Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Многоштриховые жесты мышью в проекте QReal

Курсовая работа студентки 445 группы
Осечкиной Марии Сергеевны

Научный руководитель Ю. В. Литвинов
ст. преподаватель

Санкт-Петербург
2011

Оглавление

Введение

В CASE-системах при добавлении объектов на диаграмму и выполнении элементарных действий над объектами (например, удаление), а также в некоторых редакторах при рукописном вводе текста пользователю предоставляется возможность управлять приложением с помощью жестов мышью. Распознавание команды, ассоциированной с жестом, происходит по сигналу: отпускание кнопки мыши, нажатие заданной комбинации клавиш, тайм-аут и так далее. Штрих — фигура, получившаяся в результате движения мыши между нажатием кнопки и ее отпусканием. Многоштриховые (multi-stroke) жесты — жесты, состоящие из нескольких штрихов. Жесты мышью — хороший способ сделать взаимодействие с приложениями более удобным для пользователя. Рассмотрим задачу добавления многоштриховых жестов мышью в CASE-систему QReal, разрабатываемую на кафедре системного программирования математико-механического факультета СПбГУ.

В QReal изначально предполагалось, что объекты на диаграмму надо добавлять перетаскиванием с помощью операции drag-and-drop из палитры. Чтобы ускорить добавление элементов, были реализованы одноштриховые жесты мышью [5]. Но при реализации жестов данного типа учитывались точка начала и направление движения мыши. Чтобы пользователю было удобнее работать с QReal, желательно избежать необходимости запоминания подобных деталей. Хотелось бы, чтобы нарисованный жест в случае неверного распознавания можно было дорисовать для корректного распознавания. Но поддержка жестов мышью, реализованная в QReal на данный момент, не дает такой возможности. Возникла потребность замены одноштриховых жестов мышью на многоштриховые, которые независимы от количества штрихов, точек начала и направления движения мыши. Пользователь должен изобразить жест, возможно несколькими штрихами, похожий на графическое представление требуемого объекта.

QReal – metaCASE-система, которая позволяет быстро добавлять описание новых визуальных языков с помощью метаредактора. Естественным требованием является быстрое добавление описания жестов для новых объектов, желательно без участия создателей языка, поэтому мы хотим по возможности автоматизировать процесс создания жеста. Можно было бы делать прототипы языков и без жестов, но тогда новым языкам соответствовала бы неполная функциональность системы.

Итак, задача состоит в том, чтобы улучшить в CASE-системе QReal поддержку жестов мышью. Жест должно быть просто рисовать, иначе будет проще перетащить объект из меню на диаграмму, из-за этого жесты потеряют актуальность в проекте. Надо дать пользователю возможность не запоминать способ рисования жеста, а помнить только графическое представление желаемого объекта. То есть требуется независимость от количества штрихов, точки начала штрихов, направления движения мыши.

Основные принципы работы алгоритмов распознавания

В общем случае алгоритм распознавания жестов работает следующим образом: есть список идеальных жестов, с каждым из которых сравнивается жест, изображенный пользователем. Идеальный жест — соответствующий объекту жест, по образцу которого пользователь рисует жесты. Пользователь рисует фигуру, удерживая нажатой кнопку мыши. После определенной команды программа распознает фигуру. Для этого в списке идеальных жестов ищется фигура, похожая на изображенный жест. Если требуемая фигура найдена, то выполняется ассоциированная с ней команда: в QReal генерируется объект из загруженного редактора. Существуют алгоритмы, которые позволяют корректировать параметры распознавания посредством обучения, в том числе идеальные жесты. Но для обучения необходима обучающая выборка — база жестов пользователей, где про каждый жест известно, к какому классу он относится. А чтобы ее создать, требуется время.

При распознавании жестов можно выделить следующие основные этапы:

1) Определение пути мыши — создание списка точек.

Программа получает сигнал о том, что кнопку мыши нажали или отпустили или мышь двигают, зажав кнопку. У сигнала есть метод, возвращающий координаты курсора мыши. Заносим эти координаты в список точек соответствующего штриха.

2) Фильтрация пути [4].

На этом шаге сглаживается путь мыши. Этот шаг необходим, так как различное оборудование дает разную точность при получении позиции мыши. Кроме того, если учитывать все дрожания руки, придется усложнять следующий шаг – сопоставление объекта: каждому объекту будет соответствовать слишком много разных вариантов пути мыши.

3) Построение классификатора.

Сравнение списка точек затруднительно, так как полученный список зависит от оборудования, скорости движения мыши, масштаба, — факторов, которые не обязательно учитывать при распознавании. Вводится множество признаков жеста, по которым производится сравнение. Это множество признаков называется классификатором. На пространстве классификаторов определяется расстояние.

4) Выбор объекта.

Для каждого идеального жеста определяем насколько он похож на нарисованный жест, как правило, вычисляя расстояние. Иногда схожесть жестов определяется по-другому, например, рассчитывается вероятность того, что пользователь хотел изобразить тот или иной идеальный жест. В этой работе, главным образом, рассматриваются алгоритмы, использующие расстояние: ищется объект, соответствующий минимальному расстоянию между идеальным жестом и жестом пользователя. Если это расстояние не превосходит максимально возможное расстояние до идеального жеста, генерируется объект.

Существующие решения распознавания многоштриховых жестов

Многоштриховые жесты используются для распознавания рукописного текста на лету в различных текстовых редакторах (например, при написании sms-сообщений в последних моделях Samsung), для создания объектов на формах, диаграммах и сценах в различных приложениях и CASE-системах (Visual Paradigm).

Опишем некоторые из возможных решений распознавания жестов.

