СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

БГД	-	блок генерации данных
ГЛОНАСС	-	Глобальная Навигационная Спутниковая Система
ИНС	-	инерциальная навигационная система
КНА	-	комплексированная навигационная аппаратура
КНС	-	комплексированная навигационная система
НАП	-	навигационная аппаратура пользователя
ОС	-	операционная система
ПО	-	программное обеспечение
ЭВМ	-	электронно-вычислительная машина
CSV	-	Comma-Separated Values
GPS	-	Global Positioning System
NMEA	-	National Marine Electronics Association

ВВЕДЕНИЕ

За последние годы количество устройств, работающих с системами навигации, резко возросло, и их становится все больше с каждым днем. Область применения системы навигации действительно обширна – она находит свое приложение в различных сферах, таких как наука, технологии, экономика, туризм, исследования и наблюдения. Глобальные системы спутниковой навигации уже давно используются в стандартной навигации (авиация и мореплавание), и чем дешевле и доступнее становится технология, тем больше появляется областей, в которых их применяют. К ним можно отнести археологию, геофизику, геологию, картографию, географию, экологию, геодезию, промышленность, спутниковый мониторинг транспорта, туризм и геотеггинг.

Наряду со спутниковой системой навигации существуют инерциальные системы навигации, которые имеют схожие области применения, но принципы их работы отличаются. Главной особенностью методов инерциальной навигации является автономность. На этапах тестирования и отладки комплексированной навигационной системы необходимо, чтобы все составляющие системы работали корректно и обеспечивалась высокая точность измерений. Комплексированная навигационная система подвержена воздействию различных факторов, отрицательно влияющих на ее работу, что в конечном итоге отражается на результатах измерений. На этих этапах проводится тщательная проверка всех составных частей для обеспечения корректной работы всей системы.

Актуальность задачи заключается в необходимости оперативной обработки достаточно больших объемов данных, их анализа и приема от различных навигационных модулей и эмуляторов инерциальных навигационных систем.

Цель выполнения данной работы – повышение эффективности процесса оценки характеристик комплексированной навигационной аппаратуры.

Пояснительная записка состоит из введения, исследовательского, конструкторского, технологического раздела, списка литературы и двух приложений. В исследовательском разделе рассматривается актуальность выбранной темы, исследование предметной области, анализ существующих аналогичных программных решений, структура входных и выходных данных программного модуля, состав выполняемых функций ПМ ОХА, приведены цели и задачи разработки, а также ее практическая значимость.

В конструкторском разделе были проанализированы языки программирования, среды разработки, описана архитектура программного модуля, алгоритм работы и схема данных; также приводится описание формата данных и конфигурационных файлов программного модуля, указаны формулы для расчета статистических параметров, описана разработка графического интерфейса пользователя. В технологическом разделе рассматриваются особенности программирования, методы отладки и тестирования, виды ошибок и способы их обнаружения; приводится описание отладки в среде Qt Creator с помощью отладчика gdb, тестирования с помощью класса QtTest и библиотеки для представления инженерно-технической информации. Приложение 1 содержит руководство оператора, а Приложение 2 – текст программы.

1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

2. Актуальность

Сегодня системы спутниковой навигации и инерциальные навигационные системы затрагивают многие аспекты нашей жизни: встроенные автомобильные навигаторы, GPS-приемники в поездах и автобусах, информирующие пассажиров о точном времени прибытия транспорта, геотеггинг при фотографировании с мобильных устройств, и это лишь малая часть. Помимо пользовательской аппаратуры системы навигации активно используются в промышленности, в военных целях, где погрешность точности измерений должна быть минимальна. Так, для ряда задач в геодезии десятисантиметровой точности может быть недостаточно. В частности, при наличии десятисантиметровой ошибки площадь в 600 квадратных метров может измениться на 10 $м^2$ в большую или меньшую сторону. В военном деле точность определения координат становится принципиальным вопросом, когда речь заходит о высокоманевренных объектах, к примеру, БПЛА или истребители. Поэтому перед вводом в эксплуатацию очень важно, чтобы навигационное оборудование соответствовало заявленным характеристикам, а в ряде случаев это требование является определяющим. Например, очень важно точное определение места вызова в случае необходимости оказания экстренной помощи, однако исследования показали, что в 60% экстренных вызовов с мобильных телефонов не удалось установить точное местоположение (по сравнению с 2% вызовов со стационарных телефонов). В связи с этим возникает острая необходимость в оценке характеристик навигационной аппаратуры.

3. Исследование предметной области

Запуск первого искусственного спутника земли способствовал развитию возможностей спутниковой навигации, но создание самих спутниковых навигационных систем началось только в 70-х годах прошлого века для обеспечения решения военных задач. В последние годы глобальные системы спутниковой навигации получают все большее распространение и используются как в военных целях, так и в гражданских.

На сегодняшний день в мире существует несколько различных систем навигации, имеющих разный уровень развития и территориального охвата.

Глобальная Система Позиционирования (GPS – Global Positioning System), разработанная Министерством обороны США, на данный момент является основной системой навигации наряду с системой ГЛОНАСС. Ее разработка была вызвана необходимостью создания средств наведения высокоточного оружия дальнего действия для военных, и, кроме того, требовалась универсальная система навигации, которую можно было бы массово применять в армии. В итоге было принято решение о создании системы точного позиционирования путем объединения этих двух систем. Министерство обороны США, начиная с 1960-х годов, стало развивать идею создания всепогодной, глобальной, постоянно доступной, сверхточной системы навигации и позиционирования. После запуска 24-го спутника в марте 1994 года было завершено формирование спутниковой системы GPS. А еще до его запуска США официально объявили, что для гражданских потребителей всего мира данная система поддерживает 100-метровую точность позиционирования.

В первоначальной концепции разработки спутниковой системы считалось, что точности в 100 метров для гражданских применений будет достаточно, однако во время испытаний в конце 70-х годов оказалось, что коды стандартной точности позволяют добиться гораздо лучших результатов, поэтому реальная точность составляла приблизительно 30 метров. Для того, чтобы обеспечить преимущество военных, для гражданских пользователей умышленно вводилось ограничение точности (режим избирательного доступа). Вводились ошибки как в навигационные данные, передаваемые спутниками, так и в эталонные сигналы времени. Точные данные для военных были доступны всегда.

После активного применения в гражданских целях системы GPS и ее расширения стандартной 100-метровой точности оказалось недостаточно для потребностей пользователей. Такой точности было недостаточно даже для навигации небольшой частной яхты, не говоря уже о более серьезном применении, например, в гражданской авиации или в военных целях.

Далее на рисунке 1.1 приводится иллюстрация, отражающая видимость спутников из некоторой точки с поверхности планеты. В определенный момент времени при идеальных условиях из данной точки количество видимых спутников составляет 12 единиц.

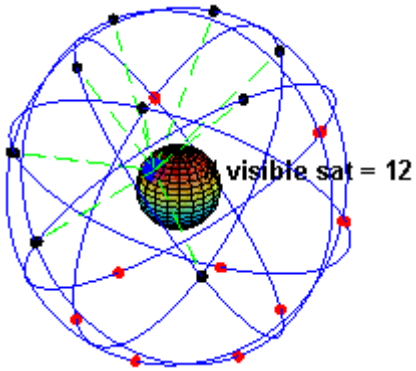

Рисунок 1.1 - Орбиты спутников системы GPS (с видимым количеством спутников)

Глобальная Навигационная Спутниковая Система (ГЛОНАСС) – отечественная система навигации, разработанная Министерством обороны. Группировка ГЛОНАСС включает в себя 24 спутника. Военно-космические силы России являются ответственными за испытание и управление системой. Развернутая в 1995 году система ГЛОНАСС не была широко распространена ввиду недостатка финансирования и небольшого срока службы спутников. В 2001 году после принятия целевой программы развития произошло «перерождение» системы, и в 2010 году ГЛОНАСС смог возобновить свою полноценную работу. Ожидается, что к 2025 году система будет глубоко модернизирована.

ГЛОНАСС предназначена для оперативного определения точного времени, координат, составляющих вектора скорости и других параметров, обеспечивая этой информацией потребителей наземного, воздушного, морского, космического базирования без ограничения на их число. По указу Президента как для российских, так и для иностранных пользователей доступ к сигналам системы ГЛОНАСС предоставляется на безвозмездной основе и без ограничений. Принцип действия системы ГЛОНАСС схож с американской системой GPS за исключением некоторых моментов. Основное отличие заключается в том, что спутникам ГЛОНАСС обеспечивается большая стабильность за счёт отсутствия резонанса (синхронности) с вращением планеты в своём орбитальном движении. Таким образом, нет необходимости в дополнительных корректировках спутников системы ГЛОНАСС на протяжении всего срока их активного использования. Тем не менее срок службы космических аппаратов системы значительно короче.

Помимо основных двух глобальных спутниковых систем на сегодняшний день существуют еще две системы, находящиеся на стадии разработки: европейская Galileo и китайская Compass (BeiDou).

Галилео (Galileo) – совместный проект Евросоюза и Европейского космического агентства, который был анонсирован в 2002 году. Изначально планировалось, что около 30 спутников этой системы будут находиться на околоземной орбите к 2010 году, но реализация плана не удалась. Ожидается, что только после 2020 года система Галилео сможет функционировать полноценно.

Бэйдоу (Compass / BeiDou) – китайская спутниковая система навигации. Планируется использование различных орбит для трех составляющих космического сегмента спутниковой системы: 27 спутников будут находиться на околоземной орбите, 3 спутника на геосинхронной орбите и 5 спутников на геостационарной орбите. Навигационная система была введена в эксплуатацию в конце 2011 года после запуска десятого спутника. На данный момент Бэйдоу работает лишь как региональная система навигации (Азия, Тихоокеанский регион), но к 2020 году планируется развить систему до глобального масштаба.

Помимо глобальных в настоящее время на стадии разработки также находятся две региональные спутниковые системы: индийская IRNSS и японская QZSS.

IRNSS — индийская навигационная спутниковая система. Планируется, что она будет использоваться только в Индии. В 2008 году был запущен первый спутник системы. Предполагаемое общее количество спутников системы IRNSS — 7.

QZSS — японская квази-зенитная спутниковая система (Quasi-Zenith Satellite System), задуманная в 2002 году. Система планировалась как средство коммерциализации, предоставляющее набор услуг для подвижной связи, вещания и широкого использования для навигации в Японии и соседних районах Юго-Восточной Азии. В 2010 году был запущен первый спутник системы QZSS. Предполагается, что космический сегмент будет состоять из трёх спутников, находящихся на геосинхронных орбитах, а также собственной системы дифференциальной коррекции.

Далее на рисунке 1.2 приводится сравнительное изображение различных орбит спутниковых систем навигации: GPS, ГЛОНАСС, Galileo, Compass, с орбитами Международной космической станции (ISS), телескопа Хаббл (Hubble) и серии спутников Иридиум (Iridium) на низкой орбите, а также геостационарной орбиты (GEO), средней околоземной орбиты (MEO) и номинального размера Земли.

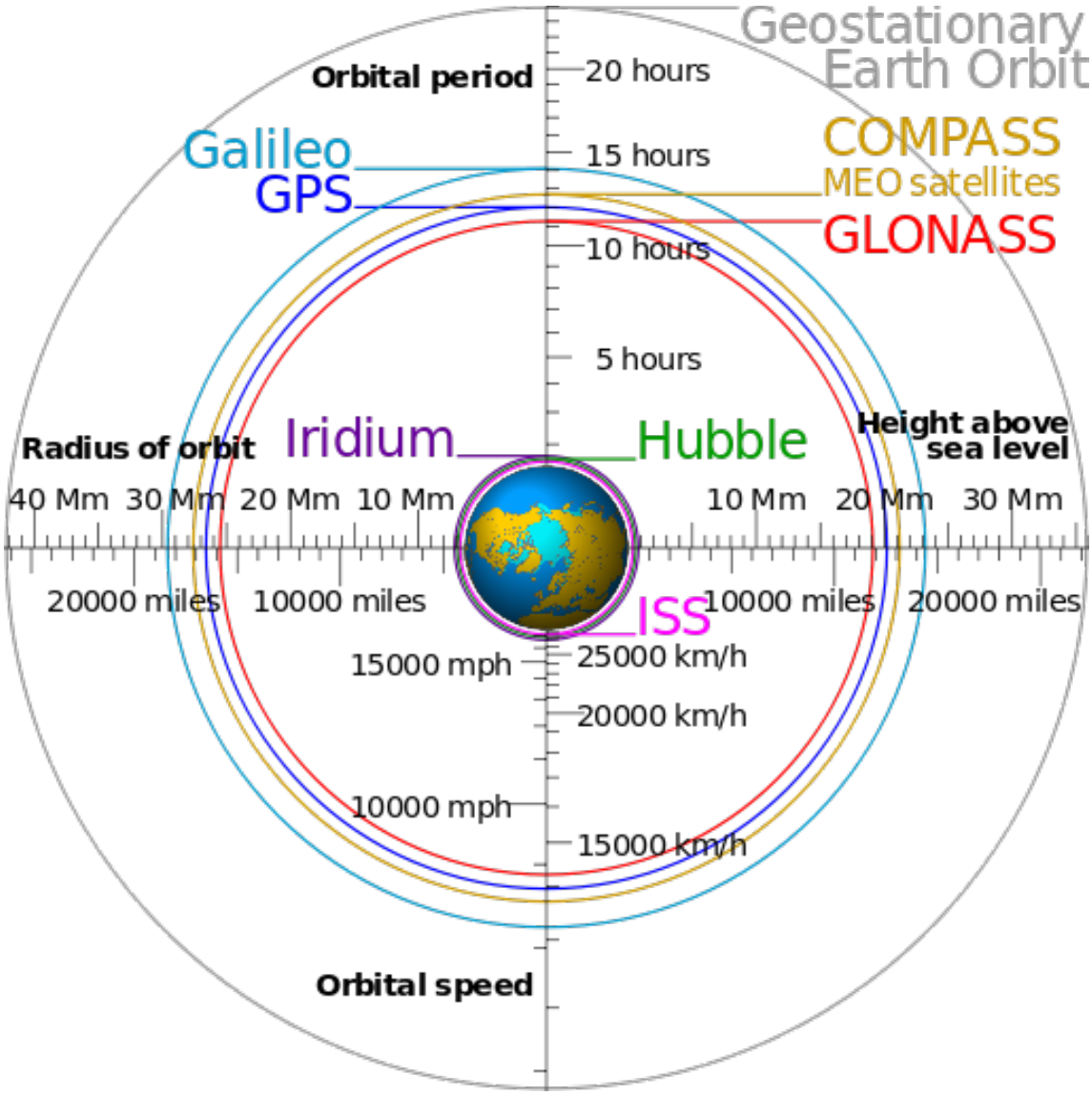

Рисунок 1.2 - Сравнение различных орбит

Как и к любой системе, к спутниковой навигационной системе предъявляются определенные требования:

- глобальность навигационно-временного обеспечения потребителей, т.е. обеспечить пользователя возможностью определения своих координат в любой точке планеты;
- непрерывность определения навигационно-временных параметров в любое время;
- неограниченность числа пользователей системы;
- высокая точность навигационно-временных определений параметров.

Для соответствия последнему требованию, представляющему интерес в данной работе, используются сложные радиосигналы, которые излучаются с навигационных спутников; правильный выбор рабочего созвездия навигационных спутников, то есть тех навигационных спутников, сигналы которых будут приниматься в обработку при

навигационных определениях; излучения достаточно мощных сигналов с навигационных спутников; использование высокоточной информации о параметрах движения навигационных спутников; оптимизации алгоритмов обработки сигналов в аппаратуре потребителя и рядом других факторов.

Обеспечение выполнения требований возможно только в сетевой структуре спутниковой радионавигационной системе, поэтому её основу составляют 3 подсистемы (сегмента), обеспечивающие высокие эксплуатационные характеристики (рисунок 1.3):

- космический сегмент (подсистема космических аппаратов);
- сегмент управления (подсистема контроля и управления);
- сегмент потребителей (навигационная аппаратура потребителей).

Помимо трех сегментов в последнее время в структуру часто включается еще одна подсистема – функциональное дополнение, позволяющее расширять функциональные возможности системы (дифференциальные режимы работы спутниковой системы).

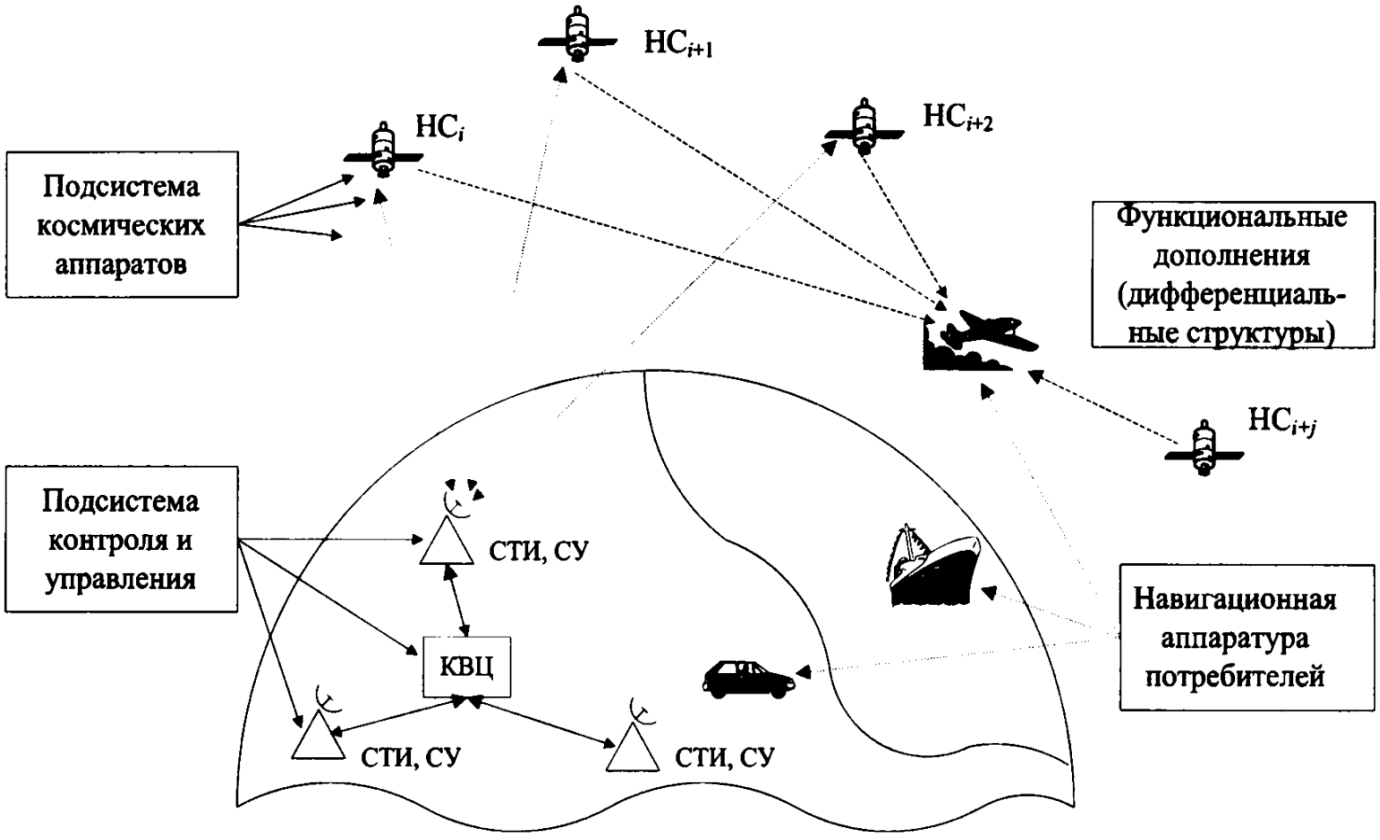

Рисунок 1.3 - Глобальная спутниковая радионавигационная система

Космический сегмент состоит из определенного числа спутников (штатное количество – 24), основной задачей которых является формирование в околоземном пространстве сплошного радионавигационного поля. 24 спутника необходимо для обеспечения связи в любое время как минимум с 4 точками. Временной и частотный эталоны, хранящиеся на борту, необходимы для обеспечения практически синхронного излучения сигналов всеми навигационными спутниками, что требуется при реализации режима пассивных дальномерных изменений в навигационной аппаратуре потребителей.

