of the rectangle and r_w, r_h are the rectangle's width and height. All the vertices inside the rectangle will also be stored in the previously mentioned `region_vector`.

B. Marking of a contour on the model

Another way to add a constraint to the deformation is to select regions that are known to remain stable. In the case of the liver, as it is standing on top of the gallbladder, the lower region is relatively stable. Thus, by making a careful correspondence of the region of interest in the preoperative model and the laparoscopic image, we can select this part in the model as a contour and make the liver move around it. This has some advantages over a normal region, including that it is a more refined and precise constraint and will give a more realistic behavior to the deformation.

Before marking the contours, the model should be positioned in such a way that the area of interest is at a bordering zone in the view. This means that right after it, the laparoscopic image should be visible instead any other part of the model. After this, the user should press the "Start Simulation" button to copy all the models into video memory and initialize the CUDA environment.

User can start marking contours by pressing the "E" key to switch to edition mode, and then the "C" key to switch to contour selection mode, making a yellow contour to appear around the model (see Figure 9). Then, the procedure to select these contours is similar to the one done for the normal vertex regions, as shown in Figure 10.

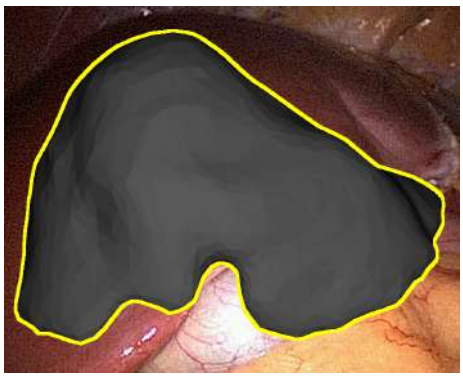


Fig. 9: Contour mode activation

The procedure to extract the contour vertices starts by segmenting the 3D model from the background image, obtaining a 2D image of the scene where the model is represented by 1's and the background by 0's. This is done with the help of the `glReadPixels()` function, giving the depth of every drawn pixel and in which the image pixels have a depth of 1. Then we map the segmentation to an interval of [0,255] and apply a Canny edge detector to extract the 2D contours, such that these contours are represented by pixels with values of 255. Now we check which of these contour pixels are inside the marking rectangle (as explained before), and then find the closest model vertices to this contour by computing the euclidean distance of the 2D projected pixels of the model and the contour pixels.

These contour vertices will also be added to the list of marked regions in the `region_vector` array, so that it can remain fixed while the others are moving.

C. Selecting a marked region/contour for manipulation

When the user presses the screen in a position where there is a visible vertex, the program tries to find the closest visible vertex to that 2D position as done in the section 4.1. The model vertices are projected into the image frame and those

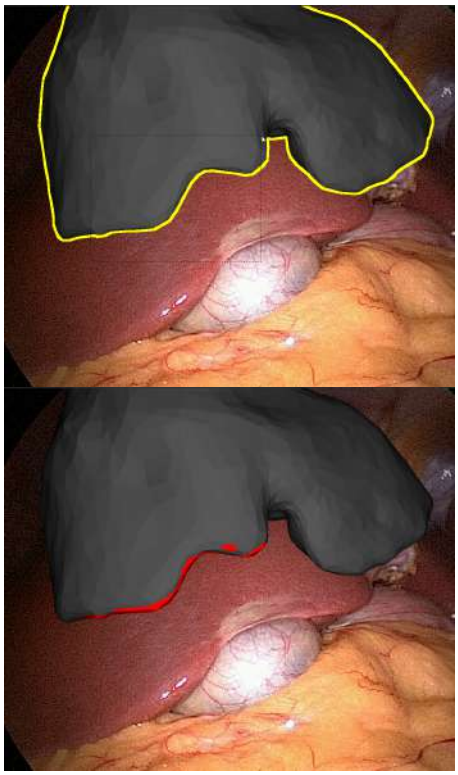


Fig. 10: Marking a contour region on the model's surface

positions are compared to the chosen point in the screen by means of a euclidean distance. The difference in this case is that, as the vertex has been already selected, the program will find the region to which it belongs thanks to the `region_vector` array. Once it is found, all the vertices of that region will be marked as being moved by setting the `moving_vertex` property to 1 (or 2 if there is a second finger in the scene).

If the finger is retired from the screen, all the vertices that were marked with `moving_vertex = 1` or `moving_vertex = 2` (according to the finger that was handling them) will be set back to 0, indicating that they are fixed regions again.

D. Manual real-time deformation by local translation and rotation

According to how the organ has deformed in certain region of interest (if there was a twist deformation rather than a shrink), then it would be better to perform a rotation of vertices rather than a displacement. Currently we are handling these two transformations with a single finger, which makes it necessary to define specific gestures that activate one transformation or the other.

We have decided to apply the same strategy as for the translation and rotation gestures in the rigid registration case. If the finger stays fixed for more than 500 milliseconds then it will switch into rotation mode, otherwise the user will directly start moving the vertices of the selected region.

The translation of the vertices goes according to the movement of the finger. First, the rotation of the model has to be taken into account so that the vertices move following that direction. As in our software this rotation is expressed as a quaternion, we will first convert it to a rotation matrix:

$$\mathbf{R} = \begin{bmatrix} 1-2q_y^2-2q_z^2 & 2q_xq_y-2q_zq_w & 2q_xq_z+2q_yq_w & 0 \\ 2q_xq_y+2q_zq_w & 1-2q_x^2-2q_z^2 & 2q_yq_z-2q_xq_w & 0 \\ 2q_xq_z-2q_yq_w & 2q_yq_z+2q_xq_w & 1-2q_x^2-2q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where q_w, q_x, q_y, q_z are the components of the quaternion that indicates the model's orientation. Then, the displacement vector is obtained by multiplying the movement of the finger in the screen with the model's rotation like:

$$\mathbf{v} = \mathbf{R}^T\mathbf{d} \tag{8}$$

where $\mathbf{v}$ is the displacement vector to be followed by the vertices and $\mathbf{d}$ is the displacement of the finger in the screen, which is the difference between the current and previous positions of that finger in x and y and is defined like:

$$\mathbf{d} = \begin{bmatrix} t_x - t_{x\_prev} \\ -(t_y - t_{y\_prev}) \\ 0 \end{bmatrix} \tag{9}$$

The vertices are then affected by this vector $\mathbf{v}$ by adding it to their current positions $\mathbf{p_v}$:

$$\mathbf{p_v} = \mathbf{p}_v + \mathbf{v} \tag{10}$$

For every new displacement of the finger the shape of the model is deformed to fit the new vertices positions, using the position-based simulation approach explained in the section 3.4. This deformation algorithm will modify the positions of all the vertices in the model while the selected region is translated, giving the impression of being pulling a model that is fixed with needles at some locations. In the Figure 11 three regions have been selected and the one in the right starts to be translated in a bottom-right direction, making the model to stretch also in that way.

As explained before, if at certain region the liver has been twisted, the user can perform local rotation of a region of vertices to fit this deformation. For the vertices to rotate according to the movements of the finger, first the centroid point c of the region is found by adding all the positions

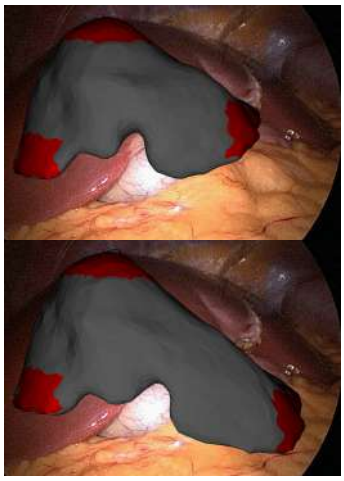


Fig. 11: Model deformation by translating one selected region

of its vertices and dividing by the total number of vertices in that region. This centroid will let the vertices to rotate around their local region center.

Then, we try to find a vector which is perpendicular to the model's rotation vector in order to obtain its axis of rotation. For this, we first define an unit vector $\mathbf{v_u}$ oriented towards the $-\mathbf{z}$ axis and then we multiply it by the model's rotation matrix to obtain a vector $\mathbf{v_p}$ perpendicular to the displacement vector $\mathbf{v}$:

$$\mathbf{v_u} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix} \tag{11}$$

$$\mathbf{v_p} = \mathbf{R}\mathbf{v}_u \tag{12}$$

Now, we compute the model's axis of rotation $\mathbf{R_a}$:

$$\mathbf{R_a} = \mathbf{v} \times \mathbf{v}_p \tag{13}$$

We can now apply this rotation to every vertex in the region. To do this, we first substract the region centroid from the actual vertex position, then multiply by the rotation matrix, and then add the centroid back to this transformation:

$$\mathbf{p_v} = \mathbf{p}_v - \mathbf{c} \tag{14}$$

$$\mathbf{p_v} = \mathbf{R_a}\mathbf{p}_v \tag{15}$$

$$\mathbf{p_v} = \mathbf{p}_v + \mathbf{c} \tag{16}$$

As done when translating a region of vertices, for every new rotation the shape of the model is deformed to fit the new vertices positions, using the position-based simulation approach. In this way, the model in that region will fit better to that twist made by the real organ. In the Figure 12 three regions have been selected and the one in the right starts to be rotated towards the back in a counter-clockwise direction, making the model to also twist in that particular area.

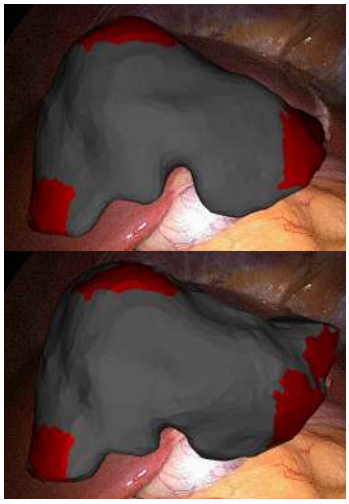


Fig. 12: Model deformation by rotating one selected region

IV. Ultrasound image registration using tactile gestures

Ultrasound (US) imaging has been an important resource in any session of laparoscopic surgery since its begginings. It provides a fast and accurate way to confirm the positions of the internal structures of an organ so that the surgeon can decide on the best way to perform an incision. However, not all clinical services are equipped with a laparoscopic US system, and not all surgeons are properly trained to use it peroperatively. Among the reasons for its difficult usage we find the separate monitor that shows the US images, the mental registration the surgeon has to do between the US plane into the laparoscopic image, and also the difficulty that represents an US image. We propose here to use it as a groundtruth to validate the registrations by comparing the locations and shapes of a tumour in both the preoperative data and the ultrasound image.

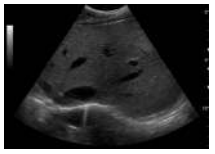
In order to reach this objective, two main steps must be achieved: Ultrasound probe calibration and ultrasound probe estimation. In the latter step, two procedures are generally executed: Pose initialization and pose refinement. In this section, we propose a solution to this pose initialization

problem involving manual registration using tactile gestures. From now on, we assume an image pair composed of an ultrasound image and a laparoscopic image on which the US probe is partially visible, both acquired at the same time.

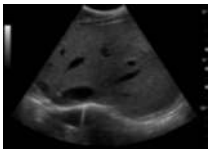
A. Loading and pre-processing of the ultrasound image

When an ultrasound image corresponding to its corresponding laparoscopic view is loaded into the program it is first converted into grayscale. Then, a gaussian blur is applied to it in order to ease the segmentation process, and thus we threshold the image by removing the black components, which is mostly the background of the ultrasound plane. As there might be some black holes inside the sonographic zone, we apply a dilation operation in the thresholded image to remove them.

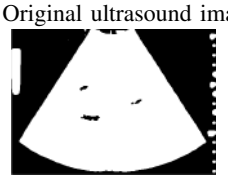
Because our area of interest is what is inside the sonographic zone, we would like to extract this information from the general image. Like so, we take the thresholded image and perform a connected-component analysis from which we will extract the highest component, corresponding in this case to the sonographic area. With this component we create a black-and-white mask and we apply it to the original image, obtaining in the end only the ultrasound data with a transparent background. The whole process of extracting this sonographic zone is illustrated in the Figure 13.



(a) Original ultrasound image

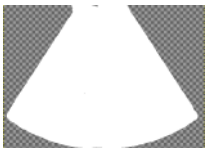


(b) Image after gaussian blur

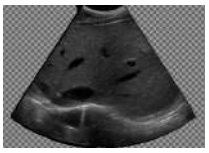


(c) Thresholding with a value of 10

(d) Dilation to remove inner holes



(e) Mask from extraction of biggest component



(f) Filtering of the original image using the mask

Fig. 13: Ultrasound image preprocessing to obtain sonographic region

B. Pose initialization of the ultrasound plane up to depth and roll angle

Our goal here is to perform a semi-automatic registration by indicating the position and orientation of the US probe in the laparoscopic image, along with the position of the probe's head. The pose of the probe can be indicated by first marking 4 points, in a way where each pair makes a line that is located in one the probe's borders (see Figure 14). From these two lines we find their intersection point, which will be then backprojected to find the point at infinity corresponding to the orientation vector that will serve as the axis in which the ultrasound plane will move. We first compute the two lines l_1 and l_2 (eqs. 5.1 and 5.2) by computing the crossproducts between the pairs of points p_1, p_2 and p_3, p_4:

$$\mathbf{l_1} = p_1 \times p_2 \tag{17}$$

$$\mathbf{l_2} = p_3 \times p_4 \tag{18}$$

$$p_i = \mathbf{l}_1 \times \mathbf{l}_2 \tag{19}$$

$$\mathbf{v_u} = K^{-1} \times p_i \tag{20}$$

$$\bar{\mathbf{v}}_\mathbf{u} = \frac{\mathbf{v_u}}{\|\mathbf{v_u}\|} \tag{21}$$

where p_1, p_2, p_3, p_4 are the 2D marks used to indicate the probe's orientation in the laparoscopic image, l_1 and l_2 are the lines obtained from these 2 pairs of points, p_i is the intersection point between l_1 and l_2, v_u is the projected intersection point in 3D space (point at infinity) while $\bar{v}_u$ is the unit vector that represents the probe's axis where the ultrasound plane will move along.

