TAGME

HOSPITAL TAG RECOGNITION SYSTEM

TECHNICAL REPORT

Prepared by
Vejey Subash Gandyer

OBJECTIVE

The objective is to design and develop a foolproof Hospital Tag Recognition system that can be deployed in mobile applications or web applications.

SIGNIFICANCE / NOVELTY

Novelty lies in the Utility factor as we will be facilitating the patients or doctors or hospital staff to automatically scan patient tags and identify their personal identification scheme of their own hospital tags issued by the hospitals.

INTRODUCTION

This TagMe project requires 3 stages namely

1. Localization stage
2. Pre-processor stage
3. Recognizer stage

1. Localization stage

In Localization stage, TagModel(hid, tagtemplate1) is created for each hospital.

The localization of tag elements is done by a module called TagElementsLocalizer. This has the form

TagElementsLocalizer(Image image, HID hid)
Input: (image, hid)
Output: (hid, tagtemplate) => TagModel

TagModel list will contain the following format:

HID	TagTemplate
123	image1
777	image2
999	image3
911	image4

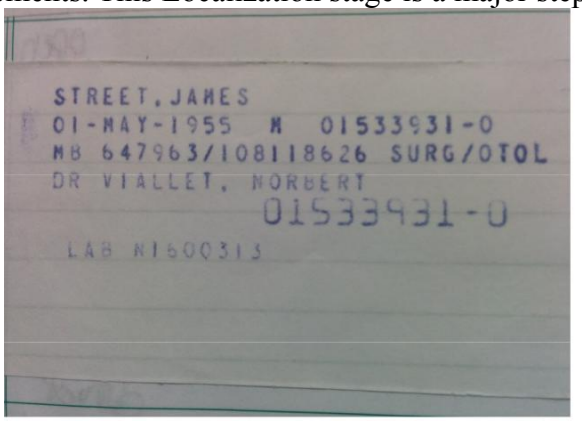
This is a Template creation stage where each hospital will send their Patient's tag image as a template with their hospital ID so that a model will be created as described in TagModel(HID with their unique template with localized tag elements with respect to each of its position it occurs in the tag).

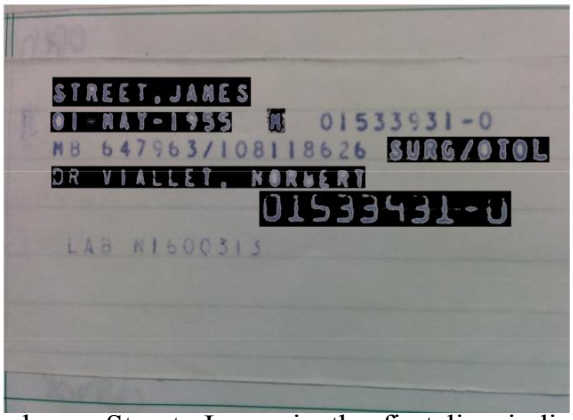
So when an API call like TagMe(image, 777) is coming, our application should immediately look up in

TagModel list and find its unique template as image2. This image2 is just not an ordinary image but it has got localization of different tag elements in that particular hospital's tag. The localization of tag elements is done by a module called TagElementsLocalizer.

The goal of this stage is to create a TagModel (list with all templates with locations localized) for storing tag templates for each hospital. Each HospitalID has got a unique template so that when an HospitalID is given as an input, the TagMe algorithm will know which image template to choose for acquiring the values from appropriate placeholders.

Illustration:
In the following Hospital's Tag Template sent as a Model, it contains the following placeholders for tag elements. This Localization stage is a major step in choosing which of the tag elements are to be





extracted and used later. In the leftmost picture above, Street, James in the first line indicates as a Patient Name. In the next line, 01-May-1955 is ofcourse the DOB field. M indicates Sex and 01533931-0 indicates MRN. In the third line, third token to the right SURG/OTOL indicates this patient belongs to which department in the hospital. The 4th line indicates Doctor Name. And the biggest font sized digits 01533931-0 indicates MRN again (redundant information). LAB n1600313 indicates which Lab.
In summary, the tag elements are as follows:

PatientName
DOB
Sex
DeptName
DoctorName
MRN
LabID

This localization stage helps in finding out these fields in the given tag. The TagElementsLocalizer module draws rectangles encompassing all the above tags as shown in the rightmost picture. As we already have apriori knowledge that each rectangle correspond to a particular field like PatientName, DOB,etc., it is essential to mark the rectangle coordinates of each of the tag elements across their TagElementsName in a file.