Сегмент управления состоит из координатно-вычислительного центра (КВЦ), систем управления и станций траекторных измерений (СТИ) для мониторинга спутников, а также наземного эталона времени и частоты. Станции траекторных измерений размещают как можно более равномерно по всей поверхности планеты. Координаты станций определены с максимальной точностью в трех измерениях. При полете спутников в зоне видимости СТИ она производит наблюдение за спутником, принимает радиосигналы, производит первичную обработку информации и обменивается данными с КВЦ, где происходит сбор и обработка данных от всех СТИ, вычисление корректирующих и координатных данных, которые подлежат загрузке в бортовую систему спутника.

Сегмент потребителей условно делится на 3 части: военные организации, гражданские организации, частные лица, но независимо от назначения оборудования оно производит прием радиосигналов навигационных спутников и их первичную обработку, выделение навигационной информации, выбор оптимального созвездия, вычисление пространственных координат и составляющих вектора скорости. Обычно первоначально определяются текущие координаты и дальность до спутников, потом производится вычисление географических координат пользователя. Путем измерения доплеровских сдвигов частоты спутников при известных векторах скорости спутников определяется вектор скорости пользователя. Затем, в зависимости от назначения навигационной аппаратуры пользователя, данные могут поступать в канал передачи, на блок управления или на устройство отображения.

В зависимости от типа используемого источника информации все навигационные системы делятся на автономные и неавтономные. К автономным системам относятся инерциальные навигационные системы, принцип действия которых основан на

интегрировании значений ускорений объекта по трем координатным осям, позволяя тем самым определять его местоположение. С помощью дополнительных расчетов и преобразований можно получить и другие параметры. Характерным для таких систем является увеличение погрешности определения координат в зависимости от продолжительности работы системы. Данная погрешность определяется как некоторая величина несоответствия действительного местоположения объекта с рассчитанным значением местоположения с помощью инерциальной системы за определенный период времени. Несмотря на высокую точность определения местоположения, суммарная погрешность измерений может очень сильно влиять на рассчитанный результат и за несколько часов работы достигать значения в несколько километров. Эти ошибки возникают из-за постоянных или медленно меняющихся во времени ошибках измерений угловых скоростей с помощью датчиков угловых скоростей, называемых «смещениями нулей» приборов. «Смещения нулей» являются случайными при каждом новом движении объекта, поэтому на этапе предварительной калибровки датчиков невозможно получить точную информацию об этих ошибках и учесть их заранее. В связи с этим инерциальные навигационные системы не могут применяться без использования специальных алгоритмов коррекции или отдельно от других навигационных систем.

В свою очередь, к неавтономным навигационным системам относятся: аппаратура всенаправленного азимутального радиомаяка, аппаратура всенаправленного дальномерного радиомаяка, глобальные навигационные спутниковые системы, совместно с которыми может использоваться автономная система для корректировки значений. Спутниковое навигационное оборудование имеет ряд преимуществ перед традиционными навигационными системами:

- глобальность и непрерывность действия;
- возможность применения специальных методов для увеличения точности определения местоположения;
- независимость работоспособности системы от внешних факторов.

Однако у спутниковых систем также есть свои недостатки. При определении местоположения система допускает неточности, которые имеют различную природу. К основным источникам ошибок данной системы можно отнести:

1. Погрешности, обусловленные режимом селективного доступа. В данном режиме Министерством обороны, которому принадлежит спутниковая система, искусственно создаются ошибки, снижая точность определения местоположения;
2. Погрешности, связанные с задержкой распространения радиоволн в ионосфере, могут приводить к ошибкам в расчетах до 30 метров;
3. Погрешности, связанные с задержкой распространения радиоволн в тропосфере, также могут приводить к ошибкам в расчетах до 30 метров;
4. Эфемеридная погрешность. В ней ошибки обусловлены расхождением между фактическим положением GPS-спутника и его расчетным положением. Значение погрешности обычно не больше 3 м;
5. Погрешность ухода шкалы времени спутника, вызванная расхождением шкал времени различных спутников;
6. Погрешность определения расстояния до спутника. Ее величина обычно не превышает 10 м.

Помимо основных источников ошибок существуют и вторичные, в той или иной мере влияющие на процесс измерения.

Для уменьшения влияния этих недостатков при определении местоположения и других параметров используются комплексированные системы навигации.

4. Тестирование комплексированной навигационной аппаратуры

Тестирование инерциальных датчиков или спутниковых приемников само по себе является непростой задачей. Более того, задача сильно усложняется при их интеграции. Линейные и угловые датчики можно испытывать отдельно с помощью центрифуг и поворотных столов, вращающихся с заданной угловой скоростью. Приемники спутниковых сигналов можно тестировать с помощью принимаемых сигналов (самый дешевый и ненадежный способ) или с помощью систем записи и воспроизведения спутниковых сигналов. Однако для полноценного тестирования таких систем необходимы дорогостоящие и достаточно длительные полевые испытания на подходящей движущейся транспортной платформе. Поэтому для проведения испытаний навигационной системы используется достойный альтернативный способ с помощью имитатора спутниковых сигналов и эмуляции в режиме реального времени сигналов инерциальных датчиков, которые генерируются в соответствии с траекторией движения объекта. Типичные характеристики (к примеру, дрейф, смещение), негативно влияющие на результаты

измерений, могут быть смоделированы в виде ошибок датчика, соответствующих имитируемой траектории движения объекта.

Основное преимущество такого способа тестирования заключается в том, что входные данные в виде результатов измерений имитируемого приемника и выходные данные эмулируемых сигналов инерционных датчиков задаются пользователем, могут быть изменены и являются хорошо повторяемыми. Это предоставляет возможность для отладки, тестирования и оптимизации навигационной аппаратуры.

Тестирование комплексированной навигационной аппаратуры проводят с помощью эмулятора сигналов инерциальных навигационных систем, которые дешевле и проще в изготовлении, но перед процедурой тестирования КНА необходимо убедиться, что сам эмулятор сигналов ИНС выдает точные значения. За эту точность в эмуляторе отвечает блок генерации данных (БГД). От блока генерации данных на обработку поступает информационный массив.

5. Входные и выходные данные

Входными данными являются данные, содержащиеся в текстовых файлах или поступающие в программу напрямую от угломерного приемника, навигационного модуля и БГД:

- первичные (вторичные) навигационные параметры;
- данные БГД (проекции скоростей и ускорений).

Выходные данные должны представлять собой результаты сравнений данных БГД и угломерного приемника/навигационного модуля, результат расчета статистических параметров и визуализация.

6. Состав выполняемых функций

Основные требования к функциональности программного модуля предполагают:

– прием, расшифровку, архивирование и анализ измерительных данных угломерного приемника и навигационного модуля;
– прием, расшифровку, архивирование и анализ измерительных данных БГД по различным последовательным интерфейсам;

– расчет первичных (вторичных) параметров ИНС из измерительных данных угломерного приемника и (или) навигационного модуля;
– преобразование полученных и рассчитанных данных в различные системы координат;
– сравнения измеренных (рассчитанных) навигационных параметров с данными БГД;
– расчет статистических параметров и визуализация.

7. Обзор аналогичных программных решений

В процессе предварительных исследований был проведен обзор существующих аналогичных решений, которые обладают схожим функционалом и решают аналогичные задачи. Их анализ, вместе с разрабатываемым модулем, представлен в таблице 1.1.

Таблица 1.1 - Сравнение аналогов

Параметры\аналоги	Spirent SimINERTIAL[1]	NAVSYS InterNav[2]	GPSOFT INS TOOLBOX[3]	ПМ ОХА
Поддержка последовательных интерфейсов	+	+	+	+
Работа в ОС Windows	-	+	+	-
Работа в ОС Linux	+	+	-	+
Поддержка большого количества ИНС	-	-	+	+
Симуляция ошибок ИНС	+	-	-	-
Построение графиков	+	+	-	+
Построение отчетов	+	-	+	+
Работа с информационным массивом из файла	-	-	+	+
Возможность расширения функционала	-	-	-	+

Условные обозначения:

\+ – указанная возможность присутствует

\- – указанная возможность отсутствует

1 – http://www.spirent.com/

2 – http://www.navsys.com/

3 – http://gpsoftnav.com/

Spirent SimINERTIAL – решение для тестирования интегрированных инерциально-спутниковых навигационных систем.

SimINERTIAL поддерживает различные популярные форматы данных инерциальных датчиков. Базовая конфигурация испытательного стенда представлена на изображении ниже.

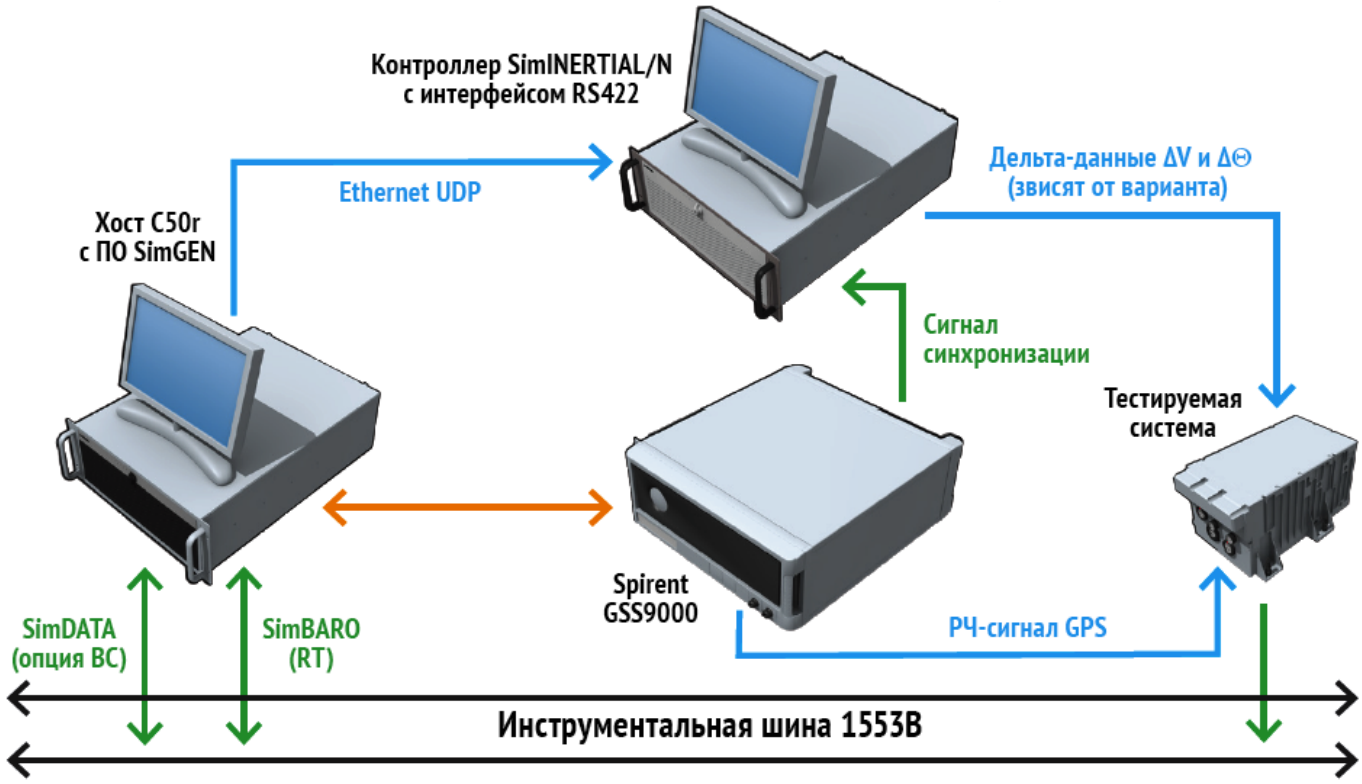

Рисунок 1.4 – Конфигурация испытательного стенда

Решение SimINERTIAL базируется на ЭВМ, оборудовано требуемым интерфейсом передачи данных. На SimINERTIAL приходит поток данных об имитируемой траектории движения от программного обеспечения SimGEN по интерфейсу Ethernet. Затем SimINERTIAL транслирует принятые данные в потоки данных реального времени так, чтобы они были подходящими для тестируемого устройства (соответствующий формат данных и скорость передачи).

NAVSYS InterNav – модульный GPS/ИНС программный продукт. ПО InterNav включает в себя функции, представленные на рисунке 1.5. Программа написана по стандарту ANSI C++ для обеспечения максимальной гибкости по интеграции различных спутниковых и инерциальных навигационных систем, и других датчиков. Полная настройка и управление всей системой осуществляется через выбор параметров в конфигурационном файле, что позволяет добиться гибкости при адаптации конфигурации

конкретного навигационного модуля или инерциальной системы. Конфигурационный файл можно полностью изменять, чтобы параметры соответствовали требованиям.

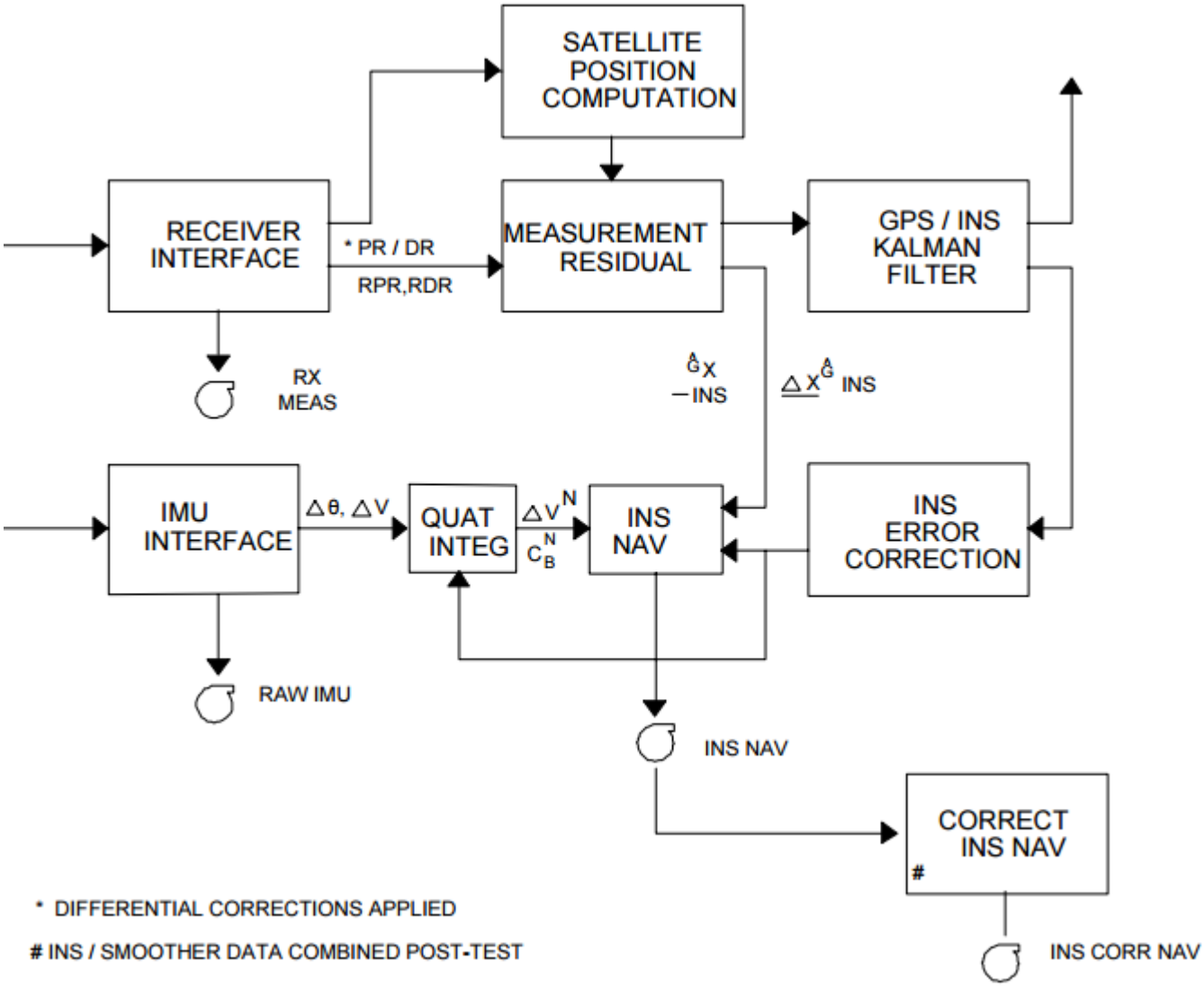

Рисунок 1.5 - Функции программного продукта NAVSYS InterNav

INS TOOLBOX – программное обеспечение компании GPSOFT, предоставляющее возможность симуляции спутниковых систем, среды распространения сигнала, измерений различных спутниковых и инерциальных систем, а также возможность обработки этих данных. Программа поддерживает буквально все приложения спутниковых систем, начиная с простого позиционирования и заканчивая системой повышения точности сигнала, учитывая разницу между известными псевдодальностями до космических аппаратов и действительными кодами псевдодальностей. Также поддерживается созвездие европейской системы Галилео, полевые испытания которой еще не были проведены.