Next step is to mark the probe's head location in the laparoscopic image (point p_5), where this point should go along the probe's axis $\bar{v}_u$. Once this location is indicated, the ultrasound image will be automatically positioned under the probe's head with the corresponding orientation (see Figure 15). The rotation process of the plane is done as follows:

$$\mathbf{p_{fx}} = \bar{\mathbf{v}}_\mathbf{u} \tag{22}$$

$$\mathbf{p_{tz}} = \mathbf{p_{fx}} \times \mathbf{p_{iy}} \tag{23}$$

$$\mathbf{p_{fy}} = \mathbf{p_{tz}} \times \mathbf{p_{fx}} \tag{24}$$

$$\mathbf{p_{fz}} = \mathbf{p_{fx}} \times \mathbf{p_{fy}} \tag{25}$$

where p_{fx}, p_{fy}, p_{fz} are the final rotation vectors, p_{iy} is the initial rotation vector in y, and p_{tz} is an auxiliar rotation

vector in z. From these rotation vectors we build our rotation matrix like:

$$\mathsf{R} = \begin{bmatrix} p_{fxx} & p_{fyx} & p_{fzx} \\ p_{fxy} & p_{fyy} & p_{fzy} \\ p_{fxz} & p_{fyz} & p_{fzz} \end{bmatrix}$$

This rotation matrix R will be used to rotate the ultrasound plane, so that in the end it will have roughly the same orientation as the probe.

As the head of the probe normally should go along the probe's axis, we would like to be sure that the selected position fulfills such constraint. We have then implemented a strategy to make the selected point to be automatically aligned along $\bar{v}_u$, which basically finds the nearest point that is along the probe's axis. Like this, we can avoid incorrect registrations that could make the ultrasound plane not to appear under the probe. To achieve this we do:

$$p_m = \frac{p_1 + p_3}{2} \tag{26}$$

$$a = p_5 - p_m \tag{27}$$

$$b = p_i - p_m \tag{28}$$

$$\bar{b} = \frac{b}{\|b\|} \tag{29}$$

$$d = a^T \bar{b} \tag{30}$$

$$\mathbf{p'_5} = p_m + d\bar{b} \tag{31}$$

where p_m is the middle point between the points p_1 and p_3 (starting or ending points of both lines l_1 and l_2), a is the vector going from p_m to p_5, b is the vector going from p_m to p_i, and d is the scalar projection of a into b, while p'_5 is the corrected head point.

Finally we backproject p_5 and align the ultrasound plane according to the indicated probe orientation and head location. The final registration process is done in the following way:

$$p_h = K^{-1} p'_5 \tag{32}$$

$$\bar{p}_h = \frac{p_h}{\|p_h\|} \tag{33}$$

$$p = \bar{\mathbf{p}}_\mathbf{h} \cdot \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T \tag{34}$$

$$p_{uh} = \bar{p}_h \frac{u_z}{p} \tag{35}$$

where p_h is the probe's head in 3D space, p is the dot product of p_h and a vector oriented to the $-z$ axis, while p_{uh} contains the coordinates in 3D space that will locate the ultrasound plane under the probe's head.

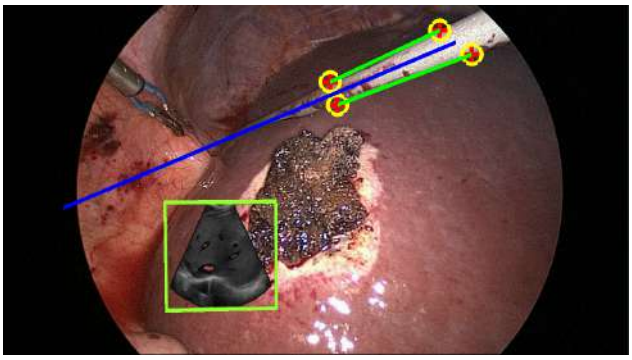


Fig. 14: Marking of ultrasound probe's pose

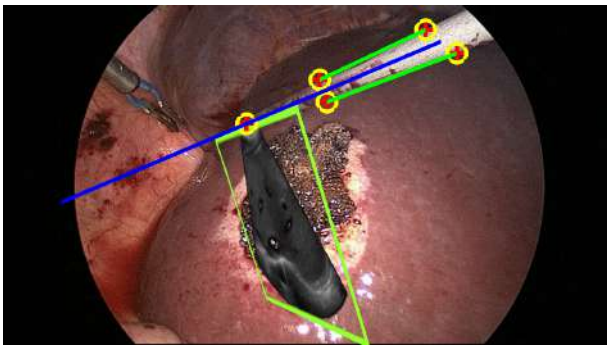


Fig. 15: Ultrasound plane aligned under the probe's head

C. Registration of the depth and roll angle

Once the ultrasound plane is located under the head of the probe and oriented in the same way, the movements of the plane using tactile gestures are restricted to only 2 degrees of freedom (DoF). These degrees include the rotation of the plane around the axis of the probe, as well as its distance along the ray through the optical center and p_5 (namely depth). This restriction will help the user not to lose the previously done registration while refining the remaining degrees of freedom.

To rotate the plane around the probe's axis the user can press on the ultrasound image and start dragging left or right. This rotation process that follows the movements of the fingers in the screen is done by rotating the trackball like:

$$d_p = p_p - p \tag{36}$$

$$d = \|d_p\| \tag{37}$$

$$\phi = \frac{d}{r} \tag{38}$$

where d_v is the difference between the points representing

the previous finger position p_p and the current one p, d is the length of the vector d, and ϕ is the angular distance between d and the radius of the ultrasound plane, which is of approx. 5cm. With both the angle ϕ and the probe's axis $\bar{v}_u$ we obtain the quaternion matrix that will make the plane rotate around the probe.

Regarding the depth of the plane, it is possible to adjust it by applying a pinching gesture (with two fingers). This will change its scale while fixing the plane to the probe's axis as well as the head. The procedure to scale the plane is done as follows:

Let us assume that the user presses two arbitrary points, $\mathbf{f}_1$ and $\mathbf{f}_2$ on the touch screen. He then moves his fingers to two new locations, $\mathbf{f}'_1$ and $\mathbf{f}'_2$. Refer to figure **??** to visualize the 3D scene. We first need to compute the backprojected points corresponding to these locations. This procedure results in four 3D points on the model, $\mathbf{F}_1$, $\mathbf{F}'_1$, $\mathbf{F}_2$ and $\mathbf{F}'_2$ respectively. Next, we compute the unit vectors u'_1 and u'_2 pointing towards f'_1 and f'_2:

$$\bar{\mathbf{u}}'_1 = \frac{f'_1}{\|f'_1\|} \tag{39}$$

$$\bar{\mathbf{u}}'_2 = \frac{f'_2}{\|f'_2\|} \tag{40}$$

Since the vectors pointing towards $\mathbf{F}''_1$ and $\mathbf{F}''_2$ are the same as the ones pointing towards $\mathbf{f}'_1$ and $\mathbf{f}'_2$, our new vectors $\bar{\mathbf{u}}''_1$ and $\bar{\mathbf{u}}''_2$ become:

$$\bar{\mathbf{u}}''_1 = \bar{\mathbf{u}}'_1 \quad , \quad \bar{\mathbf{u}}''_2 = \bar{\mathbf{u}}'_2 \tag{41}$$

Using the law of similar triangles, the following ratio holds:

$$\frac{D}{D'} = \frac{l''_1}{l'_1} = \frac{l''_2}{l'_2} \tag{42}$$

where $D = \|\mathbf{F}_2 - \mathbf{F}_1\|$ and $D' = \|\mathbf{F}'_2 - \mathbf{F}'_1\|$, l'_1 and l'_2 are lengths of position vectors $\mathbf{F}'_1$ and $\mathbf{F}'_2$ respectively and l''_1 and l''_2 are the lengths of the position vectors $\mathbf{F}''_1$ and $\mathbf{F}''_2$ which have to be computed.

$$l''_1 = l'_1 \frac{D}{D'} \quad , \quad l''_2 = l'_2 \frac{D}{D'} \tag{43}$$

We multiply the vectors calculated in equation (41) by the lengths calculated in 6.3 (a)(ii) to get the z-translated 3D points.

$$\mathbf{F}''_1 = l''_1 \mathbf{u}''_1 \quad , \quad \mathbf{F}''_2 = l''_2 \mathbf{u}''_2 \tag{44}$$

where $\mathbf{F}''_1$ and $\mathbf{F}''_2$ are new 3D points, $\mathbf{u}''_1$ and $\mathbf{u}''_2$ are vectors calculated in equation (41) and l''_1 and l''_2 are lengths calculated in equation (43)

To calculate the translation parallel to the Z-axis we take the average of the two translations required to bring the point $\mathbf{F}'_1$ to $\mathbf{F}''_1$ and $\mathbf{F}'_2$ to $\mathbf{F}''_2$.

$$\mathbf{t}_z = \frac{1}{2}\Big(\big(\mathbf{F}''_1 + \mathbf{F}''_2\big) - \big(\mathbf{F}'_1 + \mathbf{F}'_2\big) \Big) \tag{45}$$

where $\mathbf{t}_z$ is the required translation, $\mathbf{F}'_1$ and $\mathbf{F}'_2$ are the points that project onto the points $\mathbf{F}''_1$ and $\mathbf{F}''_2$ on the translated model surface and $\mathbf{F}''_1$ and $\mathbf{F}''_2$ are the 3D points on the translated model surface.

Because we want our plane to move along the probe's head axis and not on the Z-axis itself, we must compensate this as follows:

$$r = K^{-1} p_5 \tag{46}$$

$$\bar{r} = \frac{r}{\|r\|} \tag{47}$$

$$d = t_z^T \bar{r} \tag{48}$$

$$p'_h = d\bar{r} \tag{49}$$

where r is the 3D reprojection of the probe's head location in 2D, d is the distance between t_z and $\bar{r}$, and p'_h is the point that conditionates the plane position in 3D space so that it moves along the probe's head axis.

V. Tests and results

A. Deformable liver registration using tactile gestures

We have tested our method in 3 patients and measured the times we spent to perform deformable registrations using the classical contour-based registration approach and the tactile-based manual approach. We also compare the final shapes of the deformed organs and analyze the effectiveness of our approach.

In the first patient, we have used two ridge contours (red) and two silhouette contours (yellow) to make the preoperative model to fit into the image, as illustrated in Figure 16. We can appreciate the existence of a big tumour below the liver and next to the gallbladder. In total, this registration took 4 minutes 26 seconds to perform.

Afterwards, we performed a manual deformable registration with tactile gestures on the same scene using a total of 4 regions of marked vertices. The two regions in the left are used as fixed regions (as their positions match with the organ in the 2D image), while the two in the left were modified to lift up the right lobe of the liver. First, the upper-right region was translated up, then we did the same for the lower-right one. The time spent to do this registration was of 35 seconds.

In Figure 17 we can appreciate the resulting deformation for this case.

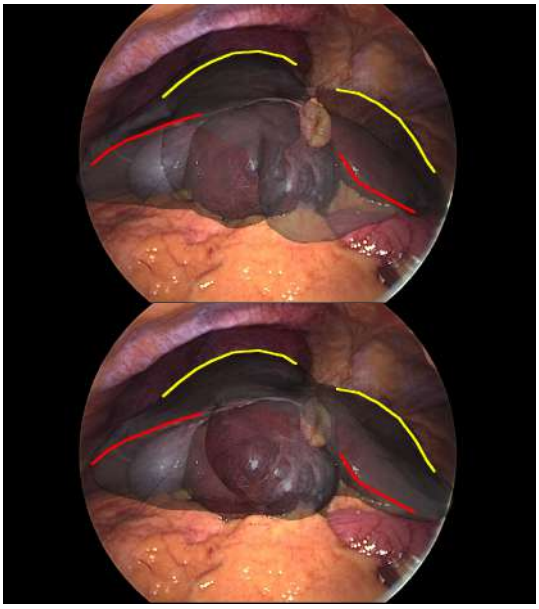


Fig. 16: Semi-automatic deformable registration of the liver using two ridge contours (red) and two silhouette contours (yellow)

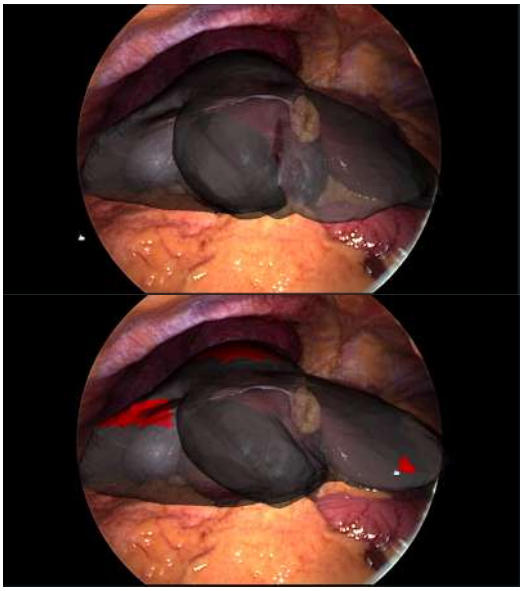


Fig. 17: Manual deformable registration of the liver using 4 regions of vertices

In the second patient, we performed registration on the liver using 3 ridge contours and 1 silhouette contour. In this case, the tumour was inside the organ and had a size of only a few cm. The time it took to do this registration was of 3 minutes 32 seconds. In Figure 20 we appreciate the effect of this deformation.

For the manual deformable case, we have marked a total of 6 regions of vertices. Among these regions, there is one contour region while the others are generic ones. The two regions in the left are used as fixed regions, and the three in the left were moved up independently. The time spent to do this registration was of 1 minute 53 seconds. In Figure 19 we can observe the resulting manual deformation.

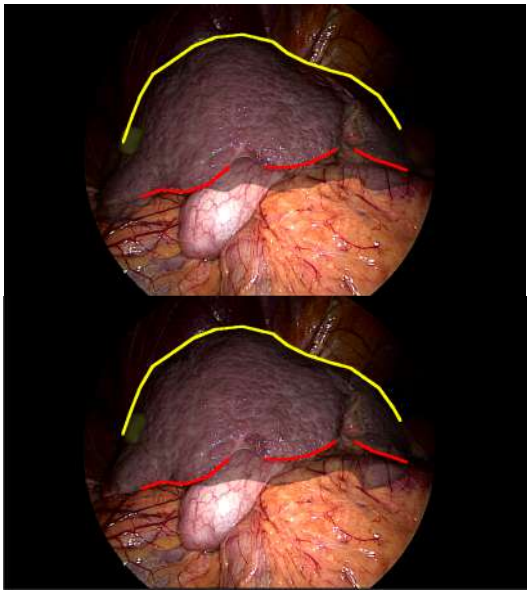


Fig. 18: Semi-automatic deformable registration of the liver using three ridge contours (red) and one silhouette contours (yellow)

The third patient's liver has been deformed using 2 ridge contours, 2 silhouette contours, and 1 ligament contour. For this case, the tumour was inside the organ but had a considerable size and thus was visible from outside, with the ligament passing just in the middle of it. The time it took to do this registration was of 4 minutes and 20 seconds. In figure **??** we can see the deformation made to fit the liver as shown in the laparoscopic image.