JSON object for TagElementsLocalizer module output:
The file has the form as explained in TagModel's image1:

```
{“HID” : “123”,
        “PatientName” : “[(x1,y1),(x2,y2)]”,
        “DOB” :“[(x3,y3),(x4,y4)]”,
        “Sex” : “[(x5,y5),(x6,y6)]”,
        “DeptName” : “[(x7,y7),(x8,y8)]”,
        “DoctorName” : “[(x9,y9),(x10,y10)]”,
        “MRN” : “[(x11,y11),(x12,y12)]”,
        “LabID” : “[(x13,y13),(x14,y14)]”
}
```

And this image1 entry has a unique ID that corresponds to HospitalID. Thus a template is created for this hospital. We have to do the same for all the hospitals in our user base. So every hospital will have a unique ID and that ID has an entry containing all the rectangular coordinates for all tag elements with its name as a JSON object. This is the most important stage in the TagMe project.

2. Pre-processor Stage

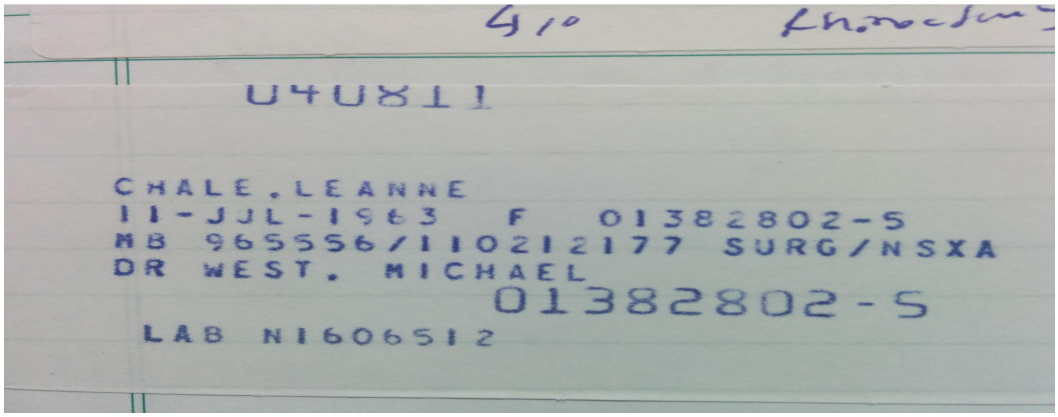
Pre-processor Stage is where pre-processing of the client's patient tag image will be cropped, aligned for further processing. This stage is the entry point of a client API call where the client will send an API call with the following format:

TagMe(Image image, HID hid)
Input: image, hid
Output: JSON object of the form

```
{“HID” : “123”, “PatientName” :
        “Chale, Leanne” “DOB” : “11-
        Jul-1963”,
        “Sex” : “F”,
        “DeptName” : “Surg/NSXA”,
        “MRN” : “01382802-5”,
        “DoctorName” : “Dr West, Michael”,
        “LabID” : “LAB N1606512”
}
```

ex: TagMe(image, 123)

where TagMe is an API that I need to expose to your client (Android and iOS mobile application), Image is the Image object and image is the actual image(captured from camera application that runs Medlinx app in the form of .png, .jpg or .jpeg as shown in **figure below**) that the client app sends to my code and HID is the HospitalID (each hospital is expected to have a single template for its patients tag as discussed in Localization stage). This uniqueness in the HID is taken advantage of in our application and at last, hid is the actual id which will have a corresponding and unique entry in the TagModel list.



Test image sent to TagMe (Uncropped, Bad Alignment)

Now TagMe algorithm will know which image template to choose for acquiring the values from appropriate placeholders for tag elements' values.

Illustration:

TagMe(image,123)

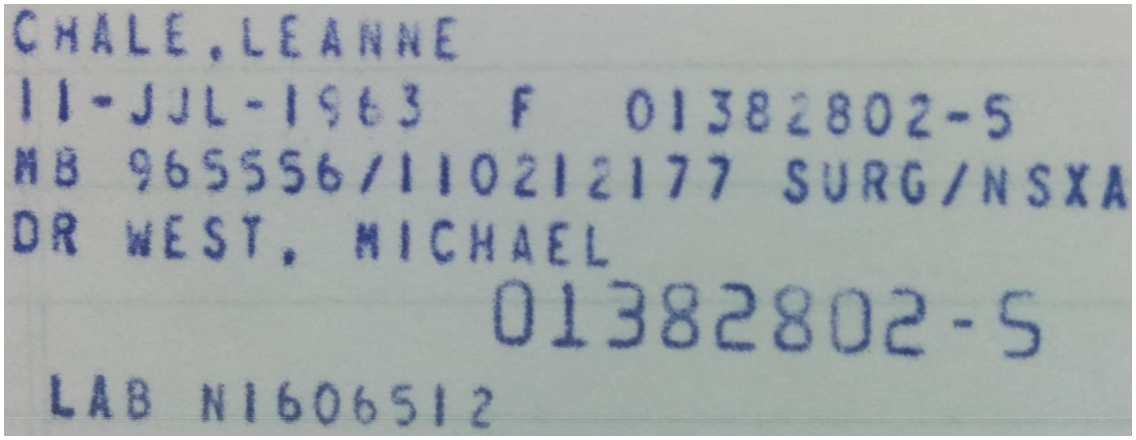
When this is the API call from the client app, TagMe algorithm does a Pre-processing of the sent image for exact cropping and alignment. After this pre-processing is done by TagMePreProcessor which takes the form,

TagMePreProcessor(image)

Input: image

Output: Cropped, Aligned image

the pre-processed image is taken for further steps.