1. Rubine алгоритм [6]. Опишем алгоритм Rubine для одноштриховых жестов. Жест представим как упорядоченную последовательность точек. К классификаторам, используемым в алгоритме Rubine, относятся косинус и синус угла между первым отрезком и осью x, длина диагонали описанного около жеста прямоугольника, расстояние между первой и последней точкой жеста, косинус и синус угла между первым и последним отрезком жеста, суммарная длина жеста, квадрат максимума скорости курсора мыши, сумма пройденных углов и протяженность штрихов. По этим классификаторам строится вектор признаков для каждого объекта из базы. С помощью обучающей выборки корректируется идеальный вектор признаков: для каждого класса жестов строится центр масс векторов признаков, сгенерированных по этим жестам. После чего в процессе обучения подбираются веса для вычисления функции расстояния.

$$v=\sum_{e=0}^{F} w_i * f_i ,$$

где F — размерность вектора признаков,

$f_0=1$,

f_i - элемент вектора признаков, стоящий на i месте,

w_i - соответствующие веса.

Веса корректируются следующим образом:

$$f0_{c,i}=\frac{1}{E_c} * \sum_{e=0}^{E_c-1} f_{c,e,i} ,$$

$f0_c$ - центр масс обучающей выборки класса c,

i - координата в векторе признаков,

E_c - количество жестов в обучающей выборке из класса c,

строим матрицу M следующим образом

$$M_{c,i,j}=\sum_{e=0}^{E_c-1} \left(f_{c,e,i}-f0_{c,i}\right)\left(f_{c,e,j}-f0_{c,j}\right) ,$$

$$M_{i,j}=\frac{\sum_{c=0}^{C-1} \frac{M_{c,i,j}}{E_c-1}}{-C+\sum_{c=0}^{C-1} E_c} ,$$

C - количество классов жестов,

$$w_{c,j}=\sum_{i=1}^{F} \left(M^{-1}\right)_{i,j} * f0_{c,j} ,$$

$$w_{c,0} = -0.5 * \sum_{i=1}^{F} w_{c,i} * f0_{c,i}$$

Расширение Rubine алгоритма на многоштриховые жесты авторы [6] определяют следующим образом: жест представляется как упорядоченный набор штрихов (упорядочение проводится в порядке прорисовки штрихов). Кроме того к вектору признаков добавляется количество штрихов в жесте, суммарное расстояние между последней точкой предыдущего штриха и первой точкой последующего, общее суммарное расстояние между штрихами, сумма углов между предыдущим и последующим штрихом, а также отношение длины каждого штриха к длине всего жеста.

Заметим, что вектор признаков меняется уже при растяжении жеста вдоль любой из осей. Это свойство относится к минусам алгоритма: в большинстве других алгоритмов растяжение или сжатие вдоль любой из осей не меняет вектор признаков. Желательно, чтобы столь простые преобразования не сильно влияли на меру схожести фигур.

В данном алгоритме принципиальное значение имеет точка начала каждого штриха, направление движения, а также последовательность штрихов.

Этот алгоритм применяется, к примеру, в утилите iGestures, подробное описание которой можно найти в [6].

2. Авторы следующего алгоритма предлагают вписать жест в прямоугольник и разбить его на 64 равные ячейки прямыми, параллельными сторонам прямоугольника, получив сетку 8 на 8. В классификатор входят те и только те ячейки прямоугольника, через которые проходит жест [6]. В качестве метрики берем расстояния Левенштейна с весами. Веса можно ввести с помощью расстояния Хэмминга, которое возвращает количество несовпадающих символов в двух строках одинаковой длины. Каждой ячейке прямоугольника сопоставим последовательность из 6 символов из алфавита {0, 1} так, чтобы соседние ячейки отличались одним символом. См таблицу. Выписывается последовательность строк из таблицы, соответствующих ячейке жеста. Две строки сравниваются с помощью расстояния Левенштейна.

000000	000001	000101	000100	100100	100101	100001	100000
000010	000011	000111	000110	100110	100111	100011	100010
001010	001011	001111	001110	101110	101111	101011	101010
001000	001001	001101	001100	101100	101101	101001	101000
011000	011001	011101	011100	111100	111101	111001	111000
011010	011011	011111	011110	111110	111111	111011	111010
010010	010011	010111	010110	110110	110111	110011	110010
010000	010001	010101	010100	110100	110101	110001	110000

Заметим, что похожие веса можно аналогичным образом задать с помощью расстояния $l1$ между двумя точками ($(x_1 - x_2) + (y_1 - y_2)$).

Авторы [6] предполагают, что имеет значение направление движения мыши.

3. Алгоритм «звезда» [6]. В данном алгоритме жест представляется как

список направлений движения мыши. Всего этих направлений 8: север, юг, запад, восток, северо-запад, северо-восток, юго-запад, юго-восток. Кроме того, в классификатор включены пропорции и отношение длин направлений. Для каждого конкретного класса объектов можно добавить индивидуальную информацию, к примеру для круга точка начала и точка конца жеста должны быть близки, а протяженность противоположных направлений примерно одинакова. В этом алгоритме, так же как и в предыдущем, учитывается начало жеста и направление движения мыши

4. Следующий алгоритм [3] есть некоторая комбинация второго и третьего алгоритма. Так же, как и в третьем алгоритме, на жест накладывается решетка, но в этом алгоритме фиксировано не количество ячеек, а их размер. Каждой точке жеста ставится в соответствие узел решетки, после чего строится набор строк из направлений жеста. Количество элементов в классификаторе соответствует количеству штрихов в жесте. Пусть $\{(x_i, y_i)\}$ – последовательность точек штриха, w – длина и ширина ячейки. $qx_i = round(x_i / w)$, $qy_i = round(y_i / w)$ – узел решетки, соответствующий точке жеста. У этого узла есть 8 соседних, точке (x_{i+1}, y_{i+1}) будет соответствовать один из этих узлов. Так как каждый штрих непрерывен, он действительно пройдет через какую-то из 8 соседних ячеек в случае, если (x_i, y_i) не последняя точка штриха. Конечная строка представляет собой набор направлений из каждой ячейки в следующую. Расстояние между жестами задается как нормализованное расстояние Левенштейна между строками.

