Implementation and Testing of Rotational Invariant $P Recognizer (March 2019)

Vigneet M Sompura, University of Florida, vigneetsompura@ufl.edu

***Abstract*— Introduction of $-family algorithms for gesture recognition has provided an easy way for developers to create and prototype applications that need gesture recognition but do not have enough resources or expertise to implement more advanced and complicated pattern matching algorithms without compromising accuracy. Unlike earlier members of $-family ($1 [3] and $N [2]), $P ignores articulation patterns to improve the space requirements and considers gestures as point clouds. It uses the minimum distance between the candidate gesture and template gesture as a criterion to recognize gestures. $P paper [1] compares many algorithms to come up with the best algorithm in terms of speed and accuracy. After comparing many algorithms, they concluded Greedy 5 (uses Weighted Euclidean-Distance as a measure [1]) to be the best among them in terms of accuracy with 98.4% accuracy for both user-dependent tests and user-independent tests[1]. However, due to the use of point clouds instead of a sequence of points, $P is unable to recognize the gestures if it is rotated. In the present, touchscreen devices are not limited to cellphones, and many larger touchscreen devices are available that can be used for collaboration of multiple users. One such example is a touch screen table that could be used by multiple users sitting on different sides. In such a scenario, the rotational invariance of the algorithm can be an important factor. This project focuses on the extension of the $P algorithm to make it rotational invariant.**

I. Introduction

This project aims to extend $P gesture recognition algorithm[1] presented by Dr. Anthony, Dr. Vatavu and Dr. Wobbrock such that it is able to recognize the gesture even if it is rotated while also correctly recognizing the gestures that are natural rotations of each other. To test the algorithm a custom dataset was collected from 3 users that contains a mix of unistroke and multistroke gestures. The gestures were chosen to support testing of rotational invariance with the ability to discriminate gestures that are natural rotation of each other. The paper presents the extended $P algorithm that allows for rotational invariance by including an extra step in preprocessing of the original $P algorithm[1]. Later, the gesture is recognized using a similar method as $P recognizer with minor modification required to accommodate the changes in the preprocessing step. The project includes a JavaFX based desktop application to demo recognition of gesture drawn using a mouse on the canvas as well as perform user dependent random 100 tests on the said dataset.

II. Dataset Details

Since the purpose of the algorithm is to extend $P recognizer[1] such that it allows for rotational invariance while successfully distinguishing between gestures that are natural rotations of each other, the MMG dataset used in the original paper wasn't sufficient for testing. Thus, a new gesture dataset was collected that includes the pairs of gestures that are natural rotations of each other (i.e. plus and X, square and diamond, eight and infinity) as well as the gestures that should be recognized as the same gesture regardless of rotation (i.e. arrow, asterisk).

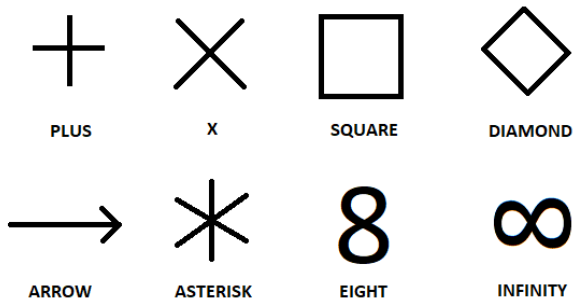

fig 1. Dataset used to test the algorithm

The dataset contains both unistroke and multistroke gestures of 8 Gesture types drawn by 3 different users. The users were all between age 24 to 26 with the average age of 25 and were enrolled in the graduate program of computer science at the University of Florida. Two of the users were male while one was female. Each user was asked to draw each gesture 10 times using the mouse on the desktop application used to collect the gesture information and save it as an XML file. To make sure the fatigue or boredom does not affect the articulation of specific gesture type, the gestures were collected in a random order. The users were told to draw the gesture shown as an image in the desktop application on nearby canvas. Since the mouse is a difficult input medium for most users for drawing gesture, the users were allowed to clear the gesture and retry if they wanted to before they submit the gesture.

Overall, 240 samples of 8 gestures were collected from 3 users with 10 repetitions each. While the testing was performed on the entire data set, the LiveDemo app uses random 8 gestures from random users for recognition.