In the manual deformable registration, we are using a total of 5 regions. We first mark a contour region around the

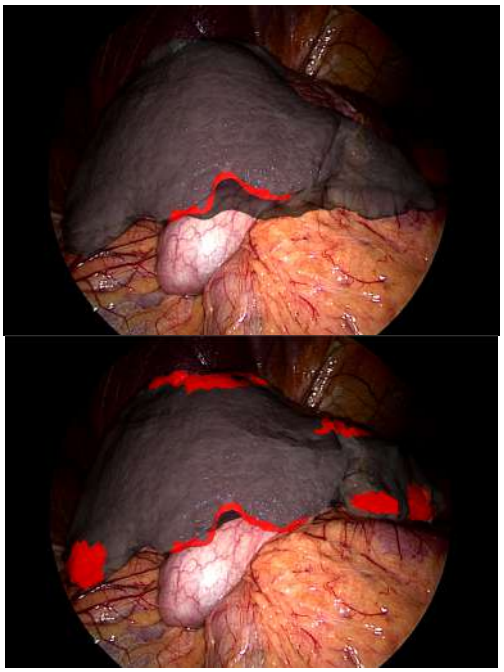


Fig. 19: Manual deformable registration of the liver using 6 regions of vertices: 1 contour region and 5 generic regions

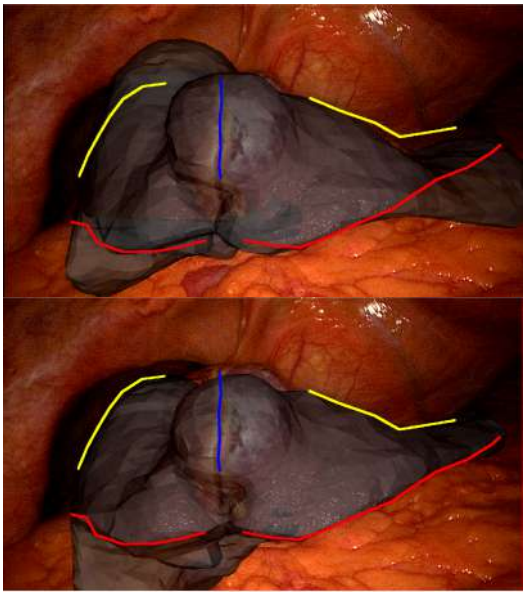


Fig. 20: Semi-automatic deformable registration of the liver using two ridge contours (red), two silhouette contours (yellow), and one ligament contour (blue)

location where the gallbladder is, and then the remaining 2 regions. The three regions in the left are used as fixed regions, while the two in the right are moved to fit the right part of the lobe. The time spent to do this registration was of 1 minute 52 seconds. In Figure 21 we appreciate the liver after manual deformation.

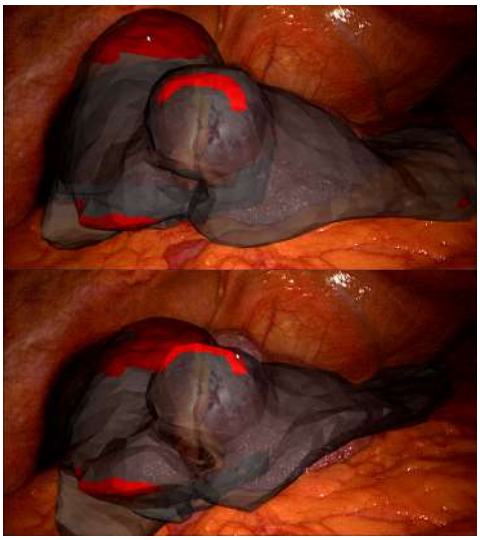


Fig. 21: Manual deformable registration of the liver using 5 regions of vertices: 3 contour regions and 2 generic regions

If we compare the times spent to deform the model using the semi-automatic and the manual approaches, we can see a general reduction on them in favor of the latter one. In the Table I we can see a summary of the deformation times for each patient and the variations between the two methods.

Deformation times			
Patient	Semi-automatic	Manual	Variation
1	04:26	00:35	7.6x
2	03:32	01:53	1.87x
3	04:20	01:52	2.32x
Average:	04:10	01:44	2.4x

TABLE I: Deformation times for the semi-automatic and manual methods

We can see how there is a substantial reduction in the time required to perform a manual deformation, compared to the semi-automatic method. Most of the extra time required by the latter is the care the user must have when selecting the contours in both the preoperative model and the organ in the camera image, as those should correspond as much as possible. In general, we see a reduction of 2x in the time

used to register the preop model manually.

B. Ultrasound image registration using tactile gestures

We had the opportunity to test our initial registration method in a patient's kidney. In this opportunity the patient was under a tumour extraction intervention, where the surgeon used an ultrasound imaging system to verify the location of the tumour in the organ. Thanks to this, we could get a synchronized stream of laparoscopic and ultrasound sequences. We take then one of these views and try to register the ultrasound image to its corresponding laparoscopic view, by means of the developed semi-automatic algorithm.

We took a laparoscopic image where the probe is located in a position where the tumour is visible. Then, in the Figure 22 we see the result of registering the US image with the laparoscopic view according to the pose of the probe.

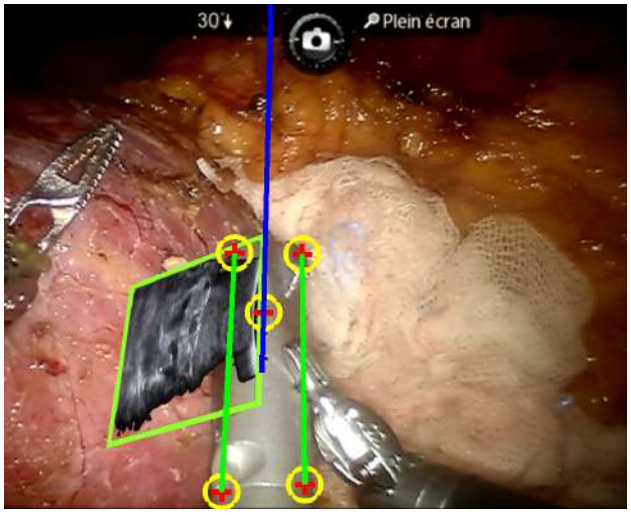


Fig. 22: Ultrasound image registered under the US probe

As seen in the registration result, we have used the marks provided by the probe to define the depth and the rotation of the ultrasound plane. Like this, we have an US image that corresponds approximately to what the probe sees under the organ. We could then use this to verify if the registration of a preoperative model has been correct.

VI. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

Regarding the effectiveness and easiness of the manual deformation strategies, we can say that our method achieves similar results as the semi-automatic one in a much shorter time. This is specially useful for surgeons as they prefer not having to wait 10 minutes for every new registration. Our approach could also be seen as a very stable alternative when a semi-automatic or automatic method fails, as it will always behave according to the surgeon's expectations.

We have successfully created a method to initialize the pose of an ultrasound image so that it is located under its corresponding probe in a laparoscopic scene. This will serve as a starting point to develop a more autonomous method in which the US plane follows the movements of the probe and the images are updated accordingly, taking as reference this initial pose.

This developed solution already can help surgeons to perform better incisions, causing less damage to the liver and offering a better quality of life to the patient in the end.

B. Future Works

As a future task, we want to extend this application to perform multiview registration of the preoperative model into several intraoperative laparoscopy images. This will help the user to have a better perception of depth in the scene, leading to a more accurate registration.

We also plan to implement a fully automatic non-rigid registration method, in the framework of a larger project that involves augmented reality for the liver. This will take as base the work done so far for the liver, specifically everything made for the Hepataug program.

As mentioned in the conclusions, there are plans to extend the application to mobile devices (like iPads or Android-based devices), so that surgeons can easily perform these registrations without the need of translating a whole workstation to the surgery room. Because the deformation algorithms normally require a lot of processing power, a good solution would be to implement a client-server structure, where the Tablet acts as the client that connects to the main workstation through an internet connection.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the collaboration given by the surgeons at the CHU de Clermont-Ferrand, France, for letting us to attend those surgeries and for the feedback they provide regarding our software.

REFERENCES

[1] Roberts, D.W., Strohbehn, J.W., Hatch, J.F., Murray, W., Kettenberger, H., 1986. A frameless stereotaxic integration of computerized tomographic imaging and the operating microscope. J. Neurosurg. 65 (4), 545549.

[2] Marcus, H.J., Pratt, P., Hughes-Hallett, A., Cundy, T.P., Marcus, A.P., Yang, G.-Z., Darzi, A., Nandi, D., 2015. Comparative effectiveness and safety of image guid- ance systems in neurosurgery: a preclinical randomized study. J. Neurosurg. 17.

[3] Cabrilo, I., Schaller, K., Bijlenga, P., 2015. Augmented reality-assisted bypass surgery: embracing minimal invasiveness. World Neurosurg. 83 (4), 596602.

[4] Zinser, M.J., Sailer, H.F., Ritter, L., Braumann, B., Maegele, M., Zller, J.E., 2013. A paradigm shift in orthognathic surgery? Acomparison of navigation, comput- er-aided designed/computer-aided manufactured splints, and classic intermaxillary splints to surgical transfer of virtual orthognathic planning. J. Oral Maxillo-fac. Surg. 71 (12). 2151e1.

[5] Wengert, C., Cattin, P.C., Duff, J.M., Baur, C., Szkely, G., 2006. Markerless endoscopic registration and referencing. In: Medical Image Computing and Computer-As- sisted InterventionMICCAI 2006. Springer, pp. 816823.

[6] Bajura, M., Fuchs, H., Ohbuchi, R., 1992. Merging virtual objects with the real world: seeing ultrasound imagery within the patient. In: ACM SIGGRAPH Computer Graphics, 26. ACM, pp. 203210.

[7] Marescaux, J., Leroy, J., Gagner, M., Rubino, F., Mutter, D., Vix, M., Butner, S.E., Smith, M.K., 2001. Transatlantic robot-assisted telesurgery. Nature 413 (6854), 379380.

[8] Ukimura, O., Nakamoto, M., Sato, Y., Hashizume, M., Miki, T., Desai, M., Aron, M., Gill, I.S., 2010. Augmented reality for image-guided surgery in urology. In: New Technologies in Urology. Springer, pp. 215222

[9] Simpfendrfer, T., Baumhauer, M., Mller, M., Gutt, C.N., Meinzer, H.-P., Rass- weiler, J.J., Guven, S., Teber, D., 2011. Augmented reality visualization during la- paroscopic radical prostatectomy. J. Endourol. 25 (12), 18411845.

[10] Nakamura, K., Naya, Y., Zenbutsu, S., Araki, K., Cho, S., Ohta, S., Nihei, N., Suzuki, H., Ichikawa, T., Igarashi, T., 2010. Surgical navigation using three-dimensional com- puted tomography images fused intraoperatively with live video . J. Endourol. 24 (4), 521524.

[11] Tansaoui, I., Ozgur, E., Bartoli, A., 2016. 3D-2D Manual Registration & Augmented Reality (Project HEPATAUG). ALCoV - ISIT, Universit dAuvergne, Clermont 1.

[12] Espinel, Y., Bashir, M., Ozgur, E., Calvet, L., Koo, B., Bartoli, A., 2017. Tactile Gestures for 3D-2D Registration Interface Applied to Laparoscopy. EnCoV - TGI, Universit Clermont Auvergne.

[13] Scratchapixel. (2018, May 15). The Perspective and Orthographic Projection Matrix. Retrieved from http://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix

[14] Bender, J., Koschier, D., Charrier, P., Weber, D., 2014. Position-based simulation of continuous materials. In: Computers & Graphics 44. Elsevier, pp. 1-10.

[15] SATT Grand Centre. (2018, May 20). Laparaug. Retrieved from http://www.sattgc.com/fr/portfolio/laparaug/

Learned Features for Matching Renders with Real Images for Industrial Visual Inspection

Pamir Ghimire and Igor Jovančević

***Abstract*— We learned a descriptor for matching real images of mechanical assemblies with simple renders of their CAD's with similar viewpoint. The descriptor is a deep Convolutional Neural Network which after training extracts features that are invariant to domain (real or synthetic) and rotation, and depend only on geometric information. A small dataset of less than 20K patch pairs extracted from real and synthetic images was created in a semi-supervised fashion for training the descriptor. Transfer learning was used due to the small size of the dataset. Descriptor was trained by minimizing triplet loss so that learned features could be compared using l_2 distance. The FPR95 score for the best descriptor was 13.8. The descriptor was used for finding local correspondences between simple renders and real images, and comparing for conformity crops of real images with those of renders by representing them as histograms based on Bag of Words of the learned features. We conclude that learning robust cross-domain rotation invariant descriptors is feasible, and such features may be of interest for various industrial applications like CAD based visual inspection, tracking, and finely registered augmented reality. To the best of our knowledge, this is the first work that presents learned features for local matching between renders and real images.**

I. INTRODUCTION

In 3D CAD based industrial visual inspection, we want to check whether mechanical assemblies that are produced according to CAD specifications actually conform with them. We are interested in using passive 2D RGB cameras for the same. The parts of the assembly that we want to inspect, henceforth auto-control or inspection elements, are expected to be with a certain pose in relation to a context. The auto-control elements can be big parts like supports and clamps or small ones like screws and caps. The context can be frames of cars, airplanes, engines, etc. Separate 3D CAD's are available for the auto-control elements (inspection CAD's) and their contexts (context CAD's). The CAD's share the same coordinate frame.

In inspecting the assemblies for auto-control elements, the viewpoint (pose + calibration) of the 2D camera relative to the inspected assembly is known to us through existing software. The respective CAD's can hence be rendered onto the 2D image plane in one of various ways so that features in the render can be compared with those in the real image inside a region of interest to produce an inspection decision.

