РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе содержит 144 страницы, 106 рисунков, 12 таблиц, 15 источников, 5 приложений.

НАВИГАЦИЯ, БИНС, СНС, КОМПЛЕКСИРОВАНИЕ, ФИЛЬТР КАЛМАНА

Тема выпускной квалификационной работы: “Моделирование комплексной системы навигации летательного аппарата”.

Цель работы – разработка модели системы навигации самолета, состоящей из комплекса нескольких навигационных систем и дискретного фильтра Калмана для оценивания состояния летательного аппарата по поступающим измерениям.

В ходе выполнения дипломной работы была создана модель комплексной навигационной системы летательного аппарата, исследовано влияние модели шумов инерциальных датчиков, частоты выдачи и отсутствия сигналов спутниковой навигационной системы на получаемую оценку. Разработано программное обеспечение, имитирующее работу системы навигации в реальном времени.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На различных подвижных объектах, для определения их положения, скоростей и ориентации в пространстве, уже давно устанавливаются инерциальные навигационные системы (ИНС). Обладая помехозащищенностью, высокой скоростью обновления информации и автономностью, они имеют существенный недостаток – накопление погрешностей с течением времени.

При этом измерения, получаемые с помощью спутниковой навигационной системы (СНС), не имеют тенденции к росту погрешностей, но в большей степени подвержены различным помехам.

Тема выпускной квалификационной работы является актуальной, так как объединение ИНС и СНС позволяет создать высокоточный навигационный комплекс, сочетающий в себе преимущества обеих систем.

В работе рассматривается схема комплексирования, позволяющая объединить показания бесплатформенной инерциальной (БИНС) и спутниковой навигационных систем самолета для оценивания его вектора состояния при помощи дискретного фильтра Калмана.

Фильтр Калмана – оптимальный рекурсивный алгоритм оценивания неизвестного вектора состояния динамической системы по известным зашумленным наблюдениям.

Задачи дипломной работы:

- Создание математической модели комплексированной навигационной системы летательного аппарата;
- Разработка программного обеспечения, имитирующего работу комплексной системы навигации в реальном времени;
- Исследование влияния частоты выдачи сигналов спутниковой навигационной системы на получаемую оценку;
- Исследование влияния отсутствия сигналов спутниковой навигационной системы на получаемую оценку;

– Исследование влияния модели шумов инерциальных датчиков на получаемую оценку;
– Расчет себестоимости разработки ВКР;
– Анализ опасных и вредных производственных факторов на рабочем месте и разработка мер по снижению их воздействия на работающего.

Выпускная квалификационная работа включает в себя три раздела.

Исследовательский раздел содержит обзор и тактико-технические характеристики летательного аппарата; рассмотрены схема комплексирования навигационных измерителей по способу компенсации; алгоритм дискретного фильтра Калмана.

Приведены модель движения летательного аппарата, приближенная модель погрешностей северного и вертикального каналов БИНС и измерений СНС; исследован процесс оценивания для трех различных частот выдачи сигналов СНС.

Представлена архитектура программного обеспечения, имитирующего работу комплексной навигационной системы в реальном времени; исследовано влияние отсутствия сигналов СНС на процесс оценивания.

Исследовано влияние аддитивных розового и винеровского шумов в составе сигналов инерциальных датчиков на получаемую оценку.

В технико-экономическом разделе приведен расчет затрат на выполнение выпускной квалификационной работы.

В разделе безопасности жизнедеятельности и экологии содержатся анализ опасных и вредных производственных факторов на рабочем месте и меры по снижению их воздействия на работающего; приведен расчет освещенности рабочего места; определены мероприятия по охране труда.

1 Исследовательский раздел

1.1 Конструкция и тактико-технические характеристики самолета

Piaggio P.180 Avanti – итальянский административный самолёт, разработанный и производимый итальянской авиастроительной компанией “Piaggio Aero”.

Самолёт выполнен по схеме моноплана с передним горизонтальным стабилизатором, Т-образным хвостовым оперением и силовой установкой с толкающими воздушными винтами.

Несмотря на некоторую внешнюю схожесть с аэродинамической схемой “утка”, самолёт не относится к этой конфигурации, так как переднее горизонтальное оперение не имеет рулевых поверхностей (рули высоты и направления расположены на хвостовом оперении, элероны – на крыльях, как и в самолётах нормальной аэродинамической схемы). На переднем горизонтальном оперении, однако, расположены закрылки, работающие синхронно с основными закрылками на крыльях.

Выпускается в различных вариантах: административный, лёгкий транспортный, аэрофотосъёмочный, санитарный и т. д.

Самолёт оснащён современными разработками в цифровой системе навигации и управления полетом.

Основные тактико-технические характеристики (ТТХ) летательного аппарата приведены в таблице 1, общий вид модели представлен на рисунке 1.

Таблица 1 – ТТХ Piaggio P.180 Avanti

Длина, м	14,41
Размах крыла, м	14,03
Высота, м	3,97
Площадь крыла, $м^2$	16,1
Масса пустого, кг	3400

Продолжение таблицы 1

Максимальная скорость, км/ч	740
Крейсерская скорость, км/ч	574
Потолок, м	12500
Скороподъёмность (на ур. моря), м/мин	899
Дальность полёта, км	2592
Максимальная взлётная масса, кг	5239
Коммерческая загрузка, кг	907
Полезная загрузка, кг	1860
Нагрузка на крыло, кг/м^2	327
Тяговооруженность, кВт/кг	0,24
Разбег, м	869
Пробег, м	872
Экипаж, количество пилотов	1-2
Пассажировместимость, человек	9
Ширина салона, м	1,85

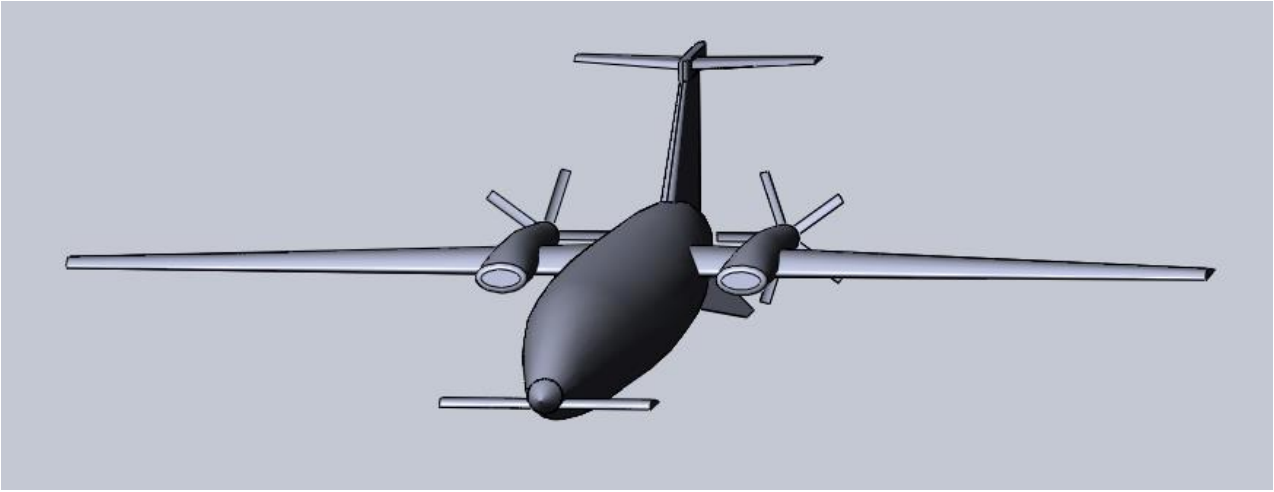

Рисунок 1 – Общий вид модели Piaggio P.180 Avanti

Рассмотрим схему комплексирования, позволяющую объединить показания инерциальной и спутниковой навигационных систем самолета для оценивания его вектора состояния.

1.2 Комплексирование измерителей по способу компенсации

Рассмотрим объединение двух навигационных измерителей, определяющих один и тот же параметр x(t).

Идея заключается в формировании разностных измерений, при которых исключается из рассмотрения сам навигационный параметр.

Структурная схема комплексирования по способу компенсации приведена на рисунке 2 [1].

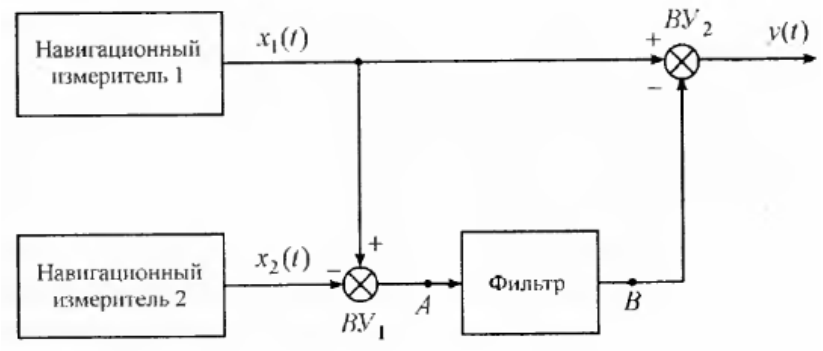

Рисунок 2 – Структурная схема комплексирования по способу компенсации

Сигналы на выходе первого и второго навигационных измерителей имеют вид:

$$\begin{aligned} x_1(t) &= x(t) + n_1(t); \\ x_2(t) &= x(t) + n_2(t), \end{aligned} \tag{1.2.1}$$

где x(t) – измеряемый навигационный параметр (полезный сигнал);

$n_1(t)$ и $n_2(t)$ – погрешности навигационных измерителей, которые предполагаются стационарными случайными процессами.

В соответствии с принципом комплексирования по способу компенсации сигналы $x_1(t)$ и $x_2(t)$ подаются на вычитающее устройство $ВУ_1$, на выходе которого образуется разностный сигнал, не содержащий измеряемого навигационного параметра (точка А на рис. 2):

$$x_A(t) = n_1(t) - n_2(t) \tag{1.2.2}$$

Разностный сигнал $x_A(t)$ поступает в линейный фильтр, передаточная функция F(s) которого должна быть такой, чтобы он в соответствии с выбранным критерием в наибольшей степени подавлял помеху $n_2(t)$ и в минимальной степени искажал помеху $n_1(t)$. Сигнал на выходе фильтра имеет вид (точка В на рис. 2):

$$x_B(s) = F(s)\, x_A(s) = F(s)[n_1(s) - n_2(s)] \tag{1.2.3}$$

На вычитающем устройстве $ВУ_2$ образуется разность:

$$\begin{aligned} y(s) = x_1(s) - x_B(s) = \\ = x(s) + n_1(s) - [n_1(s) - n_2(s)]\, F(s) = \\ = x(s) + [1 - F(s)]\, n_1(s) + F(s)\, n_2(s) \end{aligned} \tag{1.2.4}$$

Выходной сигнал комплексной навигационной системы (КНС) может быть представлен в виде:

$$y(s) = x(s) + \varepsilon(s), \tag{1.2.5}$$

где ε(s) – результирующая погрешность КНС:

$$\varepsilon(s) = [1 - F(s)]\, n_1(s) + F(s)\, n_2(s) \tag{1.2.6}$$

Можно заметить, что ε(s) не зависит от x(t). Системы, в которых погрешность не зависит от полезного сообщения x(t), называют инвариантными по отношению к x(t) [1].

В рассматриваемой схеме комплексирования в качестве фильтра используется дискретный фильтр Калмана.

1.3 Дискретный фильтр Калмана

По своему назначению любой фильтр должен подавлять помехи и с наименьшими искажениями пропускать полезный сигнал. Другими словами, фильтр дает оценку полезного сигнала.

Алгоритм фильтра Калмана состоит из двух повторяющихся фаз: экстраполяция и корректировка.

На первой фазе рассчитывается экстраполяция состояния в следующий момент времени. На второй фазе поступившая информация от измерителя корректирует спрогнозированное значение.

Фильтр предполагает, что модели состояния и наблюдения системы являются линейными, а шумы системы и наблюдений – гауссовскими некоррелированными белыми шумами с нулевым математическим ожиданием.

Рассмотрим модель дискретного фильтра Калмана.

Состояние оцениваемой системы описывается разностным уравнением вида:

$$x_{k+1} = \Phi_k x_k + \Psi_k u_k + \Gamma_k w_k, \qquad (1.3.1)$$

где x – n-мерный вектор состояния;

$\Phi_k \approx E + A_k T$ – переходная матрица состояния [n x n];

E – единичная матрица [n x n];

A_k – матрица состояния [n x n];

T – период дискретизации;

u – вектор управления размера r;

$\Psi_k \approx B_k T$ – переходная матрица управления [n x r];

B_k – матрица управления [n x r];

w – p-мерный вектор случайных воздействий (шум процесса);

$\Gamma_k \approx G_k T$ – переходная матрица возмущения [n x p];

G_k – матрица возмущения [n x p].

Модель наблюдения:

$$z_k = H_k x_k + v_k, \tag{1.3.2}$$

где z – m-мерный вектор измеренных параметров;

H – матрица наблюдения размера [m x n];

v – m-мерный вектор погрешностей измерения.

Задачей фильтра является нахождение оптимальной оценки вектора состояния на k-м шаге $\hat{x}_k$ и его ковариационной матрицы P_k.

Экстраполированная оценка вектора состояния $\hat{x}_{k|k-1}$и ее ковариация $P_{k|k-1}$ находятся на основе предыдущей оценки $\hat{x}_{k-1}$ и ее ковариации P_{k-1} (этап экстраполяции):

$$\begin{gathered} \hat{x}_{k|k-1} = \Phi_k \hat{x}_{k-1} + \Psi_k u_{k-1}; \\ P_{k|k-1} = \Phi_k P_{k-1} {\Phi_k}^T + \Gamma_k Q_{k-1} {\Gamma_k}^T \end{gathered} \tag{1.3.3}$$

С поступившим на текущем шаге измерением находится невязка e_k и ее ковариационная матрица S_k:

$$\begin{gathered} e_k = z_k - \hat{z}_{k|k-1} = z_k - H_k \hat{x}_{k|k-1}; \\ S_k = H_k P_{k|k-1} {H_k}^T + R_k \end{gathered} \tag{1.3.4}$$

Вычисляется K_k – матричный коэффициент усиления фильтра Калмана:

$$K_k = P_{k|k-1} {H_k}^T {S_k}^{-1} \tag{1.3.5}$$

Оптимальная оценка вектора состояния на k-м шаге $\hat{x}_k$ и ее ковариационная матрица P_k вычисляются следующим образом (этап корректировки в соответствии с поступившим измерением):

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k e_k;$$
$$P_k = P_{k|k-1} - K_k H_k P_{k|k-1} \quad (1.3.6)$$

Экстраполированная оценка вектора состояния $\hat{x}_{k|k-1}$ и вектора измерения $\hat{z}_{k|k-1}$ содержат всю информацию об объекте оценивания, имеющуюся на момент времени k-1, пересчитанную в соответствии с принятыми моделями движения и измерения к моменту времени k.

Вектор невязки e_k и ее ковариационная матрица S_k характеризуют новую информацию, полученную в результате измерений.

Элементы матричного коэффициента усиления K_k учитывают вклад, вносимый невязкой в результирующую оценку вектора состояния. Коэффициент усиления прямо пропорционален ковариации экстраполированной оценки и обратно пропорционален ковариации невязки.

Таким образом, чем менее точна экстраполированная оценка, и чем более точно измерение, тем больше коэффициент усиления K_k , следовательно, тем ближе оценка вектора состояния $\hat{x}_k$ к полученному измерению, а не к экстраполированной оценке, и наоборот.

1.4 Фильтрация ошибок БИНС

Рассмотрим оценивание параметров движения ЛА по схеме компенсации, с использованием инерциальной и спутниковой навигационных систем, путем нахождения оценок ошибок ИНС и их вычитания из формируемых ИНС параметров, содержащих искомую ошибку.

1.4.1 Модель системы

Рассмотрим плоское движение самолета вдоль меридиана на постоянной высоте (Рисунок 3 [1]). Таким образом, в ИНС будут задействованы 2 акселерометра и 1 гироскоп.

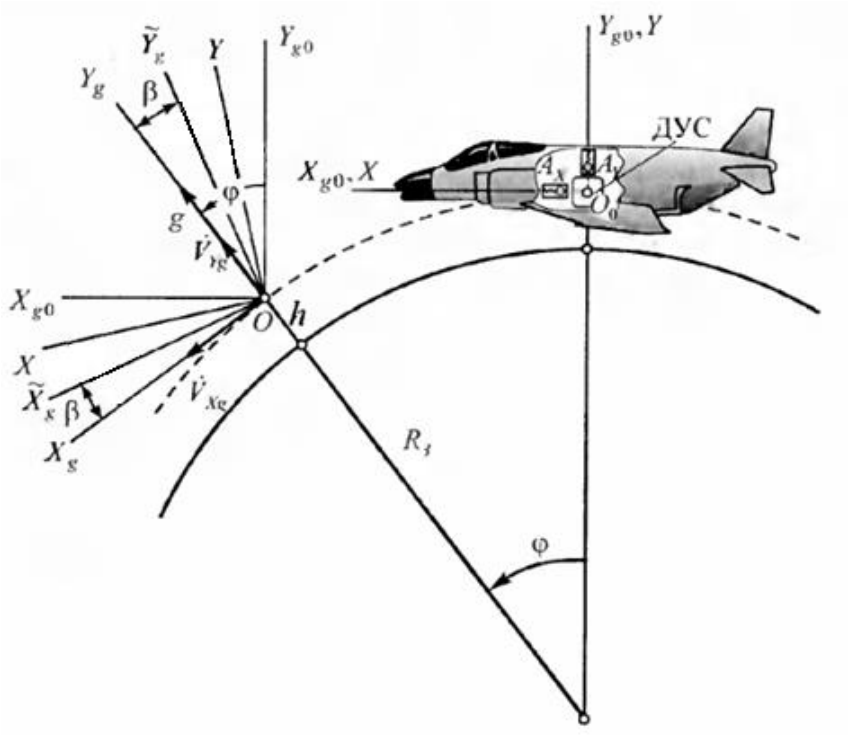

Рисунок 3 – Модель движения ЛА

Примем, что динамика ЛА описывается системой уравнений:

$$\begin{gathered} V_{Xg} = 205; \\ S = V_{Xg} t; \\ V_{Yg} = 0; \\ h = 7000; \\ \varphi = \frac{S}{R}; \\ \vartheta = 3, \end{gathered} \tag{1.4.1}$$

где $V_{Xg}, \text{м}/c -$ скорость ЛА в северном направлении;

$S, \text{м} -$ пройденное расстояние;

$V_{Yg}, \text{м}/c -$ скорость ЛА в вертикальном направлении;

$h, \text{м} -$ высота полета;

$\varphi, \text{рад} -$ широта;

$\vartheta, ° -$ угол тангажа;

$R = R_з + h, \text{м} -$ радиус-вектор от центра Земли до ЛА.

Для измерения параметров движения используем БИНС и СНС, объединенные по способу компенсации (Рисунок 4). В качестве измерений будем рассматривать разность ошибок БИНС и СНС в определении позиционных и скоростных параметров: пройденного расстояния S, высоты h, горизонтальной и вертикальной скоростей V_{Xg} и V_{Yg}.

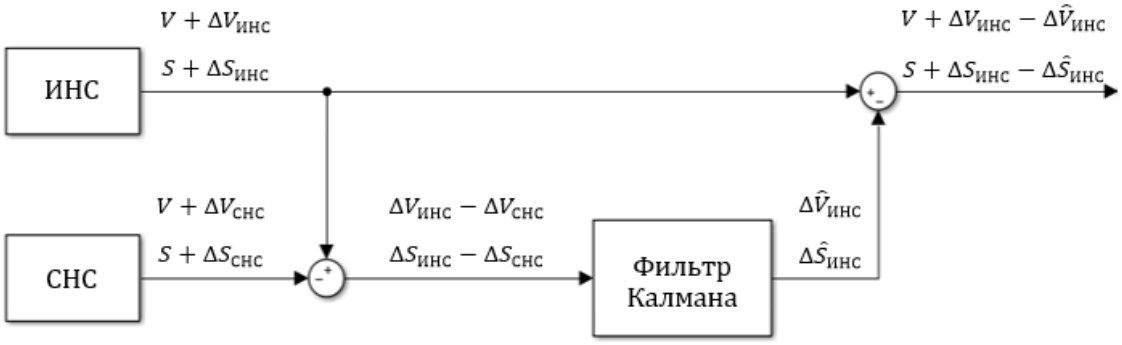

Рисунок 4 – Комплексирование информации, поступающей от двух навигационных систем по способу компенсации

Рассмотрим приближенную модель погрешностей северного и вертикального каналов БИНС [1]:

$$\begin{gathered}\dot{\beta} = -\frac{\Delta V_{Xg}}{R} - \varepsilon_z;\\ \Delta\dot{V}_{Xg} = g\beta + \delta a_{Xg};\\ \Delta\dot{S} = \Delta V_{Xg};\\ \Delta\dot{V}_{Yg} = \delta a_{Yg};\\ \Delta\dot{h} = \Delta V_{Yg},\end{gathered} \tag{1.4.2}$$

где β – ошибка построения вертикали, рад;

$\Delta V_{Xg}, \Delta V_{Yg}$ – ошибки выработки горизонтальной и вертикальной скорости, м/с;

$\Delta S, \Delta h$ – ошибки выработки пройденного расстояния и высоты, м;

ε_z – ошибка измерения проекции абсолютной угловой скорости ЛА на ось Z связанной системы координат, рад/с;

$\delta a_{Xg}, \delta a_{Yg}$– проекции ошибок измерения кажущегося ускорения ЛА на ребра X и Y географической системы координат (система координат, ось X которой лежит в плоскости местного горизонта и направлена на север, ось Y – по нормали к горизонту), м/с^2.

При этом параметры ЛА, выдаваемые БИНС (обозначим их символом "∼"), содержат ошибки:

$$\begin{gathered}\tilde{V}_{Xg} = V_{Xg} + \Delta V_{Xg};\\ \tilde{S} = S + \Delta S;\\ \tilde{V}_{Yg} = V_{Yg} + \Delta V_{Yg};\\ \tilde{h} = h + \Delta h;\\ \tilde{\varphi} = \varphi + \frac{\Delta S}{R};\\ \tilde{\vartheta} = \vartheta - \beta\end{gathered} \tag{1.4.3}$$

Измерения сформируем как разность сигналов БИНС и СНС:

$$\begin{gathered}z_1 = \tilde{V}_{Xg} - V_{Xg_{\text{СНС}}} = V_{Xg} + \Delta V_{Xg} - V_{Xg} - \Delta V_{Xg_{\text{СНС}}} =\\ = \Delta V_{Xg} - \Delta V_{Xg_{\text{СНС}}};\end{gathered} \tag{1.4.4}$$

Аналогично:

$$\begin{gathered}z_2 = \Delta S - \Delta S_{\text{СНС}};\\ z_3 = \Delta V_{Yg} - \Delta V_{Yg_{\text{СНС}}};\\ z_4 = \Delta h - \Delta h_{\text{СНС}}\end{gathered} \tag{1.4.5}$$

Погрешности СНС будем рассматривать в качестве шума измерения.

Принимаем модели погрешностей инерциальных чувствительных элементов (ε_z, δa_{Xg}, δa_{Yg}) и СНС ($\Delta V_{Xg_{\text{СНС}}}, \Delta S_{\text{СНС}}, \Delta V_{Yg_{\text{СНС}}}, \Delta h_{\text{СНС}}$) гауссовским белым шумом с нулевым математическим ожиданием.

Представим рассматриваемую систему в векторно-матричной форме:

$$\frac{d}{dt}\begin{Vmatrix}\beta\\ \Delta V_{Xg}\\ \Delta S\\ \Delta V_{Yg}\\ \Delta h\end{Vmatrix} = \begin{Vmatrix}0 & -1/R & 0 & 0 & 0\\ g & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\end{Vmatrix}\begin{Vmatrix}\beta\\ \Delta V_{Xg}\\ \Delta S\\ \Delta V_{Yg}\\ \Delta h\end{Vmatrix} + \\ + \begin{Vmatrix}-1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{Vmatrix}\begin{Vmatrix}\varepsilon_z\\ \delta a_{Xg}\\ \delta a_{Yg}\end{Vmatrix}; \tag{1.4.6}$$

$$z = \begin{Vmatrix}0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{Vmatrix}\begin{Vmatrix}\beta\\ \Delta V_{Xg}\\ \Delta S\\ \Delta V_{Yg}\\ \Delta h\end{Vmatrix} - \begin{Vmatrix}\Delta V_{Xg_{\text{СНС}}}\\ \Delta S_{\text{СНС}}\\ \Delta V_{Yg_{\text{СНС}}}\\ \Delta h_{\text{СНС}}\end{Vmatrix} \tag{1.4.7}$$

Получили следующие матрицы:

$$A = \begin{Vmatrix}0 & -1/R & 0 & 0 & 0\\ g & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\end{Vmatrix}; \tag{1.4.8}$$

$$G = \begin{Vmatrix}-1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{Vmatrix}; \tag{1.4.9}$$

$$H = \begin{Vmatrix}0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{Vmatrix} \tag{1.4.10}$$

Примем шумы системы и шумы измерений некорреллированными случайными процессами. Таким образом, матрицы интенсивностей возмущений и ошибок измерения – диагональные:

$$Q = \begin{Vmatrix} \sigma_{\varepsilon}^2 & 0 & 0 \\ 0 & \sigma_{\delta a_{Xg}}^2 & 0 \\ 0 & 0 & \sigma_{\delta a_{Yg}}^2 \end{Vmatrix}; \tag{1.4.11}$$

$$R = \begin{Vmatrix} \sigma_{V_{Xg}\text{СНС}}^2 & 0 & 0 & 0 \\ 0 & \sigma_{S\text{СНС}}^2 & 0 & 0 \\ 0 & 0 & \sigma_{V_{Yg}\text{СНС}}^2 & 0 \\ 0 & 0 & 0 & \sigma_{h\text{СНС}}^2 \end{Vmatrix}, \tag{1.4.12}$$

где $\sigma_{\varepsilon}^2, \sigma_{\delta a_{Xg}}^2 = \sigma_{\delta a_{Yg}}^2 = \sigma_{\delta a}^2 -$ дисперсии погрешностей гироскопа и акселерометров;

$\sigma_{V_{Xg}\text{СНС}}^2, \sigma_{S\text{СНС}}^2, \sigma_{V_{Yg}\text{СНС}}^2, \sigma_{h\text{СНС}}^2 -$ дисперсии шума измерения СНС.

Принятые СКО шумов представлены в таблице 2.

Таблица 2 – СКО шумов

σ_{ε} , рад/с	$\sigma_{\delta a}$, м/с^2	$\sigma_{V_{Xg}\text{СНС}}$, м/с	$\sigma_{S\text{СНС}}$, м	$\sigma_{V_{Yg}\text{СНС}}$, м/с	$\sigma_{h\text{СНС}}$, м
0.02	0.1	0.2	6	0.3	10

Примем начальную матрицу ковариации P_0:

$$P_0 = \begin{Vmatrix} \sigma_{\beta}^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{\Delta V_{Xg}}^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{\Delta S}^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma_{\Delta V_{Yg}}^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\Delta h}^2 \end{Vmatrix}, \tag{1.4.13}$$

где $\sigma_{\beta} = \sigma_{\varepsilon}$; $\sigma_{\Delta V_{Xg}} = \sigma_{\Delta V_{Yg}} = \sigma_{\delta a}$; $\sigma_{\Delta S} = \sigma_{\Delta h} = 2$ м – принятые начальные СКО ошибок оценивания вектора состояния.

Промоделируем систему с начальными условиями:

– $\beta = 1°$;

– $\Delta V_{Xg} = 0 \frac{м}{с}$;

– $\Delta S = 0$ м;

– $\Delta V_{Yg} = 0 \frac{м}{с}$;

– $\Delta h = 0$ м.

Примем частоту поступления сигналов БИНС – 10 Гц, сигналов СНС – 5 Гц.

Таким образом, шаг поступления сигналов БИНС – 0.1 с, СНС – 0.2 с.

Моделируем систему в течение 40 секунд.

1.4.2 Результаты моделирования

Для моделирования процесса был написан скрипт Matlab (Приложение А).

На рисунках 5–9 представлены зависимости переменных состояния системы – погрешностей БИНС, измерений СНС и оценок переменных состояния во времени.