8. Цель и задачи разработки программного модуля

Цель: повышение эффективности процесса оценки характеристик комплексированной навигационной аппаратуры.

Задачи:

– исследование предметной области;
– сравнительный анализ существующих программных решений;
– выбор языка и среды программирования;
– разработка схемы данных ПМ;
– разработка схем алгоритмов ПМ;
– разработка пользовательского интерфейса;
– программная реализация ПМ;
– отладка и тестирование ПМ;
– разработка руководства оператора.

9. Практическая значимость разработки

В результате разработки и внедрения ПМ ОХА ожидается:

– повышение эффективности оценки характеристик КНА;
– уменьшение временных затрат при оценке характеристик КНА;
– составление списка оцениваемых КНА для удобства дальнейшего анализа;
– организованное хранение полученных и рассчитанных данных;
– уменьшение влияния человеческого фактора при проведении расчетов;
– отсутствие последующих затрат на разработку ПО.

Выводы

В исследовательском разделе была обоснована актуальность разработки программного модуля, проведено исследование предметной области и сравнение существующих аналогичных программных решений, описан процесс тестирования комплексированной навигационной аппаратуры. Помимо этого были определены входные и выходные данные ПМ ОХА и состав выполняемых функций. Указана практическая значимость разработки.

10. КОНСТРУКТОРСКИЙ РАЗДЕЛ

11. Выбор языка программирования

Был проведен сравнительный анализ языков программирования, результат которого приведен в сводной таблице.

Таблица 2.1 - Сравнение языков программирования

Параметры	Язык				
Возможности языка программирования	C[1]	C++[2]	C#[3]	Java[4]	Python[5]
Императивный	+	+	+	+	+
Объектно-ориентированный	-	+	+	+	+
Обобщенное программирование	+/-	+	+	+	+
Поддержка исключений	-	+	+	+	+
Статическая типизация	+	+	+	+	-
Явная типизация	+	+	+	+	-
Нативная возможность компиляции	+	+	-	-	-
Ручное управление памятью	+	+	+	-	-
Контроль границ массивов	-	+	+	+	+
Шаблоны	-	+	+	+	-
Возможность подключения готовых решений	-	+	+	+	+
Встроенная поддержка в ОС Astra Linux	+	+	-	-	+
Опыт использования	+	+	+	+	+

Условные обозначения:

\+ – указанная возможность присутствует

\- – указанная возможность отсутствует

1 – http://www.open-std.org/

2 – http://isocpp.org/

3 – http://www.microsoft.com/net

4 – http://java.com/

5 – http://www.python.org/

Далее приводится описание рассмотренных языков программирования.

C – императивный язык программирования общего назначения, который поддерживает лексическую область видимости и рекурсию, и, будучи языком с системой статической типизации позволяет предотвращать нежелательные действия, а также обнаруживать ошибки на этапе компиляции. По дизайну языка его инструкции похожи на типичные машинные инструкции, благодаря чему он длительное время использовался

для написания приложений, которые были написаны на ассемблере, включая операционные системы и различное прикладное программное обеспечение для суперкомпьютеров и встраиваемых систем. Изначально язык Си был разработан в период с 1969 по 1973 гг. сотрудником компании Bell Labs Деннисом Ритчи с целью реализации операционной системы Unix. С тех пор язык Си стал самым широко используемым языком программирования за все время. Различные производители предоставляют компиляторы для большинства существующих компьютерных архитектур и встраиваемых систем. Язык Си стал основой для развития таких языков программирования, как C++, C#, Java и Objective-C и оказал огромное влияние на развитие индустрии ПО. Переносимость программ на компьютеры различной архитектуры и между различными операционными системами является основным преимуществом. К преимуществам также можно отнести логическую стройность программ, лаконичность записи алгоритмов и скорость выполнения программ, сравнимую со скоростью выполнения написанных на ассемблере программ, что связано с набором низкоуровневых инструкций, которые обеспечивают доступ к аппаратным средствам компьютера. С 1989 года Американский институт национальных стандартов ANSI регламентирует стандарт языка C. В настоящее время, помимо основного стандарта разработан международный стандарт ISO C (International Standard Organization C).

C++ – язык программирования общего назначения. Язык поддерживает такие парадигмы программирования, как процедурное, объектно-ориентированное и обобщенное программирование, модульность, абстракция данных, раздельная компиляция, виртуальные функции, контейнеры и алгоритмы, обработка исключений, а также предоставляет низкоуровневые средства программирования. При проектировании языка было уделено внимание системному программированию, встроенным и ограниченным в ресурсах системам. Особенностями языка является производительность, эффективность и гибкость при разработке программного обеспечения, включая прикладные программы, сервера (для электронной коммерции, поисковиков и SQL серверов) и высокопроизводительные приложения. C++ – компилируемый язык программирования со статической типизацией, а его реализации доступны на множестве платформ и поддерживаются такими организациями, как Free Software Foundation, IBM, LLVM, Intel и Microsoft.

C++ является одним из самых популярных языков программирования и широко применяется при разработке программного обеспечения. Область применения языка включает создание приложений для встраиваемых систем, драйверов устройств, а также развлекательных приложений (видеоигр). Существует множество различных реализаций языка C++ – как бесплатных, так и коммерческих. Наиболее популярные среди них: GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#. Синтаксис языка C++ был унаследован от языка C, а одним из принципов при его создании было обеспечение совместимости с языком C. Язык возник в 1979 году, когда сотрудник компании Bell Labs, Бьёрн Страуструп, начал работать над новым языком, который изначально назывался Си с классами, а в 1983 году он был переименован в C++. В настоящее время C++ регламентируется международной организацией по стандартизации.

Java – объектно-ориентированный язык программирования общего назначения, поддерживающий многопоточное программирование, был разработан специально с наименьшими зависимостями в реализации. Основная философия языка – «Напиши один раз – запускай, где угодно» – означает, что скомпилированный код можно запускать на всех платформах, которые поддерживают Java без необходимости повторной компиляции. Приложения, написанные на Java, обычно компилируются в байткод, который затем можно запустить на любой Java-машине вне зависимости от архитектуры компьютера. В 2016 году язык Java признан одним из самых популярных языков программирования, особенно при разработке клиент-серверных приложений, а количество разработчиков насчитывает 9 миллионов пользователей. Java был разработан сотрудником компании Sun Microsystems Джеймсом Гослингом и выпущен в 1995 году как ключевой компонент программной платформы Java. Синтаксис языка унаследовал многие черты языков C и C++, но обладает меньшими возможностями для низкоуровневого программирования. Изначально компиляторы Java, виртуальные машины и библиотеки классов распространялись компанией Sun под проприетарной лицензией, но в мае 2007 года она выпустила их под лицензией GNU GPL. К недостаткам виртуальной машины относят низкую производительность Java-программ из-за исполнения байт-кода виртуальной машиной. Несмотря на это, в последнее время создателями языка был внесен ряд усовершенствований, которые увеличили скорость выполнения программ.

C# – мультипарадигменный язык программирования со строгой типизацией. Язык поддерживает такие парадигмы, как объектно-ориентированное, структурное, функциональное и обобщенное программирование. Язык был разработан компанией Microsoft в 1998-2001 годах группой инженеров под руководством Андерса Хейлсберга. Язык был разработан для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. C# разрабатывался как простой и современный язык общего назначения с поддержкой объектно-ориентированного программирования. Язык поддерживает такие средства, как проверка типов, проверка границ массива, обнаружение использования неинициализированных переменных и автоматическая сборка мусора. C# относится к семье языков с C-подобным синтаксисом и наиболее близок к синтаксису C++ и Java. Язык поддерживает полиморфизм, делегаты, перегрузку операторов, атрибуты, свойства, события, обобщённые типы, итераторы, методы, анонимные функции с поддержкой замыканий, комментарии в формате XML и исключения. C#, опираясь на практику использования своих предшественников (C++, Java, Smalltalk, Модула и Delphi), исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Python — высокоуровневый интерпретируемый язык программирования общего назначения с динамической типизацией. Основное внимание философия языка уделяет читаемости кода, а синтаксис языка минималистичен, что позволяет программистам писать более краткий код по сравнению с такими языками, как C++ и Java. Конструкции языка позволяют писать чистый код вне зависимости от масштаба приложения. Python поддерживает несколько парадигм, включая объектно-ориентированное, процедурное и функциональное программирование. К особенностям языка можно отнести динамическую систему типов и автоматическое управление памятью, а также большую стандартную библиотеку. Интерпретатор питона доступен для множества операционных систем, что позволяет запускать код, написанный на языке Python, на большом разнообразии систем.

12. Выбор среды разработки

После выбора языка программирования был также проведен сравнительный анализ интегрированных сред разработки и выбрана среда, которая наиболее полно удовлетворяет критериям отбора.

Таблица 2.2 - Сравнение сред программирования

Параметры	NetBeans[1]	C++ Builder[2]	Code::Blocks[3]	QtCreator[4]	Visual Studio[5]
Форма распространения ПО	Свободная	Коммер-ческая	Свободная	Свободная	Коммер-ческая
Встроенный редактор графического интерфейса	-	+	-	+	+
Встроенный отладчик	+	+	+	+	+
Статический анализатор кода	-	+	+	+	+
Автодополнение	+	-	+	+	-
Профилирование	-	-	+	+	+
Работа в Linux	+	-	+	+	-
Опыт использования	+	+	-	+	+

Условные обозначения:

\+ – указанная возможность присутствует

\- – указанная возможность отсутствует

1 – https://netbeans.org/

2 – https://www.embarcadero.com/

3 – http://www.codeblocks.org/

4 – https://www.qt.io/

5 – https://www.visualstudio.com/

NetBeans IDE — кроссплатформенная интегрированная среда разработки приложений (IDE), написанная на Java. Поддерживает такие языки программирования, как C, C++, Python, Java, PHP, JavaScript, Ада и другие. Компания Oracle поддерживает и спонсируют проект NetBeans, но сама разработка NetBeans ведётся независимым сообществом разработчиков-энтузиастов (NetBeans Community) и компанией NetBeans Org. В последних версиях среды реализована поддержка профилирования, рефакторинга, выделение цветом синтаксических конструкций, автодополнение конструкций на лету и множество предопределённых шаблонов кода. Требуются установленные предварительно Sun JDK или J2EE SDK для установки и работы самой интегрированной среды, а также для разработки программ. По умолчанию среда разработки NetBeans поддерживает разработку для платформ J2EE и J2SE, а с версии 6.0 NetBeans стала поддерживать также разработку для мобильных платформ J2ME, C++ (g++)

и PHP без необходимости в установке дополнительных компонентов. В NetBeans IDE встроена поддержка различных плагинов, которые позволяют разработчикам расширять возможности среды. Одним из самых популярных плагинов является мощный дизайнер отчётов iReport (основанный на библиотеке Jasper Reports).

C++ Builder — интегрированная среда программирования, позволяющая вести быструю разработку приложений (RAD). Изначально продукт разрабатывался компанией Borland, а затем права на него перешли к Embarcadero Technologies. C++ Builder поддерживает различные системы, такие как OS X, iOS and Android, но основной системой является Windows NT (IA-32 and x64). В состав C++ Builder входят комплекс объектных библиотек (MFC, STL, CLX, VCL и др.), отладчик, компилятор, редактор кода и многие другие компоненты. Большинство компонентов, разработанных в Delphi, могут быть использованы и в C++ Builder практически без внесения изменений, однако обратное утверждение неверно. C++ Builder содержит инструменты, которые посредством перетаскивания могут действительно сделать разработку визуальной, а также упростить программирование благодаря встроенному редактору интерфейса.

Code::Blocks — свободная кроссплатформенная среда разработки, поддерживающая множество компиляторов, среди которых GCC, Clang and Visual C++. Code::Blocks разработана на языке C++ и использует библиотеку wxWidgets в качестве основного средства для разработки графического интерфейса пользователя. Открытая и расширяемая архитектура может масштабироваться за счёт подключаемых модулей. Поддерживает языки программирования C++, C, D (с ограничениями) и Fortran. Code::Blocks разрабатывается для таких операционных систем, как Windows, Mac OS X и Linux. Среду можно собрать из исходников практически под любую Unix-подобную систему, например FreeBSD, PC-BSD. Среда разработки имеет встроенную настраиваемую систему сборки, способную реализовывать параллельную сборку (задействуя при этом ядра процессора), при этом позволяет производить сборку внешними инструментами (GNU Make, Cmake и другие) с рукописными скриптами типа makefile, причем также в многопоточном режиме. Как и мощные профессиональные интегрированные среды, имеет развитые средства поддержки проектов, среди которых рабочие места для объединения родственных проектов, общие межпроектные зависимости, множественные цели.

QtCreator — кроссплатформенная свободная интегрированная среда разработки, поддерживающая C, C++, JavaScript и QML, являющаяся частью комплекта средств разработки фреймворка Qt и разработанный компанией Trolltech (Digia). В него входят графический отладчик и интегрированный дизайнер графического интерфейса пользователя, включая QtWidgets и QML. Среда поддерживает различные компиляторы, такие как GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW. Qt Creator включает в себя менеджер проектов, работающий со множеством форматов проектов, таких как .pro, CMake, Autotools и прочие. Файл проекта содержит информацию, касающуюся файлов, включенных в проект, настраиваемых шагов сборки и настройки для запуска приложения. Основная задача QtCreator — упростить разработку приложения с помощью фреймворка Qt на разных платформах. Поэтому среди возможностей, присущих любой среде разработки, есть и специфичные, такие как отладка приложений на QML и отображение в отладчике данных из контейнеров Qt, встроенный дизайнер интерфейсов как на QML, так и на QtWidgets.

Среда разработки поддерживает системы сборки qmake, cmake, autotools, qbs. Для проектов, созданных под другими системами, среда может использоваться в качестве редактора исходных кодов. Также QtCreator нативно поддерживает системы контроля версии, такие как Subversion, Mercurial, Git, CVS, Bazaar, Perforce. В QtCreator реализовано автодополнение, в том числе ключевых слов, введённых в стандарте C++11, подсветка кода (с возможностью создавать свои виды подсветок или использовать уже готовые). Также, начиная с версии 2.4, есть возможность задания стиля выравнивания, отступов и постановки скобок.

Microsoft Visual Studio – интегрированная среда разработки компании Microsoft. Используется для разработки компьютерных программ для операционной системы Windows, а также сайты, веб-приложения и веб-сервисы как в родном, так и в управляемом кодах. Visual Studio использует программные разработки платформы, такие как Windows API, Windows Forms, Microsoft Silverlight, Windows Presentation Foundation и Windows Store. Среда разработки поддерживает средство автодополнения IntelliSense, а также рефакторинг кода. Встроенный отладчик работает как на уровне исходного кода, так и на уровне машинных инструкций. В среде разработки присутствуют и другие встроенные инструментальные средства, например, дизайнер форм для создания графического интерфейса пользователя, веб-дизайнер, дизайнер классов и построитель

схемы базы данных. Visual Studio поддерживает систему плагинов, позволяющих расширять ее функционал, например, добавляет поддержку систем контроля версий или средств контроля жизненного цикла ПО (клиент Team Foundation Server: Team Explorer). Среда поддерживает различные языки программирования, среди которых C, C++, C++/CLI (Visual C++), Visual Basic .NET, Visual C#, and F#. Поддержка других языков программирования, таких как Python, Ruby, Node.js, M и прочих доступна с помощью отдельно устанавливаемых сервисов. Среда разработки также поддерживает XML/XSLT, HTML/XHTML, JavaScript и CSS. Поддерживаются различные платформы: Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Для выполнения поставленной задачи было решено использовать язык C++ в среде разработки QtCreator с использованием фрейморка Qt. Основные преимущества фреймворка Qt вместе с интегрированной средой разработки:

- наличие библиотек для работы с аппаратной частью;
- наличие средств, позволяющих вести быструю разработку графического интерфейса пользователя;
- высокая скорость разработки и прототипирования;
- полная и качественная документация;
- наличие средств, предоставляющих возможность для визуализации данных в виде графиков и схем.

13. Кроссплатформенный фреймворк Qt

Qt – кроссплатформенный инструментарий для разработки программного обеспечения с использованием языка программирования C++. Qt можно также использовать совместно со многими языками программирования, например, Python – PyQt, PySide; Java – QtJambi; Ruby – QtRuby; PHP – PHP-Qt и другие. Без изменения исходного кода фреймворк Qt позволяет запускать программы, написанные с его помощью, в большинстве современных операционных систем только перекомпилировав её для каждой ОС. В состав фреймворка входят все основные классы, способствующие быстрой и качественной разработке прикладного программного обеспечения, что достигается посредством классов для создания элементов графического интерфейса, классов для работы с базами данных, а также для работы с сетью и XML. Фреймворк поддерживает

технику компонентного программирования, является полностью объектно-ориентированным и легко расширяемым.

Система предварительной обработки исходного кода MOC - Meta Object Compiler – отличительная особенность от других библиотек. MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. За счёт этого код становится более кратким и лаконичным. Метаобъектный компилятор в заголовочных файлах ищет описания классов, содержащие макрос Q_OBJECT, и создаёт дополнительный файл исходного кода на C++, который содержит метаобъектный код.

При помощи фреймворка Qt возможно создавать и собственные плагины, размещая их непосредственно на панели графического редактора. Такая функциональность виджетов, как их отображение, размещение на экране, а также перерисовка в случае изменения размеров окна может быть расширена.

Помимо набора библиотек классов фреймворк Qt предоставляет определённую модель разработки приложений, некоторый каркас их структуры, а следуя принципам и правилам «хорошего стиля программирования на C++/Qt», можно значительно снизить частоту трудно отлавливаемых ошибок, например, ошибки утечки памяти, неосвобождённые дескрипторы ресурсов, необработанные исключения, которые встречаются при разработке программ без использования Qt на «чистом C++».