For comparing renders and real images, 2D features like contours and line segments are popular choices since rendering them from their 3D descriptions is feasible for real-time inspection and extracting them from real images is also well explored. However, such 2D features are not very informative, susceptible to false edges due to shadows and textures in real images, as well as 3D symmetries in the CAD (contours can be the same for two viewpoints). In contrast, $2\frac{1}{2}$D features, which are features extracted from rendered faces of the CAD, capture more detail associated with the viewpoint and are less susceptible to the problems mentioned before.

We propose to use $2\frac{1}{2}$D features extracted from patches centered at interest points in a simple render and a corresponding real image with similar viewpoint to compare the 2D real image with 3D CAD through a render. We propose to render the faces of the CAD and not just edges and contours to capture richer viewpoint dependent information.

Existing local descriptors like handcrafted SIFT and SURF and learned LIFT [1] vary with texture in the real image and were not designed for comparing renders with real images. We hence choose to learn a dedicated local descriptor as described in section III B. For this, it was necessary to create our own dataset of patch-pairs, described in section III A, since existing datasets like in [9] contain only real image patches. Due to a limited number of patch-pairs, we used transfer learning (section III B 1), and minimized triplet loss between descriptors so that they could be compared using l_2 distance without a dee[learned metric (section III B 2). Our descriptor shows good performance in terms of ROC curves and FPR95 score (section IV A). Nearest-neighbor matching wasn't very impressive, although some successful examples have been presented in section IV B. The learned descriptors were also used for comparing crops of real images and renders by representing them as histograms of words in a learned Bag of Visual Words (BoW) [1] based on the learned features (section IV C). We begin below in section II with review of related work.

II. RELATED WORK

A. Industrial Visual Inspection

Inspection of mechanical assemblies using 2D and 3D vision has diverse industrial applications like detection of surface defects, scratches and fissures, dimensional defects, absence of parts, etc. Some contexts for applications are assembly lines of cars, sub-sea inspections and fast paced mass manufacturing. Popular 3D sensors are LIDAR, LIMAR and Time of Flight or Structured Light cameras alongside passive 2D RGB cameras [2]. We are interested in using 2D cameras to inspect 3D assemblies produced in accordance with 3D CAD specifications. We want to know about presence/absence and misalignment of auto-control elements. [4] and [5] address similar problems. [4] compare renders of mechanical parts with real images by matching graphs of 2D features like ellipses and segments extracted

[1] http://www.robots.ox.ac.uk/~az/icvss08_az_bow.pdf

from the two images independently. [5] compare projections of 3D descriptions of similar features in CAD with ones extracted from the real image independently using image processing. Both these methods face limitations associated with 2D features described before. Comparison using 2D features was also described by Agin [3] (1980). However, he identified advantages of richer view-point dependent information in $2\frac{1}{2}$D features described as descriptions extracted from image patches. Although local features are used in inspection for real-real(image) comparison [6], to the best of our knowledge, they are absent when renders of 3D CAD (with faces, not just edges or silhouettes) are to be compared with real images.

B. Synthetic Images from CAD Renders for Learning

Outside of industrial visual inspection, synthetic images have been successfully used in recent literature for training deep networks for semantic segmentation [7] and 3D object detection [8] among other applications. 3D engines like Unity3D [7] and Blender have been important for producing large synthetic datasets with realistic shading. In these works, availability of texture mapped models has been important [7]. Google's 3D warehouse has been an important source of such models. However, invariance to color and texture cues while training object detectors has also been demonstrated [8]. When using models trained on synthetic data on real data, domain shift is an important problem. A simple way to address this is to train on a mixed dataset of real and synthetic images as suggested in [7].

C. Discriminative Feature Learning

Learned features have been recently shown to outperform handcrafted features like SIFT and SURF for matching real images [1]. Mostly, descriptors are learned independently from detectors using pairs of image patches [9] [10] [11]. The descriptors are either trained jointly with a deep-network metric by minimizing a patch pair classification loss [10] [11], or standalone by minimizing a loss based on euclidean distances between the descriptors [9]. Descriptors from networks trained in the latter manner are also called embeddings. Learning discriminative embeddings is the strongest approach in modern face recognition literature [12] [13]. Recent works in both face recognition and patch comparison literature minimize triplet loss for learning discriminative embeddigs. In patch comparison literature, patches are small, typically 64x64, while they are larger in face recognition literature, typically $\sim$ 200x200. Also, networks are shallow in the former (3 and 4 convolutional layers in [11], 3 in [9] and 5 in [10]) while they are relatively deeper in the latter (13 convolutional layers in [13], and 11 in [12]). Deep architectures in [10] and [11] are based on Alexnet [14] and in [13] on VGG16 [15].

III. Methodology

A. Creation of Patch-Pair Dataset

For training a descriptor like in [9] [10] [11], we need a dataset of patch-pairs produced from renders and real images. Since there isn't an existing one to the best of our knowledge, we create such a dataset by cropping manually registered pairs of renders and real images around interest points detected by the FAST detector[1]. FAST points were detected only in the (registered) render, not in the real image, because our 'simple' renders have edges only at locations with discontinuities in surface normal, depth from camera and element identity (auto-control or context) due to the shading described in Fig. 1. This produces patch pairs with meaningful 'geometric' features that can be used for training a descriptor invariant to texture and color variations in real images. Our CAD's do not have mapped textures or colors. A dataset of real image patches with textures was also created separately for use during training. The process of creating the dataset has been illustrated in Fig. 2. We used Unity 3D for producing our renders.

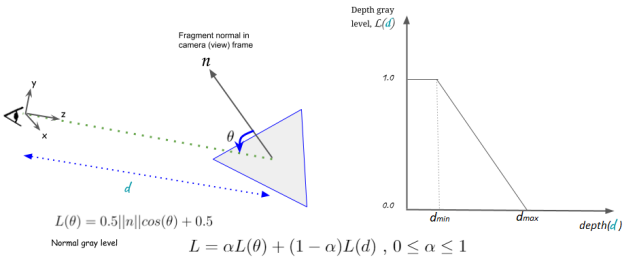


Fig. 1. Simple shading for fast rendering based on fragment normal and fragment depth along optical axis. Intensity of fragment is also varied based on whether it belongs to CAD of inspection element or that of context.

B. Training a Cross-domain Feature Extractor

There are 2 main challenges in training with the patch-pair dataset produced as described in III A :

- Small number of patch pairs are available since less than 100 image pairs were available for producing the training dataset, but large patch sizes are necessary for discriminative context, thanks to simplistic CAD's
- Domain shift due to patches coming from synthetic and real domains

Both these problems were addressed in the 'bootstrapping' stage. The weights learned at the end of this stage were used to initialize the triplet loss training stage resulting finally in a discriminative feature extractor. This strategy is similar to the one used in [13]. We also use similar notations.

1) Bootstrapping: We start with the VGG16 deep architecture as detailed in [15] (Table 2, Column D) with ImageNet weights[2]. The filters in the first convolutional layer were modified from 3x3x3 to 3x3x1 and their weights initialized by averaging the 3D filters along the 3^{rd} dimension [16]. The 1000-way linear+softmax layer was replaced with 2-way linear+softmax layer. The 2 fully connected layers following the convolutional layers were initialized from scratch using Xavier initialization [16] (Fig. 4, step 1). In the original architecture, the fully connected layers have 4096 neurons each. For us, they have L neurons each, with $L = 1024$

[1]OpenCV 3.0.0-dev documentation » OpenCV-Python Tutorials » Feature Detection and Description

[2]https://github.com/machrisaa/tensorflow-vgg

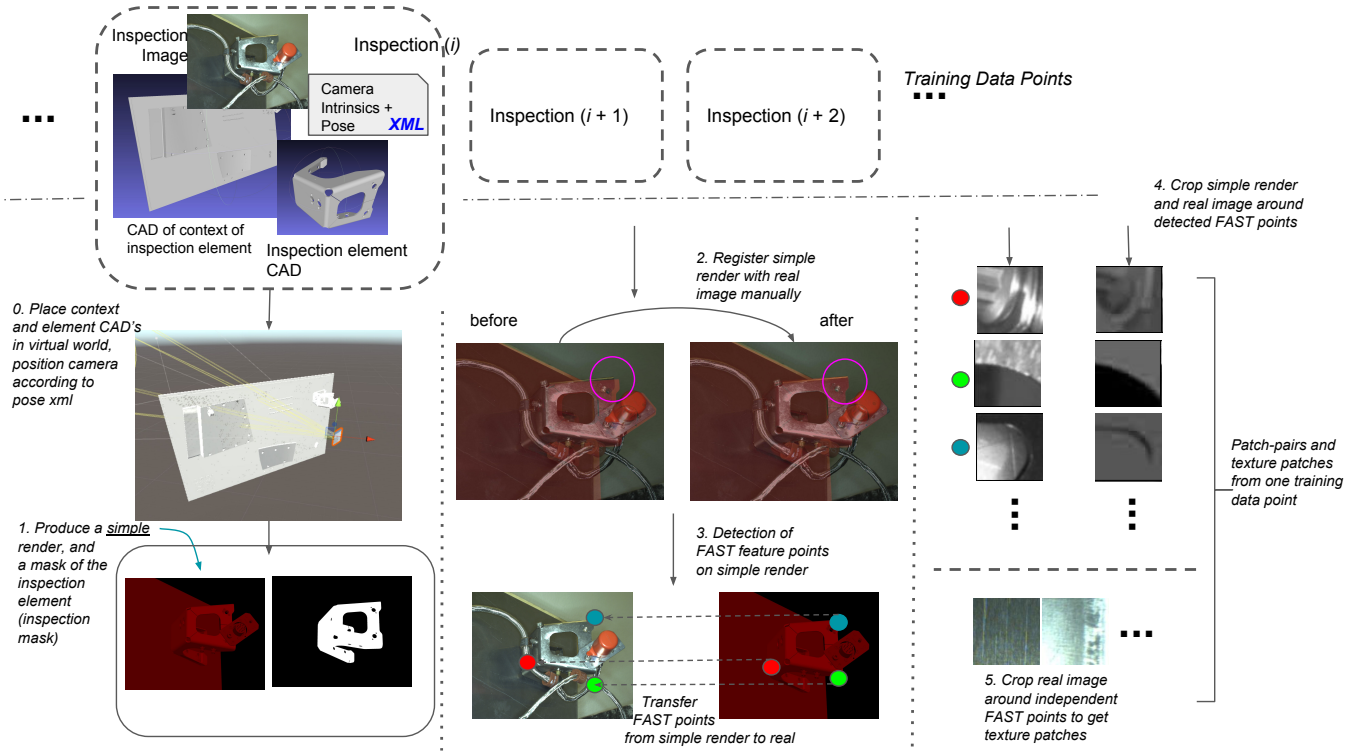


Fig. 2. Proposed method for creating patch-pair dataset given CAD's of inspection (same as auto-control) elements and their contexts along with real images with tracked viewpoint

when patches are of size 128x128 and $L = 4096$ when they are 224x224.

For bootstrapping, the patch-pair dataset described before was organized into 2 classes, one containing meaningful geometric information and the other not (texture patches). Both classes contained patches from both real and synthetic images. A cross-entropy loss (Equation 1) was minimized to classify these patches. All layers except the first convolutional and the last fully connected layers were 'frozen' during this stage (Fig. 4, step 2). Training was performed for 2 epochs with mini-batch gradient descent using batches of size 128, initial learning rate was 0.005, and dropout was used after fully connected layers for regularization with $p = 0.5$.

$$E = \frac{1}{n}\sum_{i}[y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})] \qquad (1)$$

Where $y_i \in [0, 1]$ and $\hat{y_i} = \frac{e^{x_i}}{e^{x_0}+e^{x_1}}$.

2) Triplet Loss Training: The weights learned from bootstrapping were retained except for the last 2-way linear+softmax layer. We denote the retained architecture with ϕ. It maps an input patch $P \in \mathbb{R}^M$ to $\mathbb{R}^D$; $\phi(P) \in \mathbb{R}^D$. To ϕ, we appended an l_2 normalization layer and a linear layer $W' \in \mathbb{R}^{L \times D}, L << D$, which together implement $e_P = W'\phi(P)/||\phi(P)||_2$. W' is an affine transformation layer without any bias, since bias would be canceled when minimizing triplet loss detailed in Equation 2. The thus augmented ϕ is replicated thrice to create a triplet network [12] [13] and triplet loss is minimized for batches of triplets (anchor (a), positive (p), negative(n)) produced by a batch server.

$$E(W') = \sum_{(a,p,n)\in T} max\{0, \alpha - d_{an} + d_{ap}\} \qquad (2)$$

Here, $d_{an} = ||e_a - e_n||_2$, $d_{ap} = ||e_a - e_p||_2$, and T is the set of triplets in each training mini-batch. $\alpha > 0$ is the margin parameter that describes the desired difference between average distances between matching and non-matching patches. The method and notations followed in this section are the same as those in [13].

During triplet loss training, only the 'embedding layer' W' is updated, the rest of the layers are frozen. Thus, it is W' that

learns a discriminative projection of features extracted by ϕ (Steps 3 and 4 in Fig. 4). The architecture ϕ together with the l_2 normalization W' layers is the learned cross-domain feature extractor. The extracted features are of dimension D, with $D = 512$ when patches are of size 128x128 and $D = 1024$ when they are of size 224x224. Of the $D's$ we experimented with, these were the best values.

During triplet-loss training, it is important to pick 'hard' triplets i.e., triplets for which $d_{an} < d_{ap}$ so that the network can learn from training with them. We retain only hard triplets in a batch of triplets of size 128 or 64 produced by a batch server by passing them through the network once before training for getting their embeddings. We also perform 'in triplet hard-negative mining' described in [9] by swapping anchor and positive patches if $d_{an} > d_{pn}$. This improves triplet loss training. We use the Adam optimizer with an initial learning rate of 0.005, β_1 0.9, β_2 0.999 and ϵ 1e-08. Regularization was performed via dropout added before the embedding layer.

The batch server when picking negative patches picks either a geometrically meaningful non-matching real patch respective to the anchor or a real texture patch, of which there are thousands in a 'reservoir'. We under sample the texture patches with ratio of 3:7. Random rotations are also applied to patches on the fly so that the descriptor learns to be robust to rotations.