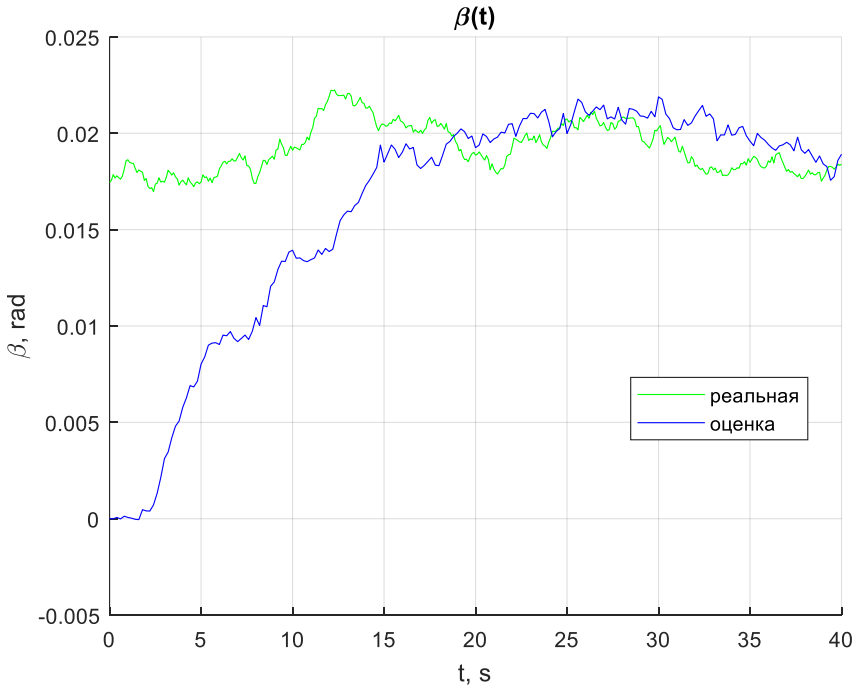

Рисунок 5 – Ошибка построения вертикали

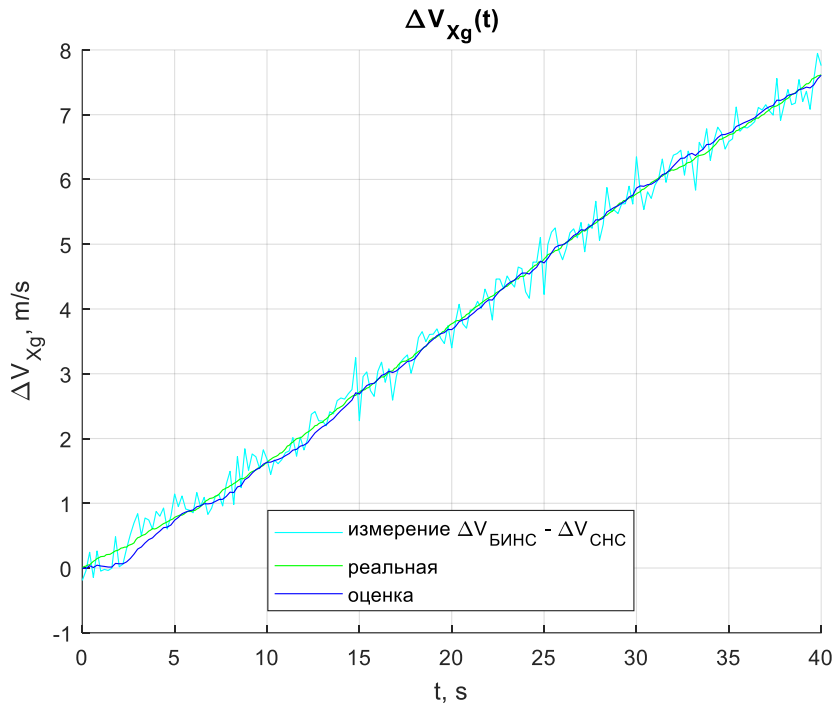

Рисунок 6 – Ошибка выработки горизонтальной скорости

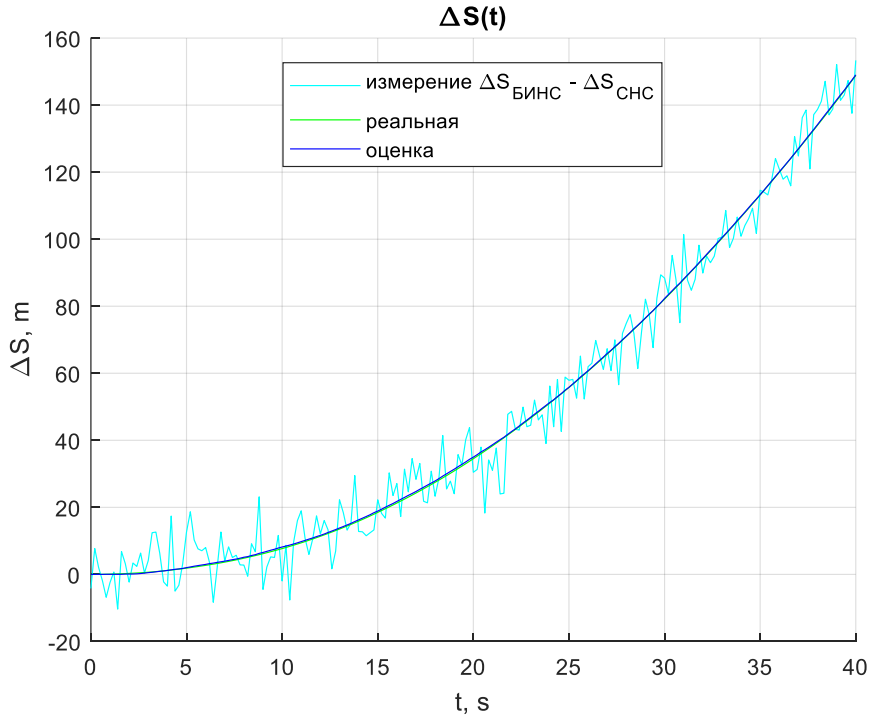

Рисунок 7 – Ошибка выработки пройденного расстояния

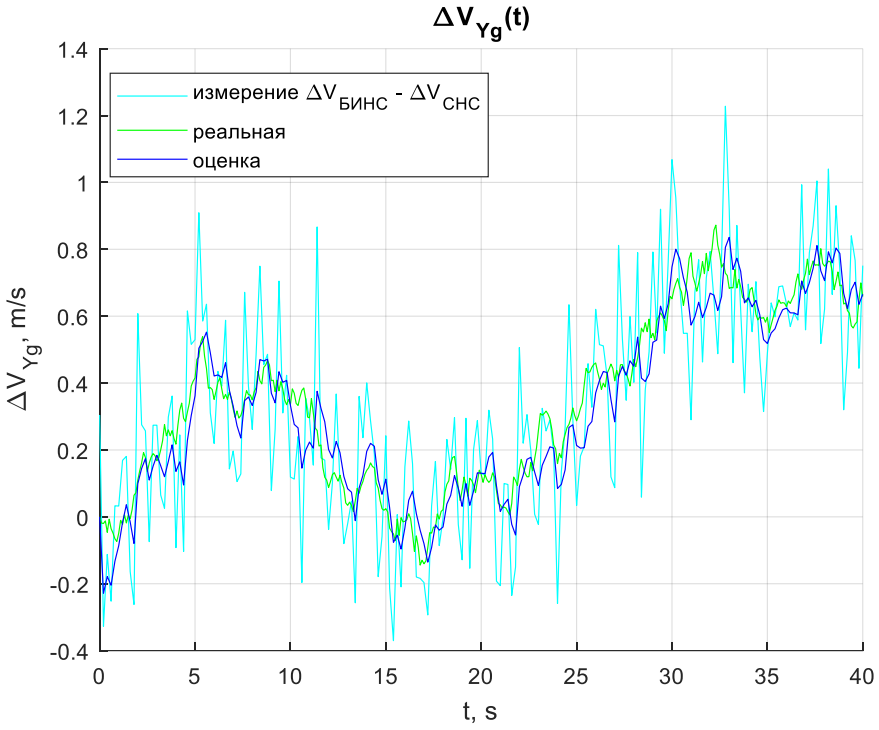

Рисунок 8 – Ошибка выработки вертикальной скорости

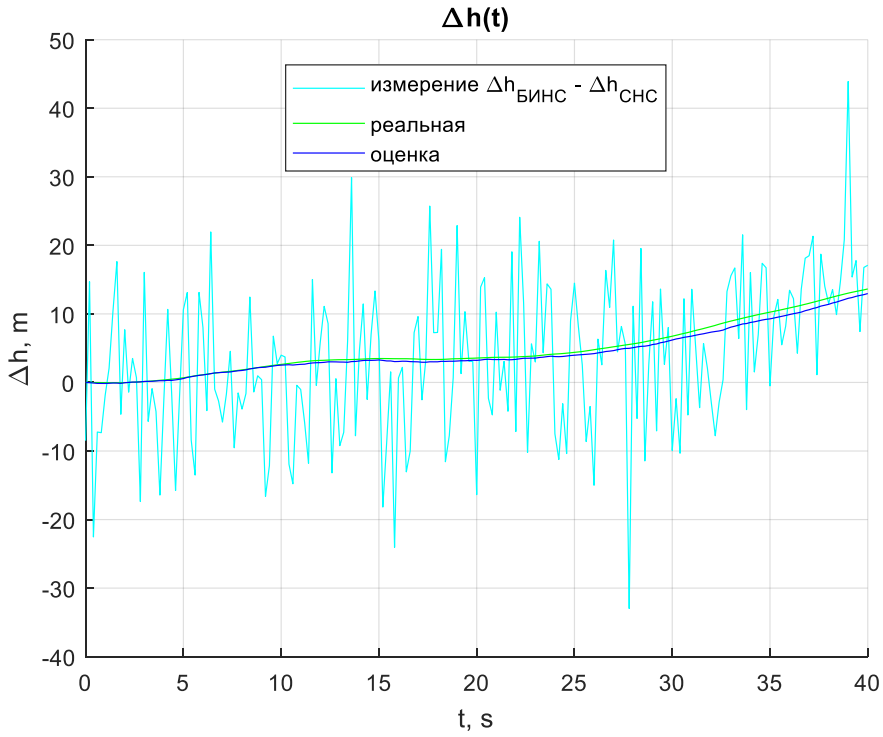

Рисунок 9 – Ошибка выработки высоты

На рисунках 10–15 представлены графики изменения во времени параметров движения ЛА, их измерений СНС и выработанных с помощью БИНС значений, а также графики полученных оценок.

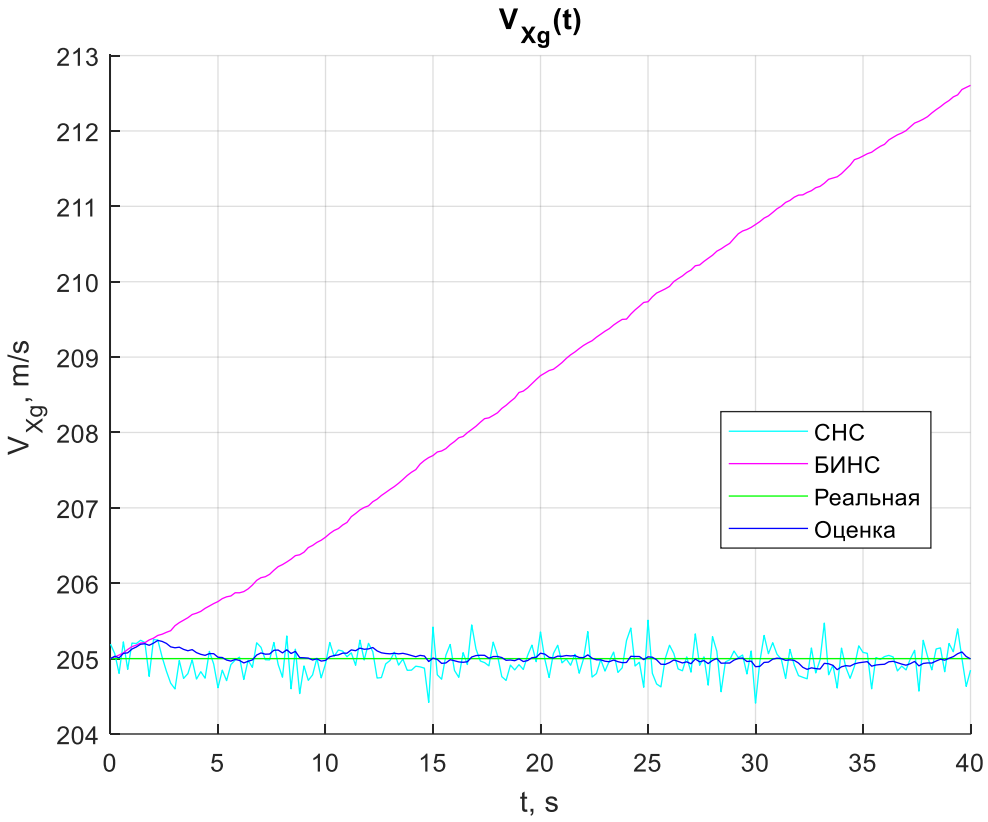

Рисунок 10 – Горизонтальная скорость

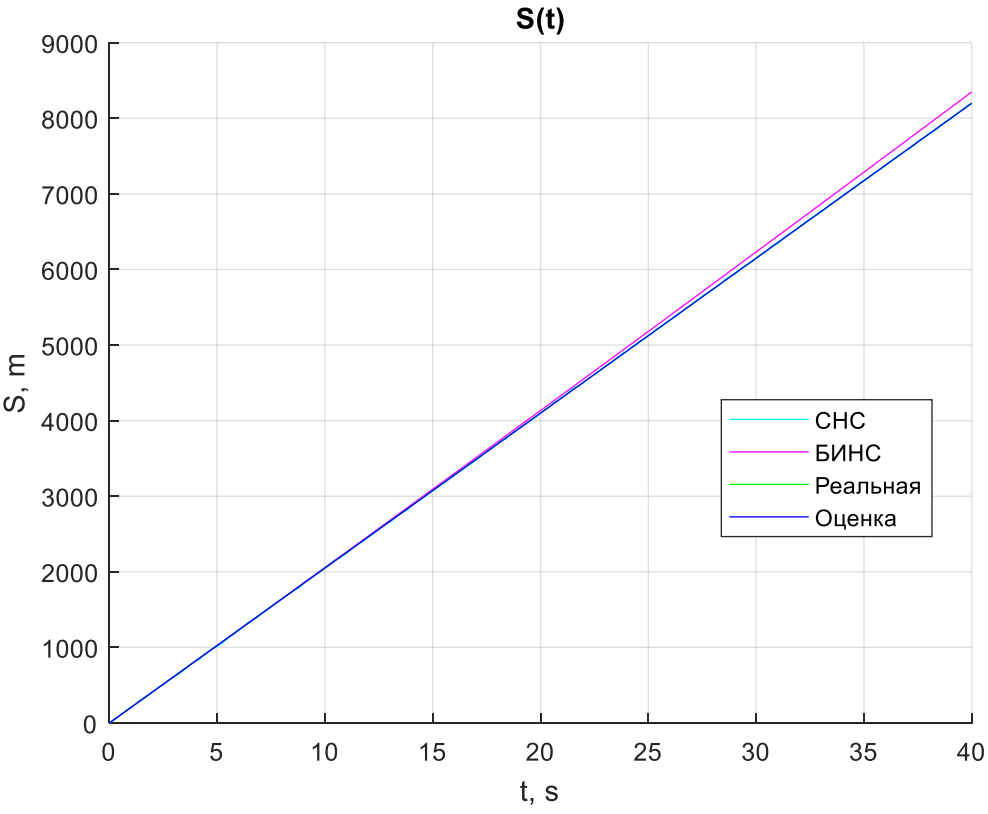

Рисунок 11 – Пройденный путь

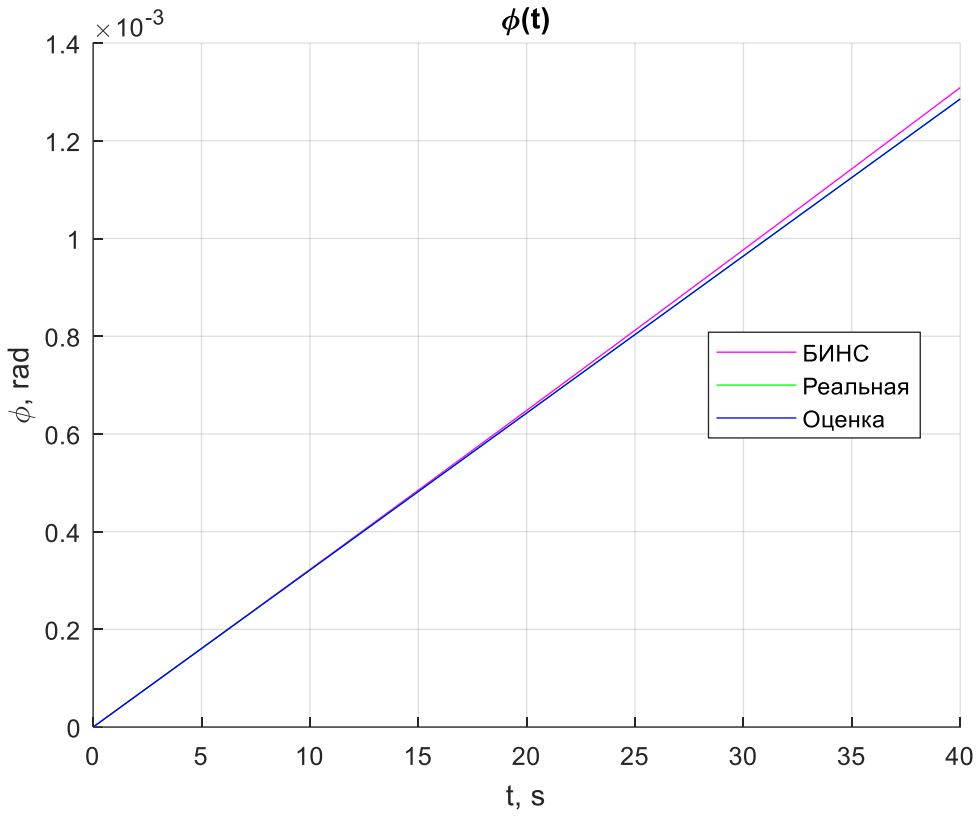

Рисунок 12 – Широта

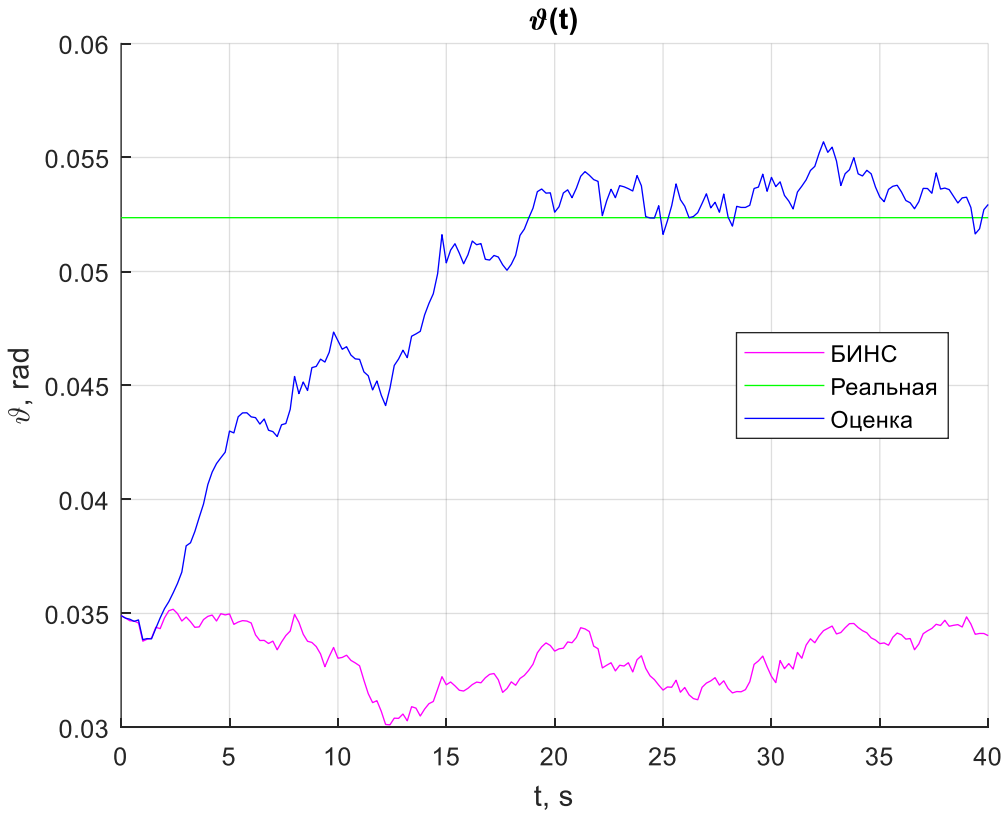

Рисунок 13 – Угол тангажа

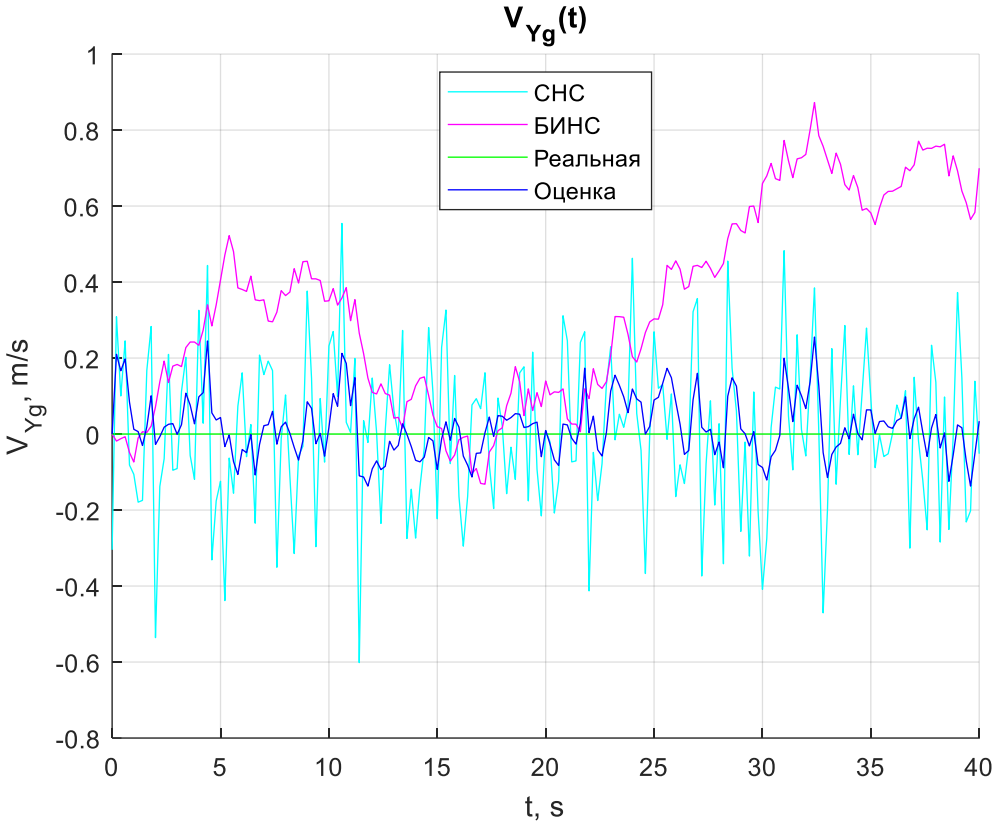

Рисунок 14 – Вертикальная скорость

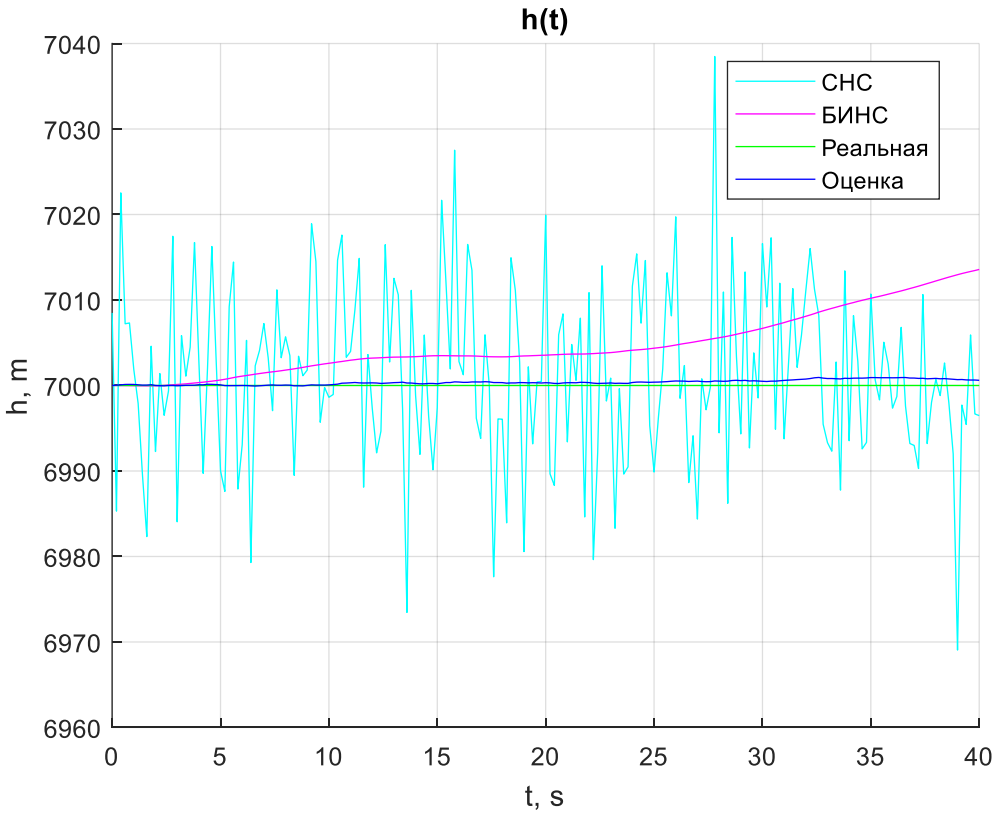

Рисунок 15 – Высота

Видно, что оценки ошибок БИНС, получаемые с помощью фильтра, близки к реальным значениям ошибок.

Погрешности БИНС с течением времени накапливаются, что приводит к уходу вырабатываемых параметров от реальных значений. При этом, рассмотренный способ компенсации позволяет вычесть оценки ошибок БИНС из ее выходных параметров, и, таким образом, получить достаточно точную оценку параметра.

1.4.3 Исследование влияния частоты сигнала СНС на получаемую оценку

Исследуем влияние частоты выдачи сигналов спутниковой системы на получаемую оценку. Для этого зададим частоту сигнала СНС сначала 2 Гц, затем 1 Гц и запустим моделирование.

Полученные результаты моделирования при заданной частоте 2 Гц сигнала СНС представлены в Приложении Б.

Результаты моделирования при частоте 1 Гц сигнала СНС приведены в Приложении В.

Рассчитаем статистические характеристики ошибок оценивания:

- Выборочное среднее:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (1.4.14)$$

- Выборочное среднеквадратическое отклонение:

$$S_0 = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad (1.4.15)$$

Результаты расчета для трех рассмотренных частот выдачи сигнала СНС представлены в таблицах 3–5.

Таблица 3 – Ошибки оценивания при частоте 5 Гц

Ошибка	$\bar{x}$	S_0
$\tilde{\beta}$, рад	0.0032	0.0047
$\Delta\tilde{V}_{Xg}$, м/с	0.0370	0.0629
$\Delta\tilde{S}$, м	-0.1063	0.1270
$\Delta\tilde{V}_{Yg}$, м/с	0.0094	0.0766
$\Delta\tilde{h}$, м	0.0550	0.1112

Таблица 4 – Ошибки оценивания при частоте 2 Гц

Ошибка	$\bar{x}$	S_0
$\tilde{\beta}$, рад	0.0035	0.0063
$\Delta\tilde{V}_{Xg}$, м/с	0.0520	0.0904
$\Delta\tilde{S}$, м	0.2587	0.1772
$\Delta\tilde{V}_{Yg}$, м/с	0.0106	0.1121
$\Delta\tilde{h}$, м	-0.0799	0.2556

Таблица 5 – Ошибки оценивания при частоте 1 Гц

Ошибка	$\bar{x}$	S_0
$\tilde{\beta}$, рад	0.0036	0.0066
$\Delta\tilde{V}_{Xg}$, м/с	0.0562	0.1191
$\Delta\tilde{S}$, м	1.5882	0.9263
$\Delta\tilde{V}_{Yg}$, м/с	-0.0117	0.1808
$\Delta\tilde{h}$, м	0.1624	0.3931

Видно, что с уменьшением частоты, выборочное среднее ошибки оценивания возрастает, также возрастает и ее среднеквадратическое отклонение.

1.5 Имитация работы системы в режиме реального времени

Разработаем программное обеспечение, имитирующее работу комплексной навигационной системы в реальном времени. Добавим также в модель вторую СНС с погрешностями, составляющими коэффициент k = 1.3 от погрешностей первой СНС. Полученная схема комплексирования систем представлена на рисунке 16.