Логичный, стройный и хорошо продуманный набор классов, предоставляющий высокий уровень абстракции, является важным преимуществом фреймворка, благодаря чему объем кода становится меньше, чем при написании приложений с использованием таких библиотек классов, как MFC. Сам код, написанный на C++/Qt, выглядит проще, стройнее и понятнее, чем схожий по функциональности код MFC или же написанный с помощью тулкита Xt. Также такой код легче развивать и поддерживать.

Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11, Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. Идет портирование на Windows Phone и Windows RT. Также идёт портирование на Haiku и Tizen.

До версии 4.0.0 под свободной лицензией распространялись только Qt/Mac, Qt/X11, Qt/Embedded, но, после выпуска в 2007 году версии 4.0.0, Qt стала доступна и на Windows. Теперь эта библиотека предоставляется под лицензией LGPL, что даёт право

использовать её при разработке собственных продуктов с закрытым исходным кодом, а также проприетарных продуктов.

Компоненты фреймворка:

- QtCore — классы ядра библиотеки, используемые другими модулями;
- QtGui — компоненты графического интерфейса;
- QtWidgets — содержит классы для классических приложений на основе виджетов;
- Qt QML — модуль для поддержки QML;
- QtNetwork — набор классов для сетевого программирования;
- QtOpenGL — набор классов для работы с OpenGL;
- QtSql — набор классов для работы с базами данных с использованием языка структурированных запросов SQL;
- QtScript — классы для работы с QtScripts;
- QtSvg — классы для отображения и работы с данными Scalable Vector Graphics;
- QtXml — модуль для работы с XML, поддерживает модели SAX и DOM;
- QtDesigner — классы создания расширений QtDesigner’а для своих собственных виджетов;
- QtUiTools — классы для обработки в приложении форм QtDesigner;
- QtAssistant — справочная система;
- QtTest — классы для поддержки модульного тестирования;
- QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы;
- QtXmlPatterns — модуль для поддержки XQuery 1.0 и XPath 2.0;
- Phonon — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств, а также по сети;
- ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt-разработчиков под Windows;
- QtDeclarative — модуль, предоставляющий декларативный фреймворк для создания динамичных, настраиваемых пользовательских интерфейсов.

В отличие от той же wxWidgets, у проекта Qt присутствует качественная документация, которая является одним из преимуществ фреймворка. Статьи документации подробно описаны и обладают большим количеством примеров. Исходный код самой библиотеки хорошо форматирован, подробно комментирован и легко читается, что также упрощает изучение Qt.

Для упрощения доступа к документации программисту предоставляется универсальное средство – QtAssistant. QtAssistant – это справочная система интерактивной документации библиотеки Qt.

14. Архитектура программного модуля и алгоритм работы

В соответствии с определением архитектуры программного обеспечения, приведенного ниже, была определена архитектура ПМ ОХА.

Архитектура программного обеспечения (software architecture) — это структура программы или вычислительной системы, которая включает программные компоненты (элементы), видимые снаружи свойства этих компонентов, а также отношения (взаимодействия) между ними. Архитектура затрагивает не только структуру и поведение, но также использование, функциональность и другие аспекты.

Для решения поставленной задачи, а также для удобства взаимодействия с пользователем и с учетом наличия в Qt сигнально-слотовой системы было решено разрабатывать программный модуль в соответствии с архитектурой, управляемой событиями (event-driven architecture, EDA). Помимо этого архитектурного стиля в основе разработки программного модуля лежит объектно-ориентированная архитектурная парадигма.

Архитектура, управляемая событиями – это концепция управления корпоративной информационной системой на основе событий, возникающих в бизнес-процессе. Событие можно определить, как «существенное изменение состояния». Например, когда покупатель приобретает автомобиль, состояние автомобиля изменяется с «продаваемого» на «проданный». Системная архитектура продавца автомобилей может рассматривать это изменение состояния как событие, создаваемое, публикуемое, определяемое и потребляемое различными приложениями в составе архитектуры.

Система, управляемая событиями, обычно содержит источники событий (или агентов) и потребителей событий (или стоков). Во фреймворке Qt за это отвечают сигналы и слоты соответственно. Стоки ответственны за ответную реакцию, как только событие возникло. Реакция может полностью или не полностью создаваться стоком. К примеру, сток может отвечать лишь за фильтрацию, преобразование и доставку события другому компоненту либо он может создать собственную реакцию на это событие. Применительно к разрабатываемому программному модулю сигналы являются источниками событий, а слоты принимают соответствующие им сигналы и производят необходимые действия, то есть формируют реакцию на поступивший сигнал.

Создание приложений и систем в рамках архитектуры, управляемой событиями, позволяет конструировать их таким образом, чтобы способствовать лучшей

интерактивности, так как системы, управляемые событиями, по своей структуре более ориентированы на непредсказуемые и асинхронные окружения.

Объектно-ориентированная архитектурная парадигма – это парадигма, в рамках которой используется объектно-ориентированный подход (ООП) к структуризации ПО на всех этапах его жизненного цикла. В основе парадигмы лежат идеи, методы и средства Объектно-Ориентированного Анализа и Проектирования (ООАП). В свою очередь, в основу ООП положены следующие принципы: абстрагирование, ограничение доступа, модульность, иерархичность, типизация, параллелизм, устойчивость.

В объектно-ориентированном подходе используются указанные ниже основные понятия.

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. При использовании термина «полиморфизм» в сообществе ООП подразумевается полиморфизм подтипов; а использование параметрического полиморфизма называют обобщённым программированием.

Класс является описываемой на языке терминологии исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект – это экземпляр класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

Объект — сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса или копировании прототипа (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Прототип — это объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка.

Алгоритм работы и схема данных программного модуля

Перед началом процесса приема и обработки данных необходимо указать системы координат для корректного преобразования координат из одной системы в другую. Также перед началом работы ПО должно установить связь с внешними устройствами и проверить ее. В процессе приема и обработки требуется производить различные действия над данными: расшифровка, анализ, преобразование координат, приведение к единой системе измерения, расчет первичных (вторичных) навигационных параметров. По окончании приема рассчитываются статистические параметры, которые сохраняются так же, как и принимаемые данные. В программном модуле реализована возможность работы с файлами логов навигационной аппаратуры. Поэтому при отсутствии связи и при наличии логов от навигационных модулей можно указать их программе и также получить результат. Результат выводится пользователю в виде файла отчета (файл лога), а также производится визуализация данных на графиках и в сводной таблице. Далее приводится схема данных и общая схема работы программного модуля.

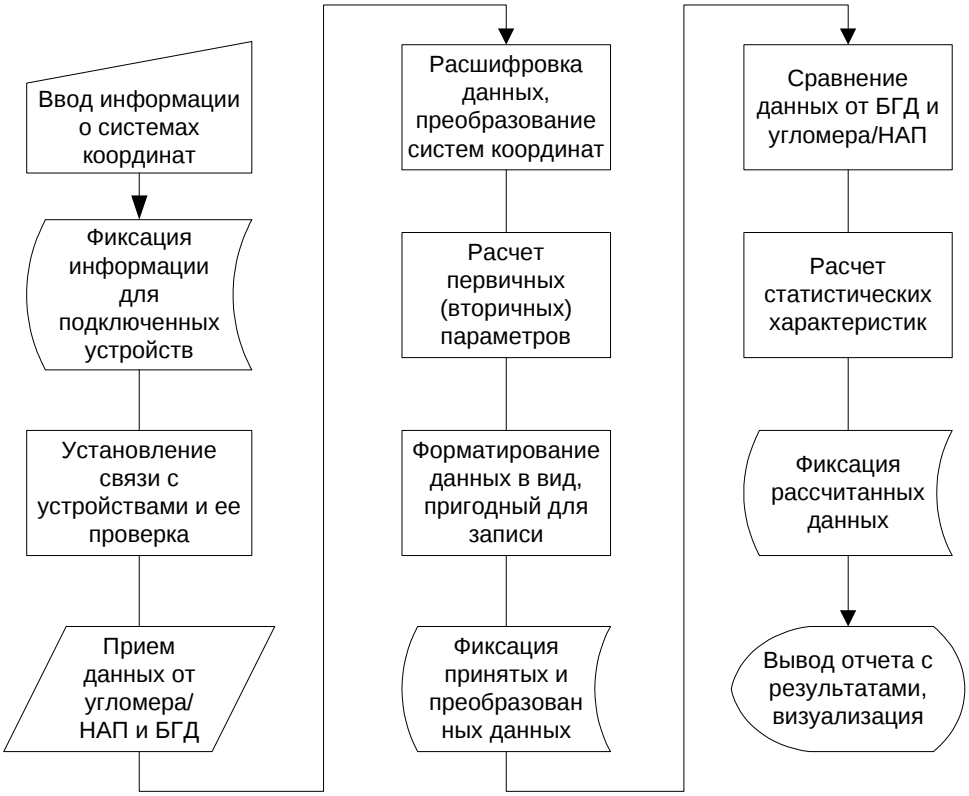

Рисунок 2.1 - Схема данных ПМ ОХА

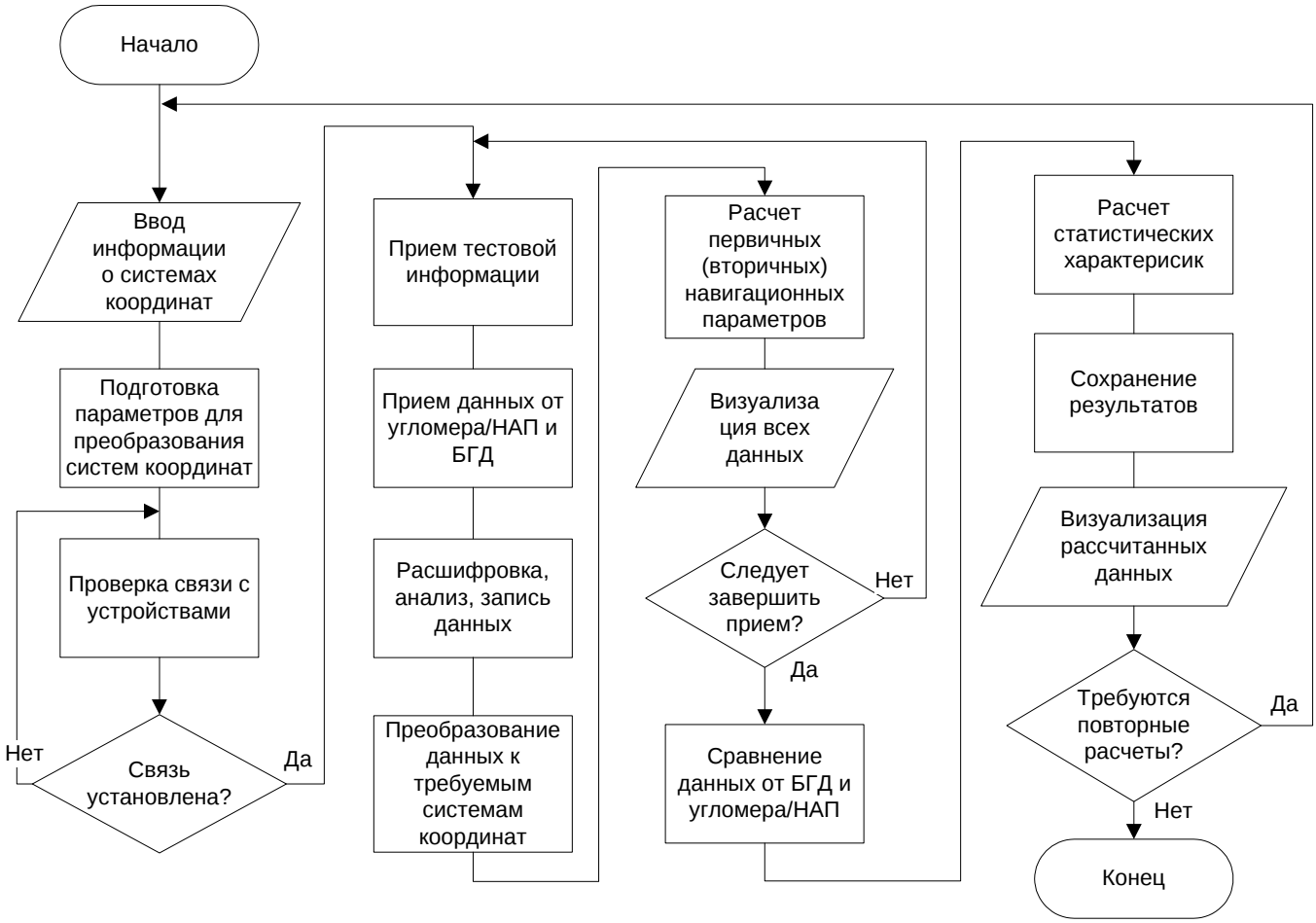

Рисунок 2.2 - Схема алгоритма ПМ ОХА

15. Форматы данных и конфигурационные файлы ПМ ОХА

Для обмена данными используется стандартизованный протокол связи NMEA-0183 или формат данных, определяемый изготовителем навигационной системы. NMEA 0183 (National Marine Electronics Association) — текстовый протокол связи морского (в основном, навигационного) оборудования между собой. Протокол стал популярен в связи с распространением GPS приёмников, использующих этот стандарт. (В настоящее время внедряется стандарт NMEA 2000). Протокол описывает не только данные, полученные с GPS приемников, но и измерения радаров, сонаров, барометров, электронных компасов и других навигационных устройств.

Все сообщения имеют общий вид и представляют собой строку значений, разделенных запятыми. Каждое отдельное сообщение является полностью «завершенным» и не зависит от других сообщений. Первым элементом является символ «$»; далее идет 5-буквенный идентификатор сообщения, где первые две буквы определяют источник сообщения; следующие три буквы определяют формат сообщения; затем список данных, разделенных запятыми, причем если какие-либо данные отсутствуют внутри строки, запятые всё равно ставятся (в конце строки некоторые поля могут совсем отсутствовать); следом – символ «*», отделяющий информационную часть от контрольной суммы, после которого стоит двухзначное шестнадцатеричное число, являющееся контрольной суммой сообщения (определяется как XOR-сумма всех байт в строке между символами «$» и «*») Завершающим символом является символ конца строки.

Список сообщений протокола NMEA

Протокол NMEA содержит описание большого списка разнообразных сообщений. Однако из всего списка в навигационной аппаратуре активно используются только около двадцати типов сообщений. В связи с большой популярностью и простотой представления данных NMEA протокол нашел применение не только в морской аппаратуре, но и в геодезических, бытовых и авиационных GPS приемниках. К распространенным типам сообщений NMEA относятся:

- AAM – прибытие в путевую точку;
- ALM – данные альманаха;
- APA – данные автопилота «A»;
- APB – данные автопилота «B»;

- BOD – азимут на пункт назначения;
- DTM – используемый датум;
- GGA – информация о фиксированном решении;
- GLL – данные широты и долготы;
- GSA – общая информация о спутниках;
- GSV – детальная информация о спутниках;
- RMB – рекомендованный набор навигационных GPS данных;
- RMC – рекомендованный минимальный набор GPS данных;
- WPL – данные путевой точки;
- ZTG – UTC время и оставшееся время до прибытия в точку назначения;
- ZDA – дата и время.

Данные внутри некоторых сообщений могут полностью дублировать информацию из других сообщений (меньше по объему) или же частично повторяют значения некоторых полей. При обработке сообщений NMEA основная информация приходится на два типа сообщений, несущих данные, необходимые для работы: сообщения RMC и GGA.

Формат сообщения RMC:

$GxRMC,HHMMSS.SS,A,BBBB.BBBB,a,LLLLL.LLLL,a,v.v,z.z,DDMMYY,x.x,a,b*

hh<CR><LF>, где

Таблица 2.3 – Описание компонентов сообщения RMC

Название	Описание
HHMMSS.SS	время UTC (часы, минуты, целая и дробная части секунды)
A	статус: V - решение не годно A - автономный режим ,D - дифференциальный режим
BBBB.BBBB,a	широта (BB - градусы, BB.BBBB - целая и дробная часть минут), север/юг (N/S)
LLLLL.LLLL, a	долгота (LLL - градусы, LL.LLLL - целая и дробная часть минут), восток/запад (E/W)
v.v	наземная скорость, в узлах
z.z	наземный курс, в градусах
DDMMYY	дата (день¦месяц¦год)
x.x, a	магнитное склонение в градусах, восток/запад (E/W)
b	режим местоопределения: A – автономный, D - дифференциальный E - ожидаемый (сопровождение при недостаточном количестве спутников), M - ручной ввод, S - режим имитации, N - данные не годны

Формат сообщения GGA:

$GxGGA,HHMMSS.SS,BBBB.BBBB,a,LLLLL.LLLL,a,b,cc,d.d,e.e,M,f.f,M,g.g,jjjj*

hh<CR><LF>, где

Таблица 2.4 - Описание компонентов сообщения GGA

Название	Описание
HHMMSS.SS	время обсервации UTC (часы, минуты, целая и дробная часть секунд)
BBBB.BBBB,a	широта (BB - градусы, BB.BBBB - целая и дробная часть минут), север/юг (N/S)
LLLLL.LLLL, a	долгота (LLL - градусы, LL.LLLL - целая и дробная часть минут), восток/запад (E/W)
b	показатель качества обсервации согласно таблице 2.5
cc	число НКА в решении
d.d	величина горизонтального геометрического фактора
e.e	высота над средним уровнем моря. Значение отрицательно, если уровень моря ниже поверхности земного эллипсоида
M	единица измерения высоты – метры
f.f	превышение геоида над эллипсоидом WGS-84; отрицательное значение означает, что поверхность геоида ниже поверхности эллипсоида
M	единица измерения – метры
g.g	возраст дифференциальных поправок. Время в секундах после получения последней дифференциальной поправки. Пустое поле используется в случае выключения дифференциального режима
jjjj	идентификатор дифференциальной станции, от 0000 до 1023. Пустое поле используется в случае выключения дифференциального режима

Таблица 2.5 – Значения показателя качества обсервации

Код	Показатель
0	Определение места не получено
1	Обсервация получена в автономном режиме
2	Обсервация в дифференциальном режиме

Помимо протокола NMEA-0183 используется формат данных CSV со знаком «;» в качестве разделителя со следующей структурой:

Таблица 2.6 - Формат данных CSV

T;	B;	L;	H;	V;	PDOP;	Ns;
количество секунд с начала суток	широта в градусах (разделитель целой и дробной части – точка)	долгота в градусах (разделитель целой и дробной части – точка)	высота в метрах (разделитель целой и дробной части – точка)	скорость в метрах в секунду (разделитель целой и дробной части – точка)	геометрический фактор ухудшения точности	количество навигационных космических аппаратов

CSV (Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. Значения отдельных колонок разделяются разделительным

символом – запятой «,». Однако большинство программ вольно трактуют стандарт CSV и допускают использование иных символов в качестве разделителя. В частности, в локалях, где десятичным разделителем является запятая, в качестве табличного разделителя, как правило, используется точка с запятой.

