# **Введение**

навигация приложение нейронный анализатор

Сегодня трудно представить жизнь без автомобиля, а поездку без навигатора. С увеличением числа дорог и усложнением инфраструктуры городов, становиться сложнее добраться из пункта A в пункт B без отслеживания текущего местоположения на карте.

Желание облегчить себе жизнь автоматическим определением локации в путешествиях и не только появилось давно, как и устройства позволяющие это сделать. Первые навигаторы появились в начале ХХ столетия (1920 г.). Устройство прибора напоминало обычные наручные часы. Они состояли из небольшой, свернутой рулоном карты и рукояток, который можно было вращать, в зависимости от передвижения по местности и по карте.

В начале 30х годов появились первые работы на тему инерциальных систем навигации. Базовые принципы таких систем были введены ещё Ньютоном в законах механики. Инерциальная навигация - это метод определения координат, скорости и угловой ориентации объекта на основе измерения и интегрирования его ускорения.

Вместе с освоением космоса появились такие спутниковые системы как GPS и GLONASS. Первые экземпляры GPS-навигаторов появились 4 октября 1947 г., что стало следствием выведения на орбиту Земли советского искусственного спутника. В достижении данной ступени развития мы должны быть обязаны военным, которые первые испытали и усовершенствовали навигационные аппараты. Американский проект «TRANSIT» был запущен в 1958 г.: он стал аналогом современной GPS-навигации. Усовершенствованная система определения координат и мониторинга объектов, перемещающихся по поверхности нашей планеты, была создана в 1960 г., благодаря выведенным на орбиту американским спутникам.

Также развитие беспроводных систем привело к росту популярности network-based navigation systems, которые работают с помощью методов триангуляции. Триангуляция - один из методов создания сети опорных геодезических пунктов, а также сама эта сеть. Заключается в геодезическом построении на местности системы пунктов, образующих треугольники, у которых измеряются все углы и длины некоторых базовых (базисных) сторон.

К сожалению, самые современные и популярные системы навигации (network-based navigation systems, GPS и GLONASS) имеют ряд недостатков: низкая точность, ограниченность навигации в условиях слабого сигнала и невозможность навигации при полном отсутствии сигнала. Чтобы решить эти проблемы можно использовать инерциальную систему навигации, так как она не имеет таких ограничений.

Навигация в современных городах с использованием спутников создает большие затруднения. Высокой уровень плотности городской застройки серьезно влияет на качество принимаемого сигнала от спутниковой системы. Возникающий помехи влекут за собой серьезное уменьшение точности вплоть до ошибки в несколько сотен метров. Это может привести к тому, что на сложных транспортных развязках водитель может легко запутаться и продолжить движение в неверном направлении. Более того, многие города имеют подземные развязки, где прием сигнала от спутниковой системы невозможен.

Другая проблема, которая может быть решена с помощью инерциальной системы навигации является навигация с учетом текущей полосы движения. Низкая точность спутниковых систем не позволяет точно определить где именно на дороге, на какой полосе ты находишься. В этом случае Навигационное ПО не всегда может правильно подсказать водителю когда нужно перестроится для подготовки к очередному маневру.

Еще одной актуальной темой является indoor навигация. Современные офисные здания и торговые центры имеют огромные размеры, что отрицательно сказывается на сложности в навигации по зданию обычными методами. Однако использования спутниковых систем или network-based систем невозможно.

В скором времени проблема навигации в крупнейших городах станет критичной для их дальнейшего развития. Но с развитием инерциальных или комбинированных систем на базе современного смартфона мы сможем дать жителям и гостям отличную возможность для навигации.

Целью данной работы является создания оптимальной инерциальной системы навигации и разработка её реального прототипа.

Для достижения этой цели необходимо решить несколько серьезных исследовательских, технологических, а также практических задач:

• исследовать существующие решения. Тема инерциальной навигации далеко не нова и существует огромное количество различных решений. Необходимо подробно изучить уже существующие наработкии выделить наиболее подходящее для практического использования в конечном продукте.

• определиться с базовыми источниками данных для нашей навигационной системы. Любая инерциальная система использует данные с различных датчиков. Акселерометр обычно является базовым. Его могут дополнять один или несколько дополнительных датчиков таких как гироскоп, компас, магнитометр, камера и другие. Необходимо проработать варианты для каждой из комбинаций и выбрать наиболее точное и простое решение.

• выбрать метод для обработки данных с датчиков, оптимальный по скорости и точности.

• найти наиболее удобный способ реализации полноценного продукта для практического применения пользователями. На рынке представлено несколько основных мобильных ОС, поэтому необходимо изучить возможности для создания приложения под каждую из них.

# **1. Теоретическая часть**

В рамках этой главы будут подробнее описаны:

) Инерциальные системы навигации;

) Существующие пути решения и реализации таких систем;

3) Машинное обучение;

4) Трудности, отмеченные в предшествующих исследованиях;

) Вывод и задачи для разрабатываемого приложения.

## **1.1 Инерциальные системы навигации**

Инерциальная навигация **-** метод навигации (определения координат и параметров движения различных объектов - судов, самолётов, ракет и др.) и управления их движением, основанный на свойствах инерции тел, являющийся автономным, т.е. не требующим наличия внешних ориентиров или поступающих извне сигналов [1].

Сейчас сложно представить жизнь человечества без систем инерциальной навигации (ИНС), так как именно они делают возможными беспосадочные авиаперелеты, полеты в космос, длительные походы подводных лодок.

Составными частями любой инерциальной навигационной системы являются блок чувствительных элементов (акселерометров и гироскопов) и вычислитель, в котором реализуется навигационный алгоритм. Точность выходной навигационной информации напрямую зависит от характеристик чувствительных элементов, входящих в состав системы, поэтому наиболее точное навигационное решение можно получить только в инерциальных системах, построенных на прецизионных гироскопах и акселерометрах.

Хочется добавить, что инерциальная навигация возможна не только на ИНС, но и на обычном смартфоне, так как практически любой девайс имеет минимально необходимый набор датчиков для её реализации. Это позволяет иметь инерциальный навигатор у себя в кармане, не используя громоздкие и дорогостоящие ИНС, как на самолётах и т.д. Конечно, у таких систем на мобильных девайсах будет ряд недостатков, основным из которых является низкая точность.

## **1.2 Существующие пути решения и реализации таких систем**

Так как данная тема не нова и беспокоит человечество довольно долгое время, существует огромное множество решений, которые применимы для создания ИНС. Каждое из них имеет как недостатки, так и преимущества.