C. Work-flow at Test-time

At test time, FAST points (or some other interest points) are detected independently in real image and the corresponding render. A patch of a fixed size is extracted around each interest point location and the learned descriptor evaluated for it. The features extracted from the images can then be used for nearest neighbor matching or converted into histograms of words based on Bag of Visual Words for comparing the pair for producing an inspection decision (See Fig. 3).

IV. Experiments and Results

We trained different networks with patch-pairs of two sizes, 128x128 and 224x224, and different values of the margin parameter 'α' (equation (2)). Table I presents the FPR95 rates, which are false positive rates when true positive rates are 95%, observed for the different descriptors. They were obtained using a test set of 10K patch pairs which were classified as either matching or non-matching based on whether the euclidean distance between their embeddings was below certain thresholds. The best score of 13.8 was observed for 224x224 descriptor with $\alpha = 5.0$. For reference,

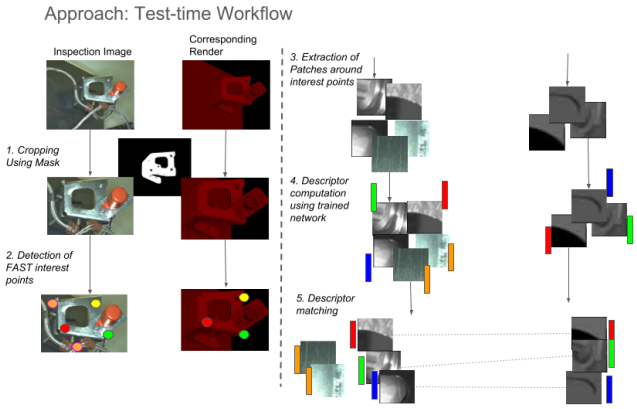


Fig. 3. Proposed workflow for using the descriptor at test-time for nearest-neighbor matching

Margin α	**128x128 Patch Descriptor**	**224x224 Patch Descriptor**
2.0	40.1	17.0
5.0	41.2	**13.8**
10.0	37.8	17.5

TABLE I
FPR95 RATES OBSERVED FOR LEARNED DESCRIPTORS.

[9] reports this score for SIFT to be between 26.0 and 30.0 when matching natural patches.

A. ROC Curves

Figure 5 presents ROC curves which are plots of true positive vs false positive rates for different descriptors mentioned in Table I. Change in α shows little effect on the ROC of the 128x128 descriptor. 224x224 descriptors for all values of α perform better than the corresponding 128x128 descriptor.

B. Nearest Neighbor Matching

We present only qualitative examples of some nearest neighbor matching between renders and real images in Figure 6 . As can be seen, the renders are very simplistic and are lacking in parts like screws and wires present in the real images. Results for one-to-one matching were generally poor despite a good FPR95 score. Good ROC does not imply good nearest neighbor matching, as explained in [11].

C. Bag of Visual Words with Learned Descriptor

Besides nearest neighbor matching for comparing image pairs, we also test histograms based on Bag of Visual Words (BoW) learned from a set of renders and real images by extracting from them ORB features (rotated BRIEF) [11]

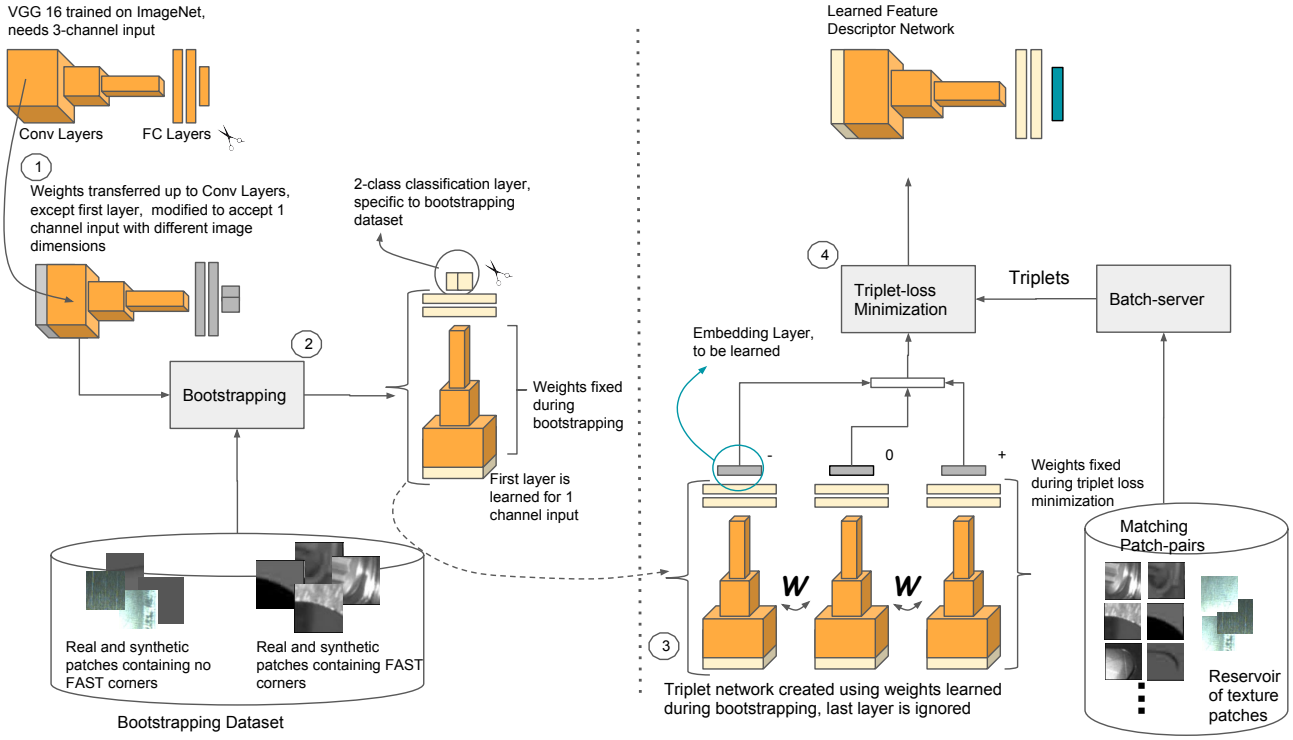


Fig. 4. Proposed method for training a descriptor using a dataset of patches created as described in Figure 2

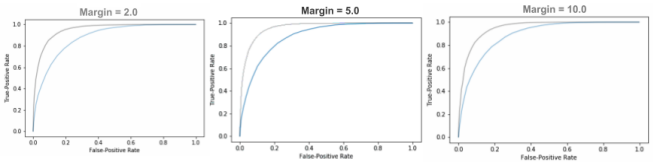


Fig. 5. ROC curves for 128x128 (blue) and 2224x224 (gray) learned descriptors for different margin parameters.

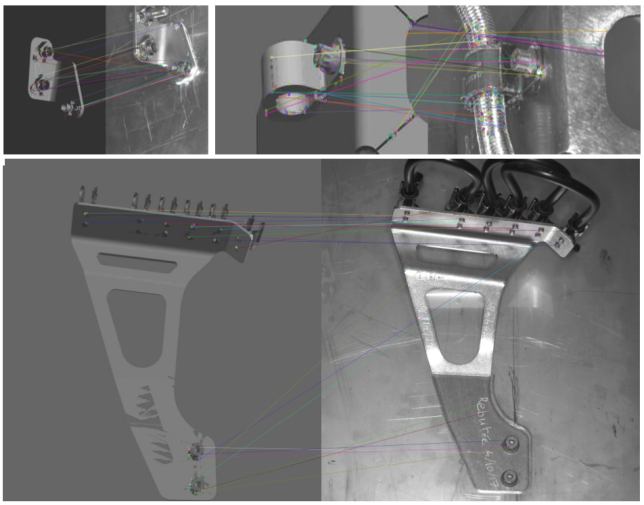


Fig. 6. Some examples of successful nearest neighbor matching between renders (left) and real images (right).

and the learned features (LF) around FAST interest points. The two learned dictionaries each contained 50 words.

For every test pair, the two features were extracted at FAST points, and histograms were created using the features and dictionaries. For each image, 2 histograms were computed, one based on ORB BoW and the other on LF BoW. One pair hence produced 4 histograms, and 2 distances, one between the ORB BoW histograms and the other between LF BoW histograms. The distances between all **matching** test pairs have been shown in Fig. 7 (e). The LF based distances between histograms were observed to be less erratic than ORB based ones. Classification by thresholding the distances was not performed because of availability of few

non-matching pairs (red in Fig. 7 (a), (b)).

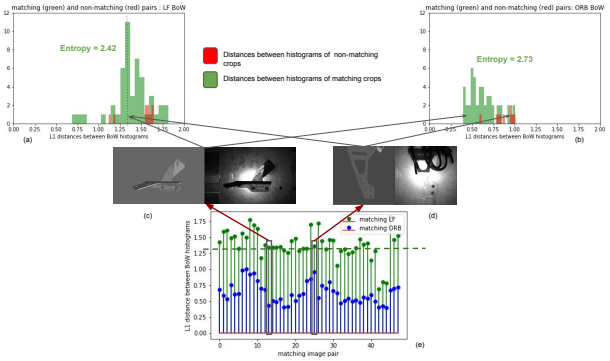


Fig. 7. Comparison of image pairs using Bag of Visual Words histograms learned from renders and real images using ORB features and the learned features (LF).

V. Discussion and Conclusions

We saw that it is possible to learn a feature descriptor to compare simple renders with real images using relatively few image pairs. The learned descriptors showed better performance when larger patches were used, possibly due to simplistic nature of the used CAD's so that bigger patches provided more discriminative context. A 2-stage training strategy was necessary for learning the descriptor, where the first stage was for learning initial weights that addressed domain-shift before learning discriminative cross-domain embeddings. The learned descriptors were used for comparing image pairs by nearest neighbor matching and through histograms based on Bag of Words built using the learned features. The learned features can thus be used in a variety of ways, much like other local features.

A. Future Works

Although we used a basic shading as described before, a slightly more detailed shading, like Phong [1], might provide better performance while still enabling fast renders. The deep architecture used was VGG16, which is expensive to evaluate and impractical for real-time inspection. Smaller networks like one in [9] need to be explored. The FAST detector was used for sake of dense detections, but its interest points are not best suited for comparing real and synthetic images. Detectors like SIFT or MSER [1] need to be tested, a detector could also be learned. Matching pairs or triplets of descriptors between image pairs needs to be tested instead of one-to-one nearest neighbor matching since correspondence between a simplistic CAD and real assembly is not one-to-one. Finally, realistically colored and textured renders can be explored for use in stead of real images so that a custom descriptor can be trained for a completely new kind of assembly with none or few real images of the same.

[1]https://learnopengl.com/Lighting/Basic-Lighting

VI. Acknowledgments

The authors gratefully acknowledge the support of Diota-Control, Toulouse, for supporting this work.

References

[1] Schonberger, Johannes L., et al. "Comparative evaluation of hand-crafted and learned local features." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2017.

[2] Timothy S Newman and Anil K Jain. A survey of automated visual inspection. Computer vision and image understanding, 61(2):231262, 1995.

[3] Gerald J Agin. Computer vision systems for industrial inspection and assembly. Computer, (5):1120, 1980.

[4] Ilisio Viana, Florian Bugarin, Nicolas Cornille, and J-J Orteu. Cad-guided inspection of aeronautical mechanical parts using monocular vision. In Twelfth International Conference on Quality Control by Artificial Vision 2015, volume 9534, page 95340I. International Society for Optics and Photonics, 2015.

[5] Jovanevi, Igor, et al. "Automated exterior inspection of an aircraft with a pan-tilt-zoom camera mounted on a mobile robot." Journal of Electronic Imaging 24.6 (2015): 061110.

[6] Claudio Cusano and Paolo Napoletano. Visual recognition of aircraft mechanical parts for smart maintenance. Computers in Industry, 86:2633, 2017.

[7] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 32343243, 2016.

[8] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In Computer Vision (ICCV), 2015 IEEE International Conference on, pages 12781286. IEEE, 2015.

[9] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In BMVC, volume 1, page 3, 2016.

[10] Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, pages 32793286. IEEE, 2015.

[11] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, pages 43534361. IEEE, 2015.

[12] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 815823, 2015

[13] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In BMVC, volume 1, page 6, 2015

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 10971105, 2012.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[16] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. arXiv preprint arXiv:1507.02159, 2015

Segmentation of Fish Instances in Underwater Imagery using Mask R-CNN

S. Guillaume, R. García, R. Prados and J. Quintana
solene_guillaume@etu.u-bourgogne.fr , {rafa, rprados, josepq}@atc.udg.edu

Abstract

Fishing vessels cannot know in real time what type of fish species is being caught by the trawl net. Fish sorting is performed on the boat, and the rejected fish, often dead, is thrown back into the sea. The main goal of the Deep Vision device is to build a system able to perform automatic recognition and measurement of the fish passing through the trawl, leading to precise selective fishing.

The system will allow to keep in the net the preferred fish, while driving back to the sea the unwanted ones by means of a motorized gate. The first step to perform selective catching consist of segmenting the fish on the pictures acquired by the Deep Vision device. The result of this segmentation can be used later for the fish species recognition and measurement in further steps. Specific algorithms have been developed to segment fish in the images acquired by the Deep Vision device. However, those algorithms are not able to appropriately segment fish when those are superposed, a situation that happens in crowded scenes.

The goal of this work is to outperform the results obtained by existing algorithms by using deep learning techniques. For this purpose, Mask R-CNN has been studied and tested for optimizing fish segmentation. The results obtained by the method are encouraging, and have demonstrated its suitability to effectively perform fish segmentation.

Introduction

The number of fish available in the sea is decreasing rapidly, and this is leading to ecological and economical problems. Until some years ago, there were no common rules among the different countries to manage the fishing, but this situation has started to change since 2015. To reach a more sustainable fishing, the European Union created the "landing obligations", aiming to eliminate progressively the discards in all the Union fisheries, for catches of species subject to catch limits [1]. The main challenge of a more selective and consequently efficient fishery comes from the inability of the fishing vessels to know precisely the species and key characteristics (such as the size) of the specimen being caught. According to a study performed during six months in the Papua New Guinea aquarium in 2016 [2], 24.2% of fish from catches was rejected and thrown back to the sea, either dead or alive. To avoid catching unwanted fish, the Scantrol Deep Vision project arises as a system allowing to automatically select the fish under the sea, by opening and closing a motorized gate located in the trawl net cod end. To perform fish selection, the Deep Vision device (see Figure 1) is equipped with a stereo camera system that takes still pictures at a relatively high frame rate, enabling the real-time study of the specimen being caught. The device is placed at some meters of the entrance of the net to allow imaging all the fish passing by.