В некоторых случаях, для обеспечения гибкости работы программы используются конфигурационные файлы ее настроек, предоставляя прямой доступ к параметрам, определяющим работу программы.

В работе программного модуля используется конфигурационный файл, в котором задается список эталонных файлов, хранящих навигационные данные в виде NMEA сообщений или CSV формате, при работе в режиме с файлами. Формат конфигурационных данных был выбран со схожим форматом ini-файлов.

ini-файл (Initialization file) — это файл конфигурации, содержащий определенные настройки приложений. Изначально конфигурационные файлы появились и использовались в операционных системах Microsoft Windows. Они могут использоваться в любой операционной системе, однако приобрели популярность в ОС Windows. Несложная, но в то же время достаточно строгая структура формата INI файлов позволяет с легкостью обрабатывать их программно, а также имеет понятный для чтения вид. Данные внутри файла просты для изменения человеком. Официальной спецификации формата файлов INI не существует. Хотя общая структура таких файлов обычно сохраняется, многие программные продукты вводят в неё дополнительные возможности или, наоборот, ограничения.

INI-файлы имеют следующий формат:

```
; комментарий
# комментарий в стиле Unix

[Section1]
; комментарий о разделе
variable1=value_1; иногда допускается комментарий к отдельному параметру
variable2=value_2

[Section2]
```

variable1= value_1

variable2= value_2

; иногда можно перечислять несколько значений через запятую

[Section3]

variable1= value_1_1, value_1_2, value_1_3

variable2= value_2

; Иногда значения могут отсутствовать

[Section4]

[ViewState]

Mode=

FolderType=Generic

В самом программном модуле используется следующий формат конфигурационного файла:

[название_файла]

path=/путь/к/файлу

timeshift=время_в_секундах

где в квадратных скобках указывается название файла,

path – относительный или абсолютный путь к файлу

timeshift – смещение по времени от начала измерений

Пример содержания конфигурационного файла, использованного в работе ПМ:

[c5kNew]

path=/scripts/c5kNew.csv

timeshift=0

16. Необходимые параметры и алгоритмы их расчета

Геодезические координаты (широта, долгота, высота)

Геодезические координаты получаем напрямую из информационных сообщений НАП. Например, из сообщения GGA протокола NMEA 0183, поля № 2-5.

Ортометрическая высота

Ортометрическую высоту получаем напрямую из информационных сообщений НАП. Например, из сообщения GGA протокола NMEA 0183, поле № 9.

Описание сообщения приведено в п. 2.5.

Номинальное атмосферное давление, соответствующее высоте объекта-носителя

Зависимость давления газа от высоты определяется так называемой барометрической формулой

$$P = P_0 e^{\frac{-\mu g h}{RT}} \quad (1)$$

где h- разность высот, м;

μ - молярная масса воздуха, 29 г/моль;

R - универсальная газовая постоянная, 8.31 Дж/(моль*К);

g - ускорение силы тяжести, 9.81 м/с^2;

T - температура воздуха (К);

P_0 - номинальное давление на исходной высоте.

Таким образом, в качестве P_0 необходимо принять нормальное атмосферное давление (101 325 Па), а в качестве разности высот h - ортометрическую высоту.

Координаты в прямоугольной связанной с Землей системе координат (по выбору оператора WGS-84, ПЗ-90.02, ПЗ-90.11)

Координаты в прямоугольной связанной с Землей системе координат получаем из геодезических координат и ортометрической высоты. Также для расчётов необходимо превышение геоида над референц-эллипсоидом, которое получаем из сообщения GGA протокола NMEA 0183, поле № 10. Система координат, в которой мы получаем геодезические данные и высоту, указана в сообщении DTM протокола NMEA 0183.

Пересчёт геодезических координат в декартовы (координаты в прямоугольной связанной с Землей системе координат) производится с помощью методов, установленных ГОСТ 51794-2008 «Глобальные навигационные спутниковые системы. Системы координат. Методы преобразований координат определяемых точек», который распространяется на системы координат, входящие в состав систем геодезических параметров «Параметры Земли», «Мировая геодезическая система» и координатной основы Российской Федерации,

и устанавливает методы преобразований координат и их приращений из одной системы в другую, а также порядок использования числовых значений элементов трансформирования систем координат при выполнении геодезических, навигационных, картографических работ с использованием аппаратуры потребителей глобальных навигационных спутниковых систем. Элементы трансформирования систем также приведены в ГОСТ 51794-2008 (WGS84 и ПЗ-90.02) и в справочном документе «ПАРАМЕТРЫ ЗЕМЛИ 1990 ГОДА» (ПЗ-90.11).

Преобразования геодезических координат в пространственные прямоугольные в соответствии с указанными документами производятся по следующим формулам:

$$\begin{cases} X = (N + H)\cos B \cos L \\ Y = (N + H)\cos B \sin L \\ Z = [(1 - e^2)N + H]\sin B \end{cases} \quad (2)$$

где X, Y, Z – пространственные прямоугольные координаты точки;

B, L, H – геодезические координаты точки;

N – радиус кривизны первого вертикала;

e –эксцентриситет эллипсоида.

Радиус кривизны первого вертикала и квадрат эксцентриситета эллипсоида вычисляются по формулам:

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 B}} \quad (3)$$

$$e^2 = 2\alpha - \alpha^2 \quad (4)$$

где a – большая полуось эллипсоида;

α – сжатие эллипсоида.

Углы пространственной ориентации (курс, крен, тангаж)

Углы пространственной ориентации получаем напрямую из информационных сообщений НАП по любому из протоколов обмена.

Введем нормальную систему координат $OX_gY_gZ_g$ (по ГОСТ 20058-80), центр которой связан с центром масс объекта, ось OX_g направлена на север по касательной

к меридиану, ось OZ_g по касательной к параллели на восток, а ось OY_g – вдоль вертикали места вверх. Также введём связанную систему координат OXYZ, центр которой связан с центром масс объекта, оси связаны с объектом. В таком случае углы курса (ϕ), крена (ψ) и тангажа (θ) – это углы поворота системы координат $OX_gY_gZ_g$ до системы координат OXYZ.

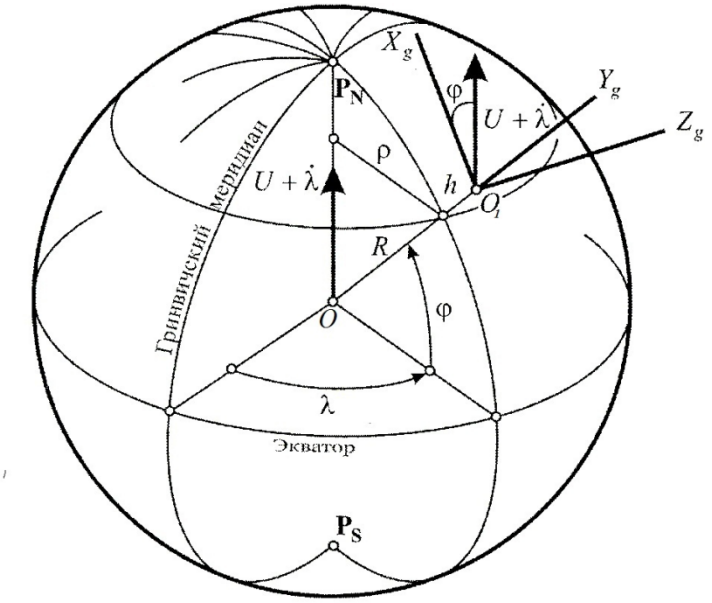

Рисунок 2.3 - Системы координат

Матрицу поворота (матрица направляющих косинусов) из связанной системы координат OXYZ (OBJ) в нормальную систему координат $OX_gY_gZ_g$ (NORM) рассчитываем по формуле:

$$C_{OBJ-NORM}=\begin{bmatrix}\cos\varphi\cos\psi\cos\theta-\cos\varphi\sin\psi-\cos\varphi\sin\psi\cos\theta-\sin\varphi\cos\psi\cos\varphi\sin\theta\\ \sin\varphi\cos\psi\cos\theta+\cos\varphi\sin\psi-\sin\varphi\sin\psi\cos\theta+\cos\varphi\cos\psi\sin\varphi\sin\theta\\ -\cos\psi\sin\theta\sin\theta\sin\psi\cos\theta\end{bmatrix} \quad (5)$$

Таким образом, любой вектор $\vec{R}_{OBJ}$ в связанной системе координат (OBJ) может быть пересчитан в нормальную систему координат (NORM) по формуле:

$$\vec{R}_{NORM}=C_{OBJ-NORM}\cdot\vec{R}_{OBJ} \quad (6)$$

Аналогичным образом рассчитываем матрицу поворота из нормальной системы координат, связанной с объектом, в геоцентрическую систему координат, связанную с Землей (ECEF), учитывая, что широта и долгота объекта (B и L, геодезические координаты, см. п. «Геодезические координаты») – это и есть углы поворота ECEF

до нормальной системы координат NORM. Тогда любой вектор $\dot{R}_{NORM}$ в нормальной системе координат, связанной с объектом (NORM), может быть пересчитан в геоцентрическую систему координат, связанную с Землей (ECEF), по формуле:

$$\dot{R}_{ECEF} = C_{NORM-ECEF} \cdot \dot{R}_{NORM} \quad (7)$$

Возможны и обратные преобразования, для этого достаточно определить обратную матрицу поворота по формуле:

$$C^{-1} = C^{T} \quad (8)$$

Поскольку геодезические координаты могут быть посчитаны в системах координат WGS-84, ПЗ-90.02, ПЗ-90.11, то и матрицы поворота могут быть вычислены для любой из этих геоцентрических систем координат, связанных с Землей (ECEF). Таким образом, мы имеем возможность пересчитать вектор в описанных системах координат в любую другую систему координат путём последовательных операций поворота.

Вектор скорости относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU)

Компоненты вектора скорости V_E, V_N, V_U относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) получаем напрямую из информационных сообщений НАП по любому из протоколов обмена. Система координат ENU – система координат, начало которой связано с центром масс объекта, а оси направлены на север, вертикально вверх и на восток. То есть эта система координат совпадает с нормальной системой координат, связанной с объектом (NORM), которая описана в п. «Углы пространственной ориентации». Следовательно, компоненты вектора скорости относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) – это проекции вектора скорости относительно Земли на оси нормальной системы координат, связанной с объектом (NORM), а вектор выражается формулой:

$$\vec{V}_{ENU} = \vec{V}_{NORM} = \begin{pmatrix} V_E \\ V_N \\ V_U \end{pmatrix} \quad (9)$$

Поскольку система координат ENU совпадает по ориентации с системой координат NORM, то и для поворота СК ENU можно применять матрицы поворота, рассчитанные для СК NORM.

Вектор скорости в прямоугольной связанной с Землей системе координат (по выбору оператора WGS-84, ПЗ-90.2, ПЗ-90.11)

Вектор скорости в прямоугольной связанной с Землей системе координат получаем путём пересчёта вектора скорости относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) (п. «Вектор скорости в связанной с объектом-носителем геодезической системе координат») из системы координат, связанной с объектом, в систему координат, связанную с Землей. Для этого используем матрицу поворота $C_{NORM-ECEF}$, рассчитанную в п. «Углы пространственной ориентации». Итоговая формула:

$$\vec{V}_{ECEF} = C_{NORM-ECEF} \vec{V}_{ENU} \quad (10)$$

Вектор ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU)

Вектор ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) получаем путём дифференцирования вектора скорости относительно Земли в той же системе координат (п. «Вектор скорости относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU)»). Формула выглядит следующим образом:

$$\vec{a}_{ENU} = \frac{d\vec{V}_{ENU}}{dt} \quad (11)$$

Вектор ускорения в прямоугольной связанной с Землей системе координат (по выбору оператора WGS-84, ПЗ-90.2, ПЗ-90.11)

Вектор ускорения в прямоугольной связанной с Землей системе координат получаем путём пересчёта вектора ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) (п. «Вектор ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU)») из системы координат, связанной с объектом, в систему координат, связанную с Землей. Для этого используем матрицу поворота $C_{NORM-ECEF}$, рассчитанную в п. «Углы пространственной ориентации». Итоговая формула:

$$\vec{a}_{ECEF} = C_{NORM-ECEF} \vec{a}_{ENU} \quad (12)$$

Линейные ускорения по заданным осям объекта-носителя

Линейные ускорения по заданным осям объекта-носителя – это компоненты вектора ускорения относительно Земли в связанной системе координат объекта-носителя. Этот вектор определяем по формуле:

$$\vec{a}_{OBJ} = C_{OBJ-NORM} \vec{a}_{NORM} \quad (13)$$

где $\vec{a}_{NORM}$ – вектор ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU) (п. «Вектор ускорения относительно Земли в связанной с объектом-носителем геодезической системе координат (ENU)»);

$C_{OBJ-NORM}$ – матрица поворота из геодезической системы координат, связанной с объектом, в связанную систему координат объекта (п. «Углы пространственной ориентации»).

Угловые скорости по заданным осям объекта-носителя

Угловые скорости по заданным осям объекта-носителя получаем путём дифференцирования углов пространственной ориентации (п. «Углы пространственной ориентации») по следующим формулам:

$$\omega_x = \frac{d\psi}{dt} \quad (14)$$

$$\omega_y = \frac{d\phi}{dt} \quad (15)$$

$$\omega_z = \frac{d\theta}{dt} \quad (16)$$

Преобразование пространственных прямоугольных координат из системы А в систему Б

Преобразование выполняется по формуле:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_Б = (1+m) \begin{pmatrix} 1 & +\omega_Z & -\omega_Y \\ -\omega_Z & 1 & +\omega_X \\ +\omega_Y & -\omega_X & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_A + \begin{pmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{pmatrix} \quad (17)$$

В этой формуле приняты следующие обозначения элементов трансформирования:

ΔX , ΔY , ΔZ – линейные элементы трансформирования систем координат при переходе из системы А в систему Б, м;

ω_x, ω_y, ω_z – угловые элементы трансформирования систем координат при переходе из системы А в систему Б, рад;

m – масштабный элемент трансформирования систем координат при переходе из системы А в систему Б.

При обратном преобразовании пространственных прямоугольных координат элементы трансформирования имеют те же значения, но с обратным знаком.

17. Расчет статистических параметров

Оценка проверяемого навигационного модуля производится после сбора навигационных параметров и сохранения в файл или в режиме реального времени, для чего требуется нижеследующие характеристики.

Оценка погрешности определения координат

Определить систематическую составляющую погрешности определения координат в плане (широты и долготы) по формулам (18), (19), например, для координаты В (широты):

$$\Delta B(j) = B(j) - B_{on}(j) \quad (18)$$

$$dB = \frac{1}{N} \cdot \sum_{j=1}^{N} \Delta B(j) \quad (19)$$

где $B_{оп}(j)$ – опорное значение координаты В в j-ый момент времени, угл. сек;

B(j) – измеренное значение координаты В в j-ый момент времени, угл. сек;

N – количество измерений.

Аналогичным образом определить систематическую составляющую погрешности определения координаты L (долготы).

Определить среднее квадратическое отклонение (СКО) случайной составляющей погрешности определения координат по формуле (20), например, для координаты В (широты):

$$\sigma_B = \sqrt{\frac{\sum_{j=1}^{N} (\Delta B(j) - dB)^2}{N - 1}} \quad (20)$$

Аналогичным образом определить СКО случайной составляющей погрешности определения координаты L (долготы).

Погрешность определения высоты

Определить систематическую составляющую погрешности определения высоты аналогично погрешности определения широты и долготы по формулам (18), (19), заменив переменные, связанные с широтой, на аналогичные, связанные с высотой.

Определить среднее квадратическое отклонение (СКО) случайной составляющей погрешности определения высоты аналогично СКО погрешности определения широты и долготы по формуле (20).

Итоговая погрешность определений высоты будет представлять собой сумму систематической и случайной погрешностей в различных соотношениях, в зависимости от необходимого уровня доверительной вероятности.

Погрешность определения составляющих вектора скорости

В целом процедура обработки аналогична процедуре обработки координат. Определить систематическую составляющую погрешности определения каждой составляющей вектора скорости аналогично погрешности определения широты и долготы по формулам (18), (19), заменив переменные, связанные с широтой, на аналогичные, связанные с составляющими вектора скорости.

Определить среднее квадратическое отклонение (СКО) случайной составляющей погрешности определения составляющих вектора скорости аналогично СКО погрешности определения широты и долготы по формуле (20).

Итоговая погрешность определений высоты будет представлять собой сумму систематической и случайной погрешностей в различных соотношениях, в зависимости от необходимого уровня доверительной вероятности.

18. Разработка графического интерфейса пользователя

Большинство программ имеет графический интерфейс пользователя (GUI, Graphical User Interface). Виджеты (widgets) – это «строительный материал» для создания графического интерфейса. Виджет – это компонент, предназначенный для отображения графической информации и способный выполнять различные действия, например, реагировать на поступающие сигналы и события или отправлять сигналы другим виджетам. Qt предоставляет широкий набор виджетов: от кнопок меню до диалоговых

окон, необходимых для создания профессиональных приложений. Наследуя классы уже существующих виджетов, можно создавать свои собственные виджеты, которые будут наиболее полно соответствовать требованиям при построении графического интерфейса.

Основные классы, используемые в фреймворке Qt для разработки графического интерфейса пользователя: QWidget, QMainWindow, QPainter.