### **Разработка системы навигации по полосам**

Основой целью авторов разработки системы навигации по полосам [2] является решение отдельно взятой проблемы. Эта система не будет заменять спутниковую навигацию, а дополнять её. В результате пользователи смогут использовать приложение, в котором будут объединены спутниковая и инерциальная системы. Приложение нацелено решить следующие задачи:

· измерять скорости по полосам, чтобы рекомендовать пользователю перестроения, если движение по полосе медленное, например, из-за того, что с левой полосы поворачивают машины. Или рекомендовать перестроиться, если на текущей полосе авария;

· решать проблему низкой точности GPS во дворах. Определить местоположение машины на извилистых дорожках придомовых территорий довольно затруднительно и GPS возвращает данные с погрешностью в районе 10 метров, если вообще ловит в такой местности. В незнакомых дворах это критично, ориентируясь на навигатор получается, что пользователь не всегда знает своё точное положение;

· определять поворот или разворот машины, так как GPS определяет маневр с опозданием. Прежде чем этот манёвр будет распознан навигатором, водитель успеет проехать несколько метров. Определяя поворот / разворот сразу, навигатор сможет быстрее перестроить маршрут, что очень важно на сложных развязках.

Приложение работает так: данные с акселерометра предварительно обрабатываются специальным образом. Для более точной работы необходимо следить за тем, чтобы положение датчика оставалось постоянным. Однако добиться этого сложно, поэтому был разработан нормализатор, целью которого является преобразование данных таким образом, чтобы ось Z была всегда вертикальной, ось X была сонаправлена с движением, а ось Y показывала отклонения от прямолинейного движения (т.е. повороты влево и вправо). Рис. 1

навигация приложение нейронный анализатор

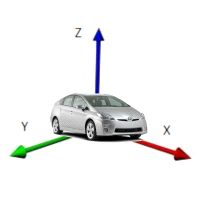


Рис. 1

В качестве классификатора используется исскуственная нейронная сеть типа feedforward с тремя hidden layers (система взаимодействующих адаптивных элементов - нейронов, каждый из которых выполняет определенное функциональное преобразование над сигналами [3]). На вход она получает данные со спутниковой системы навигации и обработанные данные с акселерометра, полученные за определенный промежуток времени.

На данный момент работа над данным проектом продолжается. Уровень точности определения действий этого приложения составляет 64%. Автор соглашается с тем, что на данный момент точность его решения низка и это не позволяет использовать его в режиме реального времени и необходимо еще провести много работ для улучшения эффективности.

### **Увеличение точности indoor навигатора**

Описанный выше метод имеет один важный недостаток - низкая точность. Чтобы решить эту проблему, можно обраться к статье [4].

В рамках исследования, Andrew R. Golding искал пути повышения точности работы системы для навигации в помещениях (indoor навигация - навигация в помещениях). Эта система была построена на базе наивного байесовского классификатора (NB - простой вероятностный классификатор, основанный на применении Теоремы Байеса со строгими (наивными) предположениями о независимости [5]), однако имела низкий уровень точности, который немногим превышал 50%. Для решения данной проблемы автор предлагает добавить так называемый data cooking. По сути это небольшой модуль, который на вход получает входные необработанные данные с датчиков, а на выходе - некоторые высокоуровневые даннные, более удобные и понятные для работы с алгоритмами машинного обучения.

Этот модуль должен решить те проблемы, которые присущи необработанным данным:

· классификатор NB предполагает независимость данных, которые подаются ему на вход. Однако данные с акселерометра, гироскопа, компаса напрямую зависят между собой. Для этого автор предлагает изменить сам подход к использованию данных с датчиков, предоставляя системе не поток данных с них, а только те данные, которые будут определены как действие. В качестве простейшего примера он приводит работу датчика света (light sensor), в котором специальный модуль определяет именно сам факт действий - включение и выключения света - и передает системе именно эти данные;

· существует проблема смещения положения датчиков, что может серьезно менять его значения. Голдинг предлагает добавить функцию, которая будет с периодичностью обновлять данные о точности, и в зависимости от точности делать поправку данных с датчиков. Примерно схожий подход имеется и в работе [2], где используется нормализатор данных с акселерометра;

· и наконец, проблема сильных шумов, которые могут привести к существенным ошибкам в определении положения. Для этого автор предлагает сделать преобразование непрерывных значений датчиков на несколько дискретных уровней. Для большей точности работы автор рассматривает возможности для автоматического задания данных дискретных уровней

Результаты этой работы впечатляют, Andrew R. Golding смог добиться очень существенных результатов. Благодаря добалению data cooking система кардинально улучшила результаты работы, увеличив точность до 98%.

### **Поиск универсального решения для навигации**

Решение узконаправленных задач всегда проще, чем поиск универсально решения. Однако, Robert E. Guinness, автор работы Поиск универсального решения для навигации [6], в рамках своего исследования постарался изучить возможности современных смартфонов для навигации в различных режимах, в том числе ходьба, бег, вождение автомобиля и движение на автобусе или в поезде. Они поставили своей целью подобрать наиболее оптимальный (с точки зрения производительности и энергопотребления) алгоритм для работы в режиме реального времени.

В рамках данной работы автором были проведены исследования широкого спектра контролируемых методов обучения для классификации, в том числе деревьев решений (DT), машины опорных векторов (SVM), наивные классификаторы Байеса(NB), сетей Байеса (BN), логистической регрессии (LR), искусственные нейронные сети (ANN) и несколько экземпляров на основе классификаторов (KSTAR, LWLand IBk). В результате данного исследования, авторы определили лучшие методы с точки зрения правильной скорости классификации. Точность работы алгоритмов приведена в таблице.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | DT | BN | LWL | KSTAR |
| Точность | 96,5% | 90,9% | 95,5% | 95,6% |

В частности, DT - алгоритм RandomForest показали лучшую общую производительность. После того, как был добавлен процесс отбора признаков для подмножества алгоритмов, производительность была немного улучшена. Кроме того, после настройки параметров RandomForest, улучшена производительность до 96,5%.

Также для работы в реальном времени, авторам было необходимо измерить вычислительную сложность классификаторов, с точки зрения затрат времени центрального процессора, необходимого для классификации. Такое исследования позволило оценить алгоритмы с точки зрения потребления энергии. В результате, классификаторы были ранжированы от самого низкого до самой высокой сложности (то есть вычислительные затраты) следующим образом: SVM, ANN, LR, BN, DT, NB, IBk, LWL и KSTAR.

Авторы пришли к выводу, что DT алгоритмы являются превосходными кандидатами для работы в такого рода системах, как с точки зрения производительности, так и с точки зрения вычислительной сложности.

**1.2 Машинное обучение**

Во всех выше перечисленных аналогах приложений для определения местоположения используются разные алгоритмы машинного обучения. Обращая внимание на статью Robert E. Guinness и высокую точность таких алгоритмов, имеет смысл и в итоговом приложении данной работы использовать машинное обучения. Для этого нужно дать определение машинному обучению и рассмотреть наиболее эффективные алгоритмы, чтобы сделать выбор в пользу лучшего, основываясь на данных, описанных в статье Robert E. и собственных.