.

III. IMPLEMENTATION DETAILS.

Predecessors of $P recognizer[1] ($1[3] and $N[2]) solved the rotational invariance by rotating the gestures by the indicative angle of the starting point. However, since the $P uses the gestures in the form of point clouds instead of a sequence of points, we do not have the notion of starting point or indicative angle. Thus, the main challenge of extending the $P recognizer was to find the rotation of the candidate that matches the template. One possible way is to check for all the possible rotations of the candidate to match with the given template. However, this process would increase the recognition time drastically and thus a better way to match the rotations of template and gesture was needed. The proposed extension in this paper uses the property of bounding box to resolve the issue. The solution lies in the fact that the bounding box depends only on the points on the extreme left/right/top/bottom (i.e. minimum and maximum X and Y coordinates of all the points in a gesture). This property of the bounding box alters the area of bounding box with the rotation. Thus, we can leverage this fact to match the rotation of candidate and template by rotating both in such a way that the area of the bounding box is minimum. Thus, the suggested algorithm performs one additional step of preprocessing than the original $P algorithm[1], rotating the gesture such that the area of the bounding box is minimum after the regular preprocessing steps of $P algorithm[1] (resampling, scaling, translation).

Since the 90° rotations of a rectangle have the same area, there will be 4 possible rotations such that the area of bounding box is minimum in general case of gesture (excludes certain gestures such as circle, hexagon, octagon, asterisk, etc. but the algorithm will still be correct due to symmetry in such shapes). Thus, to solve this issue, the algorithm checks for 4 possible rotations of normalized candidate gestures (90° rotations of each other) against the normalized template. This makes sure that one of the rotations will match the template if both candidate and template gesture are of the same gesture type solving the problem of rotational invariance.

However, the process above would fail to discriminate if the rotated gesture conflicts with another gesture in the case of gestures that are natural rotations of each other. To solve the issue, the rotation needs to be penalized such that the cloud distance of the candidate to a template that is matched after a higher amount of rotation is higher than the candidate to a template that is matched with small rotation or no rotation. The suggested algorithm tackles this issue by adding weighted rotational penalty while computing the cloud distance between candidate and template. To accommodate this change, the preprocessing step stores the amount of rotation that produced the bounding box with minimum area in the template and adds the weighted difference of rotation (in radians) between normalized template and normalized candidate to the cloud distance used in $P algorithm[1] which acts as a penalty for rotation used to differentiate between gestures that are natural rotations of each other. The weight at which the rotation should be penalized depends on the gesture types used in the dataset, thus it needs to be recomputed if the new gesture type is added. After user dependent testing the weight of 0.04 was found to achieve best recognition accuracy. (See appendices for updated/additional methods in preprocessing and formulas for computing cloud distance)

This implementation uses Java/JavaFX to for recognition algorithm and user interface for demo. The project mainly consists of 2 packages (i.e. dollarp and utilities). The package dollarp contains code and classes related to presenting a user interface for a demo while the package utilities consists of classes related to background processes for recognition of templates including main algorithm and methods parsing/preprocessing XML data.

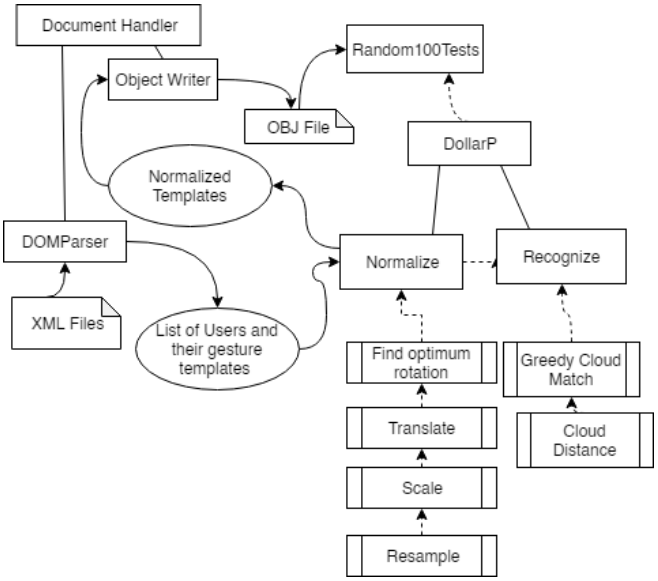

fig 2. System Architecture

A. *dollarp*

The package dollarp contains classes related to presenting a user interface for recognition. This package is mainly written using JavaFX a UI library provided with JAVA. The package contains 3 files.