Класс QWidget – базовый класс для всех объектов графического интерфейса пользователя. Виджет – самый простой базовый объект пользовательского интерфейса. Он может получать различные события: от мыши, от клавиатуры и другие события от оконной системы, и сам рисует свое изображение. Отрисовка виджета происходит в порядке наложения, а сам виджет прямоугольной формы. Класс QWidget может использовать возможности механизма сигнала и слотов, а также механизм объектной иерархии, так как унаследован от основного в Qt класса QObject. Благодаря этому виджеты могут иметь отображающихся внутри предка потомков. Это является очень важным моментом, так как каждый виджет может также являться контейнером и для других виджетов (в Qt контейнеры и элементы управления не разделяются между собой). Виджеты верхнего уровня (top-level widgets) – это виджеты без предка. Они имеют свое собственное окно. Все виджеты без исключения могут быть виджетами верхнего уровня. Для ручного изменения позиции виджета-потомка внутри виджета-предка можно воспользоваться методом setGeometry(). Используя специальные классы компоновки (layout) можно автоматически менять позицию. Вызов метода используется для отображения на экране виджета, а для скрытия – метод hide().

Большинство унаследованных от QWidget классов имеют конструктор с двумя параметрами:

QWidget (QWidget* parentwidget = 0, Qt::WindowFlags flag = 0)

Класс QMainWindow является главным окном приложения. Оно предоставляет структуру для создания графического интерфейса пользователя. У класса QMainWindow есть собственный компоновщик, имеющий центральную область, которую может занять любой виджет.

Кнопки — наиважнейший и самый часто встречаемый элемент графического интерфейса пользователя. Даже если в коде приложения не создавать ни одной, то все равно в правом верхнем углу окна будут присутствовать кнопки управления окном программы. Кнопка может быть в двух состояниях – нажата (on) или отжата (off).

Класс QAbstractButton – базовый класс для всех кнопок. Три основных вида кнопок применяются в приложениях: нажимающиеся кнопки (QPushButton), которые обычно называют просто кнопками, переключатели (QRadioButton) и флажки (QCheckBox). В классе QAbstractButton реализованы методы и возможности, присущие всем кнопкам. Все кнопки могут содержать текст, который можно передать как в конструкторе первым параметром, так и установить с помощью метода setText(). Метод text() класса QAbstractButton используется для получения текста.

На рисунке 2.4 приводится экранная форма главного окна пользовательского интерфейса. В левом углу расположен блок настроек программы, где выбирается рабочая папка проекта, указываются системы координат как устройства (файла) для проверки, так и для эталонных значений с устройства (из файла). Также выбирается файл для проверки и проверяется связь с устройством. Кнопка в блоке «Управление» активируется только при условии, что все предыдущие этапы выполнены (установление связи с устройствами или выбор эталонного файла и файла для проверки). Справа от блока настроек расположена таблица параметров. Обозначения по горизонтали: оценка математического ожидания (M), среднеквадратическое отклонение (σ), погрешность по уровню доверительной вероятности P=0.67, погрешность по уровню доверительной вероятности P=0.95. Обозначения по вертикали: B - широта, L - долгота, H - высота, V - скорость, П2D – погрешность координат в плане для M, σ, P=0.67 и P=0.95. Еще правее приводится статистика по количеству определенного типа сообщений, попаданию по времени, а также количество «выбросов», которые приходятся на долготу, широту, высоту и скорость. Нижнюю часть занимают графики погрешности для долготы, широты и высоты соответственно. По оси абсцисс – момент времени измерения, по оси ординат – разница измерений.

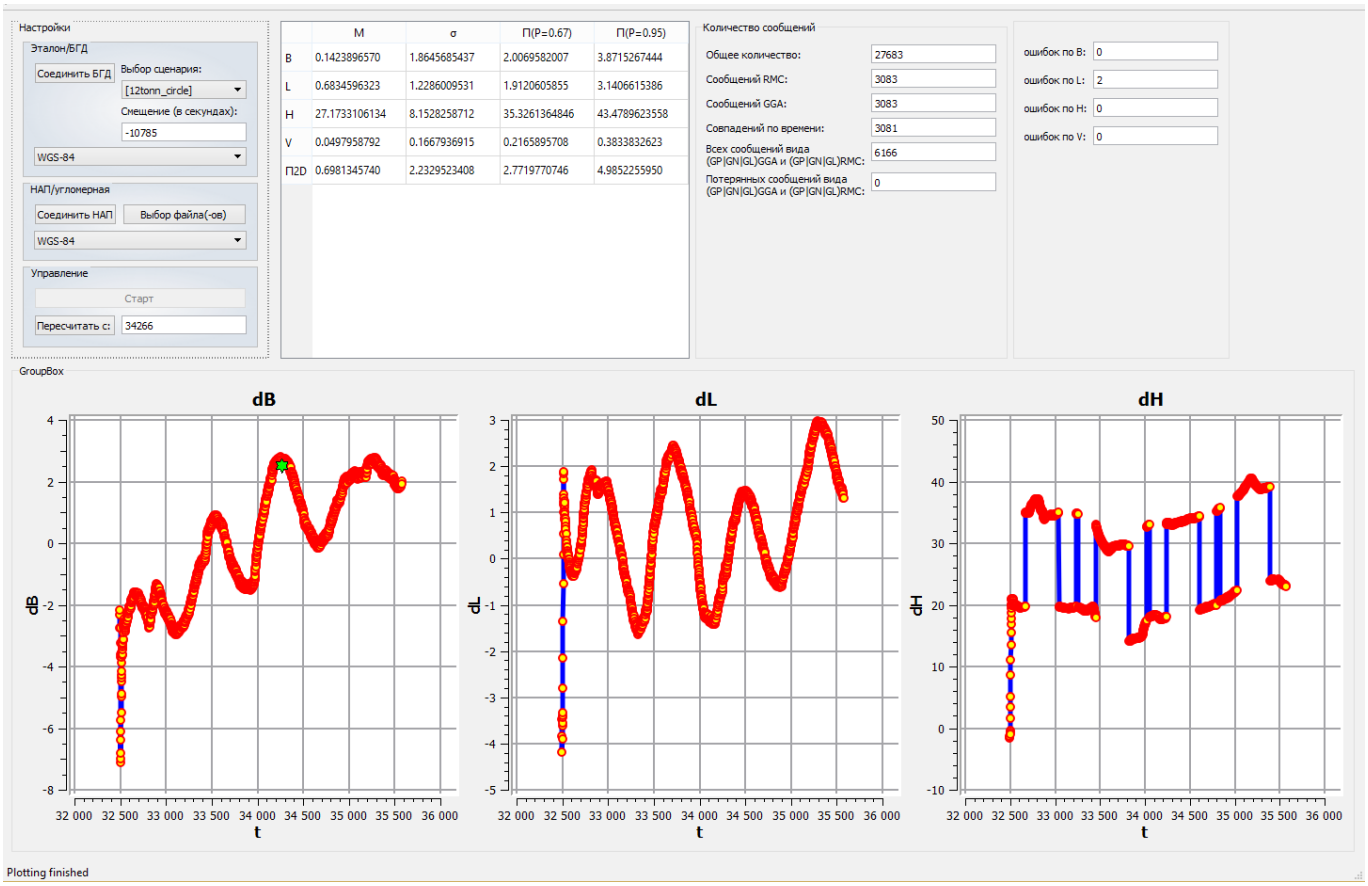

Рисунок 2.4 - Экранная форма главного окна пользовательского интерфейса

Выводы

В конструкторском разделе был описан кроссплатформенный фреймворк Qt, выбор инструментальных средств, среды разработки. Также была разработана архитектура ПМ ОХА, схема данных и схема алгоритма. Приведено описание форматов данных, с которыми работает модуль, и его конфигурационные файлы; выработаны алгоритмы и формулы расчета параметров. Кроме того, приведено описание графического интерфейса пользователя.

19. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

20. Отладка и тестирование

Отладка программы – это специальный этап в разработке программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

При разработке ПМ ОХА применялся отладчик GNU Debugger.

GNU Debugger – переносимый отладчик проекта GNU, работающий на многих UNIX-подобных системах. Отладчик умеет работать со многими языками программирования, включая Си, C++, Objective-C, FreePascal, Ada, FreeBASIC и Фортран. GDB – свободное программное обеспечение, распространяемое по лицензии GPL.

GDB предлагает обширные средства по слежению за выполнением компьютерных программ и контролю их состояния, а также по изменению хода выполнения. Пользователь может изменять значения внутренних переменных приложений и вызывать функции независимо от обычного поведения программы. GDB может отлаживать исполняемые файлы в формате a.out, COFF (в том числе исполняемые файлы Windows), ECOFF, XCOFF, ELF, SOM, использовать отладочную информацию в форматах stabs, COFF, ECOFF, DWARF, DWARF2. Наибольшие возможности отладки предоставляет формат DWARF2.

В соответствии с идеологией ведущих разработчиков FSF, GDB вместо собственного графического пользовательского интерфейса предоставляет возможность подключения к внешним IDE, управляющим графическим оболочкам либо использовать стандартный консольный текстовый интерфейс. Для сопряжения с внешними программами можно использовать язык текстовой строки, текстовый язык управления gdb/mi либо интерфейс для языка Python.

Тестирование программного обеспечения (Software Testing) – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

Для тестирования ПМ использовался компонент Qt фреймворка QtTest.

21. Виды ошибок

Существует три основных типа ошибок: ошибки этапа компиляции, ошибки этапа выполнения и логические ошибки.

Ошибки этапа компиляции, или синтаксические ошибки, происходят, когда исходный код нарушает правила синтаксиса языка программирования. Компилятор не может скомпилировать программу, пока она будет содержать недопустимые операторы или некорректные синтаксические конструкции. Когда компилятор встречает оператор, который не может распознать, соответствующий файл выводится в окне редактирования исходного кода в графической среде разработки, курсор переводится на то место, которое компилятор не смог распознать, и выводится сообщение об ошибке. Компилятор, работающий в режиме командной строки, также выводит некоторую полезную информацию. Когда он находит синтаксическую ошибку, то выводит содержащую ошибку строку с номером этой строки и сообщением об ошибке. Наиболее общей причиной ошибок этапа компиляции являются ошибки набора (опечатки), пропущенные точки с запятой, ссылки на неописанные переменные, передача неверного числа (или типа) параметров функции и присваивание переменной значений неверного типа.

Ошибки этапа выполнения, или семантические ошибки, происходят на этапе компиляции программы, которая при выполнении производит недопустимые действия. То есть программа содержит допустимые операторы языка программирования, но при выполнении операторов что-то происходит неверно. Например, программа может пытаться открыть для ввода несуществующий файл или выполнить деление на ноль. При обнаружении такой ошибки среда разработки завершает выполнение программы и выводит сообщение следующего вида: «Run-time error ## at segmentation fault».

Логические ошибки – это ошибки проектирования и реализации программы. То есть ваши операторы допустимы и что-то делают, но не то, что вы предполагали. Эти ошибки часто трудно отследить, поскольку IDE не может найти их автоматически как синтаксические и семантические ошибки. Однако среды разработки включают в себя средства отладки, помогающие найти логические ошибки.

22. Обнаружение ошибок

Ситуации, по которым определяется наличие ошибки в программе:

- отсутствует уверенность в том, что программа начала выполняться;
- программа начала выполняться, но произошел преждевременный останов с выдачей или без выдачи сообщений о системной ошибке;
- программа начала выполняться, но зациклилась;
- программа выдала неправильную информацию.

Любая из этих ситуаций требует от программиста проверки последовательности выполнения команд. Обычно для этого пригодна трассировка. Процесс обнаружения ошибок характеризуется выявлением двух мест в программе:

- точки обнаружения;
- точки происхождения.

Точка обнаружения – место в программе, где ошибка себя проявляет или становится очевидной. Точка происхождения – место в программе, где возникают условия для появления ошибки. Точка обнаружения выявляется первой и служит отправным пунктом для поиска точки происхождения. Действительная ошибка исходит не из точки обнаружения, а из точки происхождения.

23. Методы отладки

При непредвиденном поведении программы бывает, что причина достаточно очевидна, и можно быстро исправить код программы. Но другие ошибки более трудноуловимы и вызываются взаимодействием множества различных частей программы. В этих случаях лучше всего остановить программу в заданной точке, пройти ее шаг за шагом и просмотреть состояние переменных и выражений. Такое управляемое выполнение – ключевой элемент отладки.

Команды выполнения по шагам Step Over и трассировки Trace дают возможность построчного выполнения программы. Единственное отличие выполнения по шагам от трассировки состоит в том, как они работают с вызовами функций. Выполнение по шагам вызова функции интерпретирует вызов как простой оператор, и после завершения

подпрограммы возвращает управление на следующую строку. Трассировка подпрограммы загружает код этой подпрограммы и продолжает ее построчное выполнение.

Существует два способа сообщить IDE, что программу необходимо выполнить до определенной точки, а затем остановить:

Первый и простейший способ состоит в том, чтобы найти позицию в программе, где вы хотите остановиться, затем установить соответствующий флаг прерывания. Второй способ состоит в том, чтобы остановить в определенной заданной точке вашу программу. Эта точка называется точкой останова.

24. Методы тестирования

Существует несколько методов тестирования:

- тестирование программ методом «чёрного ящика» (Blackbox testing);
- тестирование программ методом «белого ящика» (Whitebox);
- тестирование программ методом «серого ящика» (Greybox);
- тестирование нефункциональных аспектов программы.

В терминологии профессионалов тестирования (программного и некоторого аппаратного обеспечения) фразы «тестирование белого ящика» и «тестирование черного ящика» относятся к тому, имеет ли разработчик тестов и тестировщик доступ к исходному коду тестируемого ПО, или же тестирование выполняется через пользовательский интерфейс либо прикладной программный интерфейс, предоставленный тестируемым модулем.

При тестировании белого ящика разработчик теста имеет доступ к исходному коду и может писать код, который связан с библиотеками тестируемого ПО. Это типично для юнит-тестирования, при котором тестируются только отдельные части системы. Оно обеспечивает работоспособность и устойчивость компонентов конструкций до определенной степени.

При тестировании чёрного ящика тестировщик имеет доступ к ПО только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Например, тестирующий модуль может виртуально нажимать клавиши или кнопки мыши в тестируемой программе с помощью механизма взаимодействия процессов и проверить, все ли идет правильно и что эти события вызывают

тот же отклик, что и реальные нажатия клавиш и кнопок мыши. Как правило, тестирование чёрного ящика ведётся с использованием спецификаций или иных документов, описывающих требования к системе.

25. Отладка с помощью GDB в среде QtCreator

QtCreator не имеет собственного отладчика. Вместо этого он предоставляет графический интерфейс к различным отладчикам.

При отладке программного модуля использовался отладчик GDB (GNU Symbolic Debugger). Процесс отладки программы с использование GDB в среде QtCretor представлен на рисунке 3.1. На изображении присутствует вид «Локальные переменные», вид «Точки останова» и стек вызовов функций.

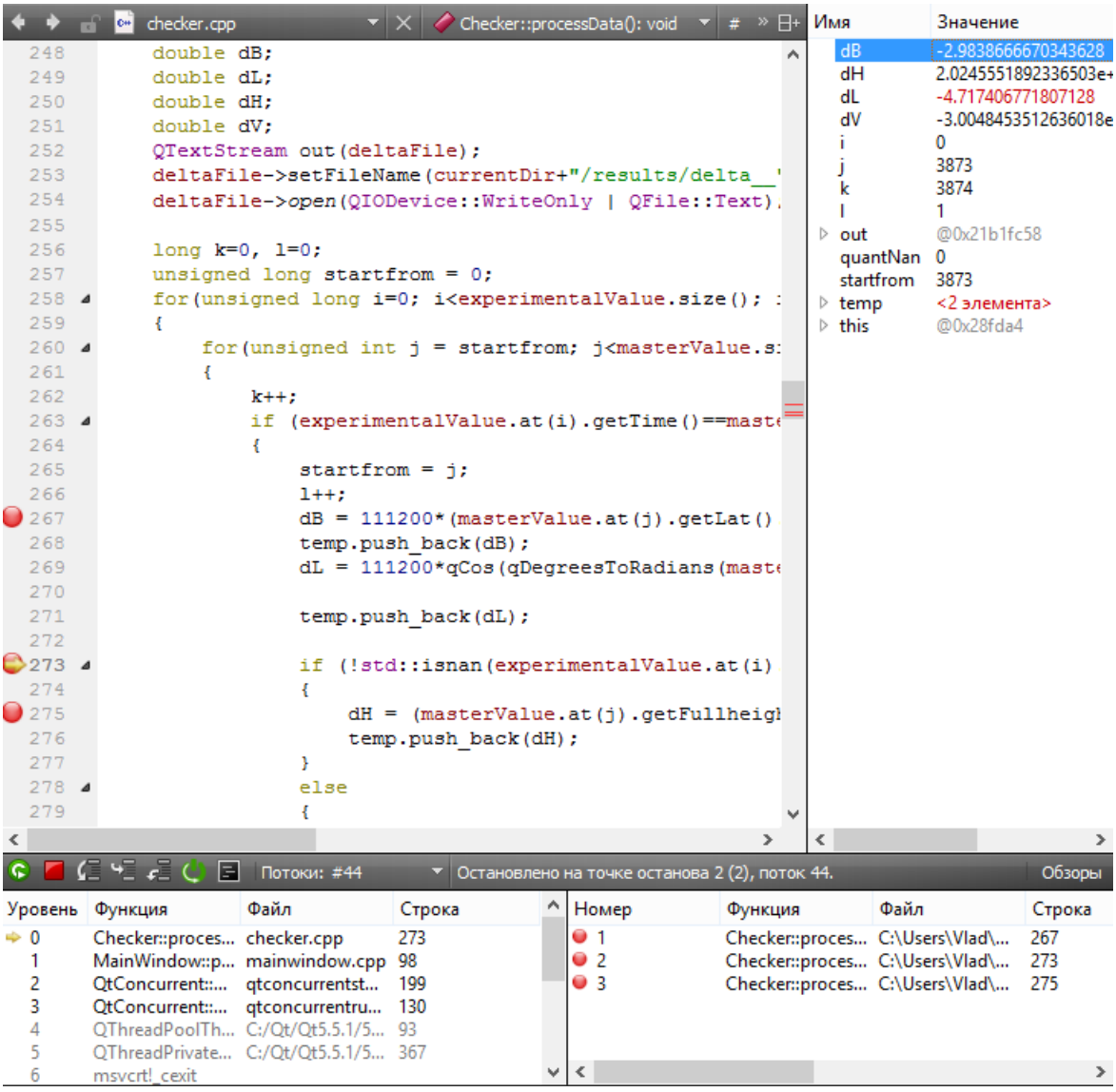

Рисунок 3.1 - Отладка программного модуля в среде QtCreator

В режиме «Отладка» взаимодействие с отладчиком возможно различными способами, включая следующие:

– прохождение программы построчно или по каждой инструкции (трассировка);
– прерывание работы программы;
– установка точки останова;
– просмотр содержимого стека вызовов;
– просмотр и изменение регистров и содержимого памяти отлаживаемого приложения;
– просмотр и изменение регистров и содержимого памяти локальных и глобальных переменных;
– просмотр списка загруженных разделяемых библиотек;
– создание и просмотр снимков текущего состояния.