Машинное обучение - обширный подраздел искусственного интеллекта, математическая дисциплина, использующая разделы математической статистики, численных методов оптимизации, теории вероятностей, дискретного анализа, и извлекающая знания из данных [7].Основной целью машинного обучения является полная или частичная автоматизация решения сложных нелинейных профессиональных задач в различных областях человеческой деятельности.

Алгоритмы машинного обучения делятся на различные группы в зависимости от времени обучения (предварительное обучение или обучение в процессе), а так же по методике обучения (обучение с учителем, без учителя, частичное подкрепление).

Существуют различные подходы к созданию алгоритмов для машинного обучения. Имеет смысл рассмотреть подробнее алгоритмы, показавшие высокую точность и производительность в статье Robert E. Guinness

### **Деревье решений (DT)**

Согласно наиболее общему определению, дерево принятия решений - это средство поддержки принятия решений при прогнозировании, широко применяющееся в статистике и анализе даны [8].



Рис. 2

Дерево решений, подобно его прототипу из живой природы, состоит из «ветвей» и «листьев» (Рис. 2). Ветви (ребра графа) хранят в себе значения атрибутов, от которых зависит целевая функция; на листьях же записывается значение целевой функции. Существуют также и другие узлы - родительские и потомки - по которым происходит разветвление, и можно различить случаи. Цель всего процесса построения дерева принятия решений - создать модель, по которой можно было бы классифицировать случаи и решать, какие значения может принимать целевая функция, имея на входе несколько переменных.

Общий алгоритм построения дерева принятия решений выглядит следующим образом:

) выбрать атрибут Q и поместить его в корневой узел;

) из тестовых примеров (или набора данных) для каждого значения атрибута i выбираем только те, для которых Q=i;

) далее, рекурсивно строим дерево принятия решений в этом потомке.

Общий принцип работы дерева принятия решений таков:

) дерево принятия решений строится по примерам, содержащим в себе конечное решение;

) в построенное дерево можно направлять n-ное количество атрибутов (конечно же, атрибуты должны быть того же типа, что и данные на этапе обучения);

) на основе уже изученной информации, дерево будет возвращать значение ветки, к которой принадлежит тот или иной набор атрибутов.

Данная структура позволяет упростить процесс обучения и часто используется в области компьютерных игр и робототехники, когда количество действий и варианты поведений заранее предопределены.

### **Сети Байеса (BN)**

Байесовская сеть (доверия) - это ациклический ориентированный граф, в котором каждая вершина (узел сети) представляет n-значную переменную, дуги обозначают существование непосредственных причинно-следственных зависимостей между соединенными переменными, а сила этих зависимостей количественно выражается в виде условных вероятностей, сопоставленных каждой из переменных [9].

Байесовские сети представляют собой удобный инструмент для описания достаточно сложных процессов и событий с неопределенностями. Основной идеей построения сети является разложение сложной системы на простые элементы. Для объединения отдельных элементов в систему используется математический аппарат теории вероятностей. Такой подход обеспечивает возможность строить модели с множеством взаимодействующих переменных для последующей разработки эффективных алгоритмов обработки данных и принятия решений. С математической точки зрения байесовская сеть - это модель для представления вероятностных зависимостей, а также отсутствия этих зависимостей [10].

Привлекательность байесовских моделей заключается в их высокой производительности, а также в интуитивно понятном представлении в виде графа [11]. (Рис. 3).

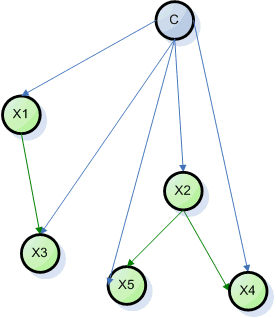


Рис. 3

**Искусственные нейронные сети (ANN)**

Искусственная нейронная сеть (artificialneuralnetwork, ANN) математическая модель, построенная по образу и подобию нервных клеток живого организма, а в частности, человеческого мозга [12].

Основным прототипом искусственной нейронной сети является биологическая нейронная сеть - совокупность нейронов (структурно-функциональная единица нервной системы, которая обрабатывает и передает информацию посредством химических и электрических сигналов.) головного и спинного мозга центральной нервной системы (ЦНС) и ганглия периферической нервной системы (ПНС). Связанные или функционально объединенные в нервной системе, они выполняют специфичные физиологические функции[13].

В отличие от биологической нейронной сети, которая до сих пор остается вопросом многих исследований и изучений в области биологии, искусственная нейронная сеть (ИНН) представляет собой упрощенную математическую модель, состоящую из узлов, именуемых искусственными нейронами, соединенных между собой связями и способными передавать и получать сигналы.

В биологической нейронной сети нейрон имеет отростки нервных волокон двух типов - дендриты, принимающие импульсы, и аксон, по которому нейрон может передавать импульс. Аксон контактирует с дендритами других нейронов через специальные образования - синапсы, влияющие на силу импульса. При прохождении синапса сила импульса меняется в определенное число раз, называемое весом синапса. Импульсы, поступившие к нейрону одновременно по нескольким дендритам, суммируются. Если суммарный импульс превышает некий порог, нейрон возбуждается, формирует собственный импульс и передает его далее по аксону. Стоит отметить, что веса синапсов могут меняться со временем, то есть будет меняться и поведение соответствующего нейрона.

Искусственный нейрон, в отличие от нейрона биологического, представляет собой более простой элемент-процессор, получающий на вход сигналы в виде каких-либо параметров для оптимизации или выходные сигналы других нейронов ИНН через некоторое количество входных каналов. Каждый входной сигнал проходит через связь, имеющую определенную интенсивность или вес. Каждый нейрон имеет некое пороговое значение. При поступлении сигнала вычисляется взвешенная сумма входов, из нее вычитается пороговое значение и в результате получается величина активации нейрона (постсинаптический потенциал нейрона, PSP). Сигнал активации преобразуется с помощью функции активации (передаточной функции) в выходной сигнал нейрона [14]. (Рис. 4).