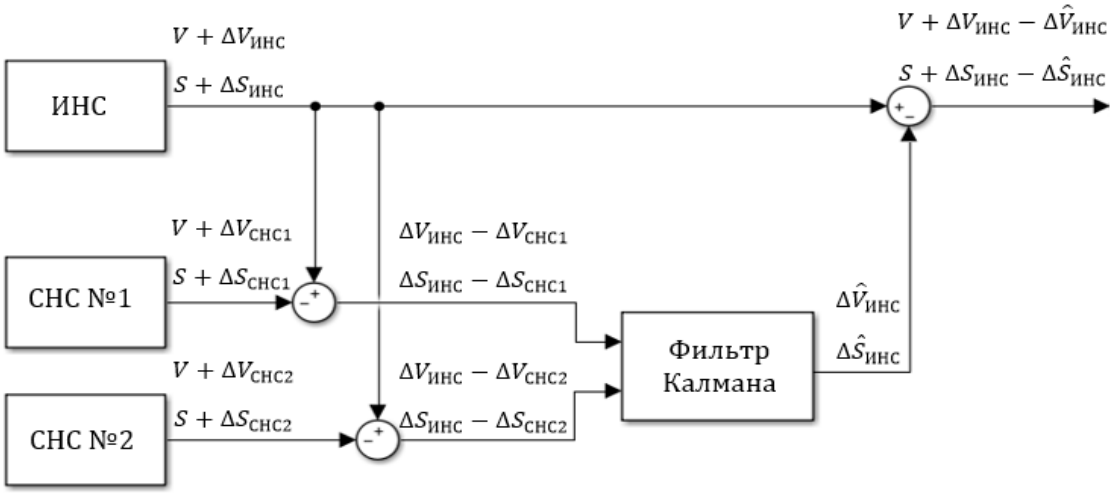

Рисунок 16 – Комплексирование трех навигационных систем летательного аппарата

1.5.1 Архитектура программного обеспечения

Для того, чтобы смоделировать работу в реальном времени, необходимо разработать имитаторы каждой системы навигации, которые с заданной частотой будут формировать измерения.

Реализуем программное обеспечение в среде разработки Qt Creator (Приложение Г).

С отправляющей стороны создадим программу, имитирующую выдачу данных от навигационных систем, в ядре которой расположим имитаторы БИНС, СНС №1 и СНС №2, отправляющие пакеты навигационных данных по протоколу UDP.

Контроль ошибок при передаче данных реализуем с помощью проверки контрольной суммы каждого пакета. В случае несоответствия контрольной суммы, пришедший пакет будет игнорирован.

С принимающей стороны создадим основную программу, реализующую прием, оценку и отображение навигационных данных.

Ядро основной программы содержит в себе сервера БИНС, СНС №1 и СНС №2, каждый из которых получает данные от соответствующего имитатора навигационной системы, передает их на графики главного окна

приложения и в объект-оценщик, реализующий алгоритм дискретного фильтра Калмана.

Схема взаимодействия программных компонентов представлена на рисунке 17.

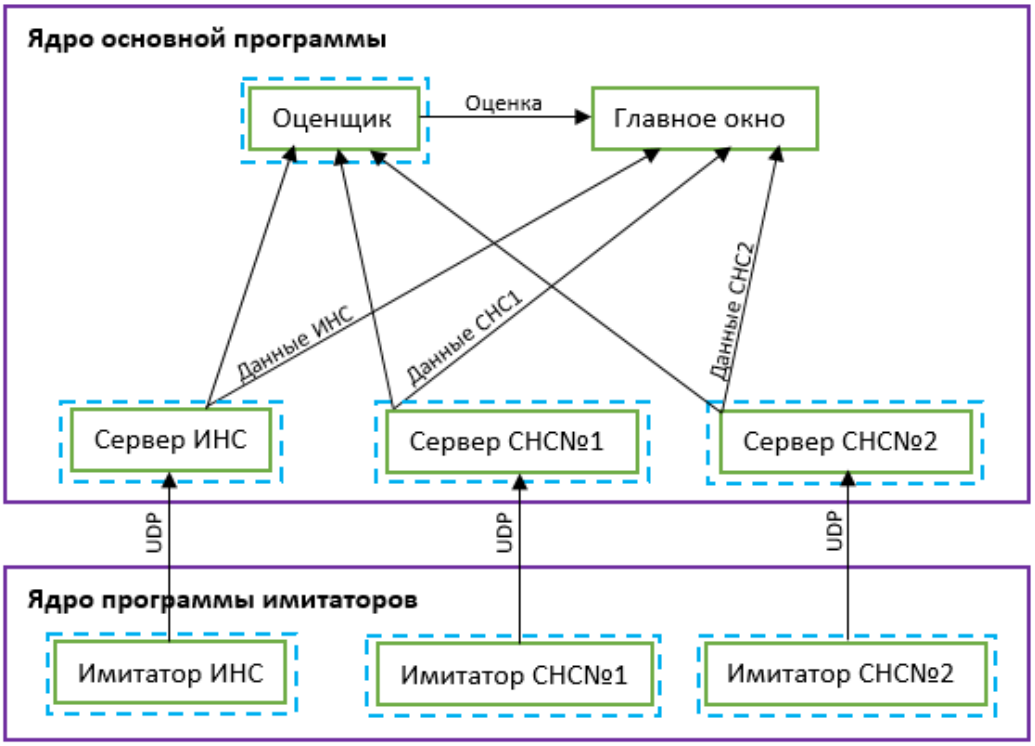

Рисунок 17 – Схема взаимодействия программных компонентов

Ядро основной программы связывает программные компоненты между собой при помощи слот-сигнальных соединений.

По прибытии данных с имитатора, сервер считывает полученный пакет и вырабатывает сигнал, содержащий навигационные данные, которые затем передаются на графики главного окна и в оценщик.

Оценщик, с приходом сигнала БИНС, экстраполирует предыдущую оценку и ее матрицу ковариации по формулам (1.3.3), а по прибытию сигнала СНС корректирует их по формулам (1.3.4) – (1.3.6), вычисляя таким образом оптимальную оценку и ее матрицу ковариации. После того, как была найдена оптимальная оценка ошибок БИНС, она вычитается из параметров ЛА,

пришедших от БИНС, содержащих искомую ошибку. Таким образом, формируется оценка текущего параметра ЛА.

Так как оценщик реализует экстраполяцию с частотой дискретизации БИНС, а оптимальная оценка формируется реже (только с приходом сигнала СНС, имеющим меньшую частоту), в формулах (1.3.3) в правой части, в случае отсутствия оптимальной оценки и матрицы ковариации для предыдущего шага, вместо них в формулы подставляется предыдущая экстраполированная оценка и экстраполированная матрица ковариации соответственно.

Для того, чтобы обеспечить параллельную выдачу/прием данных от навигационных систем, а также для разгрузки основного потока программы с графическим интерфейсом, вынесем все имитаторы, сервера и объект-оценщик в отдельные потоки. На схеме границы потоков выделены пунктирными линиями.

Для сравнения измерений навигационных систем и оценок с реальным параметром, создадим дополнительно имитатор и сервер реальных координат ЛА, который будет выдавать данные на графики координат ЛА.

В главном окне приложения разместим две вкладки, содержащие графики текущих координат ЛА и ошибок ИНС. На нижней панели расположим кнопки, позволяющие отключать/включать прием сигналов от СНС №1 и СНС №2.

В оценщике реализуем таймер, который раз в секунду будет проверять наличие пришедших за это время сигналов СНС. В случае их отсутствия, оценщик начнет выдавать экстраполированную оценку без корректировок.

1.5.2 Результаты моделирования

Зададим частоту выдачи сигналов БИНС – 10 Гц, СНС №1 и СНС №2 – 5 Гц и 4 Гц соответственно, и запустим моделирование. Результаты представлены на рисунках 18–30.

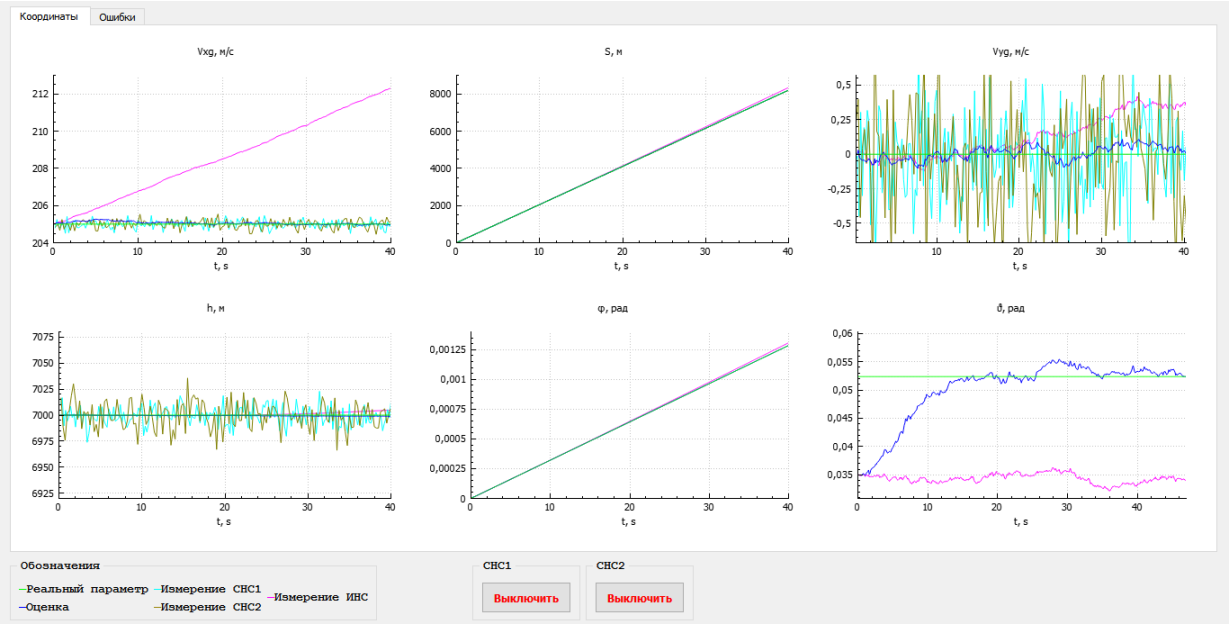

Рисунок 18 – Координаты ЛА

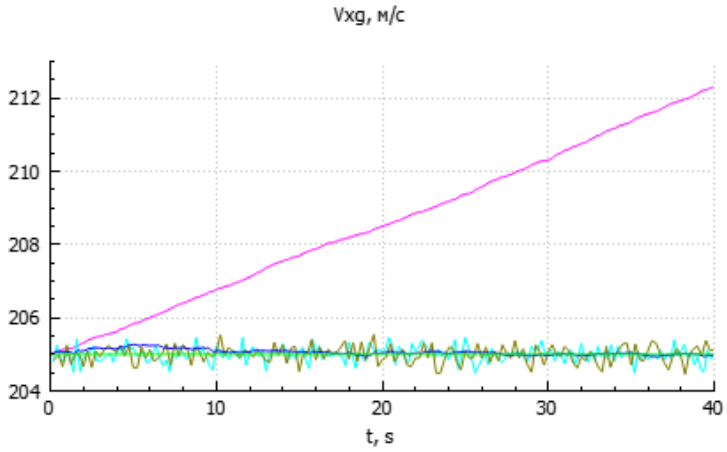

Рисунок 19 – Горизонтальная скорость

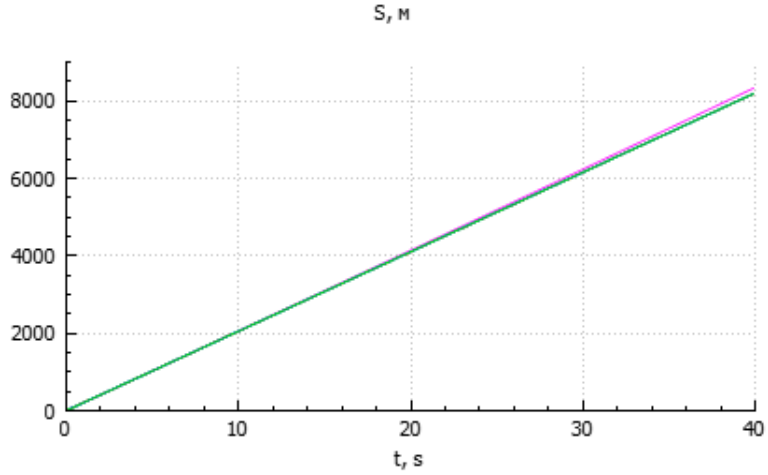

Рисунок 20 – Пройденный путь

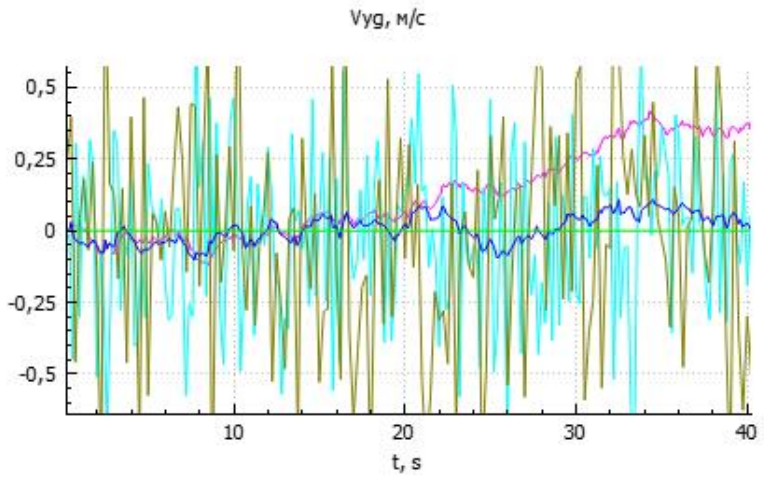

Рисунок 21 – Вертикальная скорость

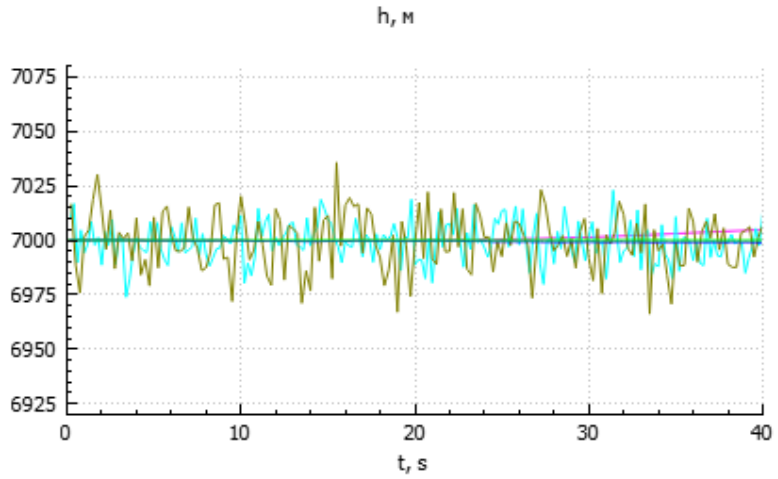

Рисунок 22 – Высота

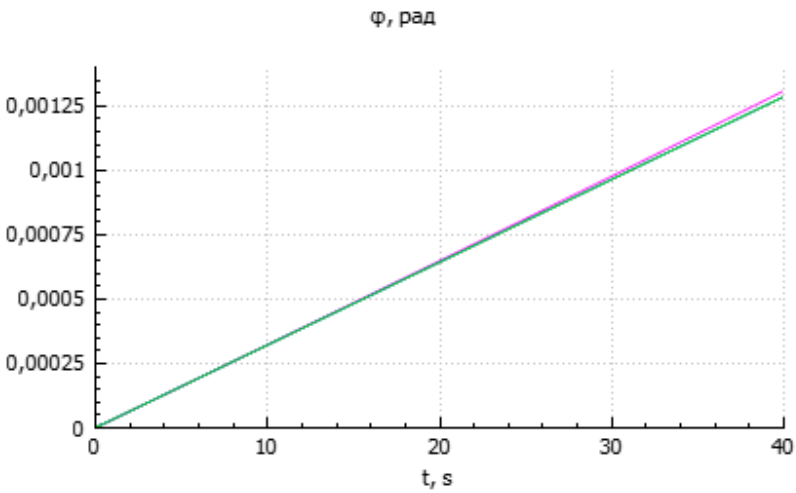

Рисунок 23 – Широта

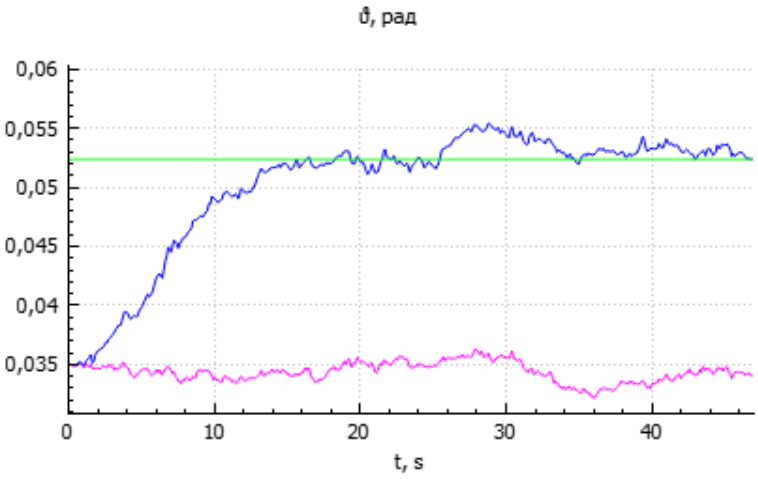

Рисунок 24 – Угол тангажа

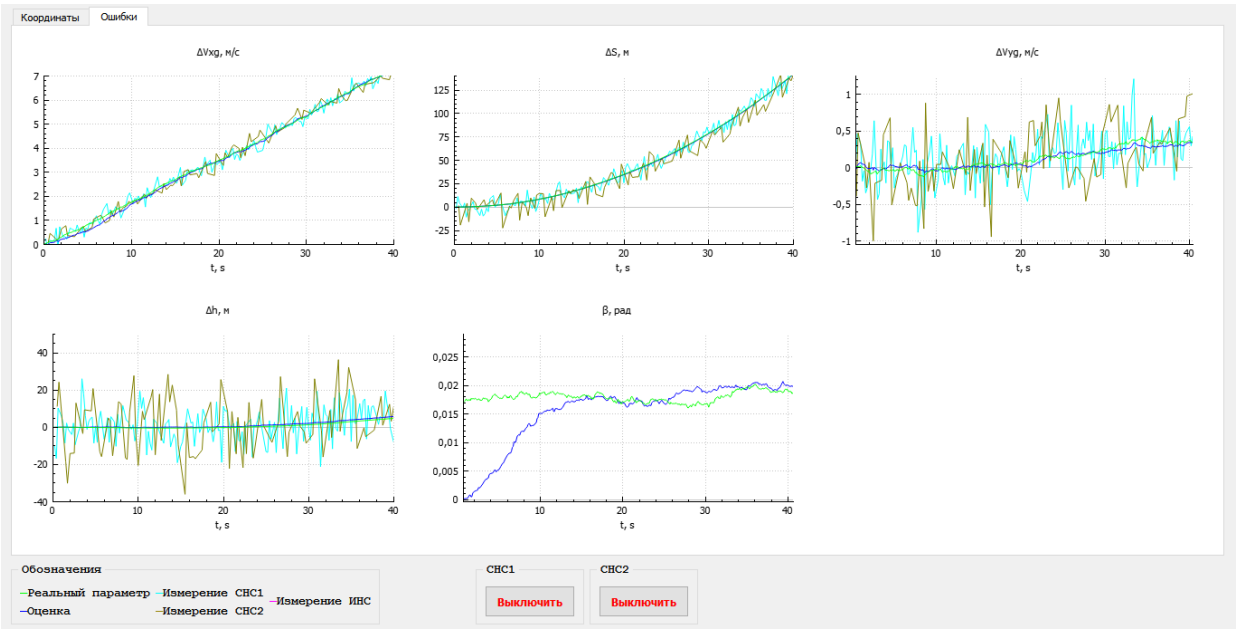

Рисунок 25 – Ошибки ИНС

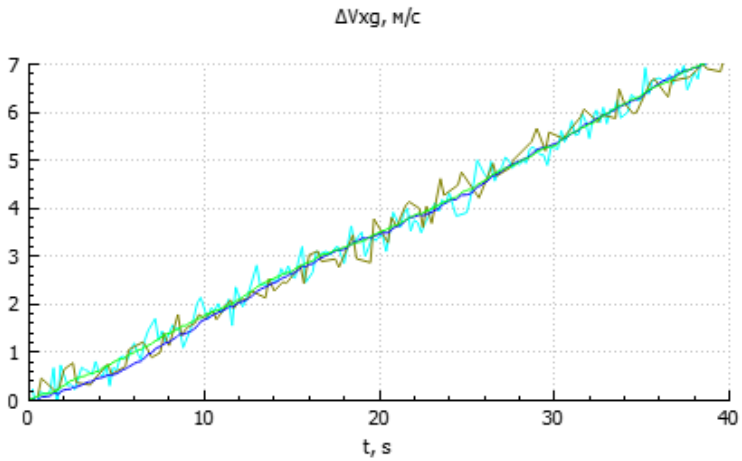

Рисунок 26 – Ошибка выработки горизонтальной скорости

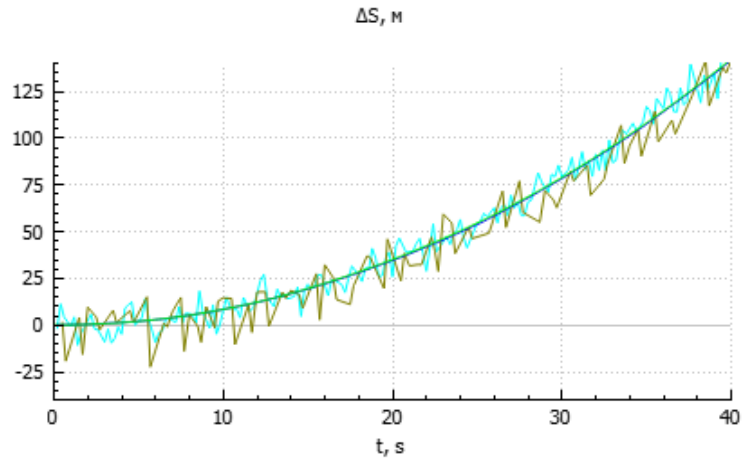

Рисунок 27 – Ошибка выработки пройденного расстояния

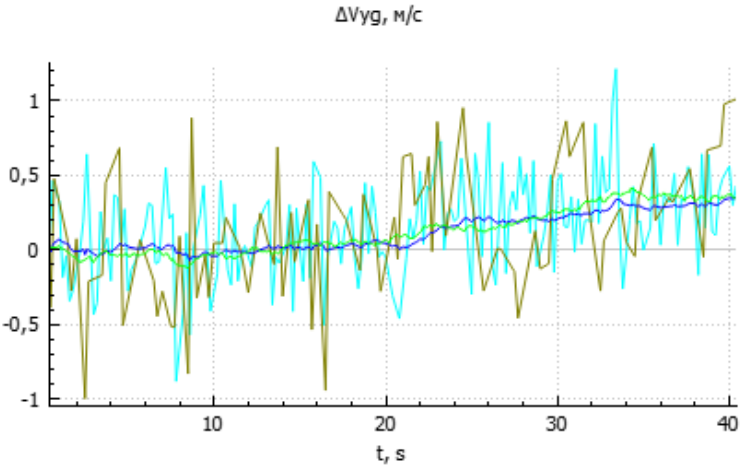

Рисунок 28 – Ошибка выработки вертикальной скорости

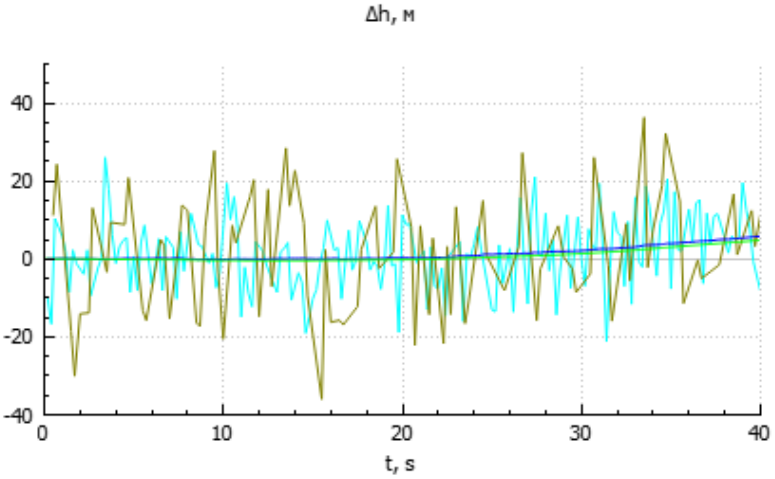

Рисунок 29 – Ошибка выработки высоты

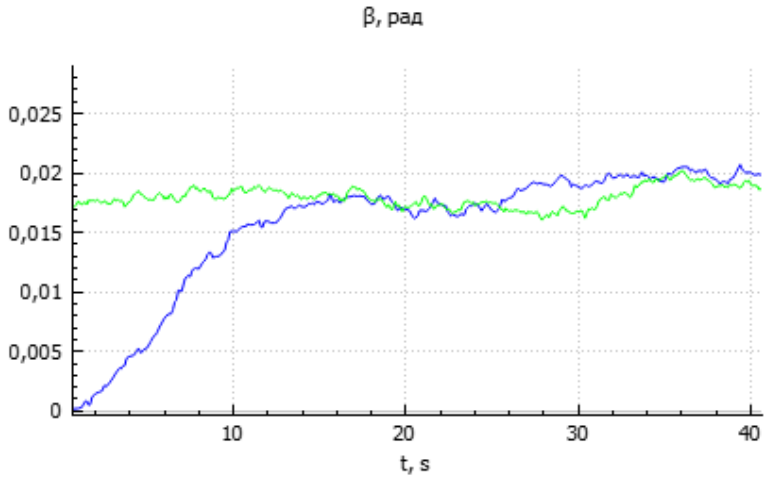

Рисунок 30 – Ошибка построения вертикали

Видно, что ошибки ИНС, а, следовательно, и координаты ЛА, оцениваются достаточно точно.

1.5.3 Результаты моделирования с отсутствием сигнала СНС

Промоделируем отсутствие сигналов спутников в течение некоторого промежутка времени (8–165 секунд). Результаты представлены на рисунках 31–43.