   Алгоритм будет хорошо работать, если масштаб жеста будет фиксирован.

   Этот алгоритм распознавания, как правило, используется для жестов при рисовании объектов формы UI, например в библиотеке CALI и инструменте ScetchiXML.

5. Можно использовать евклидово расстояние для точек жеста, не строя при этом решетку [2]. Пусть есть два штриха, приведем их к одному размеру, добавив необходимое число точек на каждом из отрезков, и масштабируем до заданной высоты и ширины. После построим матрицу D, элементами которой являются попарные расстояния:

$$D[i,j] = \sqrt{\sum_{k=1}^{S} d_k^2 * w[i,k] * w[j,k]},$$

где S – количество точек в жесте,

d_k – расстояние между k-тыми точками жеста,

$w[g,k]$ – вес точки k штриха g $\quad w[g,k] = e / Z_g$,

где $e = 1$, если k – исходная точка, полученная в процессе рисования жеста, $0 < e < 1$ иначе,

Z_g – нормализация штриха, зависящая от масштаба и количества добавленных точек.

Авторы [2] утверждают, что несмотря на то, что один и тот же объект может быть изображен в разных случаях разным числом жестов, веса сглаживают различие между жестом из одного штриха и жестом из нескольких штрихов. Два аналогичных жеста окажутся близки.

Следующие два метода применяются в комбинации с каким-либо классификатором, в том числе можно использовать любой из приведенных выше.

6. Метод k ближайших соседей [1]. Для этого метода необходима обучающая выборка. Для классификатора пользовательского жеста ищутся k ближайших соседей. Жест относят к тому классу, к которому относится большинство из k соседей.

7. Классификатор Байеса [1]. Пусть есть множество признаков, между которыми нет статистических зависимостей. Для каждого признака на основе обучающей выборки вычисляется вероятность того, что данный жест принадлежит данному классу. Затем вычисляется общая вероятность того, что данный жест принадлежит данному классу. В качестве итогового класса берется наиболее вероятный.

Из описанных алгоритмов были опробованы 2 и 3.

Такие характеристики жеста, как количество штрихов, точка начала штриха и направление жеста учитывает большинство приведенных алгоритмов: Rubine-алгоритм, алгоритм «звезда», алгоритмы 2 и 4. Можно избежать привязки к последовательности точек жеста в алгоритме 2, если упорядочить ячейки каким-либо образом. Некоторые из приведенных решений требуют длительного обучения, а кроме того, хранения дополнительной информации, например, метод k ближайших соседей. Для нас важно избежать обучения или сократить обучающую выборку до минимума из-за генеративного подхода к созданию жестов в QReal. Поэтому было принято решение опробовать другие классификаторы.

Вспомогательный список ячеек

Жесту соответствует путь мыши — список штрихов, штрихи в свою очередь представляют собой список точек. Желательно иметь дело с более простыми объектами, чем списки точек. К примеру, при переносе траектории мыши должен генерироваться один и тот же объект, но координаты точек в списке сильно меняются. Для обработки этой ситуации можно переносить жест в начало координат, но остается проблема масштабирования — подобным жестам разного размера соответствует один и тот же объект. Можно приводить жест к заранее заданному размеру, но это может повлиять на распознавание объекта.

Таким образом, даже после переноса и масштабирования довольно сложно понять, на какой из идеальных жестов больше похож нарисованный путь. Введем понятие «вспомогательный список ячеек»: сначала около жеста пользователя описывается прямоугольник со сторонами, параллельными осям координат, далее этот прямоугольник разбивается на равные ячейки прямыми, параллельными горизонтальным и вертикальным сторонам прямоугольника, затем составляем список ячеек, через которые проходит жест пользователя. Два вспомогательных списка ячеек назовем равными, если каждая ячейка одного списка содержится в другом и наоборот. Распознавание жеста пользователя сведем к распознаванию вспомогательного списка ячеек.

Предложенные параметры распознавания

Реализованные на данном этапе в QReal одноштриховые идеальные жесты генерируются по графическому описанию объекта, представленному в виде набора отрезков. Для этого строится минимальный по количеству ребер эйлеров граф, содержащий все ребра объекта. Жест выбирается как путь, проходящий через ребра графа. [5] Многоштриховые идеальные жесты генерируются аналогично одноштриховым, за исключением того, что нет необходимости строить эйлеров путь и объединять компоненты связности графического представления объекта: для идеальных многоштриховых жестов генерируется набор из идеальных штрихов.

В процессе работы над курсовой были опробованы некоторые классификаторы, основанные на вспомогательном списке ячеек.

Строки и расстояние Левенштейна. Этот метод был описан выше в существующих решениях. Строим вспомогательный список ячеек. Расстояние между списками вводим как нормированное расстояние Левенштейна. Нормировать расстояние необходимо, так как расстояние Левенштейна между длинным списком S и его частью s обычно меньше, чем расстояние между списком S и похожем на него длинным списком R. Если расстояние не нормировать, программа может распознать жест как часть требуемого объекта. Например, если среди идеальных объектов есть прямоугольник и горизонтальный отрезок, то нарисованный пользователем прямоугольник программа может распознать как горизонтальный отрезок.