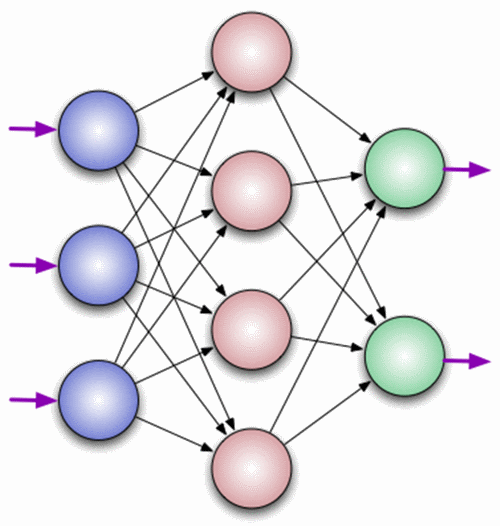


Рис. 4

Класс задач, которые можно решить с помощью нейронной сети, определяется тем, как сеть работает и тем, как она обучается. При работе нейронная сеть принимает значения входных переменных и выдает значения выходных переменных. Таким образом, сеть можно применять в ситуации, когда у Вас имеется определенная известная информация, и Вы хотите из нее получить некоторую пока не известную информацию (Patterson, 1996; Fausett, 1994).

## **.4 Проблемы**

Несмотря на то что, проблема определения местоположения машины с использование инерциальной навигации существует давно и аналоги таких систем уже используются, они до сих пор не имеют сто процентной точности.

В статье [15] перечислен ряд сложностей, с которыми разработчики сталкиваются и которые должны решить как можно более эффективным способом, чтобы увеличить надёжность результирующего приложения:

1) разные модели датчиков дают разные картинки происходящего, из-за этого одно и тоже действие на разных девайсах может выглядеть по-разному. Также показания датчиков зависят от скорости, системы подвески транспортного средства. Чтобы в результате приложение хорошо функционировало, необходимо рассматривать все возможные ситуации;

) система должна быть разработана так, чтобы уметь отличать похожие события, например перестроение из одной полосы в другую от поворота;

) нужно учитывать большую погрешность данных с датчиков, потому что это может сильно повлиять на результат. Для этого можно усреднять значения;

) использование интернета тоже может стать проблемой. Отправка большого количества данных с датчиков на сервер и назад может привести к задержке или потере данных. Таким образом, необходимо приложение, которое будет снижать использование интернета до минимума;

) ещё одна проблема это конфиденциальность. Так как приложение отправляет данные о месте нахождении пользователя, это может привести к нарушению конфиденциальности. Поэтому приложение должно зашифровывать и скрывать идентификацию пользователя, чтобы поддержать его конфиденциальность.

Анализируя теорию, описанную выше, можно сделать выводы об архитектуре будущего приложения и сформулировать более конкретные задачи для его написания.

Архитектура системы для отслеживания местоположения машины должна включать в себя следующие компоненты: сбор данных с датчиков, обработка полученных данных, отображение текущего положения машины.

· Лучшим вариантом девайса для сбора данных и отображения положения машины является мобильный девайс. Сегодня такой гаджет есть практически у любого человека и водители активно используют его вместо навигатора. Основным преимуществом этого варианта является наличие встроенных датчиков, достаточных для определения манёвра машины. Имеет смысл использовать такие датчики, как акселерометр, гироскоп, компас и GPS для определения места, в котором произошёл разрыв связи со спутником;

· Для обработки данных логичнее всего будет использовать машинное обучение. Ссылаясь на исследования, описанные в статье Robert E. Guinnessи мои личные исследования, самым простым и эффективным алгоритмом для машинного обучения будут деревья решений.

· Также стоит отметить, что в целях экономии вычислительных ресурсов телефона и заряда его батареи, стоит написать сервер-клиентское приложение, которое будет на девайсе собирать данные о манёврах и отправлять их на сервер, где с помощью машинного обучения манёвры будут классифицировать и вычисляться новое местоположение. Затем эти данные будут отправляться программе-клиенту.

Более конкретные задачи можно сформулировать так:

) Написать приложение для мобильного девайся, которое будет собирать данные Акселерометра, GPS, Компаса и Гироскопа;

) Приложение, с помощью которого можно будет размечать данные для обучения, то есть записывать показатели датчиков и явно указывать, какое действие сейчас совершается;

) Приложение, которое будет анализировать данные с помощью дерева решений;

) В приложение сбора данных добавить клиентскую часть, для отправки данных, в приложение для анализа данных добавить серверную часть, для получения данных.

**2. Практическая часть**

В данной главе будут описаны средства и методы решения поставленных выше задач.

## **2.1 Сбор данных**

Для мобильной части приложения средой разработки был выбран Intel XDK. Intel XDK - это инструмент, который позволяет создавать мобильные приложения с использованием веб-технологии. Основным плюсом этой среды разработки является то, что приложение является кроссплатформенным, а это значит, что оно сможет запускаться на мобильных девайсах с разными ОС: Android, iOS и Windows.

Цель приложения для сбора данных: записывать показатели акселерометра, гироскопа, компаса и GPS в соответствующие файлы.

Готовое приложение выглядит так Рис. 6.



Рис. 6

### **Архитектура приложения для сбора данных**

При нажатии на соответствующие кнопки начинается сбор данных и запись в файл. В файл записываются строки, содержащие время, прошедшее с момента запуска приложения и данные с датчиков.

Приложение получает данные с акселерометра, гироскопа и компаса каждые 25 мс, с GPS по мере изменения координат. Так как данные о местоположении приходят со спутника, а не вычисляются самим телефоном, изменить частоту получения данных нельзя.

Запись в файл происходит через буфер. Сначала данные записываются в соответствующие буферы, при достижение максимальной длины буфера, он записывается в файл и обнуляется. Такой метод записи в файлы экономит время: обращение к файлу на запись долгая по времени выполнения операция, записывая буфер в файл, а не отдельные данные, число обращений к файлу сокращается, а следовательно эффективность улучшается.

Также в приложении для сбора данных реализован перерасчёт показателей акселерометра.

Оси акселерометра всегда должны совпадать с осями машины: ось Х указывать в направлении движения машины, у вправо, а z наверх. Так как телефон не всегда лежит в машине неподвижно и на кочках или при поворотах может менять своё положение, направления осей телефона и машины могут расходиться. Это может приводить к серьёзным расхождениям реальных данных и полученных. Чтобы не столкнуться с такой проблемой был реализован перерасчёт данных акселерометра по осям. Для перерасчёта берётся среднее значение гироскопа по осям за 1 секунду, вычисляются матрицы поворота для расчёта новых показателей акселерометра (Рис. 7).

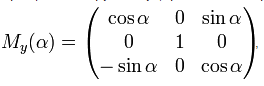
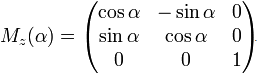
  

Рис. 7

Затем эти матрицы перемножаются:  и показатели акселерометра записанные в матрицу умножают на матрицу .

Алгоритм записи показателей акселерометра в файл. (Рис. 8).