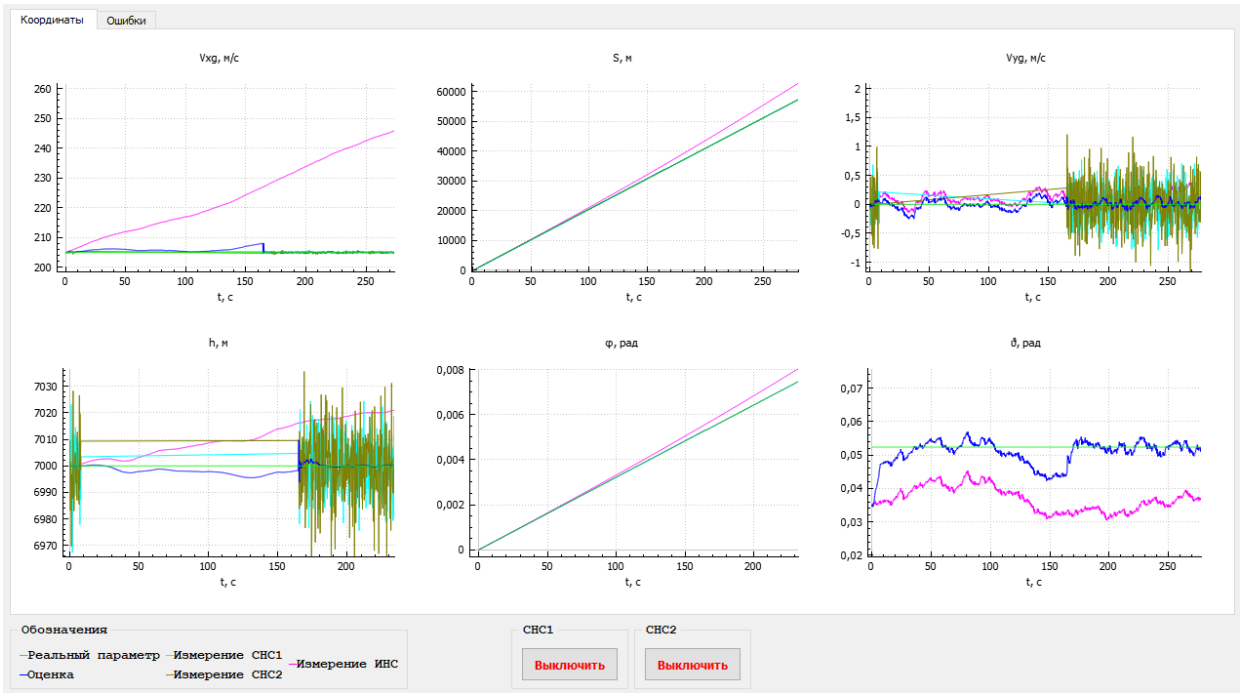

Рисунок 31 – Координаты ЛА

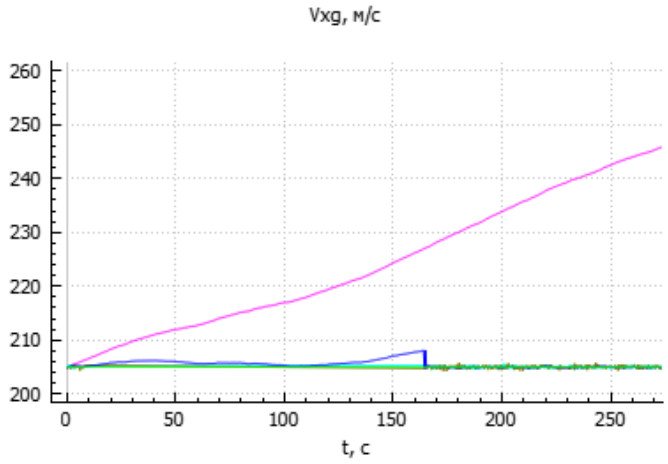

Рисунок 32 – Горизонтальная скорость

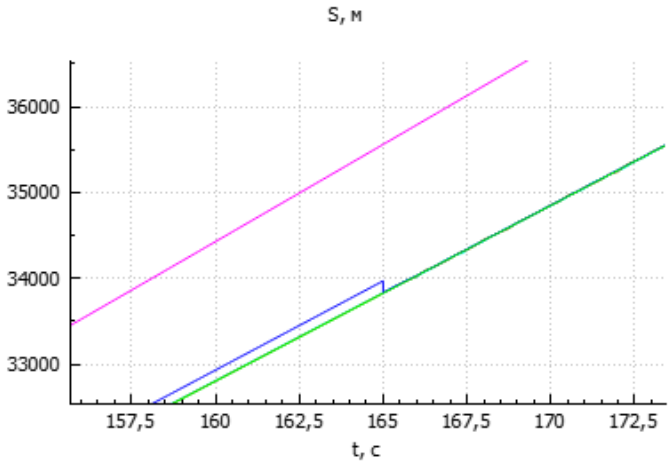

Рисунок 33 – Пройденный путь, приближенный вид

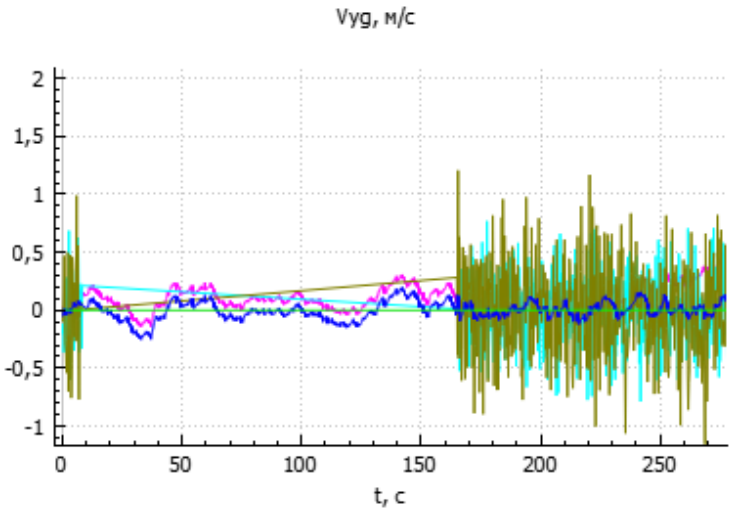

Рисунок 34 – Вертикальная скорость

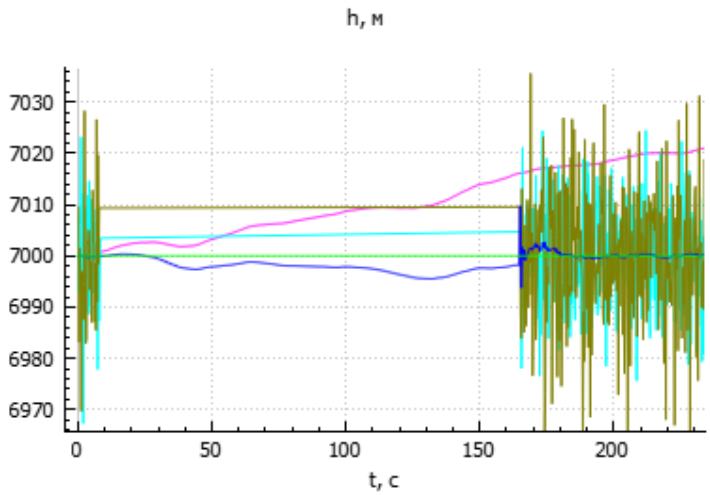

Рисунок 35 – Высота

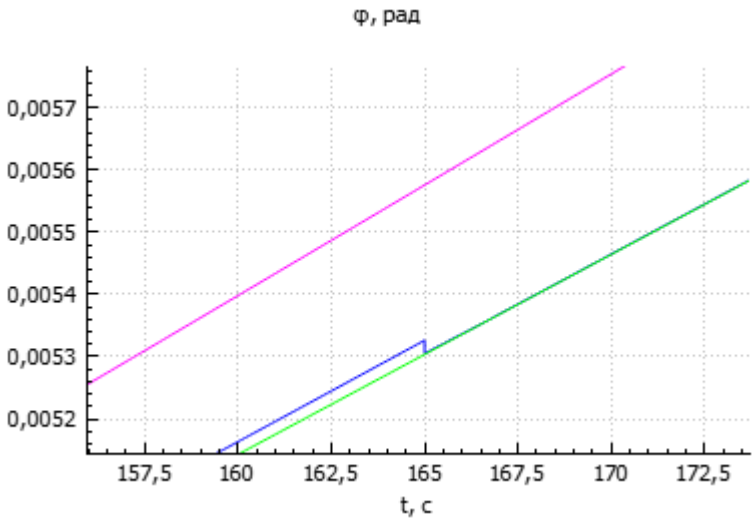

Рисунок 36 – Широта, приближенный вид

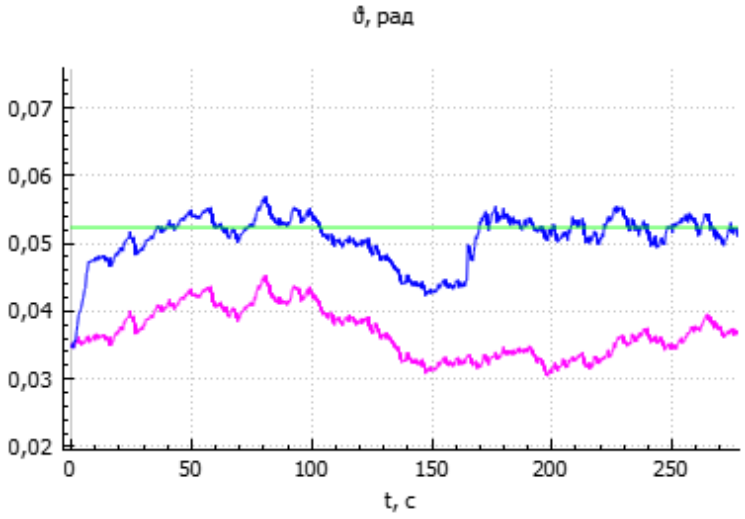

Рисунок 37 – Угол тангажа

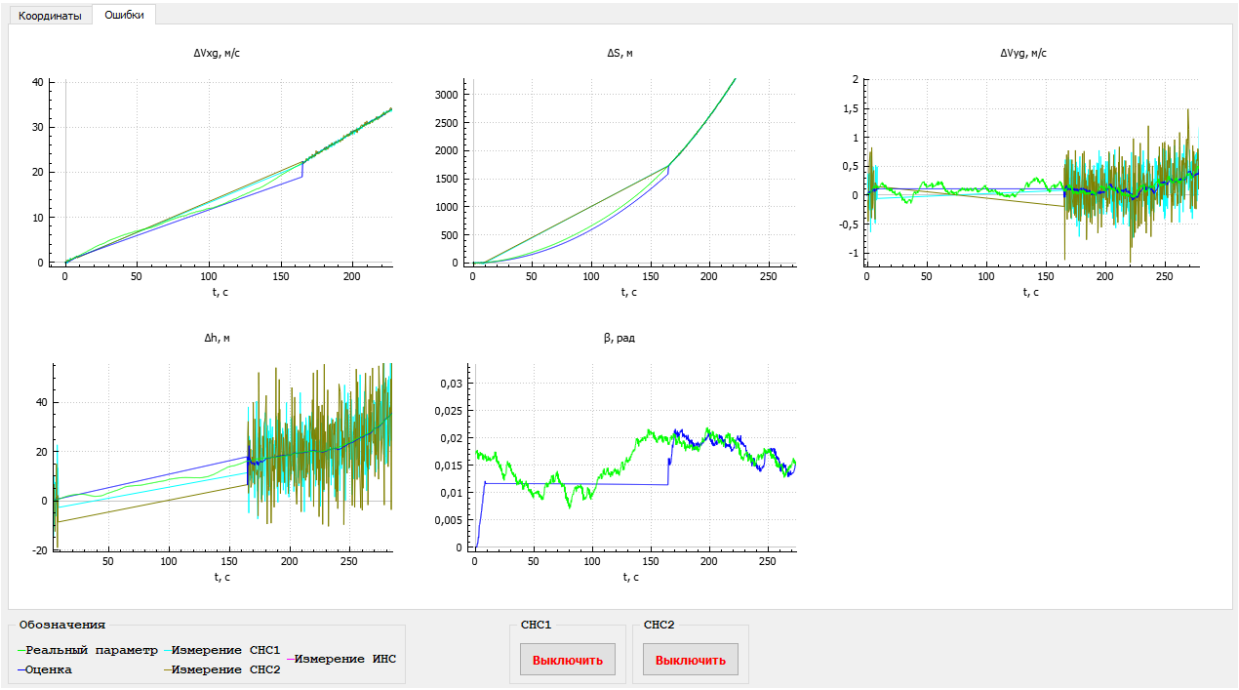

Рисунок 38 – Ошибки ИНС

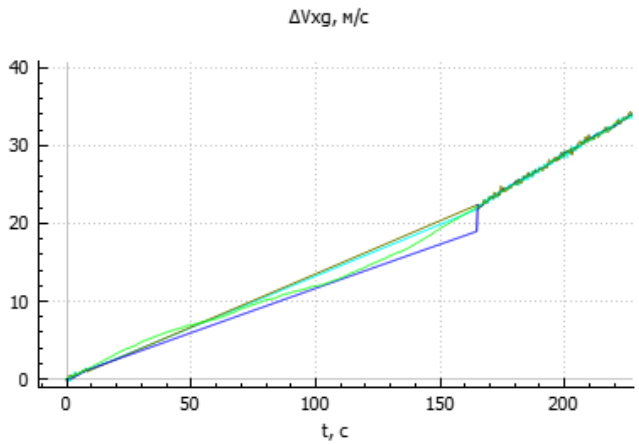

Рисунок 39 – Ошибка выработки горизонтальной скорости

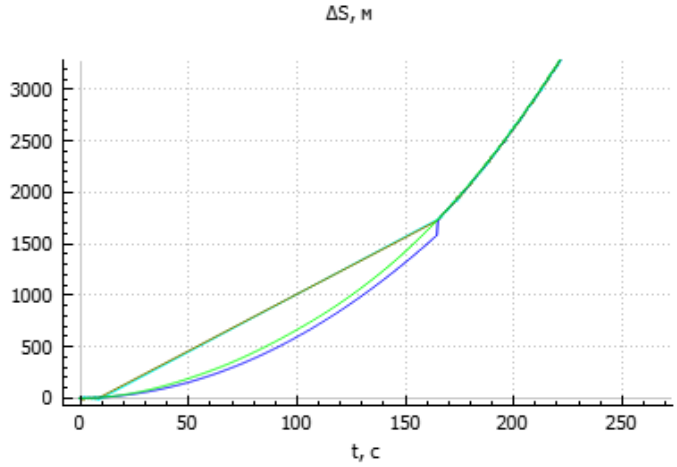

Рисунок 40 – Ошибка выработки пройденного расстояния

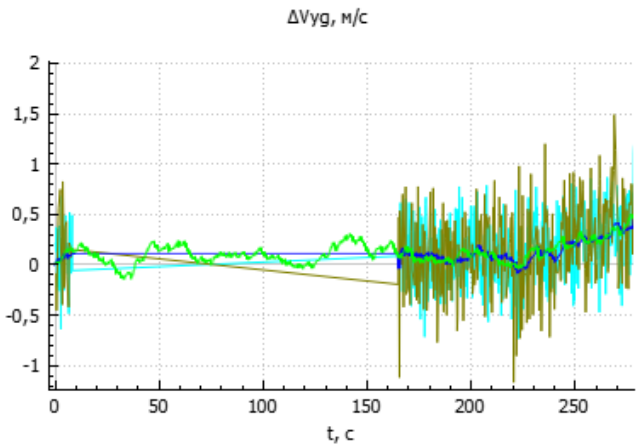

Рисунок 41 – Ошибка выработки вертикальной скорости

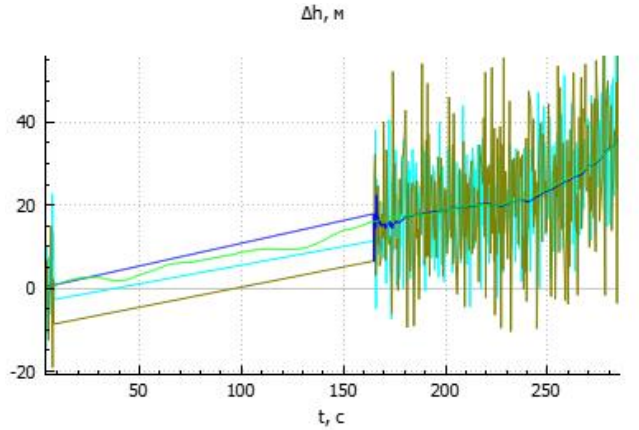

Рисунок 42 – Ошибка выработки высоты

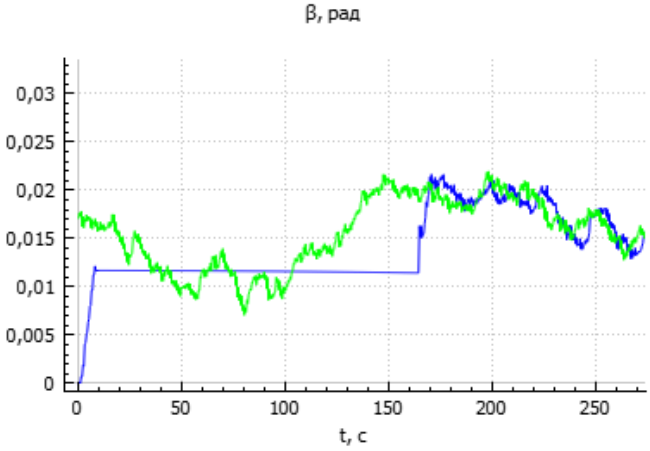

Рисунок 43 – Ошибка построения вертикали

В отсутствие сигнала СНС вырабатывается экстраполированная оценка, которая является неточной ввиду значительных шумов в системе ошибок БИНС, однако можно заметить, что уход экстраполированной

оценки от реального параметра происходит намного медленнее, чем уход параметра, вырабатываемого БИНС.

С получением сигнала СНС выполняется корректировка оценки с учетом погрешности модели системы и погрешности СНС. По графикам виден резкий скачок оценки в сторону реального параметра в момент прихода сигнала СНС.

Таким образом, можно сделать вывод, что после длительного отсутствия сигнала СНС, с его получением, точность оценивания восстанавливается.

1.6 Исследование влияния модели шумов системы на получаемую оценку

Особенностью фильтра Калмана является то, что он принимает возмущения системы белым шумом. Исследуем поведение фильтра при наличии в сигналах акселерометров и гироскопа аддитивных погрешностей, модель которых нельзя приближенно описать в виде белого шума.

1.6.1 Модель аддитивных шумов в составе сигналов инерциальных датчиков

Учтем в показаниях датчиков незначительное и изменяющееся смещение, называемое нестабильностью нуля. Данную погрешность можно приближенно описать розовым шумом.

Также учтем процессы, называемые случайным блужданием угловой скорости (RRW) для гироскопа и случайным блужданием линейного ускорения (LARW) для акселерометра, которые происходят от белого шума, встроенного в датчик и вышедшего как интегрированный сигнал.

Указанные погрешности представляют собой винеровский процесс – результат пропускания белого шума с интенсивностью RRW^2 для гироскопа и $LARW^2$ для акселерометра через интегрирующее звено.

Случайное блуждание угловой скорости ε_{rrw} можно определить, как:

$$\varepsilon_{rrw}(k+1) = \varepsilon_{rrw}(k) + w(k) \cdot T, \qquad (1.6.1)$$

где k – номер итерации;

T – период дискретизации;

w – порождающий белый шум с СКО $\sigma_{rrw} = RRW/\sqrt{T}$.

Случайное блуждание линейного ускорения δa_{larw} определяется аналогично:

$$\delta a_{larw}(k+1) = \delta a_{larw}(k) + w(k) \cdot T, \qquad (1.6.2)$$

где w – порождающий белый шум с СКО $\sigma_{larw} = LARW/\sqrt{T}$.

Примем параметры датчиков равными следующим значениям:

- $RRW = 0.00120 \frac{\text{рад}}{\text{с}\sqrt{\text{с}}}$;
- $LARW = 0.00096 \frac{\text{м}}{\text{с}^2\sqrt{\text{с}}}$.

Разделив соответствующие значения на $\sqrt{T}$, получим значения СКО порождающих белых шумов w:

- $\sigma_{rrw} = 0.0120 \frac{\text{рад}}{\text{с}^2}$;
- $\sigma_{larw} = 0.0096 \frac{\text{м}}{\text{с}^3}$.

Добавим вышеуказанные погрешности в состав ошибок гироскопа и акселерометров к уже имеющемуся белому шуму.

1.6.2 Результаты моделирования

На рисунках 44–56 представлены результаты моделирования системы с учетом аддитивных погрешностей в виде белого, розового и винеровского шумов в составе сигналов инерциальных датчиков.