Cropped and Aligned image after Pre-processing

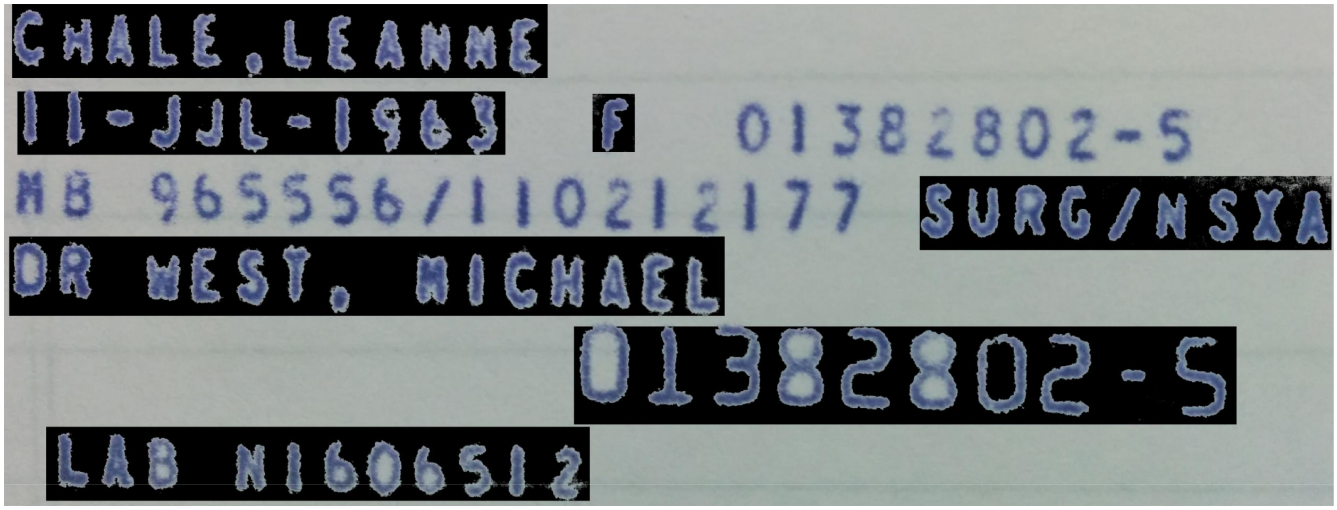
TagMe core algorithm refers TagModel table and looks up the hid value of 123. If it finds 123 in hid, it pulls that particular image template that is already stored in the localization stage for further processing.

The corresponding JSON object has the form,

```
{“HID” : “123”,
        “PatientName” : “[(x1,y1),(x2,y2)]”,
        “DOB” :“[(x3,y3),(x4,y4)]”,
        “Sex” : “[(x5,y5),(x6,y6)]”,
        “DeptName” : “[(x7,y7),(x8,y8)]”,
        “DoctorName” : “[(x9,y9),(x10,y10)]”,
        “MRN” : “[(x11,y11),(x12,y12)]”,
        “LabID” : “[(x13,y13),(x14,y14)]”
}
```

which contains the location of all the tag elements.

Now the above found rectangle coordinates will be superimposed on the client sent image as shown below.



Superimposed Rectangles for Recognition of characters

3. Recognizer Stage

Recognizer stage is where core of TagMe project's image processing algorithms rests. This is where the rectangular boxes encompassing various fields of tag elements gets recognized and information is extracted from each of the boxes that were drawn in the previous stage.

Here in this stage, OCR module will be operated on each of these rectangles in a parallel fashion and the characters are recognized and thus the tag elements' information is extracted in a quick and efficient manner.

The information extracted from the OCR module is

Chale, Leanne
11-Jul-1963
F
Surg/NSXA
Dr West, Michael
01382802-5
LAB N1606512

The output is sent in the form of a JSON object,

```
{"HID" : "123", "PatientName" : "Chale,
        Leanne" "DOB" : "11-Jul-1963",
        "Sex" : "F",
        "DeptName" : "Surg/NSXA",
        "MRN" : "01382802-5",
        "DoctorName" : "Dr West, Michael",
        "LabID" : "LAB N1606512"
}
```

Appendix I
Some Examples of Hospital Tags