Qt Creator отображает «сырую» информацию, предоставляемую отладчиками, явным и лаконичным образом с целью упростить процесс отладки, насколько возможно, без ограничения возможностей отладчиков.

В дополнение к базовой функциональности IDE, предоставляемой просмотром стека, просмотром локальных и наблюдаемых переменных, регистров и так далее, Qt Creator имеет дополнительные особенности, делая отладку основанных на Qt приложений проще. Подключаемый модуль отладчика знает о внутреннем устройстве некоторых классов Qt, таких как QString, контейнеры Qt и, что особенно важно, QObject (и унаследованные от него классы), а также большинство контейнеров C++ Standard Library и некоторых расширений gcc и Symbian. Понимание их устройства используется для представления содержимого таких классов удобным способом.

Процесс отладки в Qt Creator может включать следующую последовательность действий:

1. Нажать на область между границей окна и номером строки для активации точки останова;

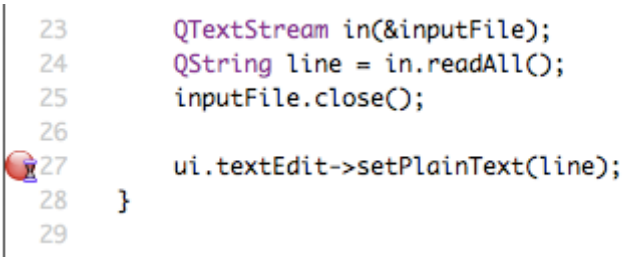

Рисунок 3.2 – Установка точки останова

2. Нажать «горячую» клавишу F5 или выбрать пункт меню «Отладка > Начать отладку > Начать отладку»;

3. Выбрать вкладку «Точки останова» (breakpoints) для просмотра и управления точками останова (активация/деактивация/удаление);

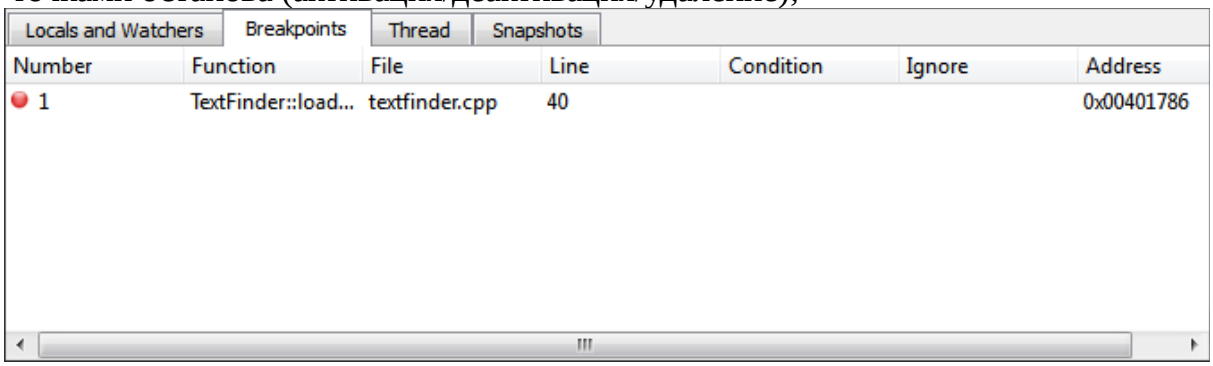

Рисунок 3.3 – Вид «Точки останова»

4. Чтобы удалить или деактивировать точку останова необходимо нажать на нее правой кнопкой мыши и выбрать пункт «Удалить точку останова» или «Деактивировать точку останова»;
5. Для просмотра содержимого переменной необходимо перейти на вкладку «Переменные»;

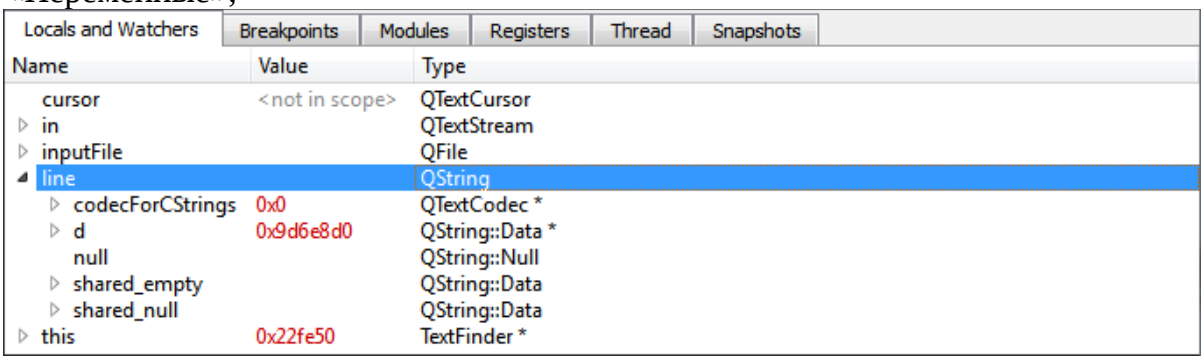

Рисунок 3.4 – Вид «переменные»

6. Если переменная представляет собой составной объект, например, список или вектор и т.п., ее можно раскрыть для просмотра содержимого;
7. Для редактирования переменной в режиме отладки можно дважды щелкнуть мышью на поле в столбце «Значение» (Type);
8. Для просмотра информации о потоках необходимо переключиться на вкладку «Потоки» (Thread);
9. Для пошаговой отладки можно воспользоваться «горячими» клавишами или использовать следующие кнопки (рис.3.5).

Рисунок 3.5 – Кнопки управления

Подключаемый модуль отладчика работает в различных режимах в зависимости от того, где и как процесс запущен и выполняется. Некоторые режимы доступны только для определённых операционных систем или платформ.

Запуск отладчика возможен в следующих режимах:

– простой режим – для отладки локально запущенных приложений на Qt с графическим интерфейсом пользователя;
– терминал – для отладки локально запущенных процессов, работающих в консольном режиме; обычно это приложения без графического интерфейса пользователя;
– подключённый – для отладки локальных процессов, которые были запущены вне среды разработки Qt Creator;
– удалённый – для отладки процессов, запущенных на другом компьютере;
– ядро – для отладки аварийно-завершившихся процессов Unix;
– post-mortem – для отладки аварийно-завершившихся процессов Windows;
– TRK – для отладки процессов, запущенных на устройствах, работающих под операционной системой Symbian.

При нажатии кнопки «Начать отладку» отладчик запускается в подходящем режиме (простой, терминал или TRK) в зависимости от настроек сборки и запуска для активного проекта. Для запуска отладчика в других режимах можно воспользоваться пунктами меню Отладка.

В режиме «Отладка» несколько прикрепляемых виджетов используются для взаимодействия с отлаживаемой программой. Часто используемые виджеты отображаются по умолчанию; используемые редко спрятаны. Для изменения настроек по умолчанию выберите «Отладка», а затем выберите «Виды». Здесь вы можете заблокировать и разблокировать положение ваших видов, а также показать или спрятать их. Среди видов, которые вы можете отобразить, есть «Точки останова», «Дизассемблер», «Модули», «Регистры», «Отладчик», «Стек» и «Поток». Расположение прикрепляемых виджетов будет сохранено для последующих сессий.

Установить точки останова можно:

– в конкретной строке кода программы;
– на определенной функции, по ее имени.

При запуске отладчика Qt Creator проверит актуальность текущей программы и пересоберёт её при необходимости. Отладчик получит контроль и запустит программу.

Замечание: Запуск программы в отладчике может занять заметное количество времени, обычно в диапазоне от нескольких секунд до нескольких минут, если используются сложные особенности (как QtWebKit). При прерывании выполнения программы по достижении точки останова Qt Creator:

– получает данные, представляющие стек вызовов текущего состояния программы;
– получает содержимое локальных переменных;
– просматривает вид «Наблюдаемые»;
– обновляет виды «Регистры», «Модули» и «Дизассемблер».

Когда отлаживаемая программа прерывается, Qt Creator отображает последовательность вызовов процедур в стеке вызовов, приводящих к текущему состоянию программы. Стек строится из кадров стека вызовов, каждый из которых представляет конкретную функцию. Для каждой функции Qt Creator попытается получить имя файла и номер строки соответствующих файлов исходных кодов. Эти данные отображаются в виде «Стек».

Так как стек вызовов, приводящий к текущему состоянию, может происходить через код, для которого нет отладочной информации, не все кадры стека могут иметь соответствующие положения в исходном коде. Эти кадры будут иметь серый цвет в виде «Стек». Если щёлкнуть на кадре с известным положением в исходном коде, текстовый редактор перейдёт в соответствующее положение и обновит вид «Локальные и наблюдаемые переменные», делая их видимыми, как будто программа была прервана перед входом в функцию.

Если прервано многопоточное приложение, то можно использовать вид «Поток» или выпадающий список с именем «Поток» в строке состояния отладчика для переключения от одного потока к другому. Вид «Стек» будет настраивать себя соответственно.

Когда бы программа ни остановилась под контролем отладчика, он получает информацию о верхнем кадре стека и отображает его в виде «Локальные и наблюдаемые переменные». Обычно он включает информацию о параметрах функции в этом кадре, а также локальные переменные.

Составные переменные структур или классов в виде будут отображены как «раскрываемые». Помимо отображаемых значения и типа пользователь может просмотреть и перейти к низкоуровневой компоновке данных объекта.

Замечание: gdb работает с оптимизированной сборкой на Linux и Mac OS X. Тем не менее оптимизация может привести к изменению порядка инструкций и иногда даже к

полному удалению некоторых локальных переменных. В этом случае вид «Локальные и наблюдаемые переменные» может показывать неожиданные данные.

Отладочная информация, предоставляемая gcc, не включает достаточной информации о времени, когда переменная инициализируется. Поэтому Qt Creator не может сказать, является ли содержимое начальной переменной «настоящими данными» или «начальным шумом». Если QObject появляется неинициализированным, его значение будет указано как «вне области». Тем не менее не все неинициализированные объекты могут быть распознаны таким образом.

Вид «Локальные и наблюдаемые переменные» также предоставляет доступ к наиболее мощной особенности отладчика – полному отображению данных, принадлежащих базовым объектам Qt. Для включения этой особенности необходимо выбрать пункт «Использовать помощник отладчика» в меню Отладка. Вид «Локальные и наблюдаемые переменные» будут автоматически переупорядочены для отображения высокоуровневого вида объектов. Например, в случае с QObject вместо отображения указателя на некоторую закрытую структуру данных вы увидите список потомков, сигналы и слоты.

Аналогично вместо отображения множества указателей и целых чисел отладчик Qt Creator покажет содержимое QHash или QMap в упорядоченном виде. Также отладчик покажет данные доступа для QFileInfo и предоставит доступ к «реальному» содержимому QVariant.

Вид «Локальные и наблюдаемые переменные» может быть использован для изменения содержимого переменной простого типа данных, такого как int или float, когда программа прервана.

По умолчанию виды «Дизассемблер» и «Регистры» спрятаны. Вид «Дизассемблер» отображает дизассемблированный код для текущей функции; вид «Регистры» отображает текущее состояние регистров CPU. Оба вида полезны для низкоуровневых команд, таких как «Перейти на одну инструкцию» и «Перейти через инструкцию».

26. Тестирование с помощью QtTest

Тесты желательно создавать до начала реализации кода. Это позволяет при написании теста лучше осмыслить и понять задачу. Скомпилированный тест

представляет собой готовую к исполнению программу. Для демонстрации рассмотрим простой пример: предположим, что нам нужно реализовать класс с методами для нахождения максимума и минимума двух чисел.

Первая задача заключается в подготовке тестовых данных, которые будут выступать в качестве образцов для тестирования. Возьмем для этой цели четыре пары чисел: (25, 0), (-12, –15), (2007, 2007) и (-12, 5).

Теперь, когда тестовые данные готовы, можно начинать писать тесты. Существуют соглашения для названия тестирующего класса и его методов, которые успели закрепиться и зарекомендовать себя на практике с наилучшей стороны. А именно:

– называть тестирующий класс именем тестируемого класса с префиксом Test. Например, если мы тестируем класс MyClass, то тестирующий класс будет называться Test_MyClass;
– называть тестовые слоты (методы) именами тестируемых методов.

В примере ниже показана программа, которая должна проводить тест методов min() и max() класса MyClass. В тестовой программе необходимо включить заголовочный файл QTest. Тестирующий класс должен быть унаследован от класса QObject для создания специальной метаинформации, содержать в своем определении макрос класса QOBJECT. Это позволит вызывать слоты класса при исполнении, включая его тестовые слоты в секции private. Макрос QCOMPARE() принимает два аргумента: полученный и ожидаемый результат, и сравнивает их. Если значения не совпадают, то исполнение тестового метода прерывается с сообщением о непройденном тесте.

```
#include <QtTest>
#include "MyClass.h"

class Test MyClass : public QObject {
Q_OBJECT
private slots:
        void min();
        void max();

void Test MyClass::min()
{
        MyClass myClass;
        QCOMPARE(myClass.min(25, 0), 0);
        QCOMPARE(myClass.min(-12, -5), -12);
        OCOMPARE(myClass.min(2007, 2007), 2007);
        QCOMPARE(myClass.min(-12, 5), -12);
}
void Test MyClass::max()
{
        MyClass myClass;
        QCOMPARE(myClass.max(25, 0), 25);
        QCOMPARE(myClass.max(-12, -5), -5);
        QCOMPARE(myClass.max(2007, 2007), 2007);
```

```
        QCOMPARE(myClass.max(-12, 5), 5);
}

QTEST MAIN(Test MyClass)
#include "test.moc"
```

При первом проведении теста полезно начать с проверки на отказ, то есть модифицировать проверяемый метод так, чтобы тест завершался неудачей. Это поможет убедиться в том, что тест действительно выполняется и проверяет то, что требуется. Для этого в методах min() и max() класса MyClass поменяем знаки сравнения на противоположные. Теперь скомпилируем и запустим тест. На экране появится следующее:

********* Start testing of Test MyClass *********

Config: Using QTest library 5.2, Qt 5.2

PASS : TestMyClass::initTestCase()

FAIL! : Test MyClass::min() Compared values are not the same

Actual (myClass.min(25, 0)): 25

Expected (0): 0

test.cpp(25) : failure location

FAIL! : Test MyClass::max() Compared values are not the same

Actual (myClass.max(25, 0)): 0

Expected (25): 25

test.cpp(35) : failure location

PASS : TestMyClass::cleanupTestCase()

Totals: 2 passed, 2 failed, 0 skipped

********* Finished testing of Test MyClass *********

Методы initTestCase() и cleanupTest() вызываются в начале и конце теста соответственно. Эти методы не трактуются как тестовые методы. Они выполняются при запуске тестов и служат для инициализации и очистки теста. Кроме того, на экране мы видим информацию о том, что тест прошел неудачно: сообщение FAIL!, имена тестов Test_MyClass::min() и Test_MyClass::max(), актуальные значения (Actual) и ожидаемые (Expected). Наш тест завершился неудачей, а это значит, что проверка работы теста удалась – он действительно способен отслеживать ошибки. Теперь поменяем обратно операторы сравнения в классе MyClass. Скомпилируем и запустим тест еще раз. Увидим на экране следующие сообщения:

********* Start testing of Test MyClass *********

Config: Using QTest library 5.2, Qt 5.2

PASS: Test MyClass::initTestCase()

PASS: Test MyClass::min()

PASS: Test MyClass::max()

PASS: Test MyClass::cleanupTestCase()

Totals: 4 passed, 0 failed, 0 skipped

********* Finished testing of Test MyClass *********

Это говорит о том, что все наши тесты прошли удачно.

Для минимизации дублирования кода модуль QtTest предоставляет возможность проведения тестов с передачей данных. Такой подход позволяет отделить тестовый код от данных, поместив их в отдельное место. Этим местом является слот, который предоставляется для каждого тестирующего слота. Он должен называться так же, как и тестирующий, но с постфиксом _data.

27. Компилятор GCC

GCC – это свободно доступный оптимизирующий компилятор для языков C, C++.

Программа gcc, запускаемая из командной строки, представляет собой надстройку над группой компиляторов. В зависимости от расширений имен файлов, передаваемых в качестве параметров, и дополнительных опций gcc запускает необходимые препроцессоры, компиляторы, линкеры. Файлы с расширением .cc или .C рассматриваются как файлы на языке C++, файлы с расширением .c как программы на языке C, а файлы с расширением .o считаются объектными. Чтобы откомпилировать исходный код C++, находящийся в файле F.cc, и создать объектный файл F.o, необходимо выполнить команду:

gcc -c F.cc, где опция – c означает «только компиляция».

Чтобы скомпоновать один или несколько объектных файлов, полученных из исходного кода - F1.o, F2.o, ... – в единый исполняемый файл F, необходимо ввести команду:

gcc -o F F1.o F2.o, где опция -o задает имя исполняемого файла.

Можно совместить два этапа обработки – компиляцию и компоновку – в один общий с помощью команды:

gcc -o F <compile-and-link-options> F1.cc ... -lg++ <other-libraries>

<compile-and-link –options> – возможные дополнительные опции компиляции и компоновки. Опция -lg++ указывает на необходимость подключить стандартную библиотеку языка C++, <other-libraries> – возможные дополнительные библиотеки.

После компоновки будет создан исполняемый файл F, который можно запустить с помощью команды ./F <arguments>. Строка <arguments> определяет аргументы командной строки программы. В процессе компоновки очень часто приходится использовать библиотеки. Библиотекой называют набор объектных файлов, сгруппированных в единый файл и проиндексированных. Когда команда компоновки обнаруживает некоторую библиотеку в списке объектных файлов для компоновки, она проверяет, содержат ли уже скомпонованные объектные файлы вызовы для функций, определенных в одном из файлов библиотек. Если такие функции найдены, соответствующие вызовы связываются с кодом объектного файла из библиотеки. Библиотеки могут быть подключены с помощью опции вида -lname. В этом случае в стандартных каталогах, таких как /lib , /usr/lib, /usr/local/lib, будет проведен поиск библиотеки в файле с именем libname.a. Библиотеки должны быть перечислены после исходных или объектных файлов, содержащих вызовы к соответствующим функциям.