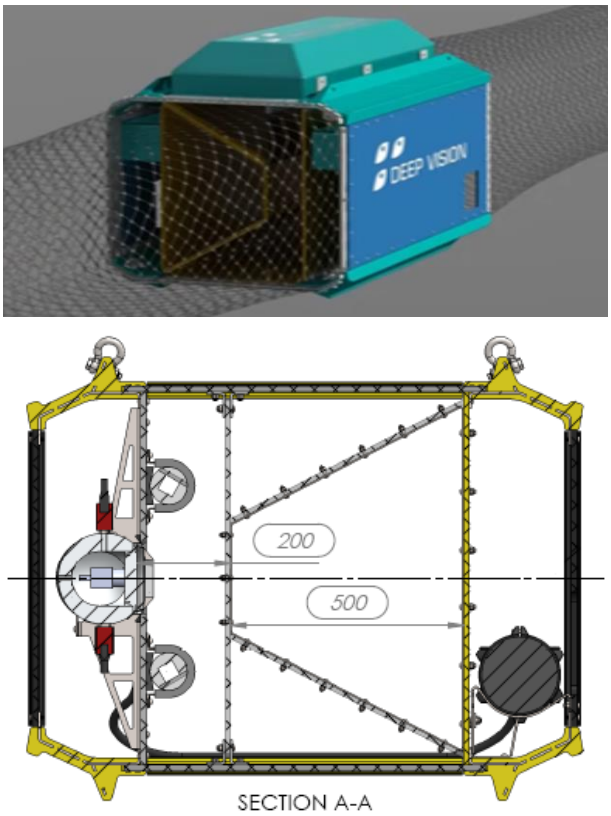


Figure 1. (Top) Scantrol Deep Vision device attached to the trawl net. (Bottom) Section scheme of the Scantrol Deep Vision image acquisition system depicting the cameras, lighting system and distances between the elements.

The method described in [3] was implemented to perform fish segmentation on the images acquired by the Deep Vision system. This algorithm performs a good segmentation when the fish on the image appear isolated, *i.e.*, when the fish are not overlapping. Nevertheless, it is unable to distinguish among fish when those are overlapping, situation that often happens in crowded scenes. Additionally, the algorithm sometimes cannot provide a precise segmentation of fish close to the image corners and the fins are undersegmented in certain

scenarios. The main objective of this paper is to outperform the results obtained in [3]. A promising deep-learning based algorithm named Mask R-CNN and able to segment and classify objects in a single step has been recently developed and presented by the Facebook research team, obtaining highly accurate results. Mask R-CNN, by performing instance segmentation (see Figure 2), is able to appropriately segment and classify objects that overlap. In our case, we are going to use images of fish acquired by the Deep Vision device in a real scenario.

State of the Art

As stated above, our problem is to be able to segment fish in a crowded scene and/or additionally to classify them. Deep learning is able to perform instance segmentation with very high accuracy, it is easy to adapt to any new environment and it is simple to use. According to [4], the most accurate or/and fast deep learning algorithms for instance segmentation are, in order of apparition: Faster R-CNN [5], Yolo [6], R-FCN [7], SSD [8], Mask R-CNN [9] and NASNet [10].

Faster R-CNN is a Convolutional Neural Network (CNN) in which the fully connected layers situated in the end of the network, called also Region Proposal Network (RPN), are proposing regions likely to contain objects. In a powerful enough computer and with a very deep network, it can run at five images per second. Yolo is a CNN predicting bounding boxes and class probabilities like Faster R-CNN. Nevertheless, and unlike Fast R-CNN, it is not analyzing one by one the extracted patches from a moving window but the entire image. This network is fast, and can process from 45 to 150 images per seconds. Unfortunately, it learns very general representations of objects, and it is less accurate than other algorithms. In fact, it performs almost the same than Fast R-CNN on PASCAL VOC 2007 [4]. R-FCN is a Fully Convolutional Network (FCN) for region detection and classification. A FCN is a CNN where the last fully connected layer is substituted by another convolution layer. Unlike Faster-RCNN, it does not apply a lot of different computations on each of the extracted patches: it shares 100% of the computations across every single output. For each class, it has learned a score map. Then the score map is compared with each region of interest (extracted by a RPN) piece by piece. R-FCN is several times faster than Faster-RCNN with comparable results. Contrarily to Faster-RCNN and FCN, Single Shot Multibox Detector (SSD) skip the part that generates regions of interest: it does not use an RPN network. For every moving window, a bounding-box and its corresponding label are found. Then Non-Maximum Suppression (NMS) and other methods are applied to discard the false positive bounding-boxes. SSD is faster than both R-FCN and Faster R-CNN and it still performs quite comparably. NASNet is an algorithm that optimizes the architecture of a CNN while this CNN is being trained on a dataset. The parameter to optimize is the number of layers. Optimizing the architecture of a network directly to a dataset should allow the network to generalize more easily and more precisely any dataset, with an architecture less heavy. NASNet provided similar results to Yolo on the PASCAL VOC 2015.

We choose Mask R-CNN [9] because it is performing segmentation in addition of detection and classification, with a very good accuracy and still a good speed to process images. As can bee seen in Figure 2, we can expect having good results for crowded scenes.

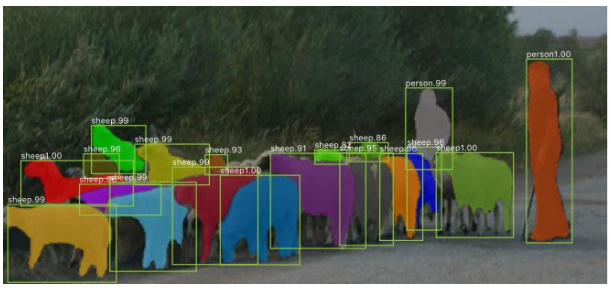


Figure 2. Output of Mask R-CNN, built using RESNet 101. Bounding boxes are represented by green frames, masks are fully colored and labels are written in white [9].

With a powerful enough computer, Mask R-CNN can process up to five images per second. According to the authors “Mask R-CNN outperforms all existing, single-model entries on every task, including the COCO 2016 challenge winners”. Mask R-CNN is an extension of Faster R-CNN, which adds some improvements to the first one. Currently, the Mask R-CNN seems to be the most appropriate framework for the task that needs to be carried out.

We need to perform the fish segmentation proposed in [3] on a large enough number of images, in order to build the ground-truth that will be used to train Mask R-CNN. Until the last year, the algorithms performing fish segmentation were scarce and none of them used an environment similar to the one setup on the Deep Vision system [3]. “Real-time Fish Detection in Trawl Nets” was conceived with the idea of taking profit of the specific acquisition environment properties of the Deep Vision system, and consequently is able to deal with the main difficulties faced to segment the fish from the background in this controlled environment. That is, the method is able to deal with the effects of non-uniform illumination, fish shadows and spurious specular reflections from the lighting system on the moving specimens. This is the main reason why the method described in “Real-time Fish Detection in Trawl Nets” has been chosen to build the ground-truth for Mask R-CNN.

Mask R-CNN

Mask R-CNN for Fish Segmentation

A CNN is a convolutional neural network used to extract image features in order to, for example, classify the image. R-CNN is first finding RoIs, where RoIs are Regions of Interests, in the input image, and then it is runing a CNN on each RoI. Some algorithms are applied on each output of CNN to classify the RoI and to find the

bounding-box, if existing. Fast R-CNN is using only one CNN, shared for all the RoIs, for feature extraction. The locations of the RoIs in the input image are computed by backpropagation. Faster R-CNN is not using an external algorithm to compute the RoIs, and this task is done by the CNN. The authors of the approach included in the CNN an RPN able to extract the RoIs. Previously, the features extraction was done twice, first for the RoIs and second for classification and computation of bounding boxes. Currently, it is done only once. Mask R-CNN is Faster R-CNN plus a branch to compute the masks. This new branch is parallel to the branch existing for classification and bounding box regression. The mask branch is a small FC network applied to each RoI. A mask is built by classifying each pixel of a RoI as *0* if it corresponds to non-object and as *1* if it corresponds to object. Then, all the pixel values equal to 1 takes the label of the corresponding RoI. The authors tested mask R-CNN with two architectures. One is a CNN made of an RPN and a ResNet network. The other one is a CNN made of an RPN and a FPN. Mask R-CNN with a FPN gives better results than Mask R-CNN with ResNet.

Creation of the Ground-truth

In [3], the illumination pattern is computed and removed from all the images. Then, fish are segmented using a difference ratio criterion between an estimated background and the image. We added to the algorithm a small piece of code to compute the masks of the segmented fish using the OTSU method. For each new datasets, we fine tuned the parameters of the algorithm. For each image, if the algorithm detects fish, it outputs the input image, the segmented fish and the mask (or masks) of the segmented fish. We applied this algorithm on different datasets, selecting the well-segmented images to build the ground truth. We added to the ground-truth only the images where the whole fish was segmented. We ensured that both the body of the fish and the fins were segmented with high accuracy. Fins are very important to enable the recognition the fish, although they are hard to segment since they often appear translucent in the images. Furthermore, we did not keep images where the fish show artifacts as a side-effect of the compensation of the non-uniform illumination. Small fish have not been segmented because they are often translucent and their sizes are quire reduced. Even for the human eye, it is generally too hard to segment these small-sized fish correctly. We organized the data in three different groups: images with overlapping fish, images with no fish in the border, and images with small fish not segmented. We want to know, for each of these groups, the consequences of adding them in the ground-truth on the quality of the results. Specially, we want to know if we can avoid to add overlapping fish in the ground-truth to simplify the creation of further ground-truths, intended, for instance, for species classification. We ended up with only *1658* high-quality segmented fish coming from the automatic algorithm, thus we performed manual segmentation. To segment precisely the fish, we used a touch computer to select a fish with a pencil. The high resolution of the computer (*2160 x 1440*) allowed for very fine segmentation. The manual segmentation has to be fast, so we implemented a MATLAB® GUI with simple and efficient functionalities to modify easily the segmented images and the corresponding masks. To be able to differentiate merged fish, we stored the masks in different grey levels, so that each fish corresponds to a mask with a different label. The Table 1 shows the current number of input images for each category. This Table would be updated as soon as the ground-truth datasets grows.

	Hand selected	**Manually segmented**	**Total**
All images	1658	656	2314
Images with small fish	242	0	242
Images with no border fish	551	80	631
Images with overlapping fish	0	22	0

Table 1. Current number of input images for each category.

Implementation of Mask R-CNN

In a first step, we created a dataset readable by the Mask R-CNN algorithm. To realize this task, we took the COCO dataset 2014 and we modified it by adding the information related to our own dataset and by removing the information concerning to any other class. The authors of Mask R-CNN included the Mask R-CNN code in Detectron, a Facebook AI Research's software system written in Python and powered by Caffe2. The research team provides a Dockerfile to access to all the tools required by Detectron, which is proposing several configuration files. We first used a network using ResNet with *50* layers. We trained then Mask R CNN on *14* input images and with *1000* iterations. This first trial took around eight minutes to train the network We finally reduced the number of ROIs per images from *512* to *14*, and we decreased the learning rate from *0.1* to *0.005*. At this step, the system was running but the output images contained multiple small masks for each fish. Later, we tried to change the anchors sizes.

After a small study of the fish size, we changed the area and the aspect ratio of the anchors. Then, the results became much better, but not good enough. We tried to train Mask R-CNN with an FPN of depth *50*, but the results did not improve significantly. In case of an "out of memory" situation, we can reduce the number of RoIs per images, and the number of images (per GPU) in the training mini-batch, but then it is needed have to increase the number of iteration and learning rate should be decreased. We set the parameter "scale" to *1392* instead of *800* to do not resize the input images, in order to increase the accuracy. We choose to pick automatically values of aspect ratio at regular intervals from the minimum to the maximum value of the aspect ratios. We applied the same criterion with the area values. Changing the anchor sizes and the rescale did not solve the error.

Finally, we cleaned and modified a bit the MATLAB® codes to created a new dataset readable by Detectron, and we used another configuration including data augmentation in the testing part. Additionally, we

decreased the learning rate parameter from *0.01* to *0.005*. This learning rate is multiplied by a coefficient at different values of iterations. We modified the different values of iterations, and we used two GPU instead of one. We called this las configuration as “Final_Model”. After having trained the “Final_Model”, we got good results by testing it on a new dataset.

Results

We trained the “Final_Model” using all the ground-truth images that we had at the time, that is, *1872* images. *80%* of the ground-truth images were used for training while the resting *20%* were used for the validation. The results are shown in Table 2.

```
(AP) @[IoU=0.50:0.95 | area =  all   | mD=100] = 0.582
(AP) @[IoU=0.50      | area =  all   | mD=100] = 0.708
(AP) @[IoU=0.75      | area =  all   | mD=100] = 0.673
(AP) @[IoU=0.50:0.95 | area = small  | mD=100] = 0.378
(AP) @[IoU=0.50:0.95 | area = medium | mD=100] = 0.704
(AP) @[IoU=0.50:0.95 | area = large  | mD=100] = 0.641
(AR) @[IoU=0.50:0.95 | area =  all   | mD=  1] = 0.664
(AR) @[IoU=0.50:0.95 | area =  all   | mD= 10] = 0.719
(AR) @[IoU=0.50:0.95 | area =  all   | mD=100] = 0.720
(AR) @[IoU=0.50:0.95 | area = small  | mD=100] = 0.484
(AR) @[IoU=0.50:0.95 | area = medium | mD=100] = 0.801
(AR) @[IoU=0.50:0.95 | area = large  | mD=100] = 0.773

Task: box
AP,      AP50,    AP75,    APs,     APm,     APl
0.3874,  0.7054,  0.2805,  0.2672,  0.5783,  0.4238

Task: mask
AP,      AP50,    AP75,    APs,     APm,     APl
0.5817,  0.7081,  0.6732,  0.3782,  0.7038,  0.6408
```

Table 2. COCO-evaluation results for instance segmentation of “Final_Model”.