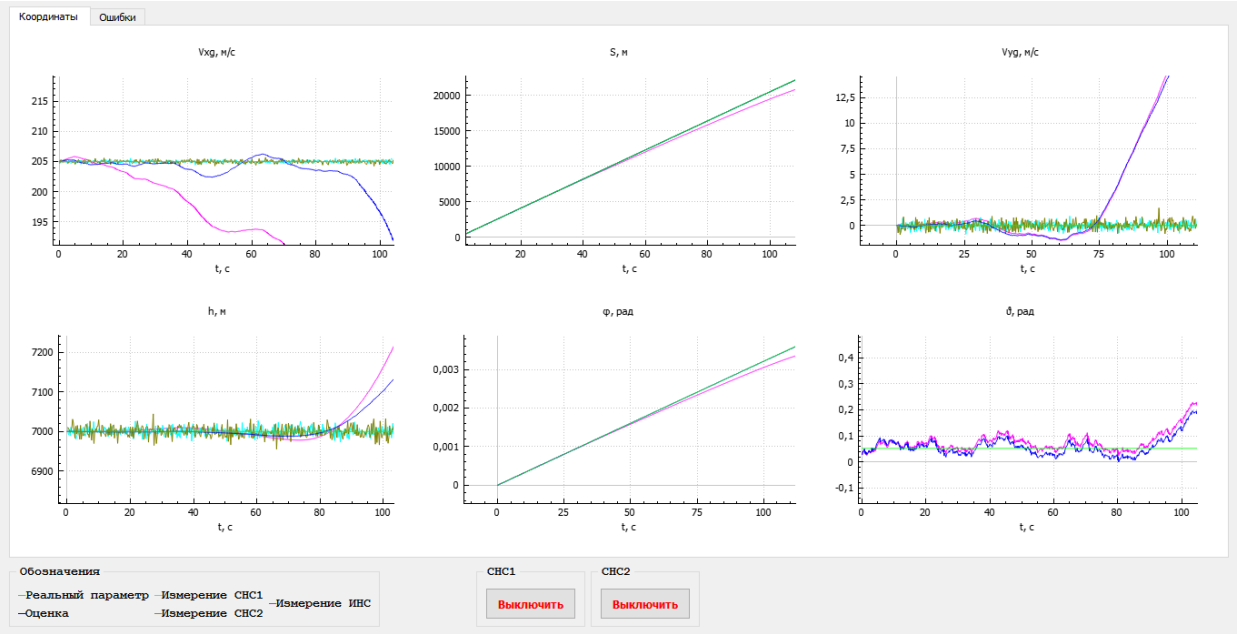

Рисунок 44 – Координаты ЛА

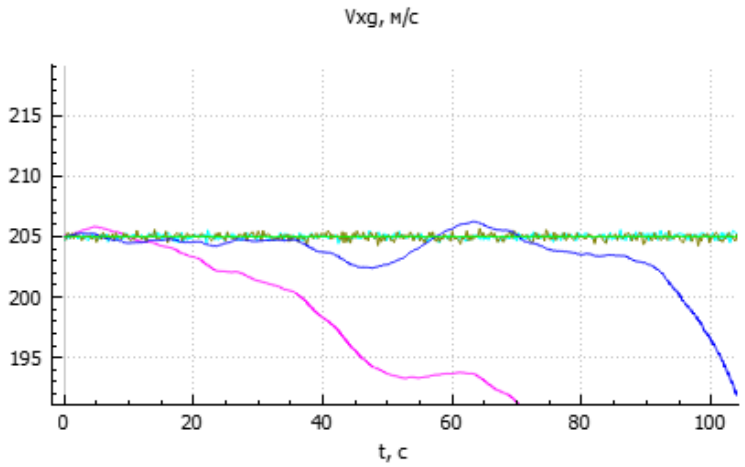

Рисунок 45 – Горизонтальная скорость

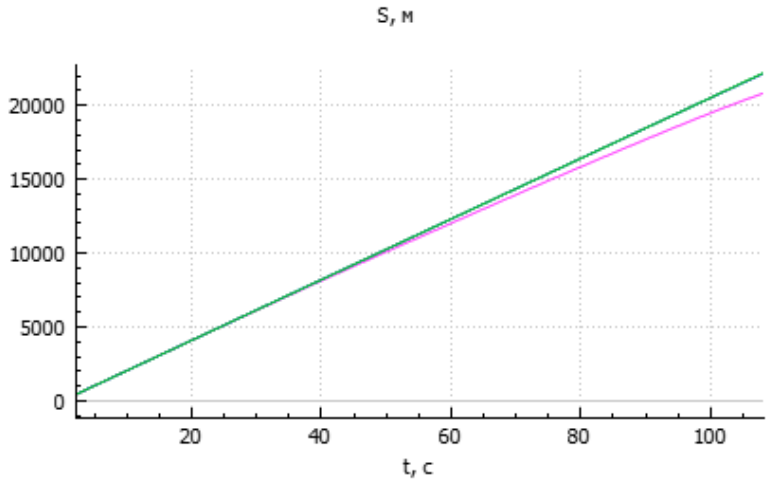

Рисунок 46 – Пройденный путь

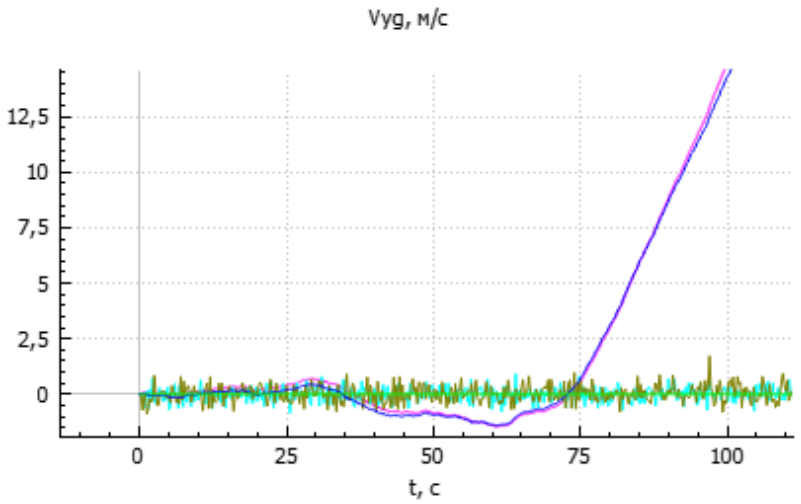

Рисунок 47 – Вертикальная скорость

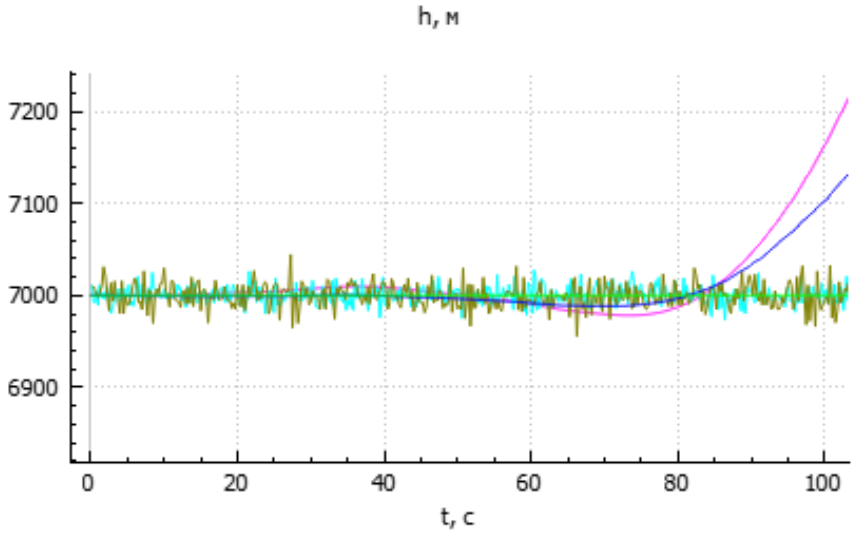

Рисунок 48 – Высота

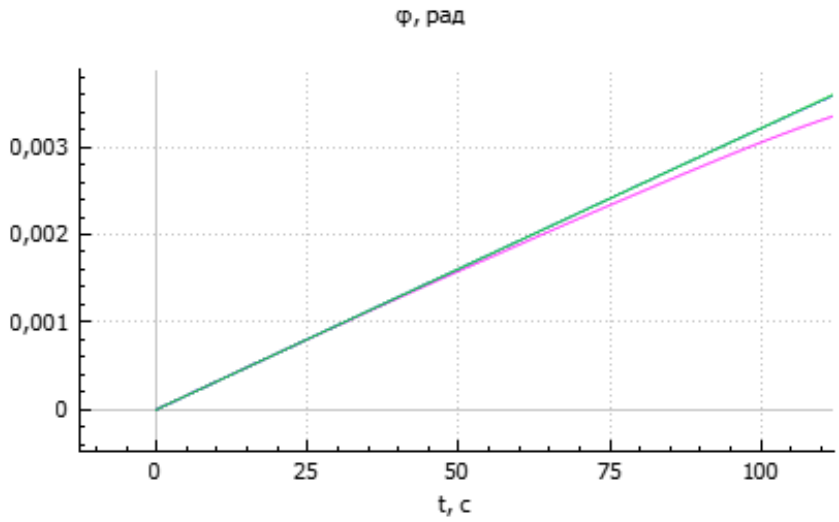

Рисунок 49 – Широта

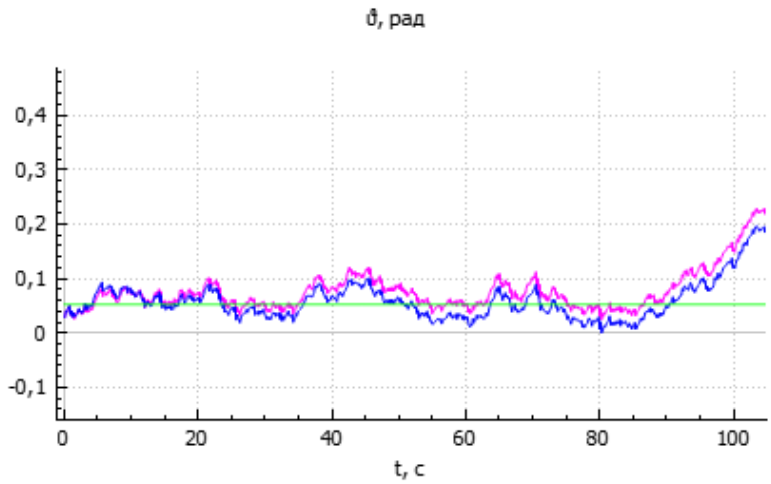

Рисунок 50 – Угол тангажа

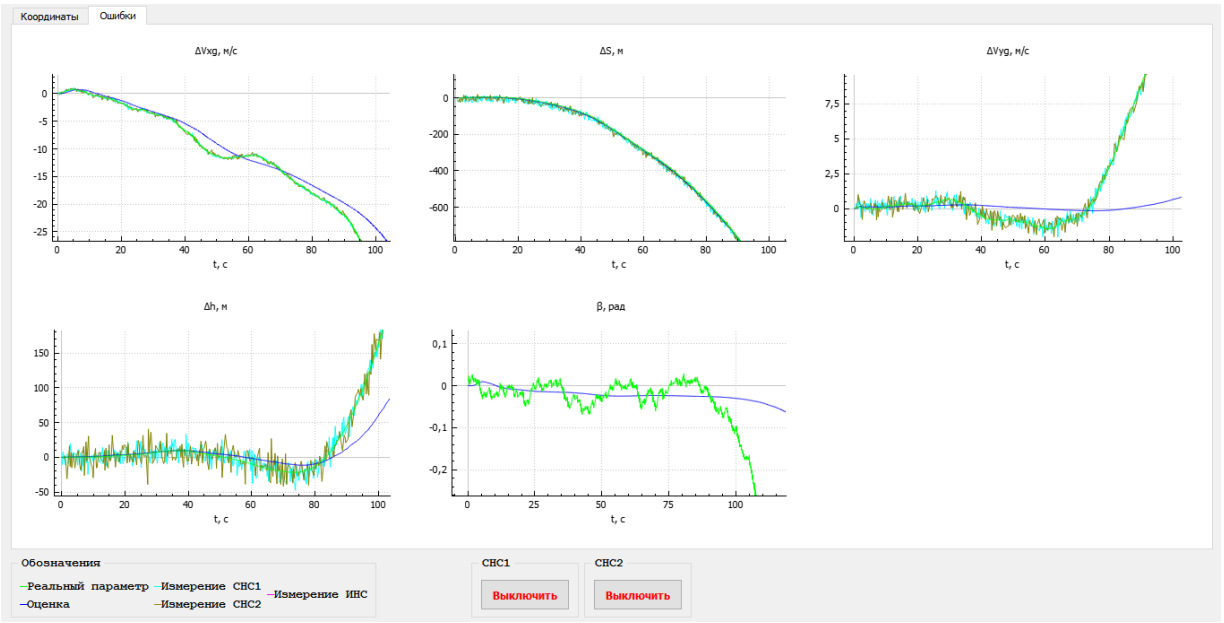

Рисунок 51 – Ошибки ИНС

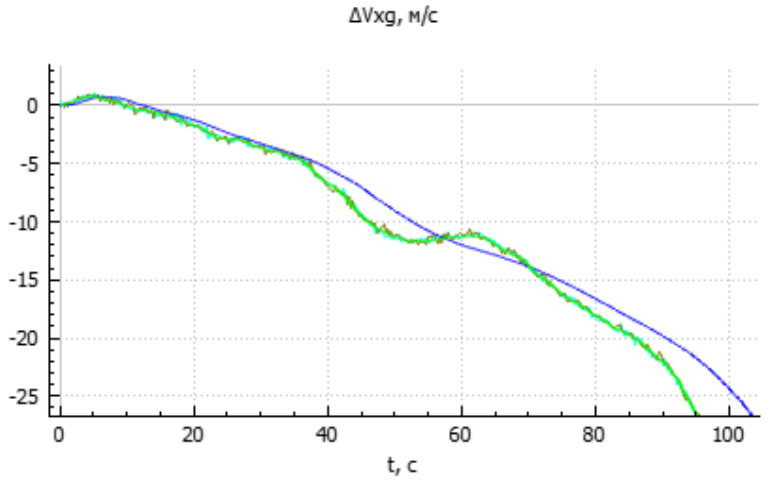

Рисунок 52 – Ошибка выработки горизонтальной скорости

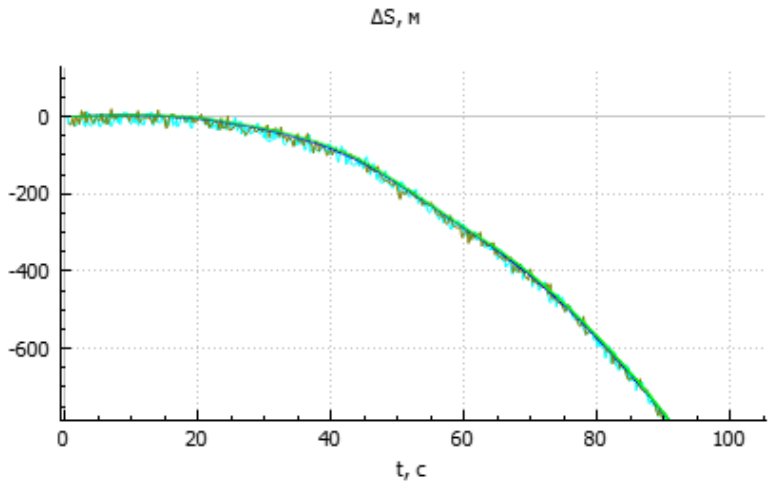

Рисунок 53 – Ошибка выработки пройденного расстояния

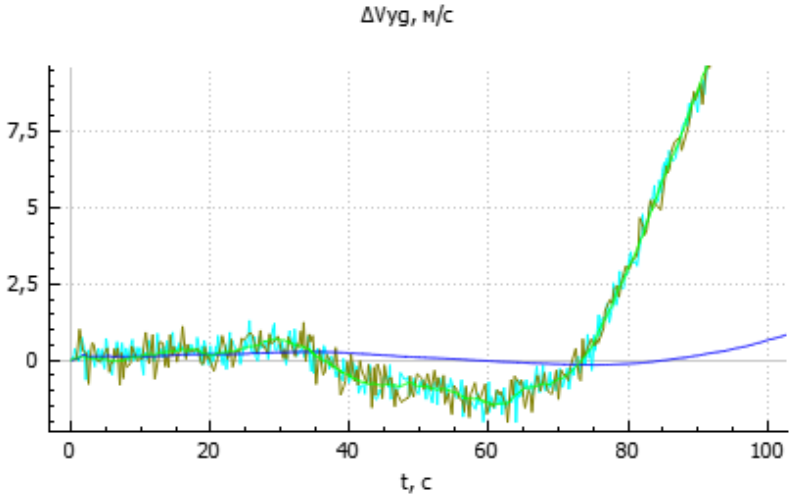

Рисунок 54 – Ошибка выработки вертикальной скорости

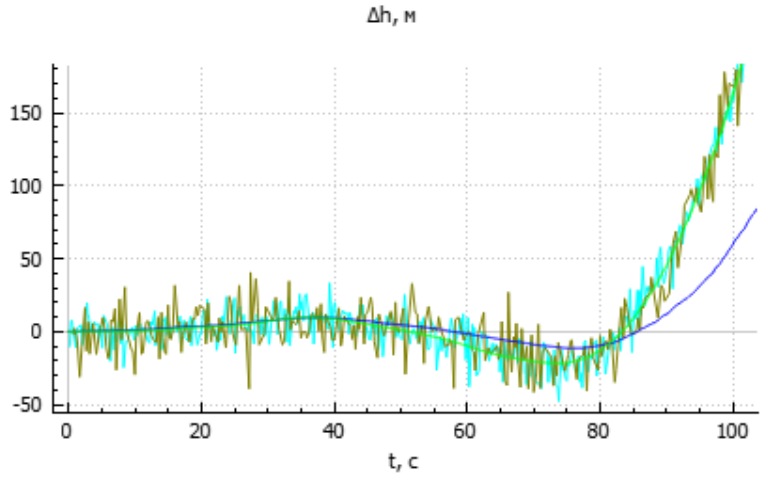

Рисунок 55 – Ошибка выработки высоты

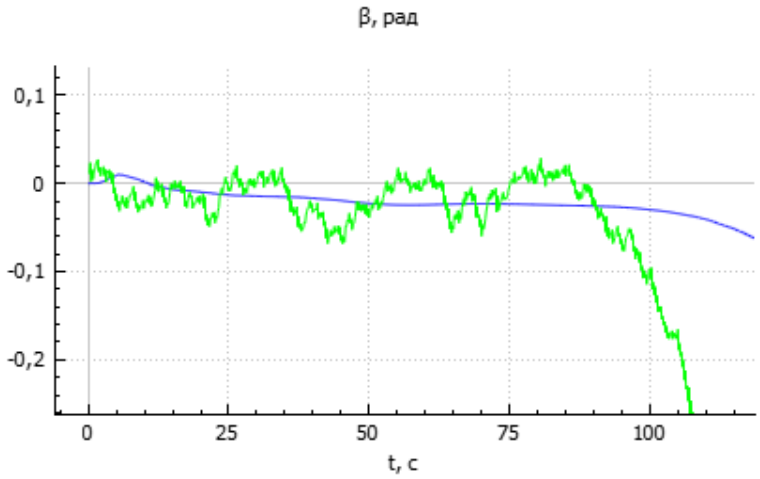

Рисунок 56 – Ошибка построения вертикали

Можно заметить, что ошибка оценивания начала возрастать со временем, следовательно, фильтр Калмана расходится.

Предотвратить расходимость фильтра можно, увеличив дисперсии на главной диагонали ковариационной матрицы Q объекта. Такое увеличение будет соответствовать введению дополнительного шума в модель динамики системы.

Для этого увеличим среднеквадратические отклонения на главной диагонали матрицы Q в 10 раз и запустим моделирование. Результаты представлены на рисукнках 57–69.

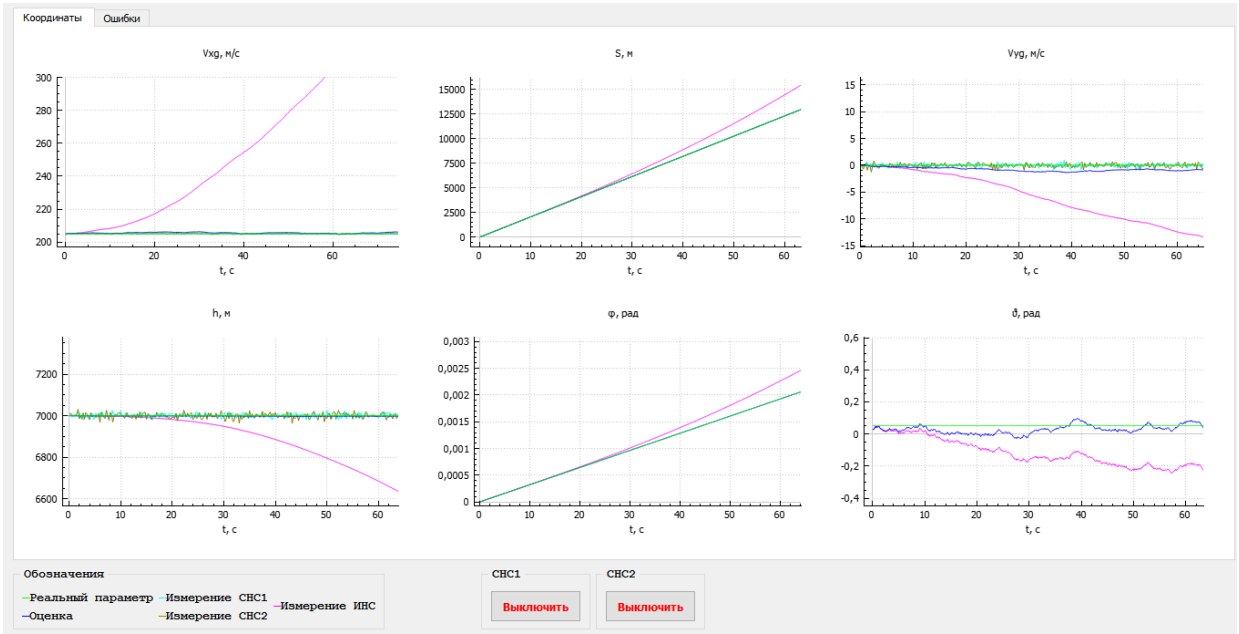

Рисунок 57 – Координаты ЛА

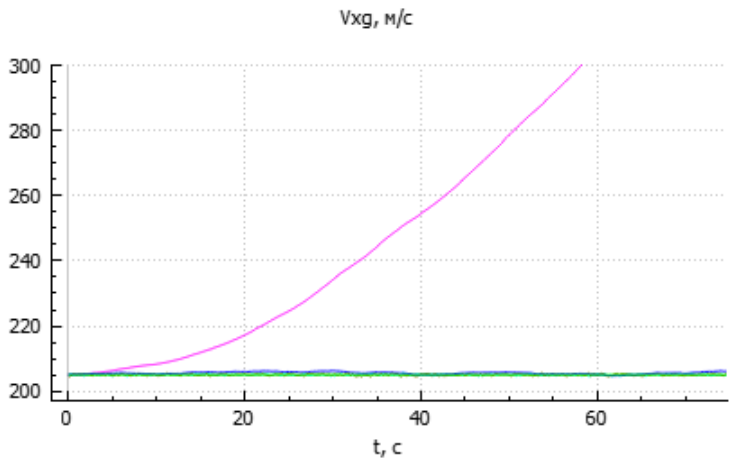

Рисунок 58 – Горизонтальная скорость

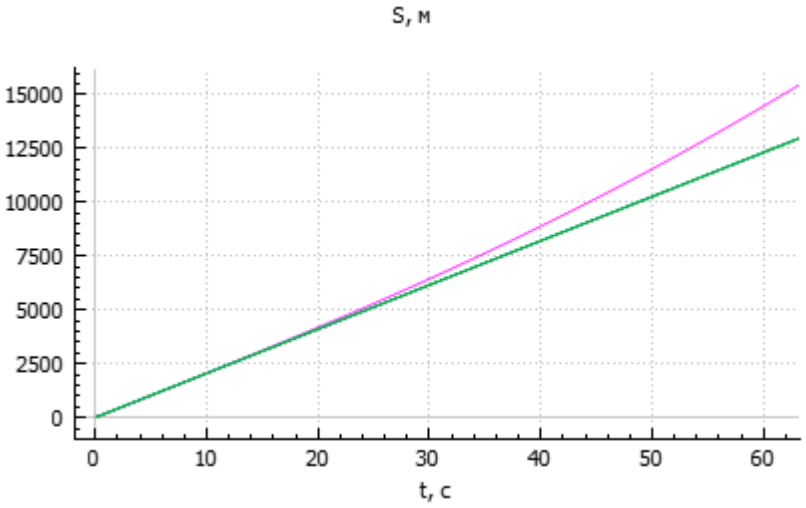

Рисунок 59 – Пройденный путь

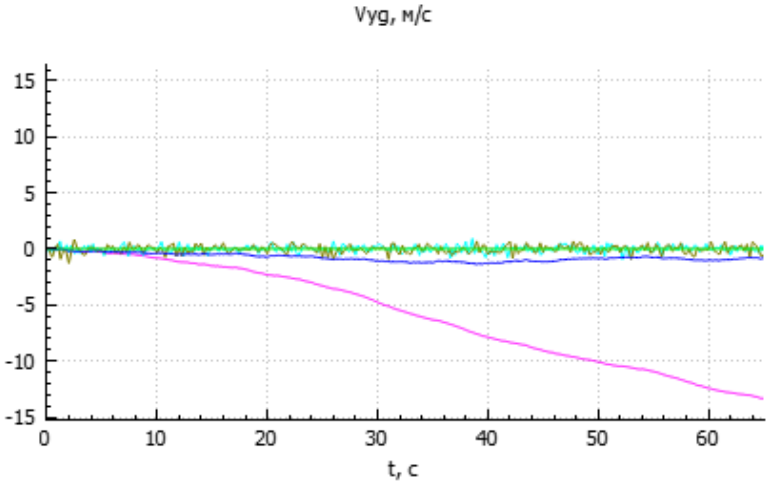

Рисунок 60 – Вертикальная скорость

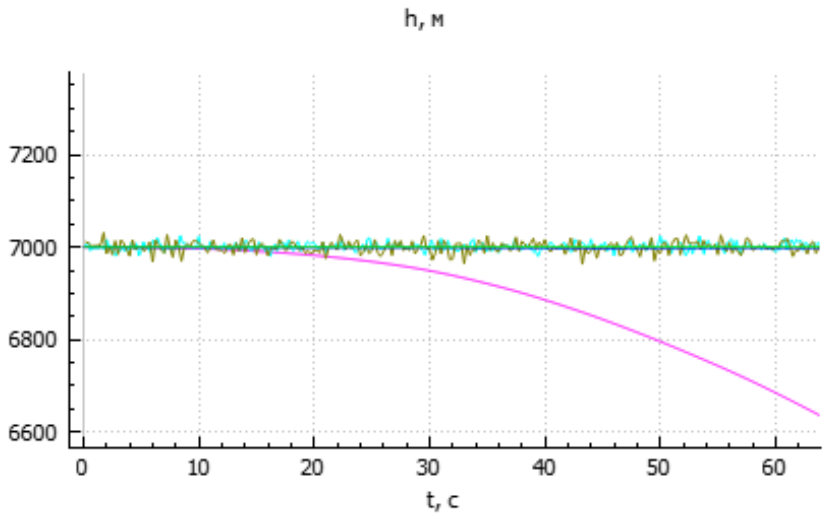

Рисунок 61 – Высота

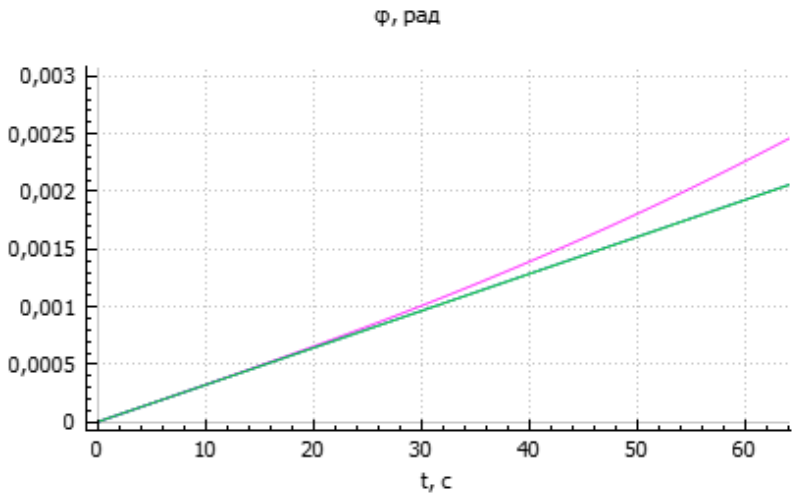

Рисунок 62 – Широта

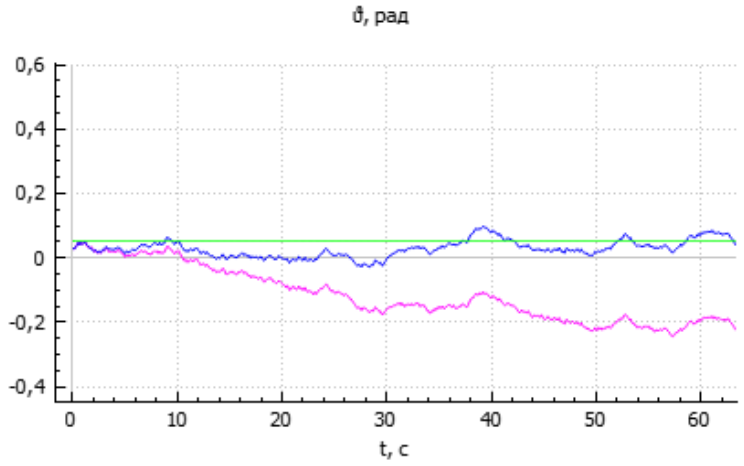

Рисунок 63 – Угол тангажа

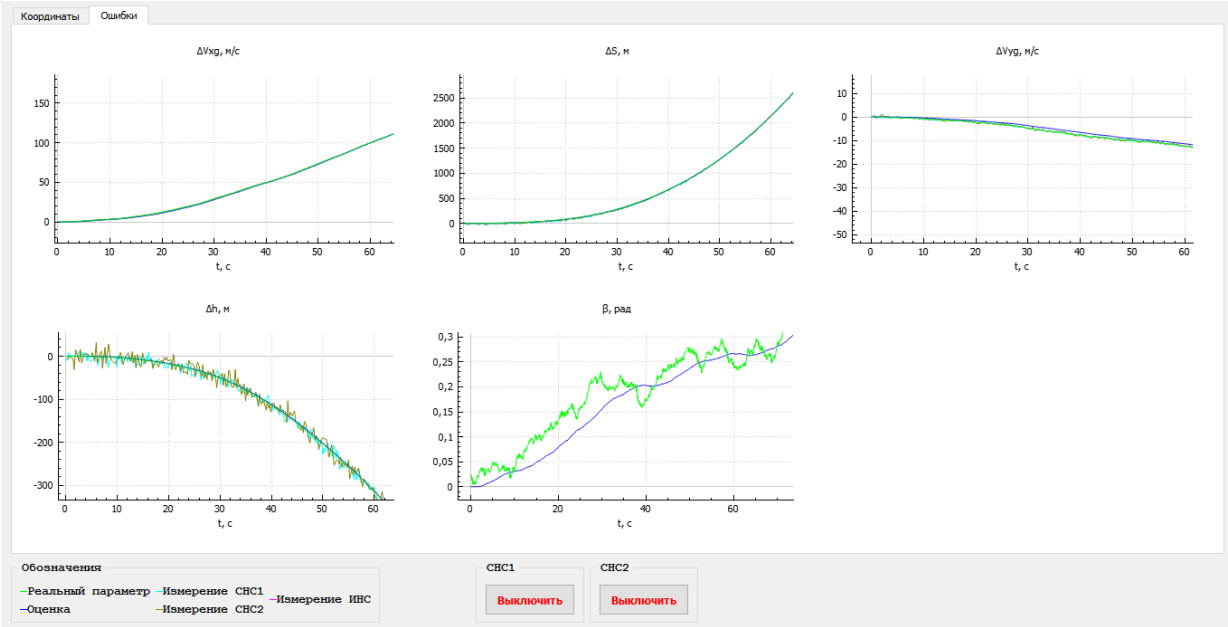

Рисунок 64 – Ошибки ИНС

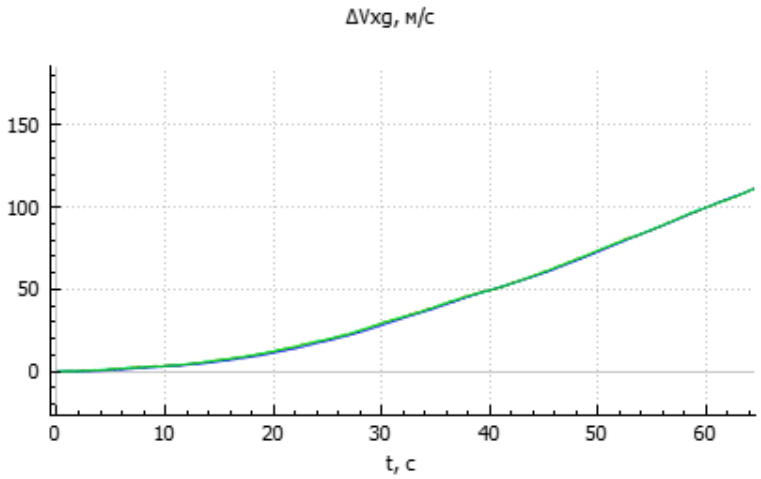

Рисунок 65 – Ошибка выработки горизонтальной скорости

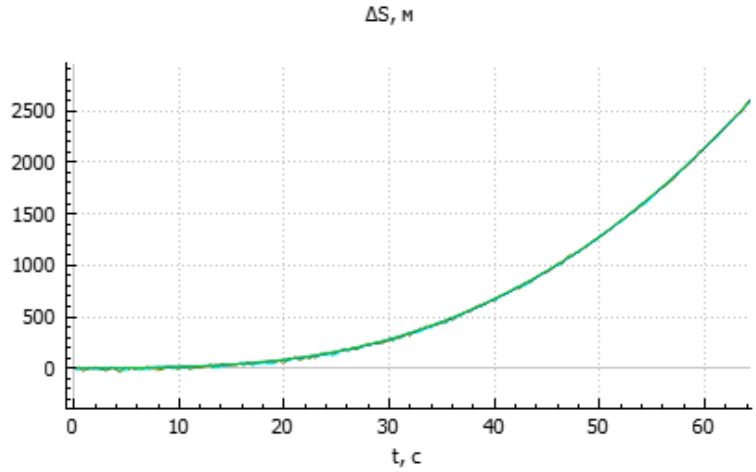

Рисунок 66 – Ошибка выработки пройденного расстояния

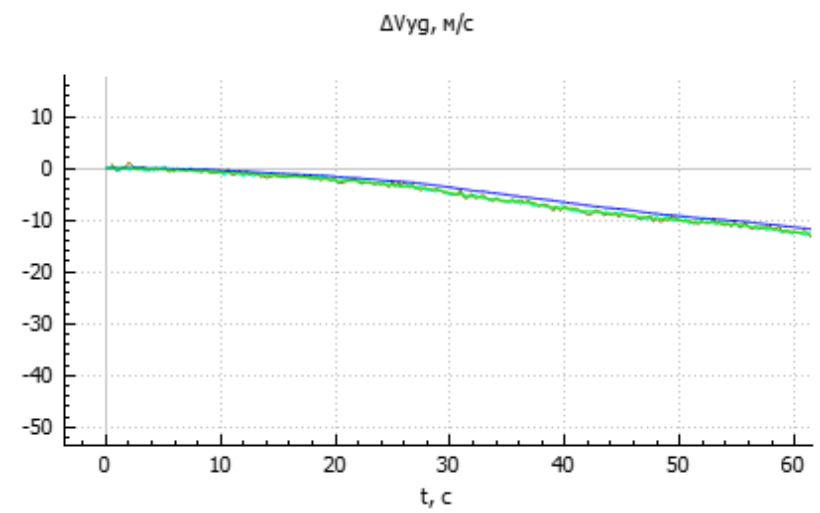

Рисунок 67 – Ошибка выработки вертикальной скорости

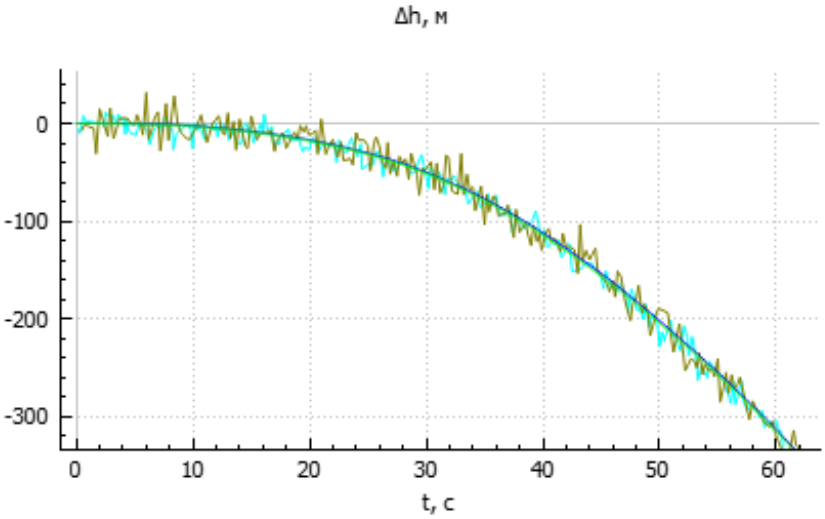

Рисунок 68 – Ошибка выработки высоты

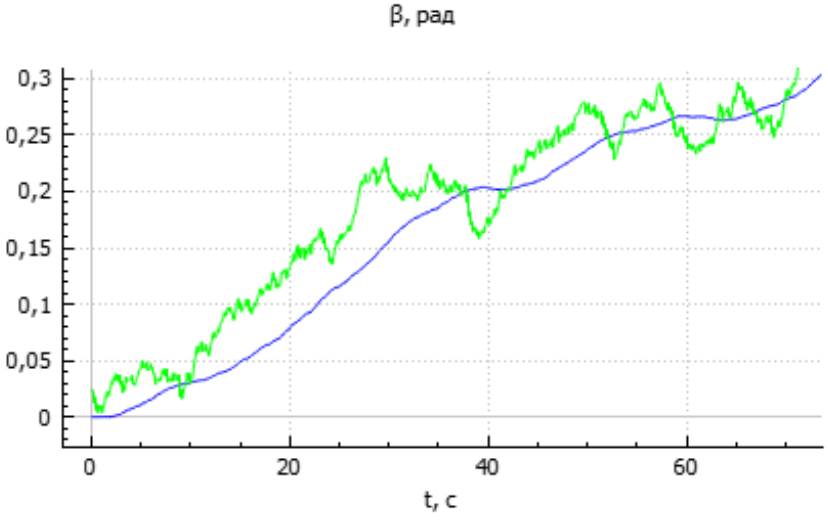

Рисунок 69 – Ошибка построения вертикали

Фильтр перестал расходиться. Оценка обладает достаточной точностью для определения вектора состояния летательного аппарата.

Дальнейшее увеличение диагональных элементов матрицы Q приводит к излишней зашумленности оценки сигналом СНС, так как в этом случае данные СНС при оценивании учитываются с большим весом, в виду предполагаемой значительной погрешности модели. Результаты моделирования при увеличении СКО на главной диагонали матрицы Q в 100 раз приведены в Приложении Д.

2 Технико-экономический раздел

Произведем расчет смет на выполнение выпускной квалификационной работы.

При расчете себестоимости разработки ВКР необходимо вычислить затраты, которые включают в себя прямые ($З_{ПР}$) и косвенные расходы ($З_{НР}$).

2.1 Прямые расходы

Прямые расходы – затраты на выполнение работ, которые могут быть непосредственно включены в себестоимость.

Прямые расходы включают:

- затраты на израсходованные материалы ($З_М$);
- затраты на электричество ($З_Э$);
- расчет заработной платы (ЗП) исполнителя, руководителей, консультантов ($З_{ЗП}$);
- амортизационные отчисления ($З_А$).

В итоге получим формулу для подсчета прямых расходов:

$$З_{ПР} = З_М + З_Э + З_{ЗП} + З_А \quad (2.1)$$

2.1.1 Затраты на материал

Затраты на материал, израсходованный при проведении исследования, определяем по формуле:

$$З_М = \sum_{i=1}^{n} Р_i \, Ц_i + З_{ТЗ}, \quad (2.2)$$

где $Р_i$ – расходное количество i -го материала, ед;

$Ц_i$ – цена i -го вида материала, руб;

$i = 1, 2, \ldots, n$ – виды материальных ресурсов;

$З_{ТЗ}$ – транспортно-заготовительные расходы.

Транспортно-заготовительные расходы – расходы, связанные с доставкой и заготовкой материальных ресурсов. Примем их равными 10% от суммы затрат на материалы. Расходы на каждый вид материалов приведены в таблице 6.