Если использовать такой классификатор и не требовать от пользователя начинать жест в определенном месте, рисовать его в заданном направлении и рисовать заданное количество штрихов, то строку надо каким-то образом преобразовать, чтобы она не зависела от перечисленных факторов. Решение проблемы — сортировка. Вот два возможных варианта сортировок.

Сортировка по координатам. Ячейки вспомогательного списка сортируем по координатам. Получаем представление конечной картинки.

Обход оболочки.

1) Левой нижней ячейке вспомогательного списка присваиваем статус текущей, после этого осуществляем обход вспомогательного списка против часовой стрелки.
2) Заносим текущую ячейку в конечный список. Если в конечный список занесены все ячейки вспомогательного списка, алгоритм прекращает работу. Иначе берем текущую ячейку списка, обходим соседние ячейки и смотрим, содержится ли одна из них во вспомогательном списке. Обход осуществляется в следующем порядке: левая нижняя, нижняя, правая нижняя, правая и так далее против часовой стрелки.
3) Как только оказывается, что во вспомогательном списке встретилась соседняя ячейка, прекращаем обход соседей. Заносим соседнюю ячейку в конечный список ячеек, теперь текущей становится она. Переходим к шагу 2. Если мы обошли всех соседей, но оказалось, что ни одна соседняя с текущей ячейка не принадлежит вспомогательному списку, переходим к шагу 4.
4) Ищем ближайшую ячейку к текущей из тех, которые принадлежат вспомогательному списку ячеек, но еще не встречались при обходе. Если таких несколько, то берем левую нижнюю. Присваиваем ей статус текущей. Переходим к шагу 2.

Приведение жестов к одному размеру. Как и в предыдущем алгоритме, вспомогательный список необходимо упорядочить, что мы и делаем, применяя одну из вышеописанных сортировок. Для нахождения расстояния между конечными списками приводим их к одной длине. Каждую ячейку первого жеста повторяем столько раз, какова длина второго списка. После чего берем расстояние Хэмминга между конечными списками. К примеру, надо определить расстояние между строками ab и abc. Длина строки ab равна 2, длина abc – 3. Чтобы привести строки к одному размеру, каждый символ первой строки надо повторить 3 раза, а второй — 2 раза. Получаем строки равной длины aaabbb и aabbcc, между которыми можно определить расстояние Хэмминга.

Расстояние между ближайшими ячейками. Хотелось бы получить расстояние, имеющее некий геометрический смысл. Мы предлагаем не преобразовывать список ячеек в строку, а рассмотреть расстояние между фигурами. Проанализируем, как можно интерпретировать похожесть жестов. Заметим, что два жеста можно назвать похожими, если для каждого объекта (отрезок, окружность, ячейка) из одного жеста можно подобрать аналогичный объект из второго жеста, так чтобы они были не сильно удалены. То есть для каждой ячейки из первого вспомогательного списка можно указать близкую к ней ячейку второго вспомогательного списка ячеек и наоборот. На основании этого введем расстояние μ между вспомогательными списками ячеек.

$M1$, $M2$ – вспомогательные списки ячеек, для которых надо определить расстояние. Пусть r – расстояние в R^2, l – норма для конечной последовательности чисел. Ячейку во вспомогательном списке будем рассматривать как точку в двумерном пространстве, для каждой ячейки q вспомогательного списка $M1$ найдем расстояние до всех ячеек вспомогательного списка $M2$ и возьмем минимальное расстояние, составим последовательность из минимальных расстояний для вспомогательного списка ячеек $M1$, аналогично для вспомогательного списка $M2$. Возьмем норму l для обеих этих последовательностей. Максимум из этих двух норм будет расстоянием между вспомогательными списками ячеек. Заметим что введенная функция μ действительно является расстоянием. Проверим выполнение трех свойств расстояния:

1) $\mu=0 \Leftrightarrow M1=M2$. <= очевидно, докажем =>. Заметим, что $\mu=0$ тогда и только тогда, когда обе нормы построенных последовательностей чисел равны 0. Норма элемента равна 0, если и только если элемент равен 0, а значит обе последовательности состоят только из 0. Значит для любой ячейки q вспомогательного списка $M1$, найдется равная ей ячейка вспомогательного списка $M2$ (равная по координатам). Для ячеек вспомогательно списка $M2$ аналогично. То есть $M1=M2$

2) $\mu(M1,M2)=\mu(M2,M1)$ Это следует из задания μ

3) $\mu(M1,M2)\le\mu(M1,M3)+\mu(M3,M2)$ Это верно, так как l является нормой, а r – расстоянием.

В качестве r были рассмотрены евклидово расстояние ($l2$), максимум модуля разности координат ($l\infty$), сумма модулей разности координат ($l1$). В качестве l были рассмотрены максимум ($l\infty$), среднее арифметическое, среднее квадратичное. Наилучший результат при распознавании показала комбинация суммы модулей разности координат в качестве r и максимум элементов в качестве l .

Заметим, что данный классификатор имеет нефиксированную размерность, то есть для разных жестов пользователя вспомогательный список ячеек может состоять из разного количества ячеек. Это может помешать в процессе обучения: как правило идеальные жесты приходится корректировать. Проще это делать, если классификатор имеет фиксированную размерность, поэтому для дальнейшего обучения желательно построить классификатор, аналогичный вышеописанному, но с фиксированной размерностью. Вот несколько классификаторов с фиксированной размерностью.