Среди множества опций компиляции и компоновки наиболее часто употребляются следующие:

Таблица 3.1 – Опции компиляции

Опции	Назначение
-c	Эта опция означает, что необходима только компиляция. Из исходных файлов программы создаются объектные файлы в виде name.o. Компоновка не производится.
-Dname=value	Определить имя name в компилируемой программе как значение value. Эффект такой же, как наличие строки #definenamevalue в начале программы. Часть = value может быть опущена, в этом случае значение по умолчанию равно 1.
-o file-name	Использовать file-name в качестве имени для создаваемого файла.

-lname	Использовать при компоновке библиотеку libname.so
-Llib-path -Iinclude-path	Добавить к стандартным каталогам поиска библиотек и заголовочных файлов пути lib-path и include-path соответственно.
-g	Поместить в объектный или исполняемый файл отладочную информацию для отладчика gdb. Опция должна быть указана и для компиляции, и для компоновки. В сочетании -g рекомендуется использовать опцию отключения оптимизации -O0
-MM	Вывести зависимости от заголовочных файлов, используемых в Си или C++ программе, в формате, подходящем для утилиты make. Объектные или исполняемые файлы не создаются.
-pg	Поместить в объектный или исполняемый файл инструкции профилирования для генерации информации, используемой утилитой gprof. Опция должна быть указана и для компиляции, и для компоновки. Собранная с опцией -pg программа при запуске генерирует файл статистики. Программа gprof на основе этого файла создает расшифровку, указывающую время, потраченное на выполнение каждой функции.
-Wall	Вывод сообщений обо всех предупреждениях или ошибках, возникающих во время компиляции программы.
-O1, -O2,-O3	Различные уровни оптимизации
-O0	Не оптимизировать. Если вы используете многочисленные -O опции с номерами или без номеров уровня, действительной является последняя такая опция.
-I	Используется для добавления ваших собственных каталогов для поиска заголовочных файлов в процессе сборки.
-L	Передается компоновщику. Используется для добавления ваших собственных каталогов для поиска библиотек в процессе сборки.
-l	Передается компоновщику. Используется для добавления ваших собственных библиотек для поиска в процессе сборки.

28. Библиотека qwt

Qwt (Qt Widgets for Technical Applications) – свободно распространяемая библиотека графических компонентов (т.н. «виджетов») для платформы Qt, позволяющая решать задачи отображения различной инженерно-технической информации, а также организации взаимодействия с пользователем посредством графических элементов управления – как общепринятых (переключатели, кнопки, шкалы), так и специализированных (компасы, термометры, индикаторы скорости, времени и т.д.). Помимо богатого набора элементов управления библиотека Qwt также включает в себя развитые средства отображения

диаграмм (графики, гистограммы и т.д.) с возможностью анимации и масштабирования отображаемых данных. Рассмотрим процесс создание и оформления графика с использованием библиотеки Qwt на простом примере.

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

#include <qwt_plot_canvas.h>
#include <qwt_legend.h>
#include <qwt_plot_grid.h>
#include <qwt_plot_curve.h>
#include <qwt_symbol.h>

#include "scrollzoomer.h"

namespace Ui {
    class MainWindow;
}

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

    QwtLegend *legend;              // легенда
    QwtPlotGrid *plotgrid;          // координатная сетка
    QwtPlotCurve *curvone,*curvtwo; // кривые

    ScrollZoomer *zoom;         // масштабированит и перемещения графика
};

#endif // MAINWINDOW_H
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // устанавливаем заголовок графика
    ui->somePlot->setTitle(QString::fromLocal8Bit("Зависимость U(t)"));

    // создаем легенду и запрещаем ее редактировать
    legend = new QwtLegend();
    legend->setItemMode(QwtLegend::ReadOnlyItem);
```

```
    ui->somePlot->insertLegend(legend,QwtPlot::TopLegend); //  добавляем легенду на
график

    plotgrid = new QwtPlotGrid; // создаем сетку
    plotgrid->enableXMin(true); // задаем отображение дополнительных линий сетки
    // назначаем цвета для линий сетки:
    plotgrid->setMajPen(QPen(Qt::black,0,Qt::DotLine)); // черный для основных
делений
    plotgrid->setMinPen(QPen(Qt::gray,0,Qt::DotLine));  // серый для вспомогательных
                                                        // линии сетки - пунктирные линии
    plotgrid->attach(ui->somePlot); // связываем созданную сетку с графиком

    ui->somePlot->setAxisTitle(
        QwtPlot::xBottom,QString::fromLocal8Bit("t, мкс")); // имя нижней шкалы
    ui->somePlot->setAxisScale(QwtPlot::xBottom,-0.25,8.25);  // задаем мин и макс
границы для нее
    ui->somePlot->setAxisTitle(
        QwtPlot::yLeft,QString::fromLocal8Bit("U, В"));     // имя левой шкалы
    ui->somePlot->setAxisScale(QwtPlot::yLeft,-1.25,1.25);    // задаем мин и макс
границы для нее

    curveone = new QwtPlotCurve(QString("U1(t)")); // первая кривая - "U1(t)"
    // разрешаем сглаживание при прорисовке
    curveone->setRenderHint(QwtPlotItem::RenderAntialiased);
    curveone->setPen(QPen(Qt::red)); // цвет прорисовки - красный

    // создаем маркеры для точек первой кривой
    QwtSymbol *symbol1 = new QwtSymbol();
    symbol1->setStyle(QwtSymbol::Ellipse);
    symbol1->setPen(QColor(Qt::black));
    symbol1->setSize(4);
    curveone->setSymbol(symbol1);

    curvetwo = new QwtPlotCurve(QString("U2(t)")); // первая кривая - "U2(t)"
    curvetwo->setPen(QPen(Qt::darkGreen)); // цвет прорисовки - темно-зеленый

    const int N1 = 128;     // кол-во для первой кривой
    const int N2 = 262144;  // кол-во для второй кривой

    // выделяем блок памяти под размещение данных X, Y ((общий для всех массивов))
    double *X1 = (double *)malloc((2*N1+2*N2)*sizeof(double));
    double *Y1 = X1 + N1;
    double *X2 = Y1 + N1;
    double *Y2 = X2 + N2;

    // шаг приращения по X1 для первой кривой
    // (зависит от количества точек N1)
    double h = 8./(N1-1);
    for (int k = 0; k < N1; k++)
    {
        X1[k] = k*h;
        // Y1(X1) – синусоида с амплитудой 1, частотой 0.5 и начальной фазой -5*PI/12
        Y1[k] = cos(M_PI*X1[k]-5*M_PI/12);
    }

    // шаг приращения по X2 для второй кривой
    // (зависит от количества точек N2)
    h = 8./(N2-1);
    for (int k = 0; k < N2; k++)
    {
        X2[k] = k*h;
        // Y2(X2) – синусоида с амплитудой 0.7, частотой 4 и начальной фазой PI/9
        Y2[k] = 0.7 * cos(8*M_PI*X2[k]+M_PI/9);
    }
```

```
    curveone->setSamples(X1,Y1,N1);
    curvetwo->setSamples(X2,Y2,N2);

    free((void *)X1); // освобождаем память

    curveone->attach(ui->somePlot); // размещаем на графике кривые
    curvetwo->attach(ui->somePlot);

    ui->somePlot->replot(); // перестраиваем график

    // назначаем тип курсора для канвы графика
    ui->somePlot->canvas()->setCursor(Qt::ArrowCursor);

    // создаем менеджер масштабирования и перемещения графика
    zoom = new ScrollZoomer(ui->somePlot->canvas());
    // устанавливаем цвет рамки, задающей новый размер графика - белый
    zoom->setRubberBandPen(QPen(Qt::white));
}

MainWindow::~MainWindow()
{
    // удаляем созданные в конструкторе переменные:
    delete legend;      // легенду
    delete plotgrid;    // координатную сетку
        // две кривые
    delete curveone;
    delete curvetwo;
    delete zoom;        // возможность масштабирование и перемещения графика

    delete ui;
}
```

Выводы

В данном разделе приводится информация о таких этапах жизненного цикла ПО, как тестирование и отладка; о том, какие бывают ошибки и способы их обнаружения; о методах отладки и тестирования. Приводится описание отладчика gdb и процесс отладки с его помощью в среде Qt Creator, а также процесс тестирования с помощью компонента QtTest фреймворка Qt. Помимо этого приведено описание компилятора gcc и графической библиотеки qwt для представления инженерно-технической информации.

ЗАКЛЮЧЕНИЕ

Результатом выпускной квалификационной работы является рабочая версия программного модуля оценки характеристик комплексированной навигационной аппаратуры. ПМ ОХА работает с распространенным протоколом NMEA, поддерживает формат данных с разделителем «;». В процессе разработки данного программного модуля были проведены его отладка и тестирование, обнаруженные ошибки были устранены. Также разработано подробное руководство оператора. Автором был получен опыт программирования в среде Qt Creator на языке программирования C++, а также опыт работы в операционной системе Astra Linux. Все поставленные цели и задачи выполнены. В рамках данной работы решены следующие задачи:

- исследована предметная область;
- обоснована актуальность изучаемой задачи;
- проведен обзор существующих программных решений и их сравнительный анализ;
- проведен выбор инструментальных средств и среды разработки ПМ;
- разработана схема данных ПМ;
- разработана схема алгоритма ПМ;
- разработан пользовательский интерфейс;
- осуществлена программная реализация ПК РД;
- проведена отладка и тестирование ПК РД;
- разработано руководства оператора.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Яценков В.С. Основы спутниковой навигации. Системы GPS NAVSTAR и ГЛОНАСС [Текст]/ Яценков В.С. – Москва: Горячая линия - Телеком, 2005. – 272 с.;
2. ГОСТ Р 51794-2008. Глобальные навигационные спутниковые системы. Системы координат. Методы преобразований координат определяемых точек [Электронный ресурс]. – Взамен ГОСТ Р 51794-2001; введ. 18.12.2008. – 2009. – 15 с. – URL: http://www.gosthelp.ru/gost/gost48175.html
3. Технология глобальной спутниковой навигации: какие бывают системы, параметры и функции [Электронный ресурс] – URL: https://habrahabr.ru/company/promwad/blog/202722/
4. Основы спутниковой навигации [Электронный ресурс] – Приложение ГНСС – URL: http://glushilka.narod.ru/Jam/gps.pdf
5. Гагарина Л.Г., Киселёв Д.В., Федотова Е.Л. Разработка и эксплуатация автоматизированных информационных систем: учебное пособие / Под ред. проф. Л.Г.Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М, 2012. – 384 с.
6. Колдаев В.Д. Основы алгоритмизации и программирования: учебное пособие / Под ред. Л.Г. Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М, 2012. – 416 с.
7. Федотова Е.Л., Портнов Е.М. Прикладные информационные технологии: учебное пособие / Е.Л. Федотова, Е.М. Портнов. – М.: ИД «ФОРУМ»: ИНФРА-М, 2013. – 336 с.
8. NMEA data [Электронный ресурс] – URL: http://www.gpsinformation.org/dale/nmea.htm
9. ГЛОНАСС. Принципы построения и функционирования / Под. ред. Перова А.И., Харисова В.Н. –4-е изд., перераб. и доп. – М.: Радиотехника, 2010. – 800 с.
10. Инерциальная навигация – [Электронный ресурс] – URL: http://www.krugosvet.ru/
11. Недостатки навигационной системы GPS – [Электронный ресурс] – URL: http://mediatek-club.ru/publ/mobilnaja_spravochnaja/nedostatki_navigacionnoj_sistemy_gps/3-1-0-61
12. Писарев С.Б. Прикладное координатно-временное обеспечение: основные тенденции современного развития // Шебшаевич Б.В. – Всероссийская конференция «Фундаментальное и прикладное координатно-временное обеспечение (КВО-2005)». Тезисы докладов. СПб.: ИПА РАН, 2005. С. 27-30.
13. Visual C# – [Электронный ресурс] – URL: http://msdn.microsoft.com/ru-ru/vcsharp/default.aspx
14. Си (язык программирования) – [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/Си_(язык_программирования)
15. C++ – [Электронный ресурс] – URL: https://ru.wikipedia.org/wiki/C++

16. Java – [Электронный ресурс] – URL: http://www.oracle.com/technetwork/java/index.html
17. Бобронников В.Т., Кадочникова А.Р. Алгоритм комплексирования бесплатформенной инерциальной навигационной системы и магнитометрической системы для решения задачи навигации летательных аппаратов [Текст] // Электронный журнал «Труды МАИ». 2013. Выпуск № 71.
18. Бадд Т. Объектно-ориентированное программирование – [Электронный ресурс] – URL: http://kit.znu.edu.ua/iLec/9sem/OOP/lit/Badd_T_-_obektno_orienntirovannoe_programmirovanie.pdf
19. Методические указания по подготовке выпускной квалификационной работы по направлению подготовки бакалавров 231000.62 «Программная инженерия» / Под редакцией Л.Г.Гагариной. – М., МИЭТ, 2014. – 38 с.
20. Фомичев А. В., Тань Л. Разработка алгоритма быстрой компенсации погрешностей комплексированной инерциально-спутникой системы навигации малогабаритных беспилотных летательных аппаратов в условиях сложной среды // Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2015. № 10. С. 252–270.
21. Степанов О. А. Интегрированные инерциально-спутниковые системы навигации // Гироскопия и навигация. 2002. № 1 (36). С. 23-45.
22. Тестирование приемников ГНСС в потребительских устройствах – [Электронный ресурс] – http://www.treatface.ru/solutions/resheniya-dlya-testirovaniya-kommunikatsionnykh-i-navigatsionnykh-ustroystv-i-sistem/testirovanie-priemnikov-gnss-v-potrebitelskikh-ustroystvakh/
23. Богданов В.С., Кедров В.Д., Тазьба А.М. Особенности построения интегрированных инерциально-спутниковых навигационных систем // Информационно-управляющие системы. 2005. № 2. С.51-54.
24. В.В. Мелешко, О.И. Нестеренко. Бесплатформенные инерциальные навигационные системы: учебное пособие. – Кировоград: ПОЛИМЕД-Сервис, 2011. – 172 с.
25. Имитаторы GPS/GLONASS-сигналов серии GSG – [Электронный ресурс] – URL: http://www.glonass-portal.ru/articles/oborudovanie/yesterday
26. Перов А.И., Шатилов А.Ю. Оценивание углов ориентации объекта с использованием спутниковых радионавигационных систем // Радиотехнические тетради. 2008. № 37. – С. 53-56.
27. Доросинский Л.Г., Богданов Л.А. ОСНОВЫ И ПРИНЦИПЫ ПОСТРОЕНИЯ ИНЕРЦИАЛЬНЫХ НАВИГАЦИОННЫХ СИСТЕМ // Современные проблемы науки и образования. – 2014. – № 5; URL: http://www.science-education.ru/ru/article/view?id=14356;

28. Л.М. Селиванова, Е.В. Шевцова. Инерциальные навигационные системы: учебное пособие. Ч. 1: Одноканальные инерциальные навигационные системы. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2012. – 48 с.
29. LearnPython – [Электронный ресурс] – URL: http://www.learnpython.org
30. Spirent: Network, devices & services testing – [Электронный ресурс] – URL: http://www.spirent.com
31. JTC1/SC22/WG21 - The C++ Standards Committee – ISOCPP – [Электронный ресурс] – URL: http://www.open-std.org
32. Qt Framework – [Электронный ресурс] – URL: https://www.qt.io
33. NetBeans IDE – [Электронный ресурс] – URL: https://netbeans.org
34. A cross-platform IDE Codeblocks – [Электронный ресурс] – URL: http://www.codeblocks.org
35. Что такое архитектура программного обеспечения – [Электронный ресурс] – URL: https://www.ibm.com/developerworks/ru/library/eeles/
36. Общие соображения по архитектуре ПО – [Электронный ресурс] – URL: https://habrahabr.ru/post/90880/
37. Гагарина Л.Г., Федоров А.Р., Федоров П.А. Введение в архитектуру проектирования программного обеспечения: учебное пособие. – М.: ИД «ФОРУМ» ИНФРА-М, 2016. – 320 с.
38. Гращенко Л.А., Фисун А.П. Теоретические и практические основы человеко-компьютерного взаимодействия: базовые понятия человеко-компьютерных систем в информатике и информационной безопасности. – Орловский государственный университет. Орел. 2004. – 169 с.
39. Акчурин Э. А. Человеко-машинное взаимодействие: учебное пособие. – М.: СОЛОН-Пресс, 2008. – 96 с.
40. Новые возможности в человеко-машинном интерфейсе – [Электронный ресурс] – URL: http://www.intuit.ru/studies/courses/10619/1103/lecture/18218
41. Справочный интернет-портал Crossplatform.ru – [Электронный ресурс] – URL: http://doc.crossplatform.ru/
42. Справочный интернет-портал для разработчиков SitForum – [Электронный ресурс] – URL: http://citforum.ru/
43. Информационный портал интернет-института «ИНТУИТ» – [Электронный ресурс] – URL: http://www.intuit.ru/
44. Макс Шлее «Qt 5.3. Профессиональное программирование на C++». – Санкт-Петербург: БХВ-Петербург, 2015. – 928 с.
45. Coded UI Test: Как написать первый тест [Электронный ресурс] – URL: http://bugscatcher.net/archives/1297

46. Справочный портал по методам тестирования «Тестирование ПО» [Электронный ресурс] – URL: http://testingworld.ru/
47. Виды Тестирования Программного Обеспечения – [Электронный ресурс] – URL: http://www.protesting.ru/testing/testtypes.html
48. Основные положения тестирования – [Электронный ресурс] – URL: https://habrahabr.ru/post/110307
49. Классификация видов тестирования – [Электронный ресурс] – URL: https://habrahabr.ru/company/npo-comp/blog/223833/
50. ГОСТ 19.505 – 79 Руководство оператора. Требования к содержанию и оформлению – [Электронный ресурс] – URL: http://www.gosthelp.ru/text/GOST1950579Edinayasistema.html

ПРИЛОЖЕНИЕ 1

ПРИЛОЖЕНИЕ 2