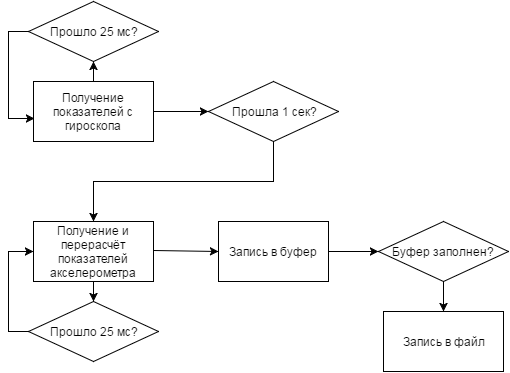


Рис. 8

## **2.2 Разметка данных**

Для того чтобы использовать машинное обучение, нужны данные, которые обучат алгоритм отличать одни действия от других. Чтобы получить данные для обучения нужно сначала собрать их, а потом разметить события, по которым алгоритм будет обучаться, то есть выбрать часть данных, указать явно действие, которое происходит и записать эти данные в файл.

Так как просто глядя на данные акселерометра и компаса сложно точно классифицировать происходящие событие, было написано приложение, которое облегчит эту задачу.

Приложение для разметки данных представляет из себя окно разделённое на три зоны, в которых параллельно воспроизводиться видео поездки, строятся графики показателей акселерометра и карта, на которой рисуется GPS трек.

Для разработки этой части приложения был выбран Qt Creator. Qt Creator - кроссплатформенная свободная интегрированная среда для разработки на С, С++ и QML. Основной плюс этой среды - наличие удобных библиотек для графического интерфейса и подробной документации.

Интерфейс приложения приведён на рисунке Рис. 9.

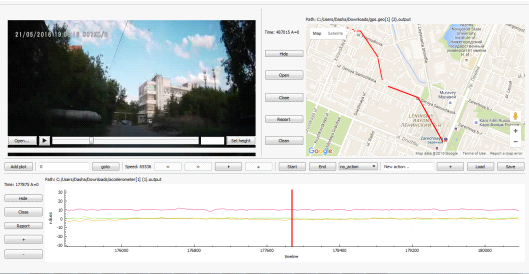


Рис. 9

### **Архитектура приложения для разметки данных**

Архитектура приложения выглядит следующим образом (Рис. 10):

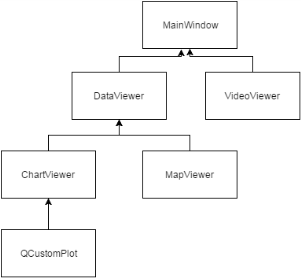


Рис. 10

Есть главное окно MainWindow, в нём есть объекты классов DataViewer и VideoViewer. VideoViewer наследуется от встроенного класса QWidget и используется для проигрывания видео. DataViewer наследуется от встроенного класса QHBoxLayout и является базовым классом для ChartViewer и MapViewer. MapViewer подгружает из интернета GoogleMap и отвечает за отображение карты и точек на ней. ChartViewer класс наследуется от QCustomPlot и отвечает за отображение графиков в реальном времени. QCustomPlot это открытая библиотека для рисования графиков с хорошей документацией и большими возможностями.

Отображение графика и GPS трека строятся параллельно видео. Для этого используется следующий алгоритм (Рис. 11):

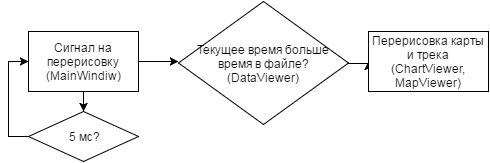


Рис. 11

Каждые пять мс. генерируется сигнал для перерисовки графика и GPS. По этому сигналу вызывается функция, которая сравнивает время с момента начала и время, записанное в файл сбора информации, которое соответствует текущим показателям датчиков. Если текущее время больше время датчиков, то запускается перерисовка.

Также в приложение реализована перемотка видео вместе с графиками и GPS треком и запись в файл размечаемых данных.

Разметка данных реализована следующим образом:

Данные постоянно записываются в буфер. После нажатия кнопки старт, данные начинают записываться в буфер с пометкой действия, которое пользователь выбрал (лэйбл), после нажатия кнопки стоп, метка действия сбрасывается и запись продолжается. После нажатия кнопки сохранить буфер записывается в выбранный файл.

## **2.3 Анализатор данных**

Для обучения было решено использовать Python с библиотекой SciKit-Learn. Эта библиотека включает в себя большое количество алгоритмов машинного обучения, а также имеет отличную документацию, что делает ее более привлекательной для тех, кто только начинает работать в области анализа данных. В ходе предварительного исследования было решено сконцентрироваться на анализе алгоритмов Decision Tree (деревья решений), так как они показывали наилучшие результаты. Эти алгоритмы часто используются в задачах, в которых обьекты имеют категориальные признаки и используется для задач регрессии и классификации.

### **Обучение**

Машинное обучение будет использоваться только для скорости и анализировать данные акселерометра по оси Х. Распознаваться будут 4 состояния: машина стоит, машина едет с равномерной скоростью, машина ускоряется и машина тормозит.

Первым делом нужно подготовить данные для обучения. Для этого:

) Округляются значения;

) Создаётся новый массив, в котором содержится среднее значение и среднеквадратичное отклонения для выборки из 10.

Когда машина стоит среднее значения показателей акселерометра и среднеквадратичное отклонение находятся в районе нуля.

Когда машина едет с равномерной скорость, среднее значение находиться в районе нуля, а среднеквадратичное отклонение выше нуля.

Когда машина ускоряется, среднее значение значительно выше нуля, когда замедляется, значительно ниже. В этих двух случаях среднеквадратичное значение не так важно.

Затем для каждого нового значения вычисляется его лэйбл. Лэйбл 1 значит, что машина стоит, лэйбл 2, что машина равномерно едет, лэйбл 3, что машина ускоряется и лэйбл 4, что машина замедляется. Сохраняется значение лэйбла, который чаще всего встречался в выборке из 10 элементов.

После строиться регрессия RandomForestClassifier по данным полученным выше. Алгоритм Random Forest называется «Случайный Лес», потому что для полученных данных он создает множество деревьев приятия решений и потом усредняет результат их предсказаний. Важным моментом тут является элемент случайности в создании каждого дерева. Ведь понятно, что если мы создадим много одинаковых деревьев, то результат их усреднения будет обладать точностью одного дерева.

### **Расчёт скорости**

Показатели акселерометра для расчёта и время, соответствующее им, загружаются в массивы, которые должны быть подготовлены для дальнейшего анализа (см. 2.3.1). Для каждого значения вызывается predict, который возвращает лэйбл, указывающий действие, которое происходило в момент снятия данных. Для вычисления текущей скорости используется среднее значение акселерометра, время и лэйбл.

Если лэйбл равен 1, то и скорость равна нулю, потому что машина находится в состоянии покоя.