Матрица расстояний до жеста. Пусть h – высота описанного около жеста прямоугольника, w – ширина прямоугольника, M – матрица $w*h$, элемент матрицы $M_{i,j}$ равен расстоянию μ от ячейки с координатами (i,j) до ближайшей ячейки вспомогательного списка. Таким образом мы имеем матрицу расстояний до вспомогательного списка. Расстояние между вспомогательными списками — это норма разности соответствующих им матриц. Введенная функция действительно является расстоянием между вспомогательными списками ячеек. Мы ввели классификатор, который является матрицей и имеет фиксированную размерность.

Количество ячеек в прямоугольнике. Следующий алгоритм заключается в подсчете количества ячеек в прямоугольнике $m*n$, где m – меняется от 1 до высоты прямоугольника, n — от 1 до ширины прямоугольника. Строится матрица $h*w$, где h – высота прямоугольника, w – ширина прямоугольника. Элемент $[i,j]$ равен количеству ячеек жеста, содержащихся в прямоугольнике $i*j$, при этом если жест проходит через одну и ту же ячейку несколько раз, в матрицу заносится только одно прохождение. Расстояние между жестами определяется как расстояние между векторами в R^{h*w} (евклидово, $l1$, $l\infty$), при этом матрица рассматривается как вектор из этого пространства. Строки матрицы выписываются в одну строку, таким образом получаем вектор. Заметим, что по построенной матрице можно однозначно восстановить построенный жест. Пусть $e[i,j]$ – наличие ячейки $[i,j]$ в построенном жесте (0 или 1). J – построенная матрица, тогда

$$e[i,j]=J[i,j]+J[i-1,j-1]-J[i,j-1]-J[i-1,j]$$

Будем считать, что за пределами прямоугольника элементы J равны 0. Заметим, что введенная функция действительно будет расстоянием для вспомогательных списков ячеек. Выполнение первого свойства расстояния следует из того, что по матрице можно однозначно восстановить вспомогательный список ячеек. Остальные свойства очевидны.

Количество ячеек в полосе. Классификатор аналогичен предыдущему, только подсчет ячеек происходит в полосе. J – построенная матрица, покажем чему равен элемент $J[i,j]$. Пусть d – число ячеек в полосе $(i-1)*w$, то есть в полосе, целиком лежащей под ячейкой $[i,j]$. k – число ячеек в полосе из ряда i от $[i,0]$ по $[i,0]$, $J[i,j]=k+d$. Заметим, что вспомогательный список ячеек однозначно восстанавливается по полученной матрице. Тогда если мы введем функцию расстояния между матрицами, эта функция будет расстоянием между вспомогательными списками ячеек.

Заметим, что все эти алгоритмы и расстояния можно сочетать, рассматривая в качестве конечного классификатора структуру из произвольных комбинаций вышеперечисленных классификаторов. При этом расстояние между вспомогательными списками ячеек задается как линейная комбинация расстояний с неотрицательными коэффициентами, один из которых строго положителен. Такая

линейная комбинация действительно будет расстоянием между вспомогательными списками ячеек.

Результаты распознавания с использованием перечисленных классификаторов

Для тестирования были выбраны объекты, использующиеся в диаграммах QReal. Приведем графическое представление выбранных объектов.

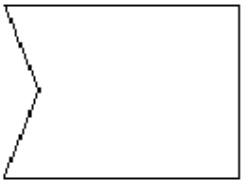


рис 1 Accept Event Action

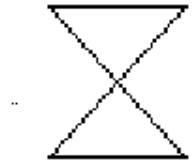


рис 2 Accept Time Event Action

рис 3 Activity Partition

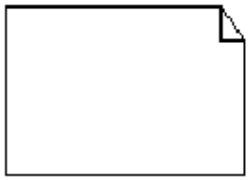


рис 4 Comment

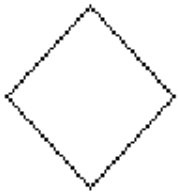


рис 5 Decision Node

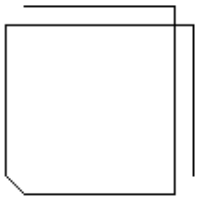


рис 6 Diagram

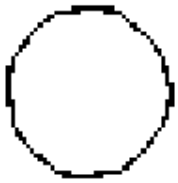


рис 7 Initial Node

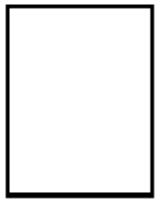


рис 8 Input Pin

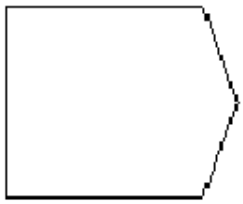


рис 9 Send Signal Node

алгоритм	рис 1	рис 2	рис 3	рис 4	рис 5	рис 6	рис 7	рис 8	рис 9	**Всего**
Распознавание пользователем	115	142	133	128	147	122	147	122	121	**1177**
Расстояние Левенштейна + сортировка по координатам	107	131	133	1	3	0	6	33	77	**492**
Расстояние Левенштейна + сортировка обходом	0	0	0	0	0	5	5	5	5	**5**
Приведение к одному размеру + сортировка по координатам	16	58	122	25	68	9	9	46	32	**385**
Приведение к одному размеру + сортировка обходом	0	49	1	47	126	42	18	1	12	**296**
Расстояние между ближайшими ячейками	112	137	133	7	142	98	143	4	61	**837**
Матрица расстояний до жеста	114	137	133	9	141	100	144	8	62	**848**
Количество ячеек в прямоугольнике	37	129	127	101	16	64	136	3	72	**685**
Количество ячеек в полосе	86	141	125	12	49	46	44	4	38	**647**
Комбинация алгоритмов	113	138	133	74	116	104	146	9	66	**899**

Заметим, что процент распознанных жестов сильно отличается для разных классов объектов. Распознаваемость зависит от того, насколько идеальный жест похож на другие идеальные жесты: чем сильнее похожи жесты, тем легче их перепутать.