I. FXMLMain.fxml
This file contains the layout of the user interface. The user – interface contains a canvas, where multistroke gesture can be drawn using the mouse, input buttons to start recognition, clear the canvas as well as a button to run user dependent tests.

II. FXMLMainController.java
This file contains the class for the controller of application. This class contains methods for each UI event including click on buttons, methods to handle and store gestures drawn on a canvas and method to invoke background recognition tasks.

III. Main.java
The main purpose of this file is to initialize the application and read dataset object if present in the same directory as the .exe file or generate the object file based on XML dataset.

B. utilities

The package utilities contains classes related to background processes required to recognize a gesture. This includes the container classes for gesture objects, class to perform recognition and class for preprocessing/ reading data from either a previously stored preprocessed object or raw XML data from MMG dataset. It also includes a class to perform user-dependent random 100 tests which is multithreaded to improve the execution time. The summary of all the classes included in this package is given below. (Full documentation is provided with source code.)

I. Point

It is a container class to represent a point in the gesture. It includes the fields to store X, Y coordinates as well as strokeNo of the specific point. Additionally, it provides methods to access those fields. Some of the method signatures are as follows.

int getStrokeID()
double getX()
double getY()
void setStrokeID(strokeId)
void setX(X)
void setY(Y)

II. Template

It is a container class to represent a gesture template or a gesture. It includes the fields to store ID, type, list of points, and angle of rotation. It provides methods to access those fields along with a method to clone the gesture for consistency during concurrent execution. Some of the method signatures are as follows.

int getID()
ArrayList<Point> getPoints()
void printInfo()
Template clone()
void addPoint(Point)

III. User

Since user dependent testing required to separate data of each user, this class serves as a container for gestures for each user. Each object of the class represents one user and contains a user id and a map of gesture type to list of templates. The class provides the following methods in addition to standard getter setter methods.

void addTemplate**(**Template**)**

IV. Result

Result class stores the result acquired through recognize() method of DollarP class. The class contains the score and template reference of the best match as well as NBest List (Top 20 only) and includes standard getter/setter methods for those fields.

V. DataHandling

It is a supporting class providing utilities for converting the data to the appropriate format required by the application. This class includes the methods to read data from a set of XML files or a previously preprocessed serialized Object file as well as storing an XML file after preprocessing it to a serialized object file which could be read later to save execution time on redundant processing. Some of the method descriptions are as follows.

Document getXMLDoc(File)

takes a file object and returns a parseable document.

ArrayList<User> parseXML(path)

takes string path pointing to dataset folder containing a set of XML files and converts it to a List of User objects each containing templates drawn by a specific user.

ArrayList<User>
preprocessXML(path, samplingRate)

takes string path to dataset folder and sampling rate and returns a List of User object as above but each template is normalized(resample → scale → translate) with the given sampling rate.

Object readObject(path)

returns an object by reading a stored object at the given path.

void writeObject(Serializable, path)

writes a serializable object to given path.

VI. DollarP

It is a supporting class providing utilities for recognition using extended $P algorithm. This class includes the methods to perform operations required for the suggested extension of $P algorithm including normalization, resampling, scaling, translation, rotation, finding template with minimum bounding box, finding distance between 2 points, compute path length, compute cloud distance, match clouds using greedy 5 cloud matching algorithm, and recognition. These methods are noted below and correspond to the original $P pseudocode [1] with some additional methods or updates to accommodate the additional methods required for the extension.

double cloudDistance(candidate, template, N, start, penaltyRate)
double distance(point1, point2)
double greedyCloudMatch(candidate, template, N, penaltyRate)
Template normalize(Template, samplingRate)
double pathlength(Template)
Result recognize(candidate, Templates, penaltyRate)
Template resample(Template, samplingRate)
Template scale(Template)
Template translateToOrigin(Template)

Template
findTemplateWithMinimumBoundingBox (Template)
Template rotate(Template, angle)
ArrayList<User> parseXML(String)

VII. Test

Provides a way to perform user-dependent random 100 tests on the dataset. The main method initializes the data set from the default location and starts recognition as described in the paper. Since the test includes 105,600 recognition test, serial implementation would take a long time to complete. Thus, this class is implemented to perform tests in parallel using multithreading. Since each user's test does not interfere with each other, to preserve consistency and validity of the output, each thread performs testing for a single user which is later merged and written into a log file. (see Offline Recognition Test Results sections for details.)