Если лэйбл равен 2, то машина едет равномерно, и скорость равна предыдущей скорости.

Если лэйбл равен 3, то машина ускоряется, и формула для вычисления скорости , где  - это новое значение скорости,  - предыдущее значение скорости,  - среднее значение акселерометра за время .

Если лэйбл равен 4, то машина тормозит и формула для вычисления выглядит как при лэйбле 3, только значение  отрицательно.

### **Погрешность**

Погрешность такого алгоритма получилась относительно небольшой. Разница показателей скорости, полученных в ходе его работы с реальной скоростью машины (видео с регистратора) колеблется в пределах 3 км/ч. Конечно у данного алгоритма есть минусы, он плохо работает на круговых развязках и резких поворотах. Погрешность и ошибки можно уменьшить, увеличив количество данных для обучения.

## **2.4 Клиент-серверная часть**

Чтобы завершить итоговое приложение, нужно доработать приложение для анализа данных и приложения для их сбора. В первом случае надо дописать серверную часть, чтобы принимать данные от клиента и возвращать новые вычисленные координаты. Во втором случае дописать клиентскую часть, чтобы общаться с сервером.

### **Клиентская часть приложения для сбора данных**

Пока есть связь с GPS, на карте отображается местоположение машины, вычисленное спутником. При потере связи с GPS, клиент начинает записывать данные акселерометра, компаса и последнюю полученную координату со спутника в буфер. Когда буфер переполняется, клиент отправляет его на сервер и при получении ответа с новой координатой, отображает её. (Рис. 13)

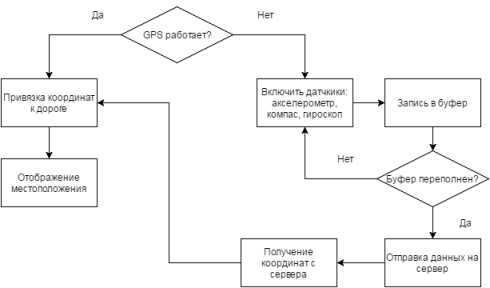


Рис. 13

Также для увеличения точности и отображения трека машины только по дорогам, используется привязка к дорогам. У Google Maps есть служба, которая привязывает координаты к дорогам (snapToRoads). Она возвращает набор данных с точками, привязанными к наиболее вероятным дорогам, по которым проехал автомобиль. Дополнительно можно запросить интерполяцию точек для получения пути, максимально соответствующего геометрии дороги. После получения координат с сервера, отправляется запрос на адрес https://roads.googleapis.com/v1/snapToRoads, который возвращает координаты, привязанные к дорогам. Затем эти координаты отображаются. Интерфейс приложения приведён на Рис. 14.

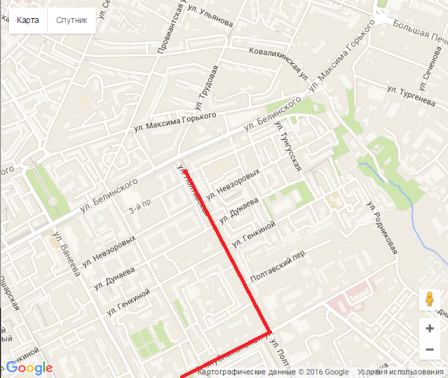


Рис. 14

### **Серверная часть приложения для анализа данных**

Так как анализ данных написан на Python, то и серверная часть будет написана на нём. Для Python есть удобная библиотека для создания сервера и обработки запросов, это библиотека Flask. Flask является микрофреймворком для создания вебсайтов на языке Python

После получения данных сервером, они парсятся, создаются массивы, в которых хранятся данные с акселерометра по оси Х, время акселерометра и данные компаса. Показатели акселерометра анализируются с помощью случайных деревьев. В результате получается массив, содержащий лэйбл, среднее ускорение и среднеквадратичное отклонение. Затем для каждого из этих значений вычисляется скорость. По значению скорости и среднего угла компаса вычисляется расстояние, которое проехала машина. Координаты (последние данные, полученные от GPS) из широты / долготы переводятся в метры (система UTM), к ним прибавляется расстояние, которое проехала машина и затем эти данные переводятся в широту / долготу и отправляются клиенту. (Рис. 15.)

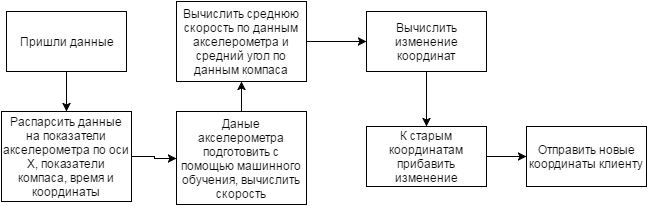


Рис. 15

# **3. Анализ точности**

В ходе работы удалось реализовать приложение, которое помогает отслеживать местоположение автомобиля, при потере связи со спутниками GPS.

Точность приложения тестировалась в следующих условиях: ровная трасса с небольшим числом поворотов, городская застройка - большое число поворотов, местность с перепадом высоты.

На трассе приложение показывает высокую точность. При отсутствии связи с GPS в течение 5 минут реальное местоположение полностью совпадало с высчитанными координатами. При отсутствии связи с GPS в течение 10 минут, появляется небольшая погрешность в пределах 3 метров, после 20 минут разрыва подключения погрешность достигала 7 метров.

В плотной городской застройке точность приложения была значительно ниже. При отсутствии связи со спутником в течение 5 минут погрешность составляет 2 метра. При отсутствии связи в течение 10 минут погрешность была в пределах 6 метров, после 20 минут погрешность составляла 15 метров. Такую невысокую точность можно объяснить тем, что количество поворотов, которые совершает машина в городской застройке значительно больше, чем на трассе. Совершая повороты, машина сбрасывает скорость, это значит, что скорость чаще меняет свои значения, что приводит к тому, что погрешность в её вычислении увеличивается. Увеличить точность можно, если расширить базу данных для обучения алгоритма.

Погрешность на местности с перепадом высоты (подъёмы, спуски) тоже довольно высокая. После 5 минут отсутствия связи со спутником погрешность варьируется в пределах 3 метров, после 10 в пределах 6 метров, и после 20 минут в пределах 10 метров. Такую погрешность можно объяснить тем, что выравнивание осей акселерометра по осям машины происходит примерно раз в одну секунду и не всегда перерасчёт показателей совпадает с началом подъёма или спуска. Это искажает данные, а вследствие этого скорость и вычисляемые координаты.

Для наглядности приведена схема. (Рис. 16).



Рис. 16

# **Заключение**

Результаты данной работы показывают, что написание определения местоположения без доступа к GPS является сложной и комплексной задачей. Использование алгоритмов машинного обучения способно в разы увеличить точность определения новой координаты.