Таблица 6 – Расходы на материалы

Наименование материалов	Единица измерения	Количество материала $Р_i$, шт	Цена за единицу $Ц_i$, руб	Расходы, руб
Бумага для печати	Пачка бумаги (500 листов)	1	410	410
Носитель информации DVD-RW	штука	1	93	93
Транспортно-заготовительные издержки, 10%	-	-	-	50,3
Затраты на материалы (итого), руб.	553,3			

2.1.2 Затраты на электроэнергию

Затраты на электричество, необходимое для работы оборудования и освещения, рассчитываются по формуле:

$$З_Э = \sum_{i=1}^{n} М_i \cdot Т_i \cdot К_u \cdot Ц, \quad (2.3)$$

где $М_i$ – паспортная мощность электроприбора (ноутбука), использованного в проведении работ, кВт/ч;

$Т_i$ – время работы электроприбора, затраченного для выполнения поставленной задачи, час;

$К_и$ – коэффициент использования мощности;

Ц – цена 1 кВт электричества, руб.

Для выполнения ВКР был использован ноутбук, паспортная мощность которого составляет $М = 0{,}117$ кВт/ч.

Выполнение работ происходило по следующему графику:

- срок выполнения ВКР – 12 недель;
- количество рабочих дней в неделю – 5 дней;
- количество рабочих часов в день – 5 часов.

Таким образом, общее время, затрачиваемое на выполнение работы, составляет $Т = 300$ часов.

Расчетные затраты на электроэнергию приведены в таблице 7.

Таблица 7 – Расчет затрат на электроэнергию

Составляющие	Обозначение	Значение
Мощность электрооборудования, кВт/ч	М	0,117
Время работы оборудования, часов	Т	300
Коэффициент использования мощности	$К_и$	0,9
Дневная цена 1 кВт, руб.	Ц	4,6
Затраты на электричество, руб	$З_э$	145,31

2.1.3 Заработная плата

Расчет заработной платы определяется основной заработной платой, дополнительной заработной платой и отчислениями на социальные нужды.

Расчет затрат на заработную плату осуществляется по следующему выражению:

$$З_{ЗП} = З_{ОСН} + З_{ДОП} + З_{СОЦ}, \quad (2.4)$$

где $З_{ОСН}$ – основная заработная плата;

$З_{ДОП}$ – дополнительная заработная плата;

$З_{СОЦ}$ – социальные отчисления.

Основная заработная плата – выплата за выполненную работу в соответствии с нормами труда.

Расчет основной заработной платы осуществляется по формуле:

$$З_{ОСН} = (ЗП_{ПР} + Д_{ТУ} + Н_{ПМ}) \cdot К_{р}, \quad (2.5)$$

где $ЗП_{ПР}$ – прямая заработная плата, $ЗП_{ПР} = Т_{ЧТС} \cdot Т_{Ф}$;

$Т_{ЧТС}$ – часовая тарифная ставка, руб/ч;

$Т_{Ф}$ – фактический фонд рабочего времени, ч;

$Д_{ТУ}$ – доплата за работы в тяжелых, вредных, особо тяжелых условиях.

Работа выполнялась в нормальных условиях труда, поэтому $Д_{ТУ} = 0$.

$Н_{ПМ}$ – надбавка за профессиональное мастерство, примем

$Н_{ПМ} = ЗП_{ПР} \cdot 0{,}12$;

$К_{р}$ – районный коэффициент, $К_{р} = 1$.

Дополнительная заработная плата – оплата за работу сверх установленной нормы, за успехи и за особые условия труда.

Дополнительная заработная плата составляет 20% от основной зарплаты: $З_{ДОП} = З_{ОСН} * 0{,}2$.

Отчисления на социальные нужды составляют 30% и определяются по формуле:

$$З_{СОЦ} = (З_{ОСН} + З_{ДОП}) \cdot 0{,}3 \quad (2.6)$$

Расчет заработной платы представлен в таблице 8.

Таблица 8 – Расчет заработной платы

Профессия	Часовая тарифная ставка, руб./ч	Фонд рабочего времени, ч	Основная з/п, руб	Доп. з/п, руб	Отчисления на соц. нужды, руб	Затраты, руб
Инженер	312	300	104832	20966,4	37739,52	163537,92
Руководитель	375	7%	8820	1764	3175,2	13759,2
Консультант по экономике	375	1%	1125	225	405	1755
Консультант по охране труда	375	1%	1125	225	405	1755
Затраты на ЗП (итого), руб.	180807,12					

2.1.4 Амортизационные отчисления

Амортизационные отчисления – отчисления части или полной стоимости основных фондов для возмещения их износа. Затраты на амортизационные отчисления вычисляются исходя из стоимости штатного оборудования, их годовых норм амортизации и времени их эксплуатации. Расчет производится по формуле:

$$З_{А} = \frac{С_{об} \cdot Н_{АМ} \cdot Т}{100\% \cdot 12}, \quad (2.7)$$

где $С_{об}$ – стоимость использовавшегося оборудования, руб;

$Н_{АМ}$ – годовые нормы амортизации, %;

Т – продолжительность работ, мес.

В процессе исследования использовался ноутбук стоимостью 30000 рублей.

Продолжительность работы на ПК составляет 300 часов. Годовая норма амортизации для компьютерного оборудования равна 33% (Постановление Правительства РФ от 1 января 2002 г. N 1, техника электронно-вычислительная, включая персональные компьютеры, относится ко второй амортизационной группе основных средств, куда включается имущество со сроком полезного использования свыше 2 лет до 3 лет включительно).

Определение затрат на амортизацию за 12 недель разработки ВКР приведено в таблице 9.

Таблица 9 – Расчет затрат на амортизацию

Составляющие	Обозначения	Значение
Стоимость оборудования, руб	$С_{об}$	30000
Годовая норма амортизации, %	$Н_{АМ}$	33
Продолжительность работ, мес.	Т	3
Затраты на амортизацию (итого), руб	$З_{А}$	2475

Суммируя вышеперечисленные затраты, получаем прямые расходы на ВКР. В таблице 10 сведены прямые затраты.

Таблица 10 – Расчет прямых расходов

Затраты	Обозначения	Значения, руб
Затраты на материалы	$З_М$	553,3
Затраты на электроэнергию	$З_Э$	145,31
Затраты на заработную плату	$З_{ЗП}$	180807,12
Затраты на амортизацию	$З_А$	2475
Прямые расходы	$З_{ПР}$	183980,73

2.2 Косвенные расходы

Косвенные расходы – это те затраты, которые невозможно отнести непосредственно к себестоимости продукта, но они связаны с поддержанием производственной деятельности в целом.

Перечень косвенных расходов может включать в себя:

– аренду и ремонт цеха;

– коммерческие издержки на реализацию продукции;

– коммерческие издержки на проведение инвентаризации;

– содержание административно-управленческого персонала;

– отопление, освещение, содержание в чистоте и ремонт зданий;

– затраты непроизводственного характера (ущерб от утери или порчи имущества и др.).

Косвенные расходы примем как 25% от суммы прямых расходов:

$$З_{НР} = 183980{,}73 \cdot 0{,}25 = 45995{,}18 \text{ руб.}$$

2.3 Затраты на разработку ВКР

Затратами на выполняемую работу является сумма прямых и косвенных расходов:

$$З_{ВКР} = З_{ПР} + З_{НР} \quad (2.8)$$

Расходы на выполнение работ за весь период разработки ВКР представлены в таблице 11.

Таблица 11 – Смета расходов на разработку ВКР

Затраты	Обозначения	Значения, руб
Прямые расходы	$З_{ПР}$	183980,73
Косвенные расходы	$З_{НР}$	45995,18
Расходы на работу	$З_{ВКР}$	229975,91

3 Раздел безопасности жизнедеятельности и экологии

Безопасность жизнедеятельности (БЖД) представляет собой систему организационных мероприятий и технических средств, предотвращающих или уменьшающих вероятность воздействия на работающих опасных и вредных производственных факторов (ОВПФ), возникающих в процессе трудовой деятельности.

Главная задача производственной безопасности – создание здоровых, безопасных и высоко производительных условий труда.

Условия труда – это совокупность факторов трудового процесса и производственной среды, в которой осуществляется деятельность человека. Они существенно влияют на состояние здоровья работающих, а также на производительность труда и, как следствие, на экономические показатели предприятий.

3.1 Анализ ОВПФ на рабочем месте программиста

Все опасные и вредные производственные факторы в соответствии с ГОСТ 12.0.003-74 по природе своего действия делятся на несколько групп: физические, химические, биологические и психофизиологические.

К физическим факторам относят:

- повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека;
- повышенное давление паров или газов в сосудах;
- повышенные уровни шума, вибрации, инфразвука и ультразвука;
- недостаточную освещенность рабочей зоны;
- повышенный уровень электромагнитных излучений и др.

Химические факторы – это вредные для организма человека вещества, подразделяющиеся по характеру воздействия (токсические, раздражающие, канцерогенные, мутагенные и др.) и по пути проникновения в организм

(органы дыхания, кожные покровы и слизистые оболочки, желудочно-кишечный тракт).

Биологические факторы – это патогенные микроорганизмы и продукты их жизнедеятельности.

Психофизиологические факторы – это статические и динамические перегрузки и нервно-психические перегрузки (умственное перенапряжение, перенапряжение анализаторов, монотонность труда, эмоциональные перегрузки).

Среди ОВПФ на рабочем месте программиста можно выделить следующие факторы:

1) Недостаточное освещение рабочего места:

Слабое освещение приводит к напряжению глаз, что при длительном воздействии ведет к ухудшению зрения.

2) Повышенная или пониженная температура воздуха рабочей зоны:

Отклонение температуры воздуха рабочей зоны от нормы может приводить к ухудшению самочувствия и заболеваниям.

3) Повышенный уровень электромагнитных излучений:

На рабочем месте источником электромагнитного излучения является монитор компьютера. Нахождение источника в непосредственной близости от человека при длительном воздействии вредит зрению.

4) Повышенное значение напряжения в электрической цепи, замыкание которой может произойти через тело человека:

Соприкосновение работающего с токоведущими частями оборудования может привести к поражению электрическим током.

5) Повышенный уровень шума на рабочем месте:

Воздействие шума, создаваемого компьютерным оборудованием, а также внешними источниками, может оказывать негативное влияние на органы слуха и на психологическое состояние человека.

3.2 Меры по снижению воздействия ОВПФ

Сведем вышеперечисленные ОВПФ в таблицу 12, рассмотрим допустимые нормы, а также меры по снижению воздействия опасных и вредных производственных факторов.

Таблица 12 – Меры по снижению воздействия ОВПФ на рабочем месте

<table>
<tr><th>Вид фактора</th><th>Нормативный документ</th><th colspan="5">Норма</th><th>Мероприятие по нормированию</th></tr>
<tr><td>1.Недостаточное освещение</td><td>СНиП 23-05-95</td><td colspan="5">$E_н$ = 300 лк</td><td>Использование дополнительного освещения</td></tr>
<tr><td rowspan="3">2.Повышенная или пониженная температура (T, ºC), влажность (φ, %) и подвижность воздуха (v, м/с)</td><td rowspan="3">СанПиН 2.2.4.548-96</td><td>Период года</td><td>T, ºC Не более</td><td colspan="2">φ, %</td><td>v, м/с</td><td rowspan="3">Установка систем вентиляции, кондиционирования и отопления</td></tr>
<tr><td>Холодный</td><td>22…24</td><td colspan="2">40…60</td><td>0.1</td></tr>
<tr><td>Тёплый</td><td>23…25</td><td colspan="2">40…60</td><td>0.1</td></tr>
<tr><td rowspan="4">3.Повышенный уровень электромагнитных изучений</td><td rowspan="4">ГОСТ 12.1.006-84</td><td rowspan="2">Доп. напряженность, В/м</td><td colspan="4">Предельные значение в диапазонах частот, МГц</td><td rowspan="4">Установка защитных экранов</td></tr>
<tr><td>0.06÷3</td><td>3÷30</td><td>30÷300</td></tr>
<tr><td>Электрического поля</td><td>500</td><td>300</td><td>80</td></tr>
<tr><td>Магнитного поля</td><td>20000</td><td>7000</td><td>800</td></tr>
<tr><td>4.Повышенное значение напряжения в электрической цепи</td><td>ГОСТ 32144-2013</td><td colspan="5">U не более 220 В, I=0.3 mA, (τ не более 10 мин)</td><td>Защитное заземление, изоляция токоведущих частей</td></tr>
</table>

Продолжение таблицы 12

<table>
<tr><td rowspan="3">5.Шум</td><td rowspan="3">ГОСТ 12.1.003-83</td><td colspan="10">Уровни звукового давления (дБ) в октавных полосах со среднегеометрическими частотами (Гц)</td><td>УЗ, дБА</td><td rowspan="3">Звукопоглощающее покрытие помещения, установка звукопоглощающих перегородок</td></tr>
<tr><td>Частота, Гц</td><td>31.5</td><td>63</td><td>125</td><td>250</td><td>500</td><td>1000</td><td>2000</td><td>4000</td><td>8000</td><td rowspan="2">50</td></tr>
<tr><td>УЗД, дБ</td><td>86</td><td>71</td><td>61</td><td>54</td><td>49</td><td>45</td><td>42</td><td>40</td><td>38</td></tr>
</table>

3.3 Расчет освещенности рабочего места

В помещении, где находится рабочее место программиста, будем использовать искусственное общее освещение. Расчет осуществляется по методу светового потока с учетом потока, отраженного от пола, стен и потолка помещения.

Рассчитаем количество ламп N в помещении длиной 6 м, шириной 4 м и высотой 3 м:

$$N = \frac{E_{\text{н(общ)}} \cdot K_{\text{з}} \cdot S \cdot Z}{\Phi_{\text{л}} \cdot U_{oy}}, \quad (3.1)$$

где $E_{\text{н(общ)}}$– нормируемая минимальная освещенность;

Z – коэффициент минимальной освещенности;

S – освещаемая площадь;

$K_{\text{з}}$ – коэффициент запаса;

$\Phi_{\text{л}}$ – световой поток лампы;

U_{oy} – коэффициент использования светового потока.

Работа программиста относится к III разряду зрительной работы (высокой точности, так как наименьший размер объекта различения равен 0.5 мм). Подразряд зрительной работы – “В”, так как контраст объекта различения с фоном – “большой”, характеристика фона – “светлый”.

Таким образом, по таблице 1 СНиП 23-05-95 находим $E_{н(общ)} = 300$ лк.

Коэффициент минимальной освещенности учитывает неравномерность освещения помещения. Для люминесцентных ламп Z = 1.1.

Освещаемая площадь S = 24 м^2.

Коэффициент запаса учитывает старение ламп и загрязнение светильников. Зависит от типа помещения и характера проводимых в нем работ, в нашем случае $K_з$ = 1.5.

Будем использовать люминисцентные лампы ЛБ-40 со значением светового потока $Ф_л$ = 2800 лм.

Коэффициент использования светового потока U_{oy} зависит от индекса помещения $i_п$ и коэффициентов отражения потока от потолка, пола и стен помещения.

Индекс помещения:

$$i_п = \frac{A \cdot B}{h_р(A + B)}, \qquad (3.2)$$

где А и В – длина и ширина помещения;

$h_р$ – высота светильников над рабочей поверхностью.

Определим высоту светильников над рабочей поверхностью.

Высота помещения Н = 3 м, рекомендуемая высота подвеса светильников $h_п$ = 2 м, тогда расстояние светильников от перекрытия $h_с$ = 1 м. Высоту рабочей поверхности примем равной $h_{рп}$ = 0.8 м, тогда высота светильников над рабочей поверхностью:

$$h_р = Н - h_с - h_{рп} = 3 - 1 - 0.8 = 1.2 \text{ м}$$

Рассчитаем индекс помещения:

$$i_{\text{п}} = \frac{6 \cdot 4}{1.2 * (6 + 4)} = 2$$

Коэффициенты отражения потока от потолка, пола и стен помещения соответственно равны 0.7, 0.3 и 0.5 (белая поверхность потолка, серая поверхность пола и светлая поверхность стен).

По таблице коэффициента использования светового потока в зависимости от индекса помещения и коэффициентов отражения, найдем $U_{oy} = 0.62$.

Рассчитаем теперь количество ламп N:

$$N = \frac{300 \cdot 1.5 \cdot 24 \cdot 1.1}{2800 \cdot 0.62} \approx 7 \text{ шт.}$$

Будем использовать 4 светильника, по две лампы в каждом.

Расстояние между соседними светильниками λ = 2 м для люминесцентных ламп с равномерным светораспределением; расстояние между крайними светильниками и стенами l = 0.8 м; расстояние между рядами светильников L = 2.3 м.

Полученные схемы размещения светильников по высоте и расположению в помещении представлены на рисунках 70, 71.

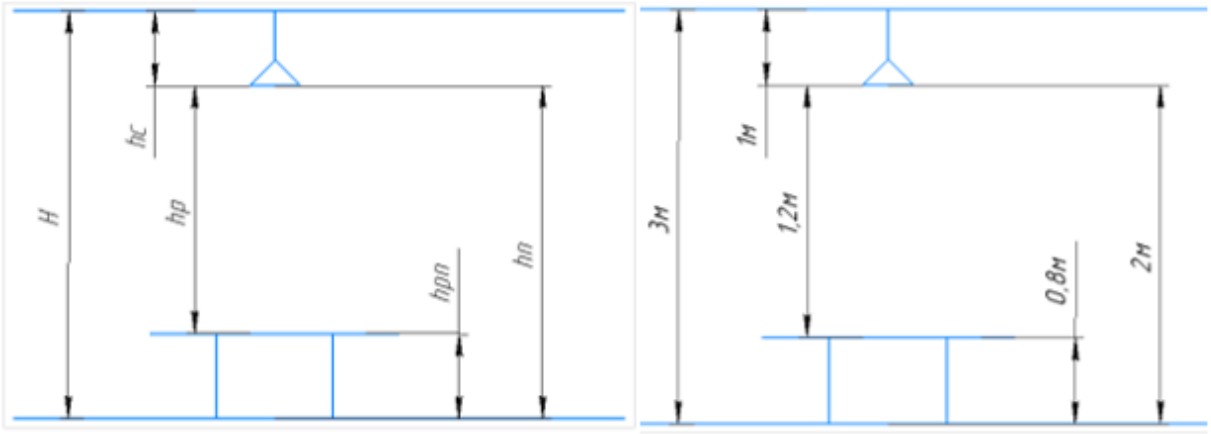

Рисунок 70 – Схема размещения светильников по высоте

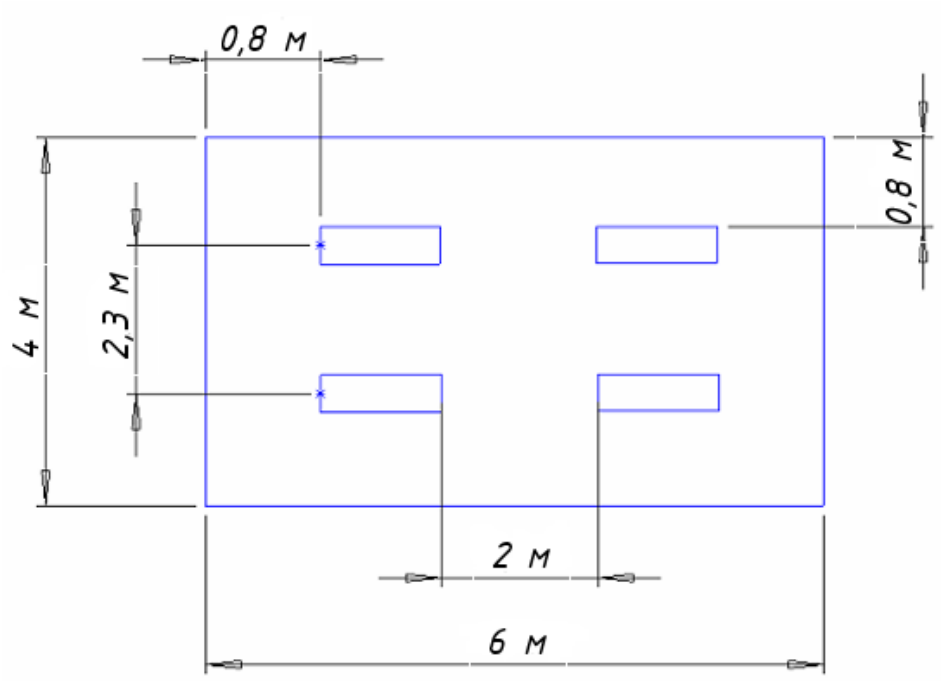

Рисунок 71 – Схема расположения светильников в помещении

3.4 Мероприятия по охране труда

Охрана труда – это мероприятия, позволяющие сохранить жизнь и здоровье работников в процессе профессиональной трудовой деятельности.

В соответствии с требованиями по охране труда на рабочем месте программиста, работник обязан проходить повторные инструктажи на рабочем месте – не реже 1 раза в 6 месяцев, периодический медосмотр – в соответствии с Приказом Минздрава № 302н, очередную проверку знаний требований охраны труда – не реже 1 раза в год.

Работник обязан:

– соблюдать правила внутреннего трудового распорядка;

– соблюдать требования инструкции по охране труда, инструкции о мерах пожарной безопасности, инструкции по электробезопасности;

– соблюдать правила личной гигиены;

– извещать своего руководителя о любой ситуации, угрожающей жизни и здоровью людей, о каждом несчастном случае, происшедшем на производстве, об ухудшении состояния своего здоровья;

– уметь оказывать первую помощь пострадавшему, знать место нахождения аптечки и средств пожаротушения, а также уметь ими пользоваться.

На рабочем месте монитор должен находиться на расстоянии 50–70 см от глаз и иметь антибликовое покрытие, которое также должно обеспечивать снятие электростатического заряда с поверхности экрана, исключать искрение и накопление пыли.

Нельзя загораживать заднюю стенку системного блока или ставить его вентиляционными отверстиями вплотную к стене, это ведет к нарушению охлаждения компонентов системного блока и его перегреву.

Режим работы и отдыха работника должен зависеть от характера выполняемой работы. При вводе данных, редактировании программ, считывании информации с экрана непрерывная продолжительность работы с ПК не должна превышать 4 часа за рабочий день при 8-ми часовом рабочем дне. Через каждый час работы необходимо делать перерывы на отдых по 5–10 минут или каждые два часа по 15–20 минут. Для снятия общего утомления, в перерывах необходимо проводить упражнения, улучшающие состояние нервной, сердечно-сосудистой и дыхательной систем, улучшающие кровообращение, снижающие мышечное напряжение и утомление глаз.

ЗАКЛЮЧЕНИЕ

Таким образом, в дипломной работе был исследован процесс оценивания параметров движения летательного аппарата по схеме компенсации, с использованием объединения данных бесплатформенной инерциальной и спутниковой навигационных систем, путем нахождения оценок ошибок БИНС и их вычитания из формируемых БИНС параметров, содержащих искомую ошибку.

В ходе выполнения работы разработана модель навигационной системы летательного аппарата, состоящей из БИНС, СНС и дискретного фильтра Калмана.

Разработано программное обеспечение, моделирующее работу комплексной системы навигации в режиме реального времени, с возможностью имитации отсутствия сигналов спутниковых навигационных систем и работы в режиме экстраполяции, без корректировок оценки сигналами измерения СНС.

Написанное программное обеспечение может быть использовано в тренажерных технических средствах, где требуется моделирование работы навигационной системы летательного аппарата. Также в перспективе имитаторы могут быть заменены на реальные системы навигации.

Исследовано влияние аддитивных розового и винеровского шумов в составе сигналов инерциальных чувствительных элементов, влияние частоты выдачи и отсутствия сигналов спутниковой навигационной системы на получаемую оценку.

Анализ результатов показывает, что дискретный фильтр Калмана позволяет оценить вектор состояния рассматриваемой системы по имеющимся зашумленным измерениям. При этом оценивается весь вектор состояния, несмотря на то, что измеряется он не полностью.

Также нужно отметить, что по мере увеличения частоты выдачи сигнала спутниковой навигационной системы, оценка становится более

точной. Это можно объяснить тем, что фильтр в таком случае успевает совершить больше итераций оценивания за то же самое время.

При пропадании сигнала измерения СНС, формируемая экстраполированная оценка дает недостоверные координаты при имеющихся возмущениях системы, тем не менее, такая оценка точнее параметров, выдаваемых БИНС. С поступлением сигнала СНС происходит корректировка, и точность оценки восстанавливается.

С учетом аддитивных погрешностей, не являющихся белым шумом, в составе сигналов инерциальных датчиков, появляется несоответствие априорных данных о шумах системы, заложенных в фильтр в виде матрицы ковариации возмущений, истинным шумам, в результате чего фильтр расходится.

Увеличение элементов на главной диагонали матрицы ковариации возмущений позволяет учесть дополнительные шумы объекта, и, таким образом, предотвратить расходимость фильтра.

В технико-экономическом разделе рассчитана себестоимость разработки выпускной квалификационной работы. Себестоимость составила 229 тыс. 975 руб. 91 коп.

В разделе безопасности жизнедеятельности и экологии произведен анализ опасных и вредных производственных факторов на рабочем месте программиста, приведены меры по снижению их воздействия на работающего; определены мероприятия по охране труда; произведен расчет освещенности рабочего места, – в соответствии с результатами расчета, для комфортной работы в качестве осветительных приборов в помещении необходимо использовать 4 светильника, по 2 люминисцентные лампы ЛБ-40 в каждом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы построения бесплатформенных инерциальных систем / В.В.Матвеев, В.Я.Распопов – СПб.: ГНЦ РФ ОАО "Концерн "ЦНИИ Электроприбор", 2009. – 280 с.
2. Теория автоматического управления: Учеб. для вузов. Ч. II. Теория нелинейных и специальных систем автоматического управления. / А. А. Воронов, Д. П. Ким, В. М. Лохин и др. – М.: Высш. шк., 1986. – 504 с.
3. Strapdown Analytics, Part 2 / Paul G. Savage, Strapdown Associates, Inc., Maple Plain, Minnesota, 2000.
4. Piaggio P.180 Avanti [Электронный ресурс]. – URL: http://www.avantievo.piaggioaerospace.it/ (дата обращения 01.04.2022)
5. Основы траекторной обработки радиолокационной информации. Ч. 2 / А. А. Коновалов – СПб.: СПбГЭТУ “ЛЭТИ”, 2014. – 180 с.
6. Kalman Filtering. Theory and Practice with MATLAB, 4th Edition / Grewal M. S., Andrews A. P., 2015.
7. Understanding Kalman Filters [Электронный ресурс]. – URL: https://www.mathworks.com/videos/series/understanding-kalman-filters.html (дата обращения 17.04.2022)
8. How a Kalman filter works, in pictures [Электронный ресурс]. – URL: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/ (дата обращения 18.04.2022)
9. Беспроводные технологии на автомобильном транспорте. Глобальная навигация и определение местоположения транспортных средств. Учебное пособие / В.М. Власов, Б.Я. Мактас – М.: Инфра-М, 2017 – 184 с.
10. Технико-экономическое обоснование дипломных проектов: учеб. пособие для втузов / Л.А. Астреина, В.В. Балдесов, В.К. Беклешов; под ред. В.К. Беклешова – М.: Высш. шк., 1991. – 175 с.

11. Экономическое обоснование дипломных проектов: методические указания / С. А. Манжинский, А. С. Соболевский. – Минск: БГТУ, 2010. – 47 с.
12. Безопасность жизнедеятельности. Кн.1. Коллективные средства защиты: Справ. пособие по дипломному проектированию. Изд. 2-е, испр. и доп. / Под ред. Н.И. Иванова и И.М. Фадина; БГТУ – СПб., 2003.
13. Основы эргономики и безопасность труда: учеб. пособие / Н.И. Чепелев, С.Н. Орловский, А.Ю. Щекин. – Красноярск, 2018. – 253 с.
14. ГОСТ Р 55710-2013. Освещение рабочих мест внутри зданий. Нормы и методы измерений. – М.: Стандартинформ, 2016.
15. СНиП 23-05-95. Строительные нормы и правила Российской Федерации. Естественное и искусственное освещение. – М.: Госстрой России, ГУП ЦПП, 2003.