Классификатор, подсчитывающий количество ячеек в прямоугольнике, имеет невысокий процент распознавания, но хорошо распознает некоторые жесты, которые

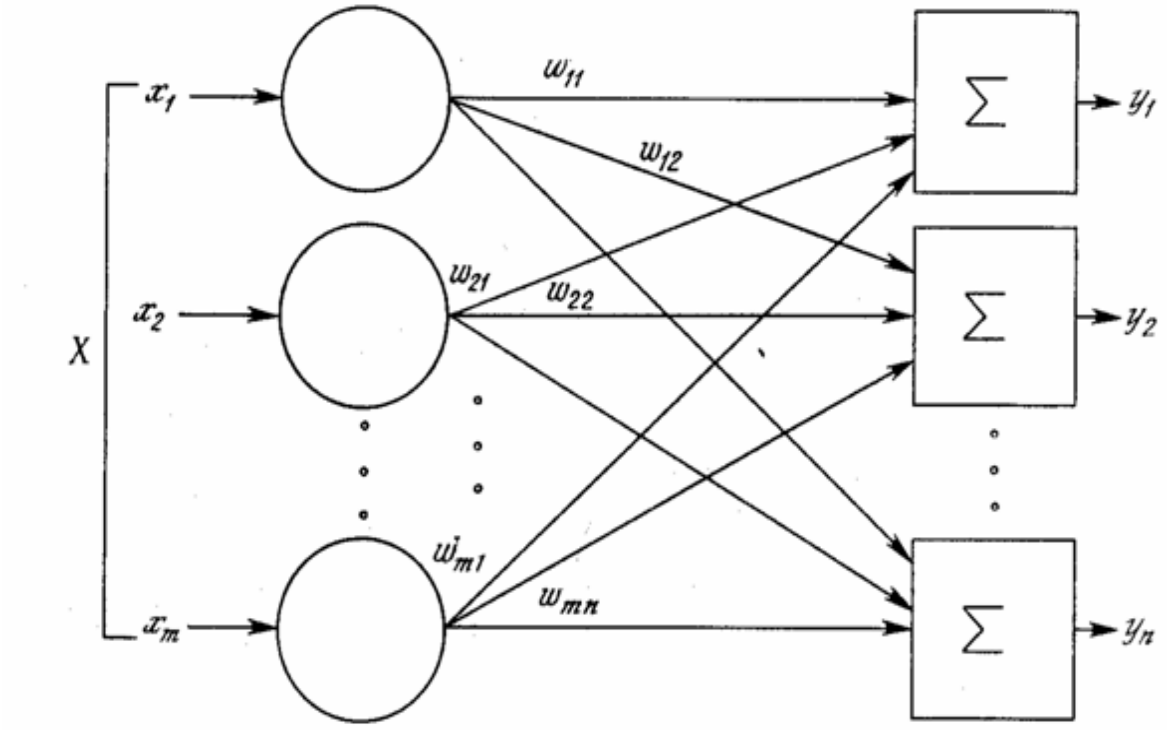
плохо распознаются другими классификаторами. Поэтому было принято решение скомбинировать данный классификатор с классификатором, дающим наилучшую итоговую распознаваемость. Коэффициенты для линейной комбинации расстояний были подобраны эмпирически. Результаты распознавания жестов с помощью комбинации классификаторов приведены в последней строке.

Обучающие алгоритмы

Для улучшения распознаваемости пользовательских жестов можно использовать обучение. На основе базы жестов пользователя, о каждом из которых известно, к какому классу он относится, можно изменить параметры алгоритма распознавания так, что и другие жесты пользователя, подаваемые на вход, будут распознаваться с большей вероятностью.

Уже описанный метод k ближайших соседей и метод Байеса относятся к алгоритмам обучения.

Широкое распространение среди алгоритмов обучения получили нейронные сети. Обучение нейронной сети подразумевает нахождение коэффициентов связей между нейронами (см. рисунок «нейронная сеть»). Но нередко обучать нейронную сеть приходится довольно долго. Например, в [7] обучающая выборка в 1,5 раза превосходит тестовую. Как уже упоминалось, хотелось бы как можно сильнее сократить обучающую выборку, иначе генеративный подход при создании жестов утратит смысл. К тому же, как правило, при распознавании жестов мышью на вход нейронной сети подается вектор, который весьма неоднозначно определяет исходный жест. Например, входной вектор может состоять из значений средней кривизны, среднего угла, скорости курсора мыши и так далее. Может получиться так, что перед обучением разным идеальным жестам соответствуют одинаковые вектора признаков.



Нейронная сеть

Поэтому был выбран алгоритм k-средних, предполагающий корректировку идеального жеста и максимального расстояния до жеста. Для каждого класса жестов задана обучающая выборка. Изначальный классификатор, соответствующий

идеальному жесту, заменяется на центр масс классификаторов, соответствующих жестам из обучающей выборки для этого класса.

Как правило, центр масс нескольких точек вычисляется как сумма векторов от начала координат до этих точек, разделенная на количество точек. Проблема классификаторов с нефиксированной размерностью в том, что довольно трудно определить начало координат, сумму классификаторов и деление классификатора на скаляр, так чтобы классификатор не утратил своего геометрического смысла.

При обучении были использованы следующие свойства центра масс:

1) Центр масс одного классификатора совпадает с этим классификатором.