Написанное в ходе этой работы приложения, нельзя использовать массово, так как у него есть ряд недостатков:

) Невысокая точность. Не смотря на то, что приложение имеет точность выше, чем у большинства аналогов, погрешность в вычислениях всё ещё большая. Улучшить точность определения местоположения можно, например, расширив базу для машинного обучения, а также добавив алгоритмы машинного обучения для определения угла поворота.

) Данные для обучения собирались с использованием нескольких девайсов. Для массового использования это число должно доходить до нескольких десятков, потому что разные девайсы будут иметь разные показания акселерометров на одном и том же манёвре, что может приводить к неправильной классификации действий. Эта проблема тоже решаема расширением базы машинного обучения.

Несмотря на свои недостатки, приложение всё-таки показывает неплохие результаты, относительно аналогов и может быть использовано в качестве основы для дальнейших исследований.

# **Список литературы**

1. ТеКнол. Инерциальная навигация - инерциальные навигационные системы (ИНС), [Электронный ресурс]. Режим доступа: http://www.teknol.ru/analitycs/ins.

2. Alexander Ponomarev. Machine learning in navigation devices: detect maneuvers using accelerometer and gyroscope, [Электронный ресурс]. Режим доступа: https://blindmotion.github.io/2015/04/11/ml-in-navigation/.

3. Е.Н. Ефимов, Т.Я. Шевгунов. Построение нейронных сетей прямого распространения с использованием адаптивных элементов. - Московский авиационный институт (национальный исследовательский университет). 13 августа 2012 г.

4. Andrew R. Golding, Neal Lesh. Indoor navigation using a diverse set of cheap, wearable sensors. - MITSUBISHI ELECTRIC RESEARCH LABORATORIES, TR99-32 December 1999. (http://www.merl.com/publications/docs/TR99-32.pdf).

5. Наивный байесовский классификатор [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/ Наивный\_байесовский\_классификатор

6. Robert E. Guinnes. Beyond Where to How: A Machine Learning Approach for Sensing Mobility Contexts Using Smartphone Sensors. - Sensors 2015, 15 (5), 9962-9985; doi:10.3390/s150509962.

. Mitchell T. Machine Learning. - McGraw-Hill Science/Engineering/Math, 1997. ISBN 0-07-042807-7.

8. Деревья решений и алгоритмы их построения [Электронный ресурс]. Режим доступа: http://datareview.info/article/derevya-resheniy-i-algoritmyi-ih-postroeniya/

9. Pearl, J. Probabilistic. Reasoning in Expert Systems: Networks of Plausible Inference / J. Pearl. - San Francisco: Morgan Kaufmann, 1988. -552 p.

. Bidjuk P.I., Terent'ev A.N. Postroenie i metody obuchenija bajesovskih setej. - Tavrijs'kij visnik informatiki i matematiki, No2/2004 - s. 139-154.

. Desmarais, M.C., & Baker, R.S. (2012). A review of recent advances in learner and skill modeling in intelligent learning environments. User Modeling and User-Adapted Interaction. - Volume 22 Issue 1-2, April 2012. Pages 9-38. (http://dl.acm.org/citation.cfm? id=2198192).

12. Каллан Р. Основные концепции нейронных сетей / Пер. с англ. М.:Вильямс. - 2001. 287 с.

13. Уоссермен Ф. Нейрокомпьютерная техника / пер. с англ. Ю.А. Зуев, В.А. Точенов. [Электронный ресурс]. Режим доступа: http://www.codenet.ru/progr/alg/nks/.

. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. - М.: Горячая линия - Телеком, 2001. - 382 с. - ISBN 5-93517-031-0.

15. Gunjan Chugh, Divya Bansal and Sanjeev Sofat. Road Condition Detection Using Smartphone Sensors: A Survey. - International Journal of Electronic and Electrical Engineering. ISSN 0974-2174, Volume 7, Number 6 (2014), pp. 595-602.

16. Random Forest с примерами на R. [Электронный ресурс]. Режим доступа: http://www.algorithmist.ru/2012/05/random-forest-r.html.

. Нейронные сети. [Электронный ресурс]. Режим доступа: http://www.statsoft.ru/home/textbook/modules/stneunet.html.

18. Google Maps. Привязкак к дорогам. [Электронный ресурс]. Режим доступа: https://developers.google.com/maps/documentation/roads/snap? hl=ru#\_2.

. Коваленко В.В., Лысов. А.Н. Малогабаритная инерциальная система. Учебное пособие. - 2010.

. IX САНКТ-ПЕТЕРБУРГСКАЯ МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ ПО ИНТЕГРИРОВАННЫМ НАВИГАЦИОННЫМ СИСТЕМАМ. [Электронный ресурс]. Режим доступа: http://www.elektropribor.spb.ru/cnf/icins02/rfinprg.html.

# **Приложение**

Клиент, карта

Map.js

/\*jslint browser:true, devel:true, white:true, vars:true \*/

/\*global $:false, intel:false, acc:false, moment:false \*/

/\*global addClass:false, removeClass:false \*/

.map = window.map || {}; // don't clobber existing acc object

// The console.log() messages sprinkled in this file are for instruction and debug.

// If you reuse this code you do not need to include them as part of your own app.

// Set to «true» if you want the console.log messages to appear.

.LOG = true;.consoleLog = function() { // only emits console.log messages if acc.LOG!= false

«use strict»;(map.LOG) {args = Array.prototype.slice.call (arguments, 0);.log.apply (console, args);

}

};

// TODO: add use of browser DeviceMotionEvent and DeviceOrientationEvent

// see: http://www.html5rocks.com/en/tutorials/device/orientation/

var \_map;lines = [];apiKey = 'AIzaSyBgI8P5iu\_CVhG\_hBKVQAiNdzS8ShIce5o';

.addLine = function (lat\_current, lng\_current, lat\_next, lng\_next) {.consoleLog (lat\_current);flightPlanCoordinates = [

{lat: lat\_current, lng: lng\_current},

{lat: lat\_next, lng: lng\_next}

];flightPath = new google.maps. Polyline({: flightPlanCoordinates,: true,: '#FF0000',: 1.0,: 2

});.setMap(\_map);.push(flightPath);newLoc = new google.maps. LatLng (lat\_next, lng\_next);

\_map.setCenter(newLoc);

}

.removeLine = function(id) {= lines[id].getPath().getAt(0);[id].setMap(null);.splice (id, 1);

\_map.setCenter(prevCenter);

}

map.removeAllLines = function () {= lines[0].getPath().getAt(0);

\_map.setCenter(firstCenter);.forEach (function (element) {.setMap(null);

});.splice (0, lines.length);