Average Precision (AP) and Average Recall (AR) of bounding-boxes and masks are higher for medium-sized fish, a bit lower for large-sized fish, and lower for small-sized fish. When the number of fish detected by mini-batch is large, the global AR value is a bit higher. Global AP is ranging only from *0.673* to *0.708* while IoU threshold is rangin from *0.75* to *0.50*. The global AP is equal to *0.582* when the IoU value is between *0.50* and *0.95*. The reason for the AP and AR to be fish-size dependent may be the fact that the medium-sized fish are the ones with more samples in the training datasets, while this number is lower for the other fish sizes.

Currently, the obtained results are not precise enough to be able to recognize all the different types of fish, but there are anyway encouraging. The illumination correction had not been applied on the dataset used to validate the “Final_Model”, so we better when using images with the non-uniform illumination compensated are expected. The “Final_Model” is not performing well when trying to detect small fish, and the best performance occurs when middle size fish are identified.

We are currently training “Final_Model” on each of these three datasets separately, so that we will have up to three output models. Without overlapping fish in the ground-truth, the results of “Final_Model” for overlapping fish on images with no illumination compensation are not good enough, but at least the different fish specimen are appropriately identified (see Figure 3).

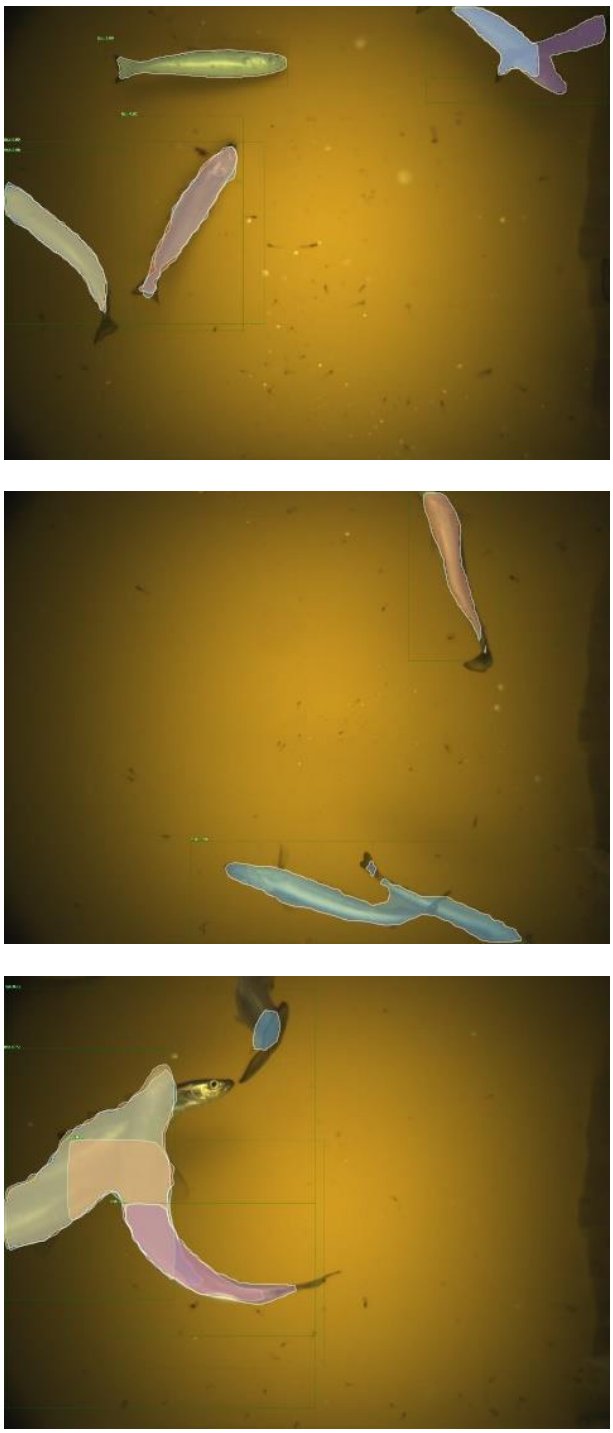


Figure 3. Qualitative results obtained by testing “Final_Model” (which is a Mask R-CNN N network with a FPN) on a new dataset.

Conclusions

In this work, we tried to improve the results obtained by the segmentation approach presented on the paper “Real-time Fish Detection in Trawl Nets”, specially in the case where several fish overlap, given that the algorithm is not able to deal with this situation. Increasing the robustness of the method would allow further steps such as fish tracking and species recognition.

To reach this goal, we decided to use Mask R-CNN, because it is fast and accurate, and performs instance segmentation. We had to build the ground-truth to train

the Mask R-CNN. The outputs of [3] where used to create the initial ground-truth. From this first output, we had to select fish correctly segmented. Manual segmentation has been also performed to increase the ground-truth. Mask R-CNN using Detectron has been finally applied, and we have been able to train it using 1872 images.

The results obtained during this thesis are encouraging, but they are not currently accurate enough to enable fish recognition in a future steps. The main reason explaining this performance is the lack of an appropriate amount of labeled data. Mask R-CNN does not achieve good enough results in this situation, but it has proved to be able to identify and differentiate two or more different fish in some of the tested images.

We are currently in the process of increasing the size of the training datasets by performing manual segmentation in cases where overlapping fish appear, to be able to train and test Mask R-CNN with the largest possible number of images, and increase consequently the quality of the results. After this work, much better results are expected.

Bibliography

[1] “Commission Delegated Regulation (EU) 2015/2438 of 12 October 2015 establishing a discard plan for certain demersal fisheries in north-western waters”. In: Official Journal of the European Union (Oct. 2015). URL:http://eurlex.europa.eu/legal-content/EN/TXT/

[2] Thane A. Militz et al. “Fish Rejections in the Marine Aquarium Trade, An Initial Case Study Raises Concern for Village-Based Fisheries”. In: (March 2016). URL: http://journals.plos.org/plosone /article?id=10.1371/journal.pone. 0151624.

[3] Real-time Fish Detection in Trawl Nets - R. Prados, R. García, N. Gracias, L. Neumann, Håvard Vågstøl, June 2017

[4] Arthur Ouaknine « Review of Deep Learning Algorithms for Object Detection » Feb 5 2018 https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

[5] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, January 2016

[6] You Only Look Once: Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (Submitted on 8 Jun 2015 (v1), last revised 9 May 2016 (this version, v5))

[7] R-FCN: Object Detection via Region-based Fully Convolutional Networks

Jifeng Dai - Yi Li - Kaiming He - Jian Sun, 21 June 2016

[8] SSD: Single Shot MultiBox Detector – Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg (Submitted on 8 Dec 2015 (v1), last revised 29 December 2016 (this version, v5))

[9] Mask R-CNN Kaiming He - Georgia Gkioxari - Piotr Dollár - Ross Girshick, 24 January 2018

[10] NASNet : Learning Transferable Architectures for Scalable Image Recognition - Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le (Submitted on 21 Jul 2017 (v1), last revised 11 April 2018 (this version, v4))

Classification of pattern algorithm for novel microwave sensor systems

Author : Despoina Melidoni – despoina.melidoni@gmail.com

Supervisors: Professor Desmulliez Marc - m.desmulliez@hw.ac.uk
Dr Altmann Yoann - y.altmann@hw.ac.uk
Dr. Pavuluri Sumanth Kumar - sumanth_kumar.pavuluri@hw.ac.uk

Abstract Food industries require the development of sensor that provide continuous measurement and monitoring of the quality of food products. Microwave sensors are a widely known technology that allows the evaluation and analysis of food products. However, this technology is mostly applied on lab conditions, which differs a lot from those in food industries. A solution to this need is provided by the FoodSense sensor system, invented and developed by "MicroSense Technologies" Ltd . The purpose of this research is to analyze the data collected by the sensor and use classification methods to train models that could optimally recognize and correctly label the provided data to the corresponding recipe. The classification methods used for this project are Support Vector Machine (SVM), Decision Trees and K-Nearest Neighbor (K-NN).

***Index Terms* — classification, decision trees, foodsense, k-nearest neighbor, microwave sensor, support vector machine**

I. Introduction

The quality and composition of food products are of significant importance during all stages of a food production flow line. But what is considered "good quality"? Based on the paper [1], quality is defined as "The distinctive trait, characteristic, capacity or virtue of a product that sets it apart from all others". Taking this into consideration, and with the knowledge that the consumer's awareness is high in today's social and economic environment, obliges food manufacturing companies to have great standards regarding the quality of their products. The technology of microwave sensors is an accurate fit to fill this need. Many researches on microwave techniques were performed through the last years, resulting in practical implications in pharmaceutical, oil and food industries [2] [3] [4] [5].

More precisely, the dielectric-based sensors have shown to provide better results, due to the fact that most food products have enough water concentration, that combined with water's strong interaction with the electric field at microwave frequencies, can render even better results [6]. Different food compositions have distinct dielectric properties, such as permittivity and conductivity. These properties affect the reaction of each food composition to an external electromagnetic field.
This technology offers real-time monitoring, is non-destructive, contactless and non-invasive, in contradiction to other complex chemical procedures. A statement of the basic principle behind this kind of sensor was given by the authors of this article [6]. "Depending on the type and amount of a chemical ingredient in the liquid under test, a variation of the complex dielectric permittivity is produced. This perturbation of the dielectric permittivity affects the electromagnetic response of the sensor, thus allowing to obtain information about the composition of the solution."

Generally, microwave measurements can be categorized into frequency and time domain measurements. The first method determines the permittivity by calculating the transmission or reflection coefficient. On the other hand, the method using time domain frequencies, traces the time domain waveform and then can either compare it with the one before the measurement or apply Fourier-Transform (FT) and follow the frequency domain method [7].

There are many techniques available to analyze the dielectric properties of liquids in the frequency domain that was mentioned earlier. Some of these techniques use a wide range of frequencies [8] [9] [10], while others, using resonant perturbation methods, provide results with more accuracy [11] [12] [13] [14].

II. Processing of raw data from sensor

A. FoodSense Sensor System

The microwave sensor system that was used for the collection of data in this project is called "FoodSense Sensor Electronic System" (Figure 1). It was invented by "MicroSense Technologies" Ltd [15], which is a spin out company from Heriot Watt University [16]. The sensor provides precise and real-time monitoring and beneficial intelligence to running processing pipelines of food manufacturing industries, that lack online measuring

systems. Currently the sensor is installed in two food production flow lines, of two different companies. In order to make a correct analysis of the data collected by the sensor, both food companies, submitted the overview of the process they follow during the time the sensor was working. This schedule overview is a sheet containing information regarding the date and time of each product processing period and how the procedure moved from one product to the next. It also contained information about the cleaning cycles that occurred in between some productions of recipes

Figure 1: FoodSense Sensor Electronic System

B. *Initial Data Analysis*

The first step was to process the data separately for each recipe, to assess their integrity in comparison with the schedule overview provided by the companies. The schedule overview is a sheet containing information regarding the date and time of each product processing period and how the procedure moved from one product to the next. It also contained information about the cleaning cycles that occurred in between some productions of recipes. During this part, some irregularities in the signal were noticed, that later, using the variance and the second order derivatives of the data as a whole, concluded that they were caused by existing outliers in the data. Afterwards, using linear interpolation these outliers were removed. Furthermore, in order to achieve better results during the procedure of classification, the data of each recipe had to go through the process of PCA.

III. CLASSIFICATION OF DATA

Following the results of PCA, the data from all the recipes were combined together in one matrix, where was added an extra column at the end, that contained the information of which measurement belonging to what recipe. For example, the rows of the matrix that contain the measurements from recipe 26, are labeled at the final column as class 1, where the rows containing the measurements for recipe 51, are labeled in the final column as class 2. The complete mapping for the correlation between recipes and labels/classes, is shown in Figure 2.
This matrix was used in the following classification methods.

Class	
1	Recipe 26
2	Recipe 51
3	Recipe 39
4	Recipe 31
5	Recipe 35
6	Recipe 37

Figure 2 : Mapping between labels/class and recipes

A. *Support Vector Machine (SVM)*

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. An SVM performs very similar to a human brain regarding pattern classification, being able to recognize and distinguish an object by its features [17] [18] [19]. During SVM's recognition procedure, a hyperplane is formed that separates two classes with a maximum distance. The closest samples to the hyperplane are called support vectors.
As seen in Figure and *Figure* , the data are classified correctly in almost all cases. The overall accuracy of the SVM classification model was 92%. The significant errors are found in class 6, where as the confusion matrix in *Figure* verifies, 45% of the data belonging in class 6 were classified incorrectly in class 3. In a similar way, but not at the same extend, 4% of the class's 3 data were wrongly labeled as class 6. Moreover, 2% of class's 1 data were falsely labeled as 5 and respectively 6% of data belonging in class 5, were classified as class 1.

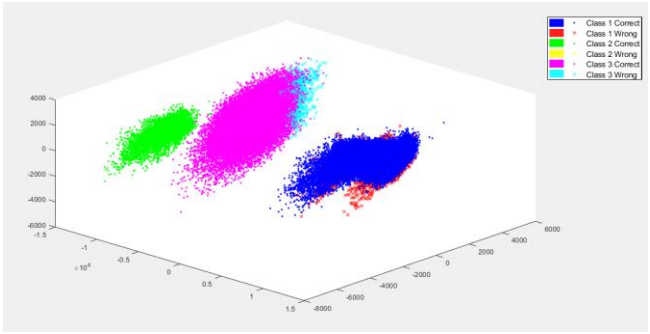


Figure 3: SVM Model – Classes 1 , 2 , 3

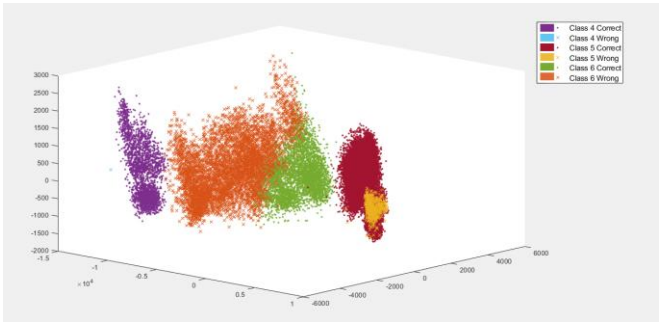


Figure 4: SVM Model – Classes 4 , 5 , 6

There is an apparent correlation between classes 1 and 5, and 3 and 6. Another look at the schedule overview reveals that in both cases, there is a transition from one recipe to another, without a cleaning cycle in between, which potentially causes these errors in the classifier model. Moreover, there is a need for more calibration data provided by the sensor, but currently there is a lack of them. This potentially could cause the errors at the classification model.