IV. Offline Recognition Test Results.

User-dependent random 100 testing on the dataset described above was done in two stages (i.e. first to compare the effect of different penalty rates on recognition accuracy, and second to compare the performance of the extended version of $P algorithm with the original $P algorithm[1].

A. *Effect of different Penalty Rates*

To check the effect of different penalty rates on recognition accuracy and select the best penalty rate, the user dependent random 100 test was performed on the dataset described above. For random 100 test, for each user, the testing was performed on multiple training set sizes (1,2,4,8 training samples). for each training sample set size T, T number of random samples were selected from each gesture type for training and 1 addition gesture for each gesture type was selected for testing. Since in the original dataset all the gestures were oriented as shown in the image, it would not be able to represent the possibility of rotation to verify rotational invariance. To solve this issue the gesture to be tested was randomly rotated between -45° to 45° before performing the recognition. This process was repeated 100 times for each training set size and each penalty rate. The average recognition accuracy was registered for each penalty rate (0 – 0.1 with 0.01 increments).

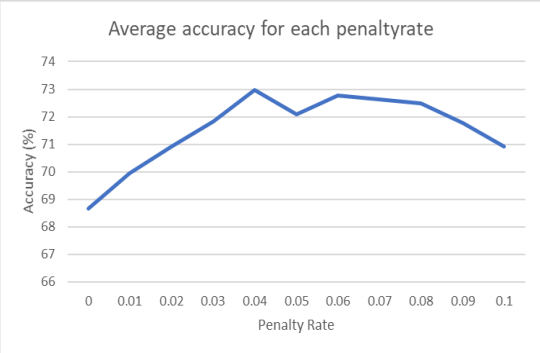

fig 3. Effects of penalty rate on accuracy

Based on the results of 105,600 tests (11 penalty rates X3 users X 4 training set sizes X 8 gesture types X 100 iterations) it was found that the initially the increase of rotational penalty improves the performance but increasing it further makes the penalty of rotation too high that it overpowers the measure of Euclidean distance. It can be clearly seen from the chart shown in fig. 3 that the penalty of 0.04 works the best for the given dataset.

B. *Comparison with original $P Algorithm*

Since the original $P algorithm was tested on MMG Dataset[1], the results of the user dependent random test described above cannot be directly compared with the results of original $P paper[1]. Thus, the dataset shown in fig 1. was tested on both original $P algorithm[1] and extended $P algorithm presented in this paper. Since the original $P algorithm does not allow rotational invariance, for a better comparison between the two algorithms, to tests were performed with each algorithm (i.e. one without rotating the candidate gestures and one with candidate gestures rotated randomly between -45° to 45°)