}

.initialize = function () {.consoleLog («googleMaps initialization»);myOptions = {: new google.maps. LatLng (56.317168, 44.025813),: 15,: google.maps. MapTypeId.ROADMAP,: true

};

\_map = new google.maps. Map (document.getElementById («map\_canvas»), myOptions);.consoleLog («googleMaps initialization done»);

//map.runSnapToRoad(123);

}

// Snap a user-created polyline to roads and draw the snapped path.runSnapToRoad = function(path) {pathValues = [];(var i = 0; i < ChileTrip1.length; i++) {.push (ChileTrip1 [i].toUrlValue());

}

map.consoleLog («ebana»);.get ('https://roads.googleapis.com/v1/snapToRoads', {: true,: apiKey,: pathValues.join(«|»)

}, function(data) {.processSnapToRoadResponse(data);.drawSnappedPolyline();newLoc = new google.maps. LatLng (lat\_next, lng\_next);

\_map.setCenter(newLoc);

});.consoleLog (jQuery.);

}

// Store snapped polyline returned by the snap-to-road method..processSnapToRoadResponse = function(data) {= [];= [];(var i = 0; i < data.snappedPoints.length; i++) {latlng = new google.maps. LatLng (.snappedPoints[i].location.latitude,.snappedPoints[i].location.longitude);.push(latlng);.push (data.snappedPoints[i].placeId);

}

}

// Draws the snapped polyline (after processing snap-to-road response)..drawSnappedPolyline = function() {snappedPolyline = new google.maps. Polyline({: snappedCoordinates,: 'black',: 3

});.setMap(map);.push(snappedPolyline);

}

Клиент, расчёт скорости.

Speed.py

#!/usr/bin/python

(\_\_doc\_\_)

# Import the necessary modules and librariesthreadingtimenumpy as npmatplotlib.pyplot as pltscipy import statssklearn.tree import DecisionTreeRegressorsklearn.ensemble import RandomForestClassifiersklearn import preprocessing

.set\_printoptions (precision=3, suppress=True)\_for\_position = []

get\_acc\_array (raw\_X, values):

# initialize new\_X array\_X = ()

# round by\_X = np.around (raw\_X, decimals=1)

# reshape in 'rows' = 'len (raw\_X)/values', columns = 'values'= len (raw\_X)/values\_X.resize (rows\*values, 1)\_X = np.array (raw\_X).reshape (rows, values)

# this axis need to get line regrassion parametersrow in raw\_X:\_X = np.append (new\_X, [np.average(row), np.std(row)])

# print np.array([np.average(row), np.std(row)]);new\_X

get\_label\_array (raw\_Y, values):\_Y = ()\_Y = np.array (raw\_Y).astype(int)\_Y.resize (len(raw\_Y)/values, values)row in raw\_Y:= np.bincount(row)

# print np.argmax(counts)\_Y = np.append (new\_Y, [np.argmax(counts)])new\_Y

get\_grid(data):\_min, x\_max = data[:, 0].min() - 1, data[:, 0].max() + 1\_min, y\_max = data[:, 1].min() - 1, data[:, 1].max() + 1np.meshgrid (np.arange (x\_min, x\_max, 0.01),.arange (y\_min, y\_max, 0.01))

def plot\_data (X\_test, Y\_predicted, X, Y, lim\_x, lim\_y):.figure().ylim (lim\_y).xlim (lim\_x), yy = get\_grid (X\_test[:, [0, 1]])

#predicted = regr.predict (np.c\_[xx.ravel(), yy.ravel()]).reshape (xx.shape).pcolormesh (xx, yy, Y\_predicted, cmap='seismic').scatter (X[:, 0], X[:, 1], c=y, s=len (X[:, 1]), cmap='seismic').show()

calculate\_speed (data, times, regr, values):

# TODO: static vars in C-style?time\_indexspeedprev\_time

\_time = times [values\*time\_index] - prev\_time\_data = data [time\_index]= get\_speed (speed, delta\_time/1000, speed\_data, regr, values)

\_time = times [values\*time\_index]«Time:%f»% prev\_timespeed\_data\_index += 1

time\_index < len(times):\_call\_time = (times [values\*time\_index] - prev\_time)/1000.0. Timer (next\_call\_time, calculate\_speed, [data, times, regr, values]).start()«Next call:%f»% next\_call\_time:len(times)«time is out % d»% time\_indexspeed

get\_speed (speed, time, data, regr, values):

i= 0

# predictdata= np.array(data).reshape (1, 2)\_test = regr.predict(data)= data.item((0, 0))acceleration

#global t\_s

#t\_s = t\_s + acceleration \* time \* 3.6

#print «C speed is % 2.f»% t\_sY\_test[0] == 1:= 0\_s = 0Y\_test[0] == 2:= speedY\_test[0] == 3:= speed + acceleration \* time \* 3.6Y\_test[0] == 4:= speed + acceleration \* time \* 3.6 #! acceleration have sign +/-

# print speed«Predicted % d»% Y\_test[0]«M speed is % 2.f»% speed\_for\_position.append(speed)= i + 1speed

##########################################

# Start point ############################

##########################################start\_speed():= 10

# load check data= open («test\_data/accelerometer\_test.output»).readline() # skip the header= np.loadtxt (f2, delimiter=', ', usecols=(0,2))\_X\_test = data2 [:, 1]\_test = get\_acc\_array (raw\_X\_test, values).reshape (len(raw\_X\_test)/values, 2)

# predict, yy = get\_grid (X\_test[:, [0, 1]])\_test = regr\_1.predict (np.c\_[xx.ravel(), yy.ravel()]).reshape (xx.shape)

# plot

#plot\_data (X\_test, Y\_test, X, y, [-5.0, 5.0], [-0.1, 1.5])

#exit();

# load time data= data2 [:, 0]

#print times

# start calculating speed\_index = 7530/values\_time = 0= 0\_speed (X\_test, times, regr\_1, values)

init():= 10=10

= open («train\_data/speed\_acc\_data.output»).readline() # skip the header

# load accelerometer training data

# 0 column is time

# 1 is X (turns)

# 2 is Y (speed)

# 3 is Z (vertical)

# 4 is action\_id= np.loadtxt (f, delimiter=', ', usecols=(2,4)).close()\_X = data[:, 0] # len is N= len (raw\_X)/values= get\_acc\_array (raw\_X, values).reshape (rows, 2)len(X)

# load action\_ids (labels)

# 0 is not used

# 1 is wait

# 2 is constant motion

# 3 is acceleration

# 4 is deceleration\_Y = data[:, 1]y= get\_label\_array (raw\_Y, values)len(y)

# Fit regression modelregr\_1\_1 = RandomForestClassifier (n\_estimators=trees)\_1.fit (X[:, [0,1]], y)(regr\_1.feature\_importances\_)