2) Пусть M_n - классификатор, соответствующий центру масс n классификаторов заданного класса,

D_n - максимальное расстояние от M_n до классификатора жеста из заданного класса,

k - еще один классификатор, соответствующий жесту данного класса,

$d=dist\left(M_n,k\right)$ - расстояние между центром масс и новым классификатором, тогда

$$dist\left(M_{n+1},M_n\right)=d*\left(1/n+1\right)=d0_M ,$$
$$dist\left(k,M_n\right)=d*\left(n/n+1\right)=d0_k$$

Будем строить точку, находящуюся на расстоянии $d0_M$ от точки M_n и на расстоянии $d0_k$ от точки k (свойство 2). В общем случае такая точка может не существовать в метрическом пространстве. Тогда будем строить точку, отношение расстояний от M_n и k до которой наиболее близко к требуемому. Или таких точек может быть несколько, важно чтобы алгоритм в этом случае возвращал хотя бы одну точку, обладающую таким свойством.

В алгоритме, вычисляющем расстояние между двумя ближайшими ячейками, используется классификатор нефиксированной размерности. Чтобы вычислить расстояние между двумя классификаторами, рассматриваются пары ячеек, находящиеся на минимальном расстоянии друг от друга. Для каждой ячейки первого вспомогательного списка ищется ближайшая к ней ячейка второго вспомогательного списка, и для каждой ячейки второго вспомогательного списка — ближайшая к ней ячейка первого вспомогательного списка. После чего пары ячеек рассматриваются как концы отрезка на плоскости, и берется точка, делящая отрезок в заданном соотношении $1/n$, считая от первой ячейки, если первая ячейка относится к классификатору центра масс, считая от второй ячейки в ином случае. Полученные точки рассматриваются как ячейки, возможно с нецелыми координатами, и составляют классификатор, который мы будем считать новым центром масс. Заметим, что свойство 2 выполняется.

Алгоритм, использующий взвешенное расстояние Левенштейна, тоже использует классификатор нефиксированной размерности. Есть два классификатора, находящиеся на расстоянии d друг от друга. Чтобы получить классификатор «между» двумя данными, надо рассмотреть последовательность операций из замен и удаления ячеек, соответствующую расстоянию Левенштейна, и выполнить часть из этих операций. Надо выполнять операции замены и удаления ячеек из классификатора, который является центром масс, пока расстояние от центра масс до нового классификатора меньше чем $d*\left(\frac{1}{n+1}\right)$.

В процессе обучения надо редактировать не только центр масс, но и максимальное расстояние от центра масс класса до произвольного классификатора

для жеста из данного класса. Необходимо корректировать максимальное расстояние так, чтобы шар B_n с центром M_n и радиусом D_n принадлежал шару B_{n+1} с центром M_{n+1} и радиусом D_{n+1}. Новый классификатор должен принадлежать новому шару.

Таким образом $D_{n+1} \geq \frac{d*n}{n+1}$.

$k_0 \in B_n$ - классификатор, относящийся к классу жестов, который соответствует B_n

По свойству треугольника $dist\left(k_{0,} M_{n+1}\right) \leq dist\left(k_{0,} M_n\right) + dist\left(M_n, M_{n+1}\right)$.

Исходя из двух неравенств выберем $D_{n+1} = max\left(\frac{d*n}{n+1}, D_n + \frac{d*1}{n+1}\right)$.

Результаты обучения

Обучающая выборка состояла из 45 жестов, каждому из 9 объектов соответствовали 5 жестов пользователя.

классификатор	Рис 1	Рис 2	Рис 3	Рис 4	Рис 5	Рис 6	Рис 7	Рис 8	Рис 9	**После**	До
Распознавание пользователем	115	142	133	128	147	122	147	122	121	**1177**	1177
Расстояние между ближайшими ячейками	93	140	133	98	144	103	145	48	85	**989**	837
Матрица расстояний до жеста	93	141	133	97	145	96	140	56	94	**995**	848
Количество ячеек в прямоугольнике	58	111	93	102	55	100	123	68	28	**738**	685
Количество ячеек в полосе	78	138	132	86	105	100	136	62	57	**904**	647
Комбинация классификаторов	89	141	133	117	138	120	142	93	88	**1061**	899

Заметим, что увеличение процента распознанных жестов зависит не только от обучающей выборки, но и от классификатора. К примеру, классификатор, подсчитывающий количество ячеек в полосе, после обучения показал улучшение распознаваемости на 40 %. В то время как классификатор, подсчитывающий количество ячеек в прямоугольнике, дал улучшение распознаваемости всего на 8 %. При последующей работе можно ввести двойное обучение — обучать классификаторы вышеописанным способом, при этом рассматривая комбинацию не 2, а более классификаторов. Как уже было сказано расстояние при этом вычисляется как линейная комбинация расстояний с неотрицательными коэффициентами.

Генерация обучающей выборки

После проведенных экспериментов предложенные классификаторы не дали удовлетворительного результата без обучения. Максимальный процент распознанных жестов был равен 76 %. Была предпринята попытка избежать обучения пользователем посредством генерации обучающей выборки. Хотелось бы, чтобы приложение могло выполнить обратную к распознаванию задачу. А именно, имитировать хотя бы часть жестов, близких к тем, которые рисует пользователь.

Причем так, чтобы сгенерированная выборка описывала максимальное число жестов для каждого класса, которые может нарисовать пользователь.