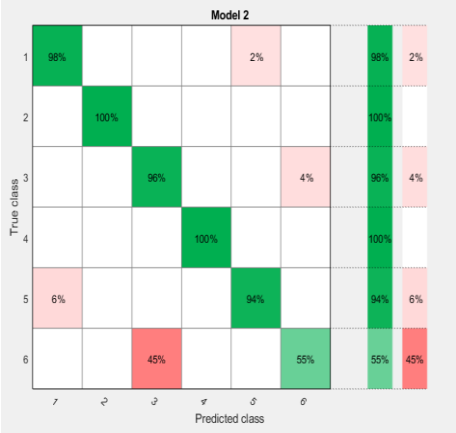


Figure 5: Confusion Matrix of SVM Model

B. *Decision Trees*

Decision Tree Classifier is a simple and widely used classification technique [20] [21] [22]. It applies a straightforward idea to solve the classification problem. Decision Tree Classifier poses a series of carefully crafted questions about the attributes of the test observation. Each time it receives an answer, a follow-up question is asked until a conclusion about the class label of the observation is reached. *Figure 2* shows an example of a decision tree [23].

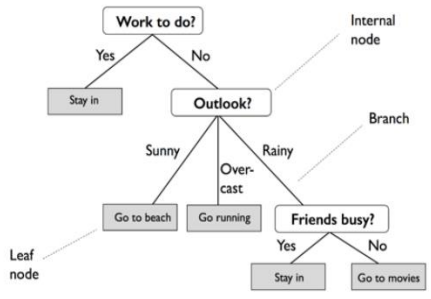


Figure 2: Decision Tree Example

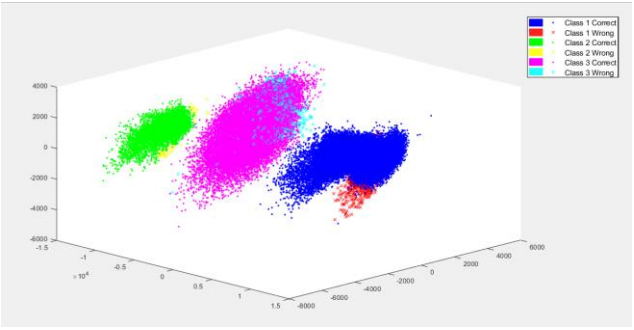


Figure 3: Decision Tree Model – Classes 1 , 2 , 3

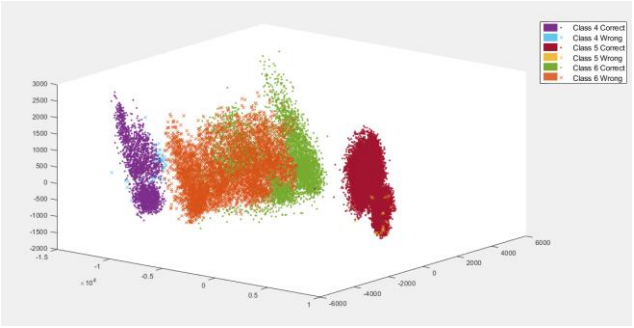


Figure 4: Decision Tree Model – Classes 4, 5 , 6

The results of the Decision Tree model have similarities with those of the SVM model, but with better overall accuracy at 95%. Again, a significantly high percentage of class's 6 data were incorrectly classified as class 3. And vice versa but with lower percentage. The correlation between classes 1 and 5, and 3 and 6 are observed again (Figure 9). In addition, more associated classes can be found, such as 2 and 3, 2 and 4, 3 and 4. Look back again at the schedule overview, there is indeed a time when transitions between the respective recipes occurred, without a cleaning cycle after each one.

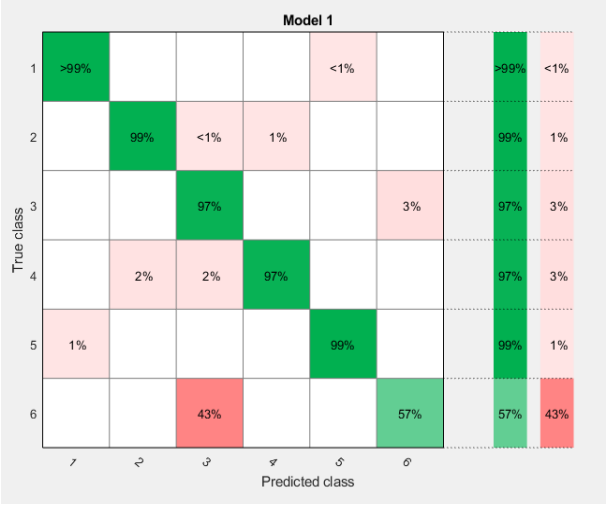


Figure 5: Confusion Matrix of Decision Tree Model

C. *K-Nearest Neighbor (K-NN)*

K-NN is a non-parametric method used for classification and is considered as one of the simplest and thus most common machine learning algorithms. All the training data are used during the testing phase and an object is classified by a majority vote of its neighbors, with the object being assigned to the class, that is the most common among its *k* nearest neighbors. If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor. There are various methods to calculate the distance between the neighbors, with the most popular being the Euclidean distance,

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

, where $p = (p_1, p_2,..., p_n)$ and $q = (q_1, q_2,..., q_n)$ are two points in Euclidean n-space, that is the set of all the ordered n-tuples and it is denoted by R^n.

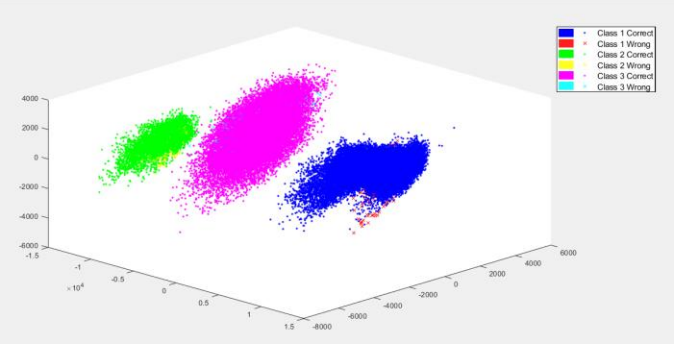


Figure 7: K-NN Model – Classes 1 , 2 , 3

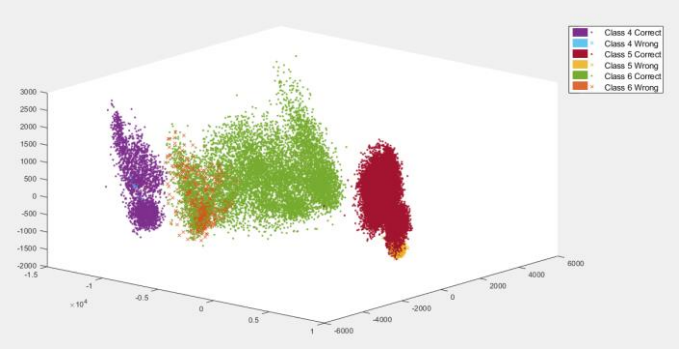


Figure 8: K-NN Model – Classes 4 , 5 , 6

The K-NN model gives much better accuracy results (97.4%) compared to the SVM and Decision Tree models. The classification of the data was completed with higher precision and with less incorrect data points (*Figure 7*, *Figure 8*). From the PCA results, the data appeared to be clearly distinguishable and easily separable, thus the K-NN method was expected to have the best results, since the classification of its point is based on its neighbor. The confusion matrix of the K-NN model (*Figure 6*) reflects the same conclusion. The correlations of classes 1 and 5, 3 and 6 are observed again (as with SVM and Decision Trees models) with the addition of 2 and 4 that was found in previous model as well, inferring the same conclusion as before.

Figure 6: Confusion Matrix of K-NN Model

IV. Conclusion

The aim of this master thesis was to analyze and subsequently to classify the data collected using the FoodSense sensor system. The first step was to process the data separately for each recipe, to assess their integrity in comparison with the schedule overview provided by company A. During this part, some irregularities in the signal were noticed, that later, using the variance and the second order derivatives of the data as a whole, concluded that they were caused by existing outliers in the data. Afterwards, using linear interpolation the outliers were removed. Following that, three methods of classification were used, Support Vector Machine (SVM), Decision Trees and K-Nearest Neighbor (K-NN), where the latest provided the highest accuracy (97.4%). In all methods existed some consistent correlations between some recipes that caused errors to appear in the classification models, that as it was observed can be attributed to transitions from one recipe to the next, without a cleaning cycle in between. Also, the current lack of additional raw data may be affecting these results.

A. *Recommendations*

There are areas of this project that could benefit from more research, that could not be accomplished during the time period of this master thesis. Other outlier detection methods could be tried and compared to find the one providing the optimal results. The same applies for the method used for the linear interpolation, were other methods could be used to remove the outliers in a faster and more efficient way. In conclusion, more classification methods could be tested to

search for one resulting in higher validation accuracy and thus enabling the creation of a real-time recognition and classification system. Moreover, the use of additional raw data collected from the sensor could provide better overall results in the classification process.

ACKNOWLEDGMENT

I would like to express my deep gratitude to my master thesis supervisor, Professor Desmulliez Marc for his enthusiastic encouragement, his useful critiques on this research and his assistance in keeping my progress on schedule. This work would not have been possible without his continuous support and the environment he provided to work at the Heriot-Watt University. I am also particularly grateful for the assistance given by Dr Altmann Yoann and Dr Pavuluri Sumanth Kumar for their patient guidance, their valuable and constructive suggestions during the planning and development of this research work. Their willingness to give their time, outside their working hours, has been very much appreciated as well as, their immense knowledge that assisted me during the time of research and writing of this thesis.

Last but not the least, I would like to thank my parents, whose love and guidance are with me in whatever I pursue. My special thanks are extended to my siblings and my life partner that supported me throughout my studies.

REFERENCES

[1] M. Ferree, "What is Food Quality," Food Distribution Research, pp. 36-39, February 1973.

[2] K. Saeed, A. .. Guyette, I. C. Hunter and R. D. Pollard, "Microstrip resonator technique for measuring dielectric permittivity of liquid solvents and for solution sensing," in Proc. IEEE Int. Microw. Theory Tech.Soc. Symp., 2007.

[3] K. Joshi and R. Pollard, "Sensitivity analysis and experimental investigation of microstrip resonator technique for the in-process moisture/permittivity measurement of petrochemicals and emulsions of crude oil and water," in Proc. IEEE Int. Microw. Theory Tech. Soc. Symp. Dig., 2006.

[4] K. H. Theisen and T. Diringer, "Microwave concentration measurement for process control in the sugar industry," Proc. SIT, vol. 60, pp. 79-92, 2000.

[5] O. L. Bo and E. Nyfors, "Application of microwave spectroscopy for the detection of water fraction and water salinity in water/oil/gas pipe flow," Non-Crystalline Solids, vol. 305, pp. 345-353, 2002.

[6] S. Trabelsi and S. O. Nelson, "Microwave sensing of quality attributes of agricultural and food products," IEEE Instrumentation & Measurement Magazine, pp. 36 - 41, 21 January 2016.

[7] Z. Meng, Z. Wu and J. Gray, "Microwave Sensor Technologies for Food Evaluation and Analysis – Methods, Challenges and Solutions," Transactions of the Institute of Measurement and Control, 20 September 2017 .

[8] J. M. Anderson, C. L. Sibbald and S. S. Stuchly, "Dielectric measurements using a rational function model," IEEE Trans. Microwave Theory Tech., vol. 42, pp. 199-204, February 1994.

[9] M. D. Migliore, "Partial self-calibration method for permittivity measurement using a truncated coaxial cable," Electron. Lett, vol. 36, pp. 1275-1277, 2000.

[10] D. Misra, M. Chabbra, B. R. Epstein, M. Microtznik and K. R. Foster, "Noninvasive electrical characterization of materials at microwave frequencies using an open-ended coaxial line: Test of an improved calibration technique," IEEE Trans. Microw. Theory Tech., vol. 38, pp. 8-14, January 1990.

[11] K. Saeed, R. Pollard and I. C. Hunter, "Substrate integrated waveguide cavity resonator for complex permittivity characterization of materials," IEEE Trans. Microw. Theory Tech., vol. 56, pp. 2340-2347, October 2008.

[12] U. Raveendranath, S. Bijukumar and K. Matthew, "Broadband coaxial cavity resonator for complex permittivity measurements of liquids," IEEE Trans. Instrum. Meas., vol. 49, pp. 1305-1312, December 2000.

[13] L. F. Chen, C. K. Ong, C. P. Neo, V. V. Varadan and V. K. Varadan, Microwave electronics: Measurement and materials characterization, New York, USA: Wiley, 2004.

[14] R. Inoue, Y. Odate, E. Tanabe, H. Kitano and A. Maeda, "Data analysis of the extraction of dielectric properties from insulating substrates utilizing the evanescent perturbation method," IEEE Trans. Microw. Theory Tech., vol. 54, pp. 522-532, February 2006.

[15] "MicroSense Technologies," [Online]. Available: http://microsensetechnologies.co.uk/.

[16] "Heriot Watt University," [Online]. Available: https://www.hw.ac.uk/.

[17] R. Brereton and G. Lloyd, "Support vector machines for classification and regression," Analyst, vol. 135, no. 2, p. 230–267, 2010.

[18] S. Ari, K. Hembram and G. Saha, "Detection of cardiac abnormality from PCG signal using LMS based least square SVM classifier," Expert Syst. Appl., vol. 37, no. 12, p. 8019–8026, 2010.

[19] T. Wu, S. Huang and Y. Meng, "Evaluation of ANN and SVM classifiers as predictors to the diagnosis of students with learning disabilities," Expert Syst. Appl, vol. 34, no. 3, p. 1846–1856, 2008.

[20] Y.-Y. Song and Y. Lu, "Decision tree methods: applications for classification and prediction," Shanghai Arch Psychiatry, vol. 27, no. 2, p. 130–135, 25 April 2015.

[21] C. C. Aggarwal, Data Classification: Algorithms and Applications, CRC Press, 2015.

[22] O. Z. Maimon and L. Rokach, Data Mining With Decision Trees: Theory And Applications, World Scientific Publishing, 2015.

[23] S. Raschka, Python Machine Learning, Packt Publishing Ltd, 2015.