ПРИЛОЖЕНИЕ А

Скрипт модели в среде Matlab

```
clear; clc; close all;

%% Инициализация
h = 0.1;            % Шаг выдачи данных БИНС
h_SNS = 0.2;        % Шаг выдачи данных  СНС
k = h_SNS / h;
t_end = 40;         % Время окончания моделирования

n = t_end / h + 1;  % Количество итераций БИНС в процессе моделирования
t = 0 : h : t_end;  % Вектор времени БИНС

n_SNS = t_end / h_SNS + 1; % Количество итераций СНС в процессе моделирования
t_SNS = 0 : h_SNS : t_end; % Вектор времени  СНС

g = 9.8;            % Гравитационная постоянная
R_earth = 6371000; % Радиус Земли
r = R_earth + 7000;% Высота полета ЛА = 7 км

% СКО случайных ошибок  БИНС
sig_gyro   = 0.02; % СКО гироскопа
sig_acc_X  = 0.1;  % СКО акселерометра X
sig_acc_Y  = 0.1;  % СКО акселерометра Y

% СКО ошибок СНС
sig_nav_Vxg = 0.2;
sig_nav_S = 6;
sig_nav_Vyg = 0.3;
sig_nav_h = 10;

% Матрица наблюдения
H = [0 1 0 0 0;
     0 0 1 0 0;
     0 0 0 1 0;
     0 0 0 0 1];

% Матрица шума системы
Q = diag( [sig_gyro^2 sig_acc_X^2 sig_acc_Y^2] );

% Матрица шума измерения
R = diag( [ sig_nav_Vxg^2 sig_nav_S^2 sig_nav_Vyg^2 sig_nav_h^2 ] );

% Начальные усл. для интегрирования системы
inits = [  1 * pi/180;  0;  0;  0;  0  ];

% Начальное значение вектора состояния
X_real(:, 1) = inits;
X_real_t_SNS(:, 1) = X_real(:, 1);

% Начальное значение вектора измерений
delta_SNS(:, 1) = [ normrnd(0, sig_nav_Vxg);
                    normrnd(0, sig_nav_S);
                    normrnd(0, sig_nav_Vyg);
                    normrnd(0, sig_nav_h) ];
z(:, 1) = H * X_real - delta_SNS(:, 1);

% Начальное значение вектора оптимальных оценок
```

```
X_est(:, 1) = zeros(1, 5);

% Начальное значение ошибки оценки
error(:, 1) = X_real_t_SNS(:, 1) - X_est(:, 1);

% Матрица ковариации
P = zeros(5);
P(1, 1) = sig_gyro^2;
P(2, 2) = sig_acc_X^2;
P(3, 3) = 2^2;
P(4, 4) = sig_acc_Y^2;
P(5, 5) = 2^2;

% Матрица состояния
A = [
    0 -1/r 0 0 0;
    g   0  0 0 0;
    0   1  0 0 0;
    0   0  0 0 0;
    0   0  0 1 0;
    ];

% Матрица возмущения
G = [
    -1 0 0;
     0 1 0;
     0 0 0;
     0 0 1;
     0 0 0;
    ];

% Координаты ЛА в нач. момент времени
Vxg(1) = 205;
S(1) = 0;
fi(1) = 0;
theta(1) = 3 * pi/180;
Vyg(1) = 0;
height(1) = 7000;

j = 2; % счетчик измерений СНС
%% Решение
for i = 2 : 1 : n
    %% Переходные матрицы для системы БИНС
    F = eye(5) + A*h;
    GAMMA = G*h;
    %% Нахождение реального вектора состояния
    w = [  normrnd( 0, sig_gyro);
           normrnd( 0, sig_acc_X );
           normrnd( 0, sig_acc_Y );  ];
    X_real(:, i) = F * X_real(:, i-1) + GAMMA * w;

    %% Каждый k-й сигнал БИНС сопровождается сигналом измерения СНС
    if mod(i, k) == 0
        %% Координаты ЛА в текущий момент времени
        Vxg(j) = 205;
        S(j) = Vxg(j)   * t_SNS(j);
        fi(j) = Vxg(j)/r * t_SNS(j);
        theta(j) = 3 * pi/180;
        Vyg(j) = 0;
        height(j) = 7000;
```

```
        %% Сохранение вектора состояния для моментов времени измерений СНС
        X_real_t_SNS(:, j) = X_real(:, i);

        %% Переходные матрицы для системы СНС
        F = eye(5) + A*h_SNS;
        GAMMA = G*h_SNS;

        %% Нахождение экстраполированной оценки вектора состояния
        X_extrap = F * X_est(:, j-1);

        %% Моделирование текущих измерений
        delta_SNS(:, j) = [ normrnd(0, sig_nav_Vxg);
                            normrnd(0, sig_nav_S);
                            normrnd(0, sig_nav_Vyg);
                            normrnd(0, sig_nav_h) ];
        z(:, j) = H * X_real_t_SNS(:, j) - delta_SNS(:, j);

        %% Нахождение опт. оценки вектора состояния и ковариационной матрицы
P
        P_extrap = F * P * F' + GAMMA * Q * GAMMA';

        v = z(:, j) - H * X_extrap;

        S_matrix = H * P_extrap * H' + R;

        K = P_extrap * H' * S_matrix^(-1);

        % Опт. оценка
        X_est(:, j) = X_extrap + K * v;

        % Ковариационная матрица
        P = P_extrap - K * H * P_extrap;

        % Ошибка оценки
        error(:, j) = X_real_t_SNS(:, j) - X_est(:, j);

        j = j + 1;
    end
end

%% Графики переменных состояния во времени
figure;
hold, grid on
title('\beta(t)');
ylabel('\beta, rad');
xlabel('t, s');
plot( t, X_real(1, :), 'g', 'DisplayName', 'реальная'  );
plot( t_SNS,  X_est(1, :), 'b', 'DisplayName', 'оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\DeltaV_{Xg}(t)');
ylabel('\DeltaV_{Xg}, m/s');
xlabel('t, s');
plot( t_SNS, X_real_t_SNS(2, :) - delta_SNS(1, :), 'c', 'DisplayName',
'измерение \DeltaV_{БИНС} - \DeltaV_{СНС}'  );
plot( t, X_real(2, :), 'g', 'DisplayName', 'реальная'  );
plot( t_SNS,  X_est(2, :), 'b', 'DisplayName', 'оценка' );
```

```
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\DeltaS(t)');
ylabel('\DeltaS, m');
xlabel('t, s');
plot( t_SNS, X_real_t_SNS(3, :) - delta_SNS(2, :), 'c', 'DisplayName',
'измерение \DeltaS_{БИНС} - \DeltaS_{СНС}'  );
plot( t, X_real(3, :), 'g', 'DisplayName', 'реальная'  );
plot( t_SNS,  X_est(3, :), 'b', 'DisplayName', 'оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\DeltaV_{Yg}(t)');
ylabel('\DeltaV_{Yg}, m/s');
xlabel('t, s');
plot( t_SNS, X_real_t_SNS(4, :) - delta_SNS(3, :), 'c', 'DisplayName',
'измерение \DeltaV_{БИНС} - \DeltaV_{СНС}'  );
plot( t, X_real(4, :), 'g', 'DisplayName', 'реальная'  );
plot( t_SNS,  X_est(4, :), 'b', 'DisplayName', 'оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\Deltah(t)');
ylabel('\Deltah, m');
xlabel('t, s');
plot( t_SNS, X_real_t_SNS(5, :) - delta_SNS(4, :), 'c', 'DisplayName',
'измерение \Deltah_{БИНС} - \Deltah_{СНС}'  );
plot( t, X_real(5, :), 'g', 'DisplayName', 'реальная'  );
plot( t_SNS,  X_est(5, :), 'b', 'DisplayName', 'оценка' );
legend('Location', 'Best');
legend show

%% Статистические параметры ошибок оценок
for i = 1 : 5
    M(i) = mean( error(i,:) );
    D(i) = sqrt( var(error(i,:)) );
end
M
D

%% Координаты

% Реальные параметры
coords = [ Vxg; S; fi; theta; Vyg; height ];

% Ошибки БИНС
dVxg = X_real_t_SNS(2, :);
dS = X_real_t_SNS(3, :);
dfi = dS / r;
dtheta = -X_real_t_SNS(1, :);
dVyg = X_real_t_SNS(4, :);
dh = X_real_t_SNS(5, :);
% Показания БИНС
coords_INS = coords + [ dVxg; dS; dfi; dtheta; dVyg; dh ];
```

```
% Оценки ошибок БИНС
dV_est = X_est(2, :);
dS_est = X_est(3, :);
dfi_est = dS_est / r;
dtheta_est = -X_est(1, :);
dVyg_est = X_est(4, :);
dh_est = X_est(5, :);
% Оценки параметров
coords_est = coords_INS - [ dV_est; dS_est; dfi_est; dtheta_est; dVyg_est;
dh_est ];

%% Графики координат
figure;
hold, grid on
title('V_{Xg}(t)');
ylabel('V_{Xg}, m/s');
xlabel('t, s');
plot( t_SNS, coords(1, :) + delta_SNS(1, :), 'c', 'DisplayName', 'СНС' );
plot( t_SNS, coords_INS(1, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(1, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(1, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('S(t)');
ylabel('S, m');
xlabel('t, s');
plot( t_SNS, coords(2, :) + delta_SNS(2, :), 'c', 'DisplayName', 'СНС' );
plot( t_SNS, coords_INS(2, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(2, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(2, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\phi(t)');
ylabel('\phi, rad');
xlabel('t, s');
plot( t_SNS, coords_INS(3, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(3, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(3, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('\vartheta(t)');
ylabel('\vartheta, rad');
xlabel('t, s');
plot( t_SNS, coords_INS(4, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(4, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(4, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('V_{Yg}(t)');
```

```
ylabel('V_{Yg}, m/s');
xlabel('t, s');
plot( t_SNS, coords(5, :) + delta_SNS(3, :), 'c', 'DisplayName', 'СНС' );
plot( t_SNS, coords_INS(5, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(5, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(5, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show

figure;
hold, grid on
title('h(t)');
ylabel('h, m');
xlabel('t, s');
plot( t_SNS, coords(6, :) + delta_SNS(4, :), 'c', 'DisplayName', 'СНС' );
plot( t_SNS, coords_INS(6, :), 'm', 'DisplayName', 'БИНС' );
plot( t_SNS, coords(6, :), 'g', 'DisplayName', 'Реальная'  );
plot( t_SNS, coords_est(6, :), 'b', 'DisplayName', 'Оценка' );
legend('Location', 'Best');
legend show
```