Попробуем понять, какие признаки остаются инвариантными для жестов из одного класса. Предположим, пользователь рисует идеально сглаженный жест, то есть если он хочет провести отрезок, то проводит отрезок: помимо дрожания руки есть множество факторов, влияющих на искажение жеста. Приходится игнорировать дрожание руки, так как иначе пришлось бы генерировать слишком много различных жестов, хотя некоторым из них будет соответствовать один и тот же классификатор. Заметим, что параллельность при изображении жестов как правило не сохраняется, пропорции тоже, но пользователь все же рисует треугольник в виде треугольника, прямоугольник — в виде четырехугольника, близкого к нему, окружность — в виде эллипса. Кроме того, пользователь старается сохранять выпуклость фигуры, то есть если угол не превосходит π в идеальном жесте, то и в жесте пользователя этот угол тоже не превосходит π . То есть требуется преобразование, которое не сохраняет ни углов, ни пропорций, ни параллельность, но сохраняет выпуклость, количество и взаимное расположение фигур. Этому свойству удовлетворяет перспективная проекция [8].

Перспективную проекцию плоской фигуры можно задать тремя параметрами — углом поворота вокруг $0x$ и вокруг $0y$, точкой расположения наблюдателя (камеры). Были испробованы разные подходы к выбору углов — углы выбирались произвольно независимо друг от друга, менялись их значения в различных интервалах, независящих друг от друга или зависящих друг от друга, но удовлетворительного результата это не дало. Единственное непреложное требование — повернутый идеальный жест не должен пересекать картинную плоскость. Изначально улучшения в распознавании после обучения не было. Когда на вход обучения стали подавать только те жесты, которые не входят в шар, соответствующий заданному классу жестов, точность распознавания возросла на 2%.

Заключение

Были протестированы несколько классификаторов для распознавания многоштриховых жестов, объединены два лучших из них.

Был опробован алгоритм обучения для классификаторов фиксированной размерности и расширен на некоторые классификаторы нефиксированной размерности. Была предпринята попытка реализовать самообучение алгоритма.

На основании тестирования классификаторов без обучения был сделан вывод, что распознавание без обучения не удовлетворит пользователя. Но начальный порог распознавания и скорость обучения позволяют разработчикам не переучивать систему каждый раз при добавлении нового жеста, а предоставить пользователю возможность самому обучать приложение в процессе работы. Под начальным порогом распознавания подразумевается процент распознанных без обучения жестов, скорость обучения — отношение увеличения распознанных жестов к количеству жестов в обучающей выборке.

На данный момент лучшая распознаваемость после обучения составляет 90 % жестов пользователя для тестовой выборки из 1177 жестов. Это вполне удовлетворительный результат для распознавания жестов. Планируется добавить многоштриховые жесты в QReal, предварительно создав обучающую базу жестов для всех объектов.

В дальнейшем планируется добавить возможность изменять объект уже после того, как он был распознан. Хочется сделать распознавание по тайм-ауту — тогда пользователю не придется прибегать к помощи клавиатуры.

На будущее ставится задача распознавать не только отдельные объекты, но и диаграммы объектов. Возникает проблема вложенности объектов — в QReal есть объекты, графическое представление которых включает в себя графическое представление другого объекта. К примеру есть объект Actor, изображаемый в форме фигуры человека с головой в виде окружности, а есть объект Initial node, графическое представление которого есть окружность. Распознать такого рода объекты будет довольно сложно.

Реализацию программной части можно скачать по адресу git://github.com/qreal/qreal.git, версия от 19.04.2011, ник автора osechkina_masha. Инструмент для распознавания жестов мышью находится в папке tools/MouseGestures.

Список литературы

1) Cesar F. Pimentel, Manuel J. da Fonseca, Joaquim A. Jorge, Experimental evaluation of a trainable scribble recognizer for calligraphic interface // Graphics recognition : algorithms and applications, editors: Dorothea Blostein, Young-Bin Kwon, Ontario, Canada, 2001, p. 81-85
2) Gesture recognition based on manifold learning, Heeyoul Choi, Brandon Paulson, Tracy Hammond // Structural, syntactic, and stastistical pattern recognition, editors: Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, Marco Loog, Orlando, USA, 2008, p. 247-250
3) Trainable sketch recognizer for graphical user interface design, Adrien Coyette, Sascha Schimke, Jean Vanderdonckt, Claus Vielhauer // Human-computer interaction – INTERACT 2007, Cecilia Baranauskas, Philippe Palanque, Julio Abascal, Simone Diniz Junqueira Barbosa, Rio de Janeiro, Brazil, 2007, p. 125-130
4) Xuejiang Cheng, Self-adjusting digital filter for smoothing computer mousemovement // Freepatentsonline. URL http://www.freepatentsonline.com/5661502.pdf (Дата обращения: 14.02.2010)
5) Осечкина Мария, Курсовая, Визуальное программирование при помощи мыши// сайт кафедры системного программирования СПбГУ URL http://se.math.spbu.ru/SE/YearlyProjects/2010/YearlyProjects/2010/345/Osechkina_report.pdf (Дата обращения: 23.04.2011)
6) Beat Signer, Moira C. Norrie, Ueli Kurmann, iGesture: A Java Framework for the Development and Deployment of Stroke-Based Online Gesture Recognition Algorithms // Eth Zurich – Department of Computer Science, URL http://www.inf.ethz.ch/personal/signer/publications/2007-snk-tr561.pdf (Дата обращения: 1.02.2011)
7) Don Willems, Ralph Niels, Marcel van Gerven, Louis Vuurpijl, Iconic and multi-stroke gestures recognition // Radboud University Nijmegen, URL http://www.cs.ru.nl/~marcelge/papers/willems2009.pdf (Дата обращения: 3.02.2011)
8) Эдвард Эйнджел, Интерактивная компьютерная графика. Вводный курс на базе OpenGL, 2-е издание, пер. с англ. В. Т. Тертышный, издательский дом «Вильямс», 2001, стр 221 - 223
9) MacQueen, *J. B*. Some Methods for Classification and Analysis of Multivariate Observations // Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, 1967, p. 281–297