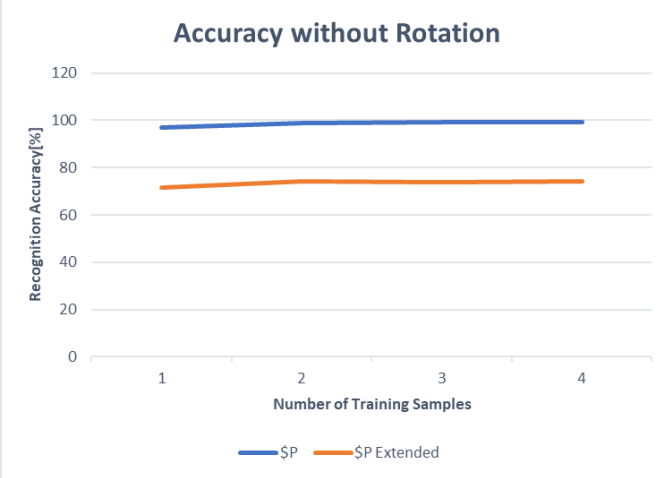

fig 4. comparison of accuracy without rotating gestures

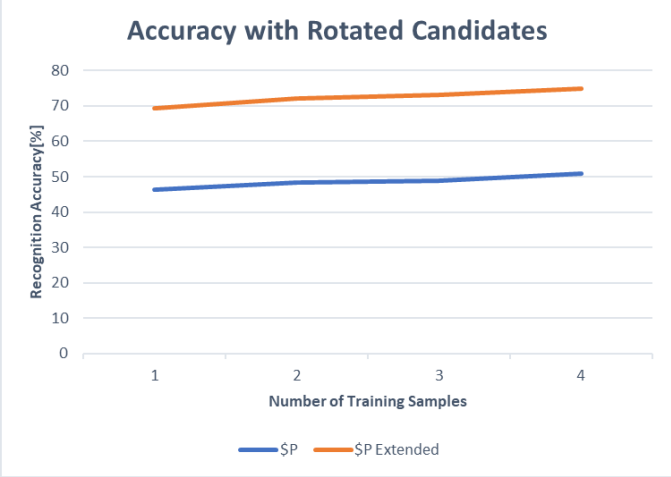

fig 5. comparison of accuracy with rotated gestures

As shown in fig 4, it is clear that the original $P performs much better if the gestures aren't rotated. The reported accuracy for original $P algorithm[1] was more than 96% even with 1 gesture while the recognition accuracy of the extended $P algorithm was about only 74% even with 8 training examples. Both the algorithms showed an increase in accuracy with the increase of the number of training samples.

However, in the case where the gestures are rotated, the extended $P algorithm outperformed the original $P algorithm. While both showed an increase in accuracy with increase in the number of training samples, the original $P algorithm was never able to reach the accuracy of the extended algorithm. The accuracy of the extended $P algorithm was 69.37% with 1 training example and reached 74.87% accuracy with 8 training samples. On the other hand, the original $P algorithm reached only 50.83% accuracy even with 8 training examples.

In conclusion, in the application where gestures are not rotated $P would be an ideal choice compared to the extended algorithm while the extended algorithm would be a better choice if the application requires rotational invariance.

REFERENCES

[1] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures As Point Clouds: A $P Recognizer for User Interface Prototypes. In Proc. of the 14th ACM Int. Conference on Multimodal Interaction (ICMI '12). ACM, New York, NY, USA, 273–280. DOI: http://dx.doi.org/10.1145/2388676.2388732W.-K. Chen, *Linear Networks and Systems*. Belmont, CA, USA: Wadsworth, 1993, pp. 123–135.

[2] Anthony, L., and Wobbrock, J. O. A lightweight multistroke recognizer for user interface prototypes. In GI '10 (2010), 245–252.

[3] Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In UIST '07 (2007), 159–168.

[4] Anthony, L., and Wobbrock, J. O. $N-Protractor: A fast and accurate multistroke recognizer. In GI'2012 (2012), 117–120.

APPENDICES

A. *Additional Methods*

```
Template findTemplateWithMinimumBoundingBox(Template temp)
    MinArea  = ∞
    MinAngle = null

    for angle in -45 to 45:
        Template t = Rotate(temp, angle); // Rotate is same as $1[3]
        xmin = ymin = ∞
        xmax = ymax = - ∞
        for point p in t
            xmin = min(xmin, p.x)
            ymin = min(ymin, p.y)
            xmax = max(xmax, p.x)
            ymax = max(ymax, p.y)
        area = (xmax-xmin)*(ymax-ymin)
        if(MinArea ≥ area)
            MinArea = area
            MinAngle = angle

    return Rotate(temp, MinAngle)
```

B. *Updated CloudDistance*

The original $P algorithm[1] uses the sum of euclidean distances between points as the distance between two templates. The extended algorithm adds the weighted rotation difference between candidate and template to each Euclidean distance of points.

$$\text{distance}(\text{points}, \text{tmpl}) = \text{Euclidean-Distance}(\text{points}_i, \text{tmpl}_j) + \text{penaltyRate} * |\text{points.rotation} - \text{tmpl.rotation}|$$