ПРИЛОЖЕНИЕ Б

Результаты моделирования при частоте 2 Гц сигнала СНС

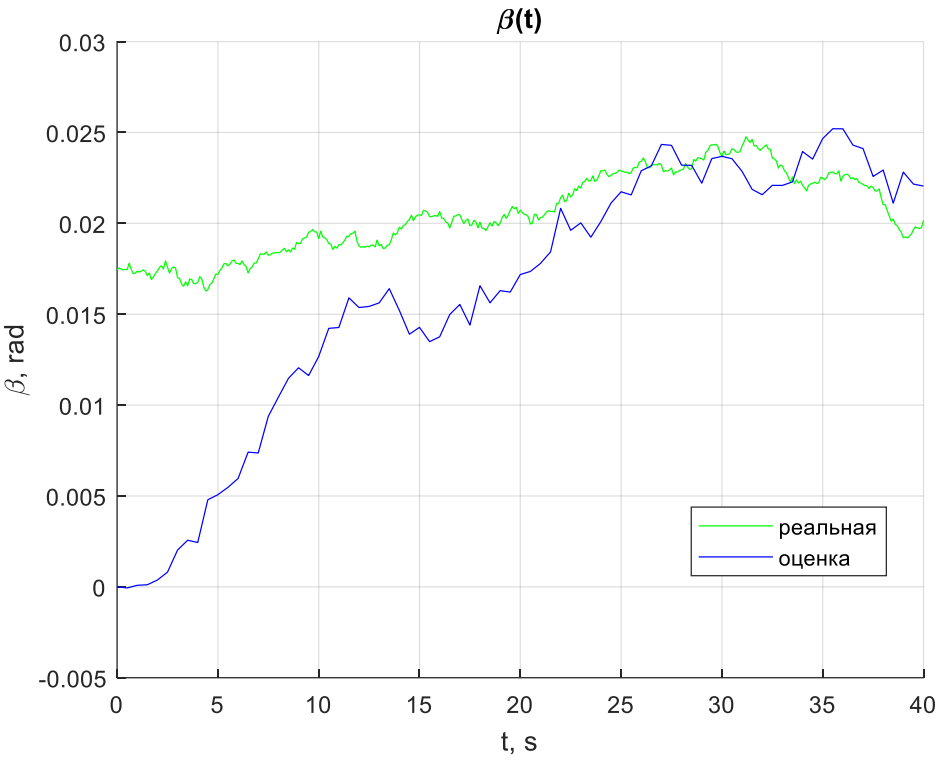

Рисунок Б.1 – Ошибка построения вертикали

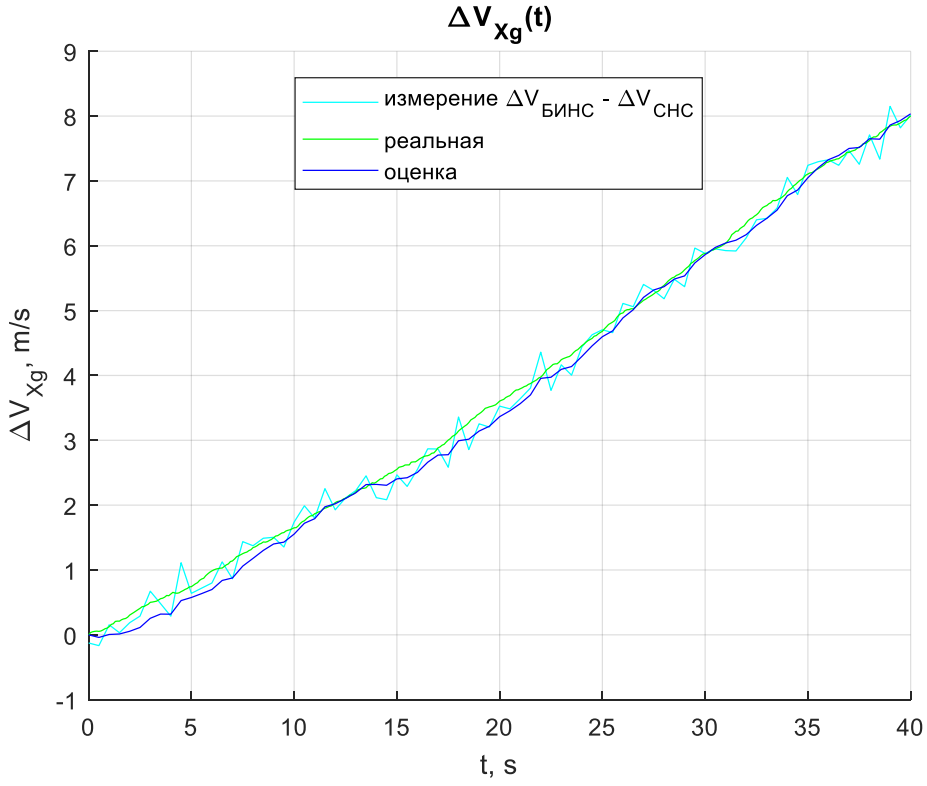

Рисунок Б.2 – Ошибка выработки горизонтальной скорости

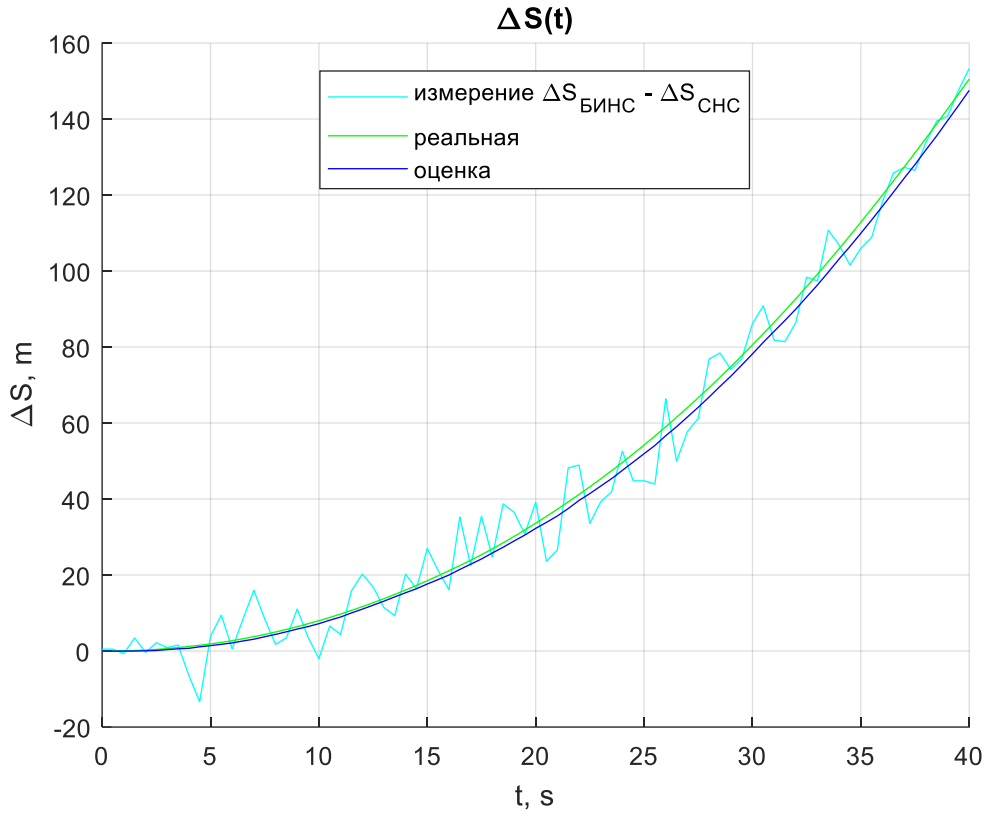

Рисунок Б.3 – Ошибка выработки пройденного расстояния

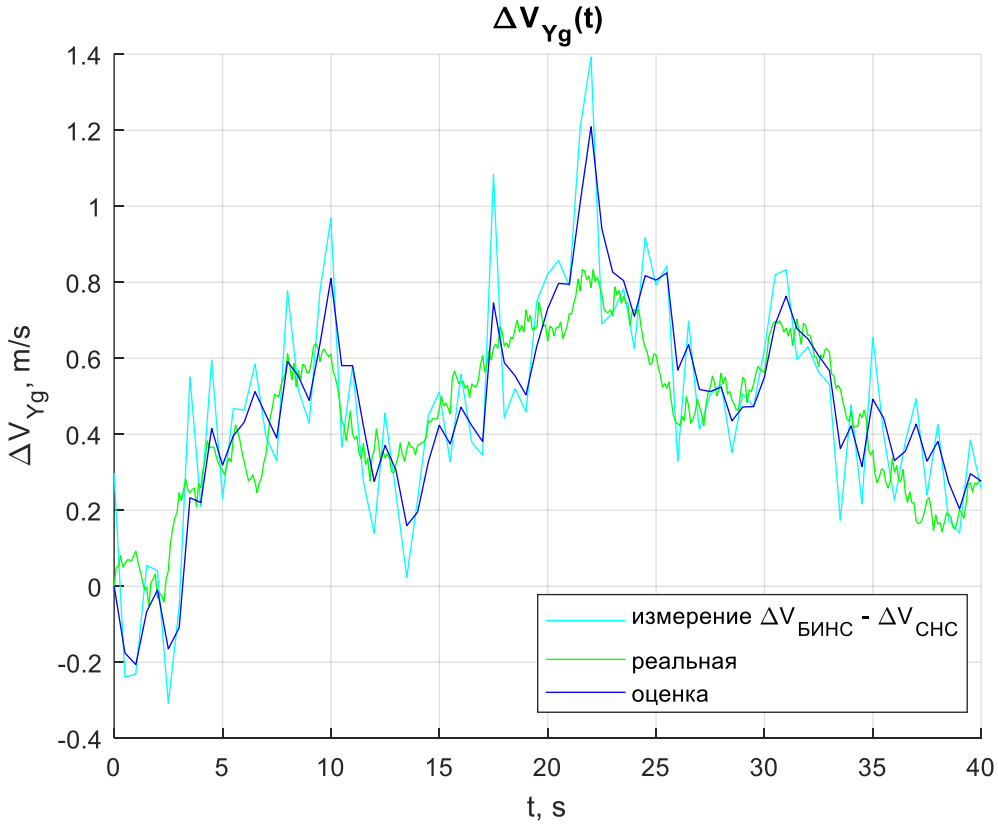

Рисунок Б.4 – Ошибка выработки вертикальной скорости

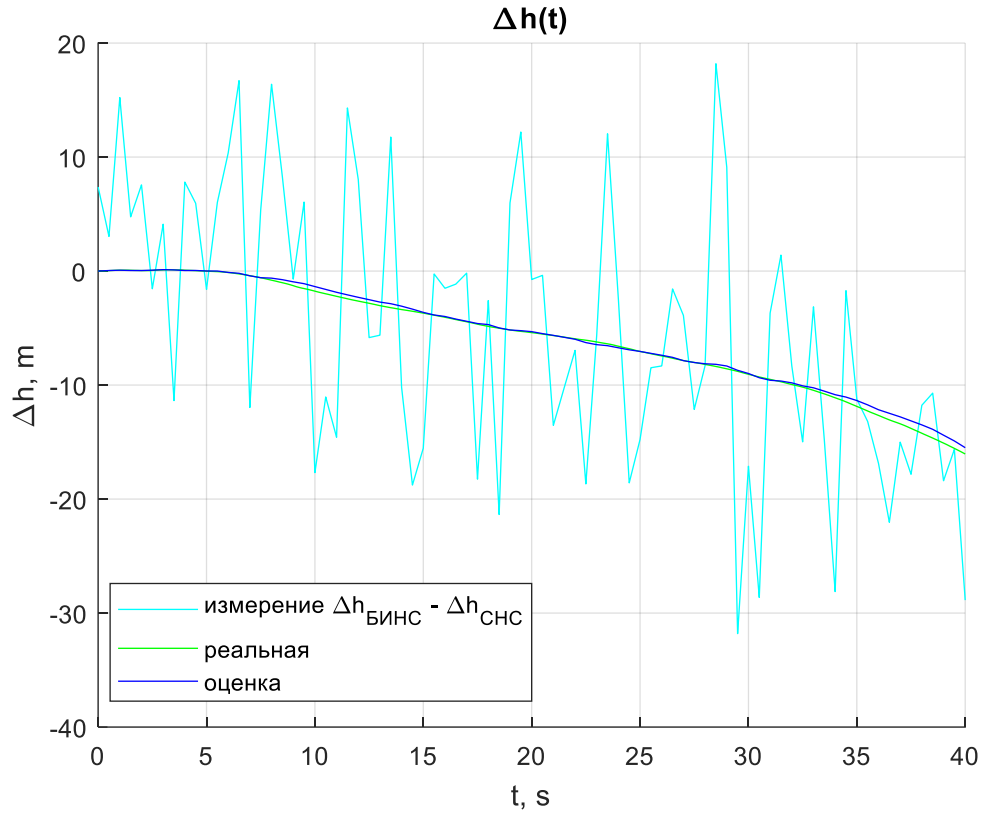

Рисунок Б.5 – Ошибка выработки высоты

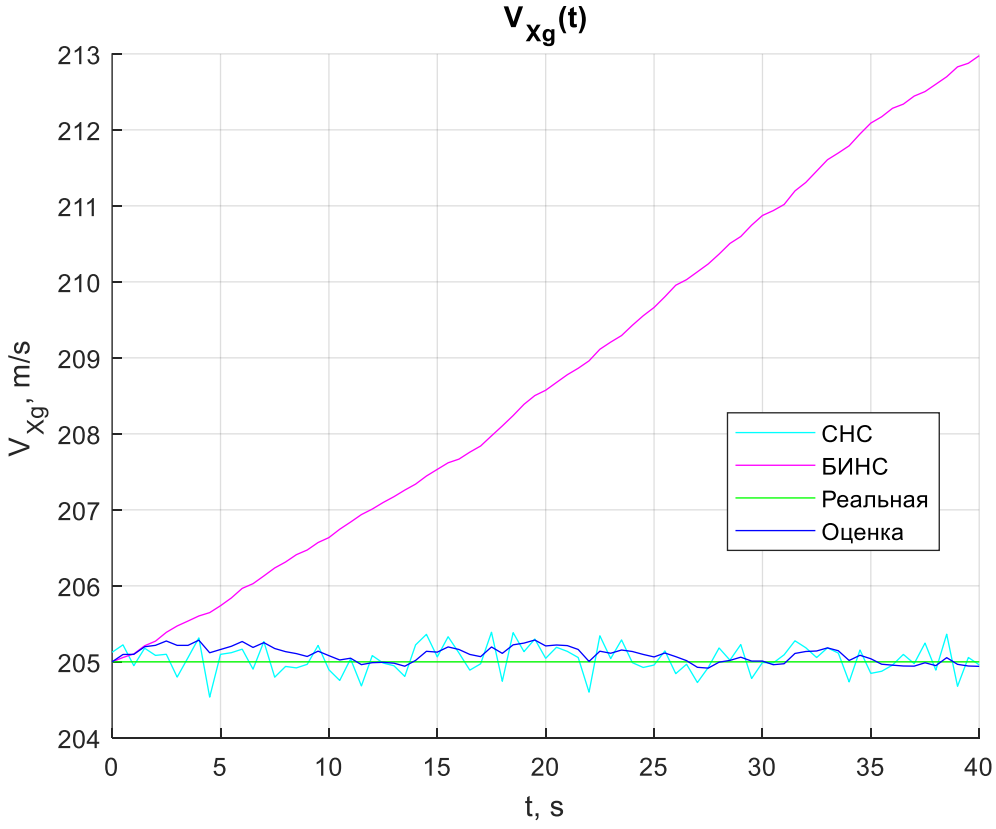

Рисунок Б.6 – Горизонтальная скорость

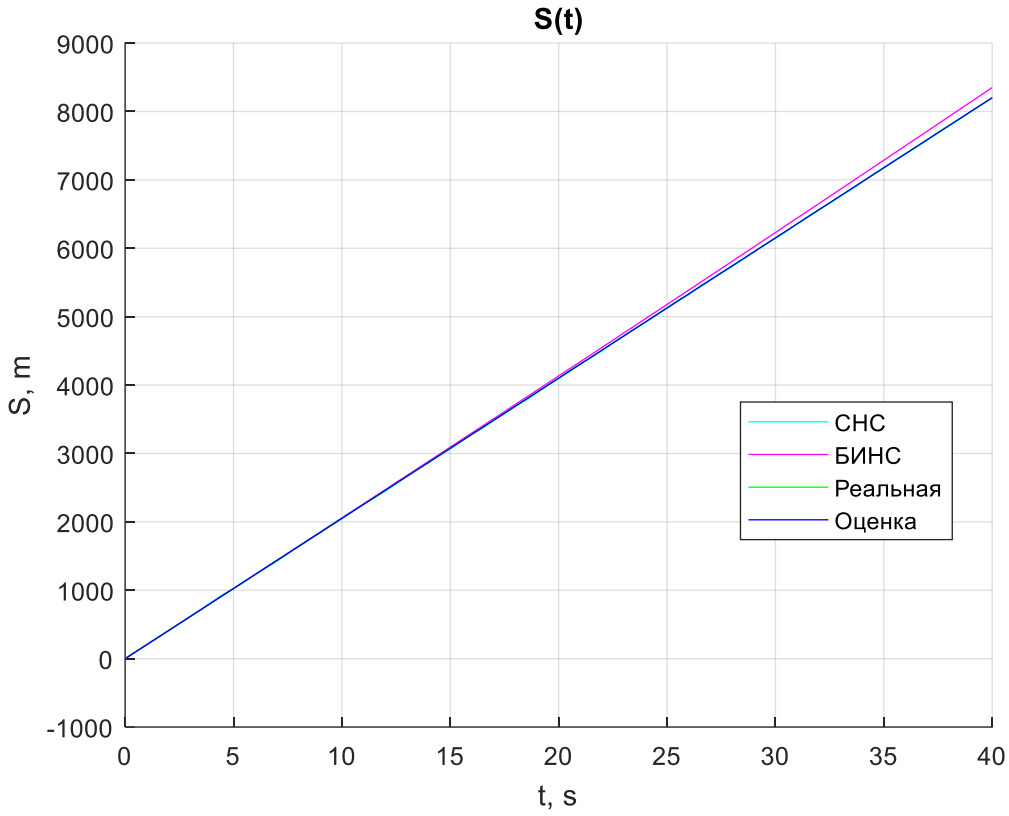

Рисунок Б.7 – Пройденный путь

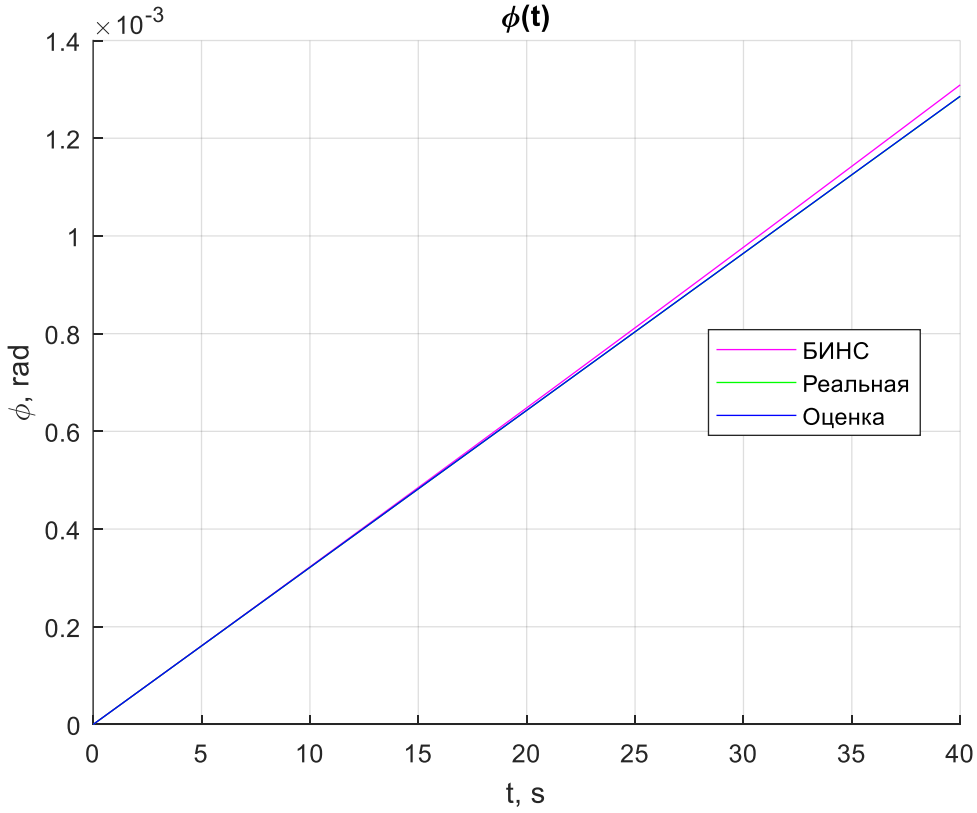

Рисунок Б.8 – Широта

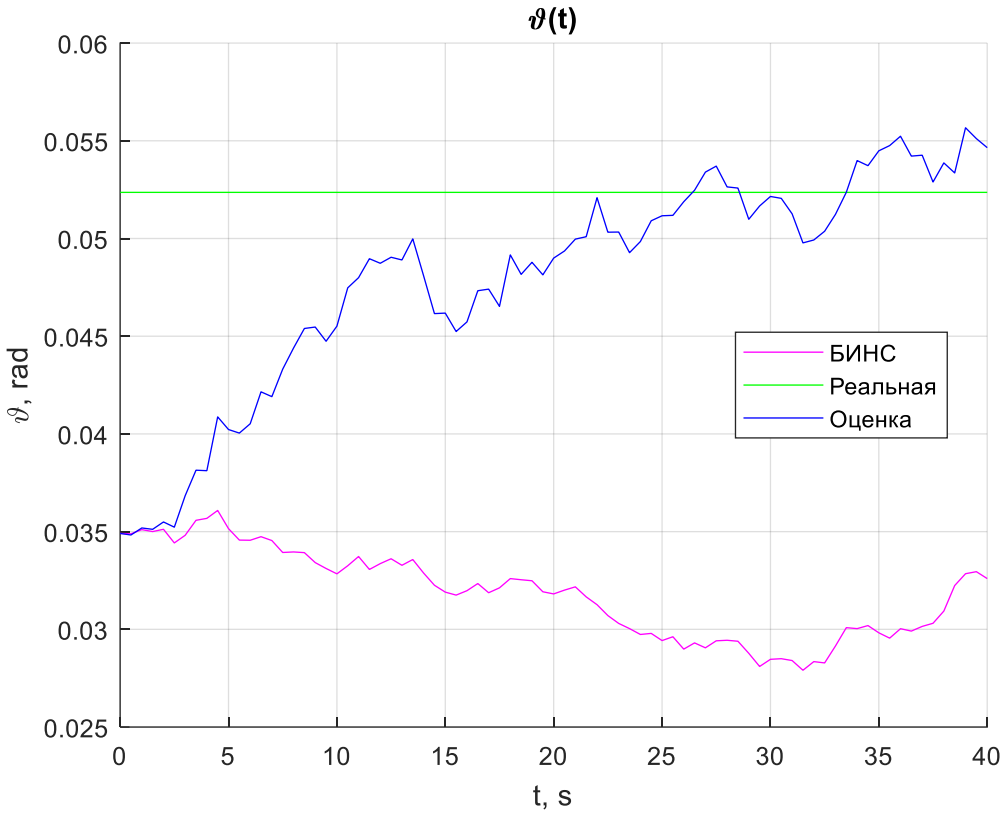

Рисунок Б.9 – Угол тангажа

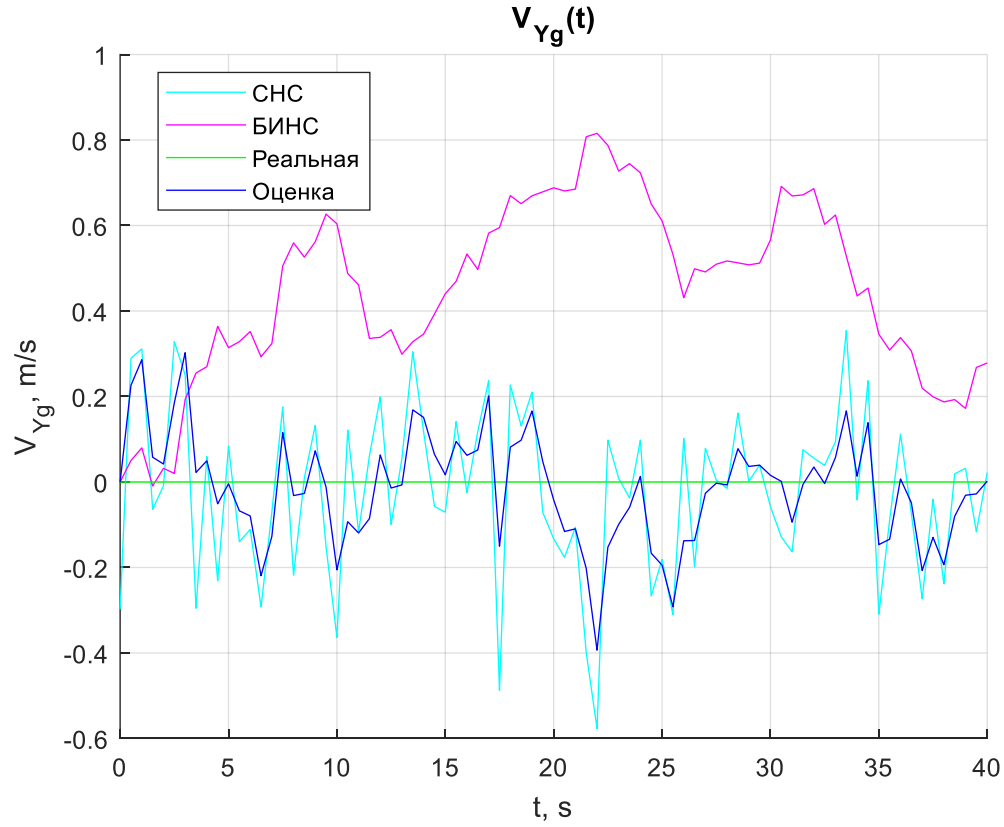

Рисунок Б.10 – Вертикальная скорость

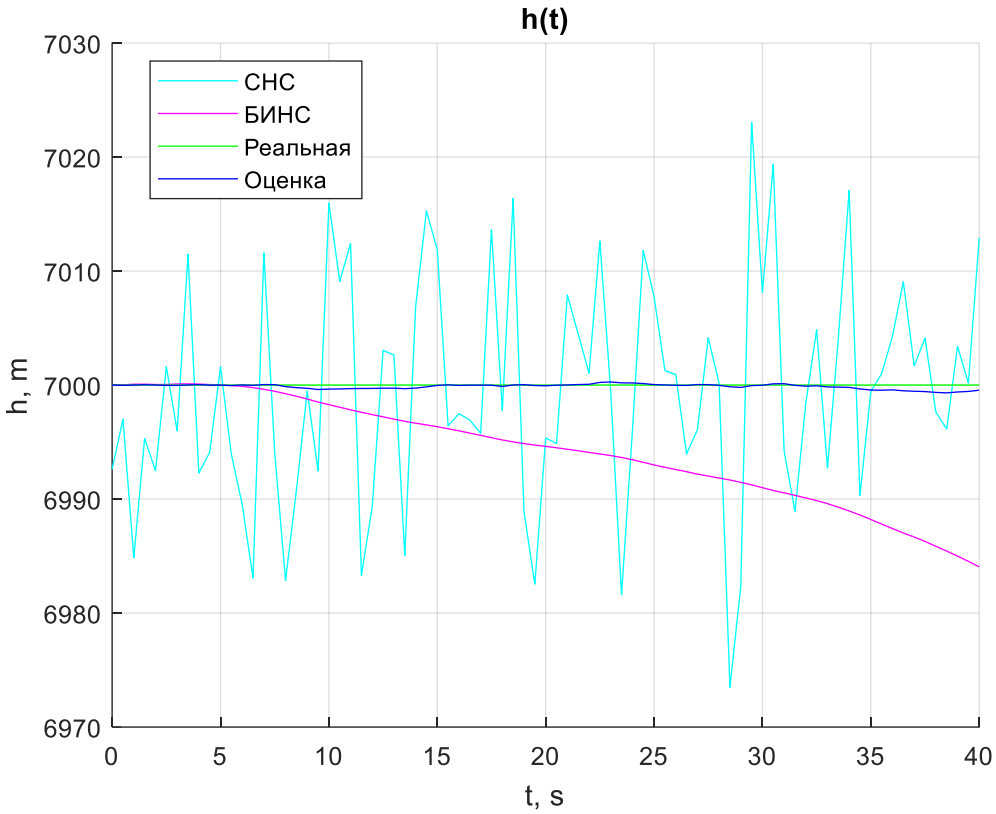

Рисунок Б.11 – Высота

ПРИЛОЖЕНИЕ В

Результаты моделирования при частоте 1 Гц сигнала СНС

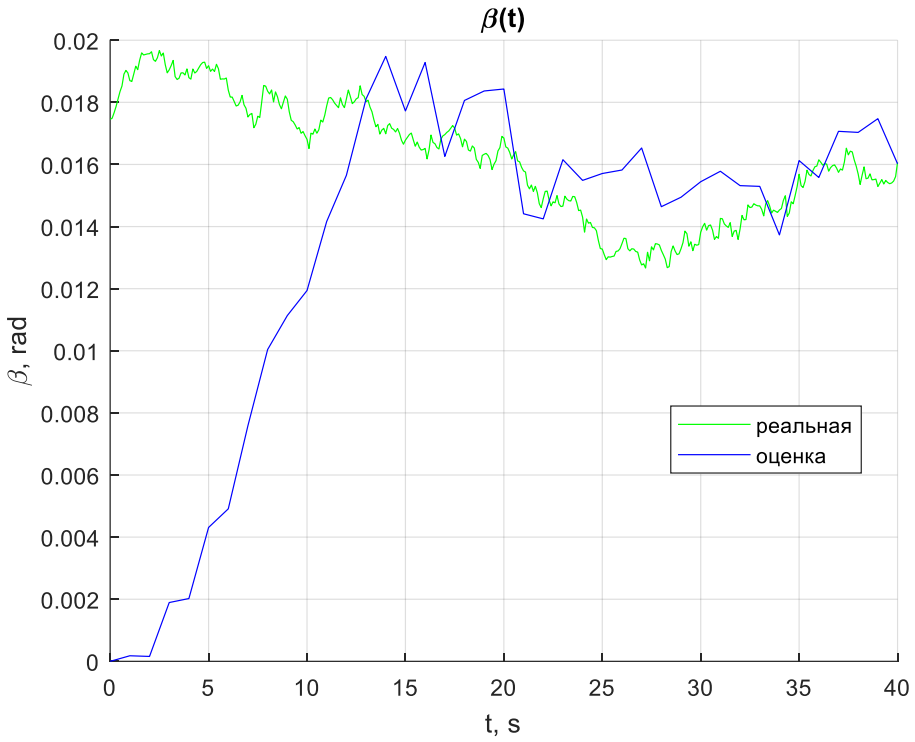

Рисунок В.1 – Ошибка построения вертикали

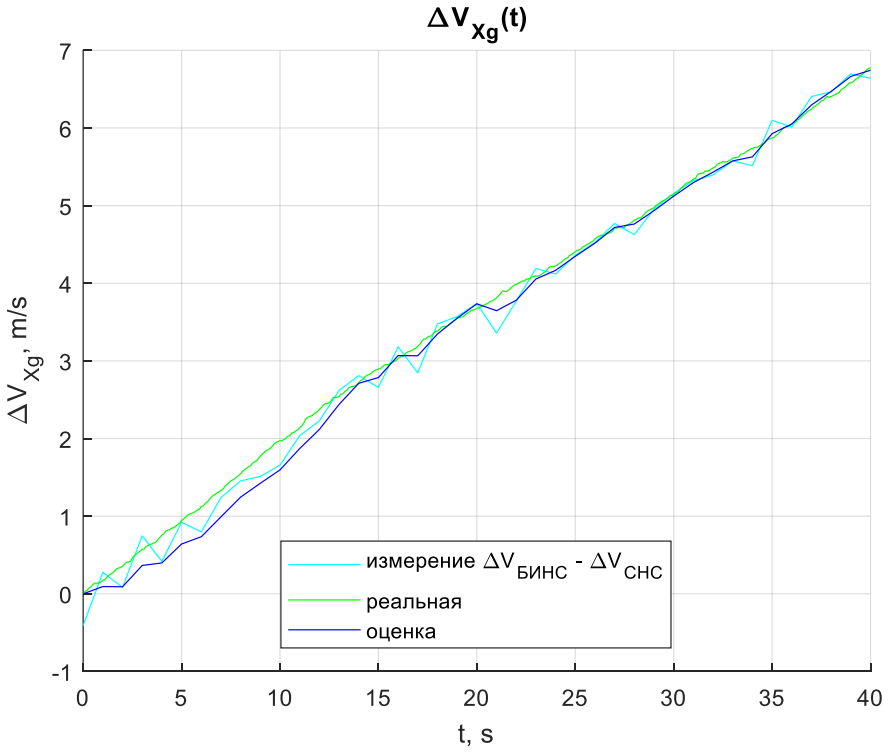

Рисунок В.2 – Ошибка выработки горизонтальной скорости

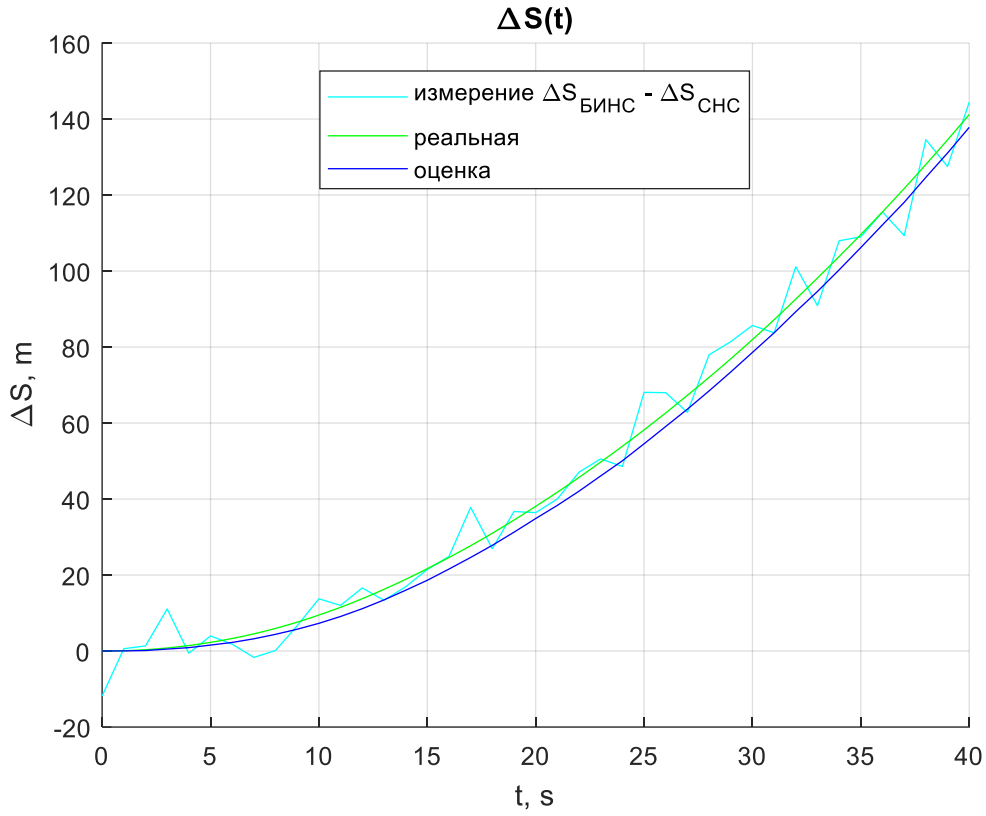

Рисунок В.3 – Ошибка выработки пройденного расстояния

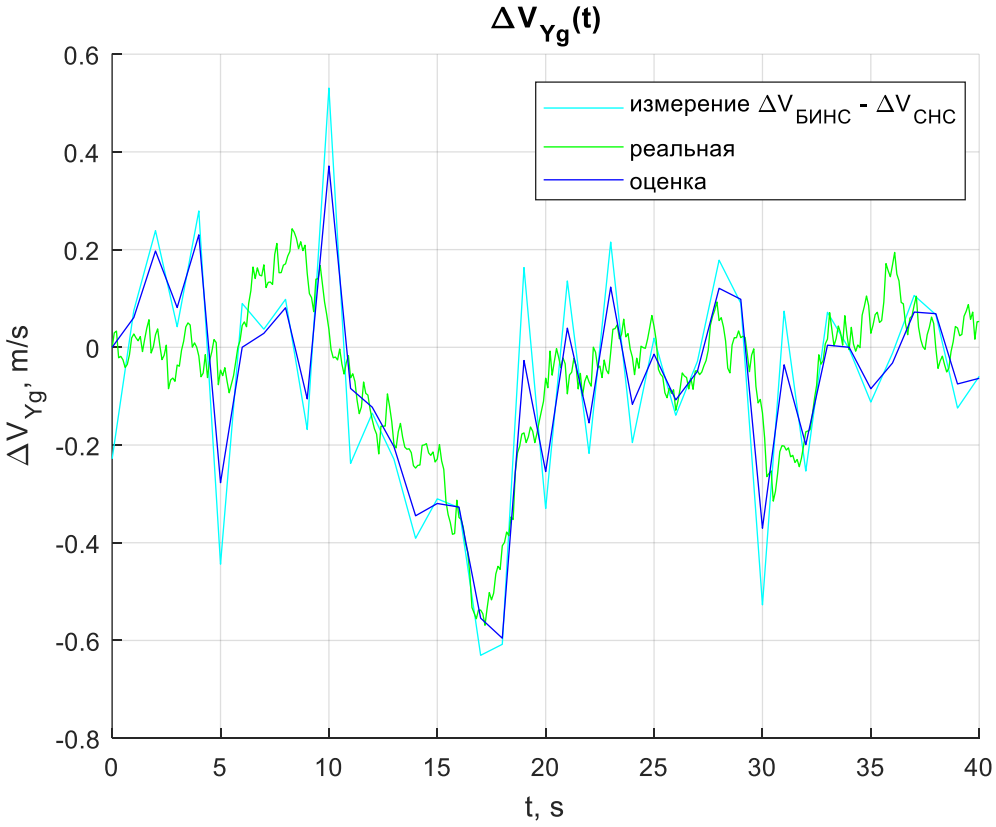

Рисунок В.4 – Ошибка выработки вертикальной скорости

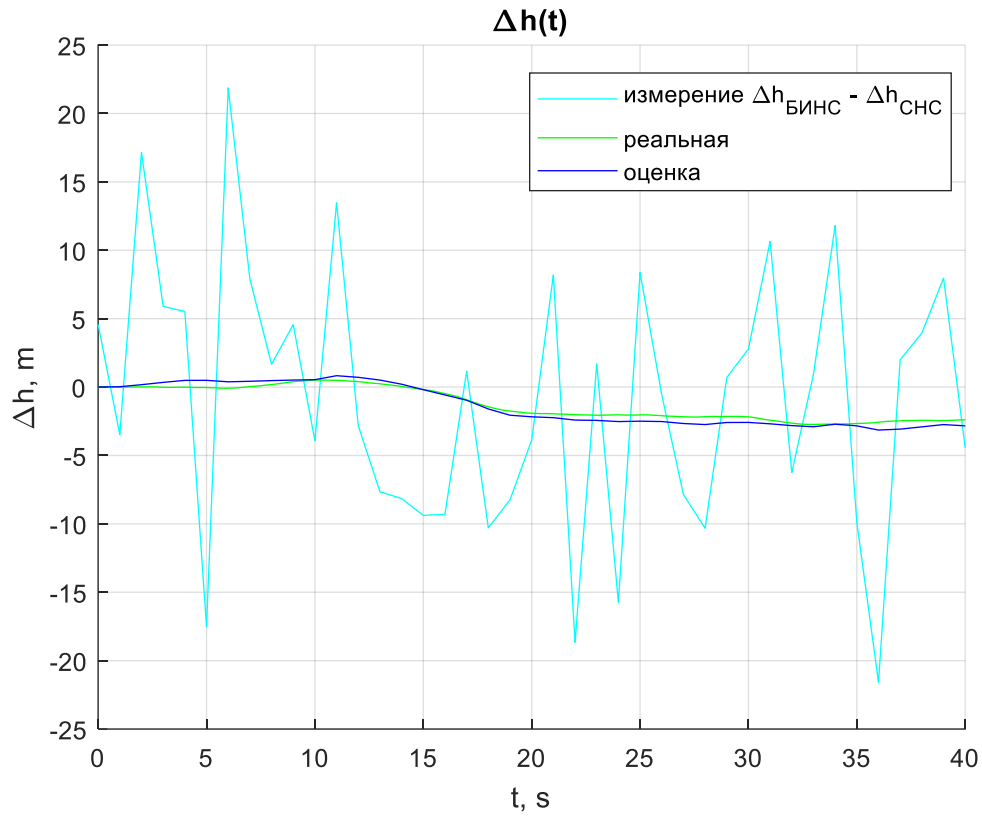

Рисунок В.5 – Ошибка выработки высоты

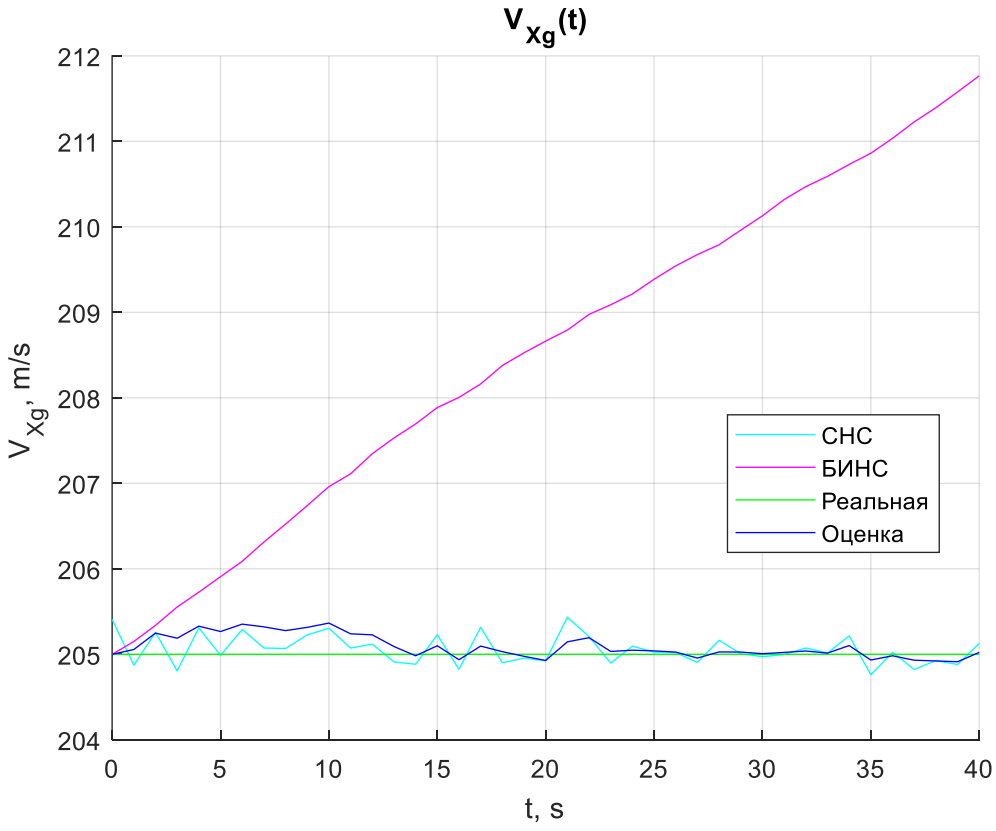

Рисунок В.6 – Горизонтальная скорость

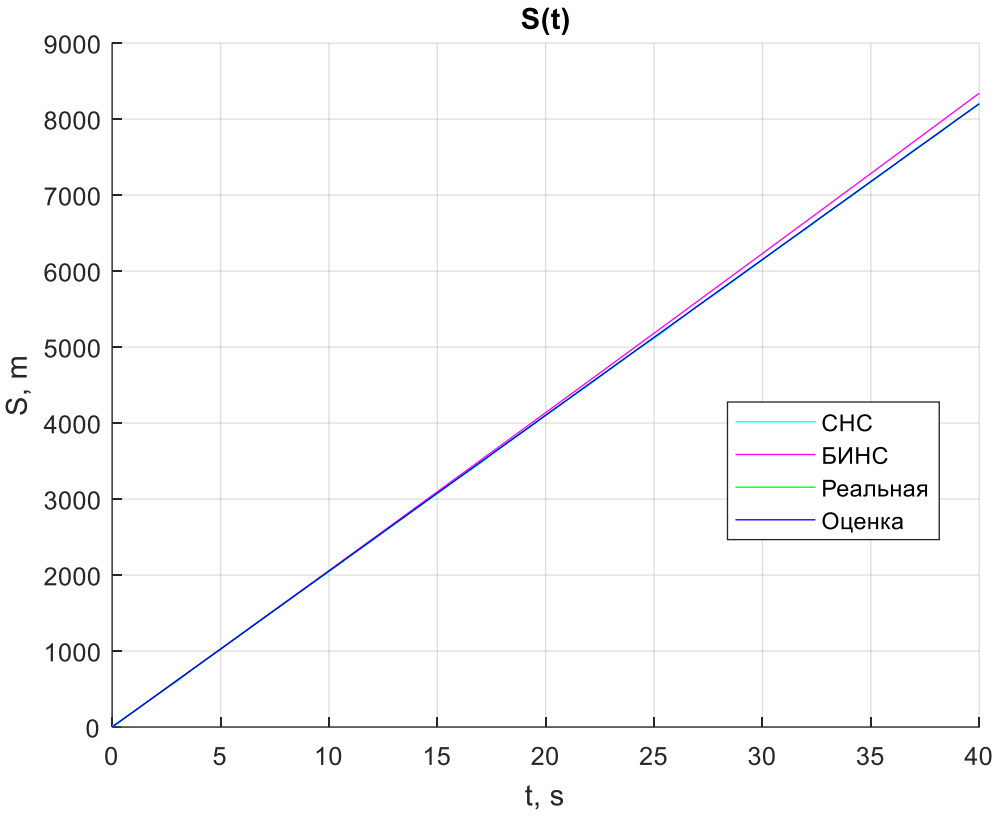

Рисунок В.7 – Пройденный путь

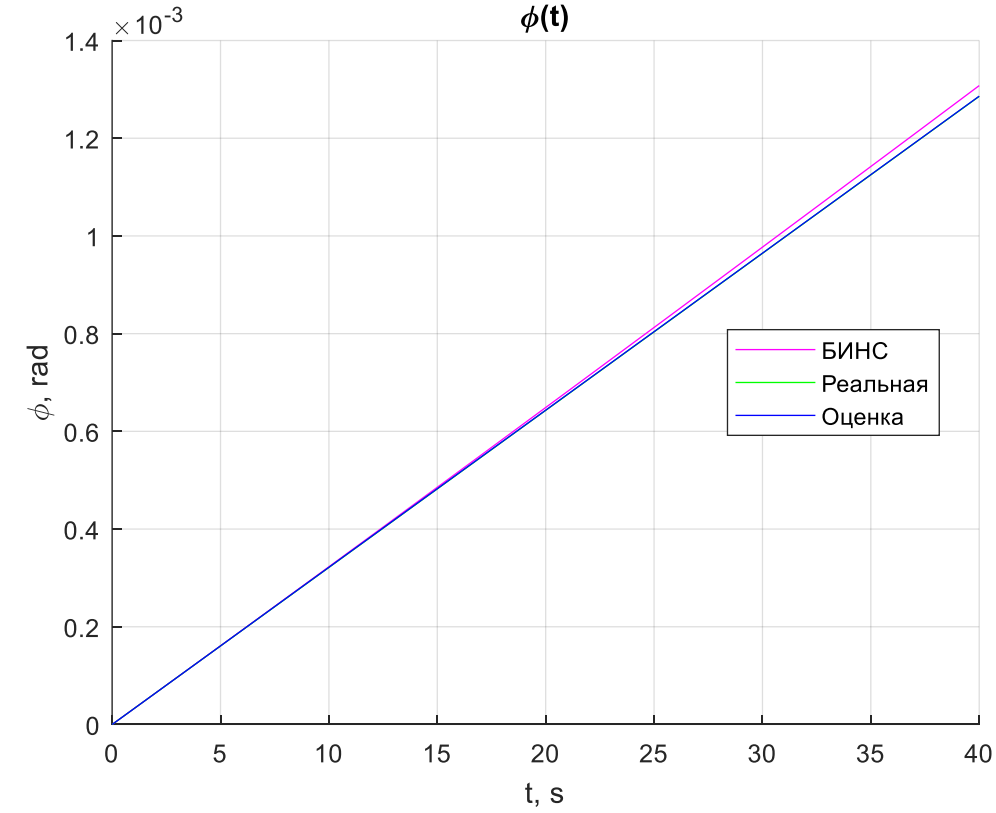

Рисунок В.8 – Широта

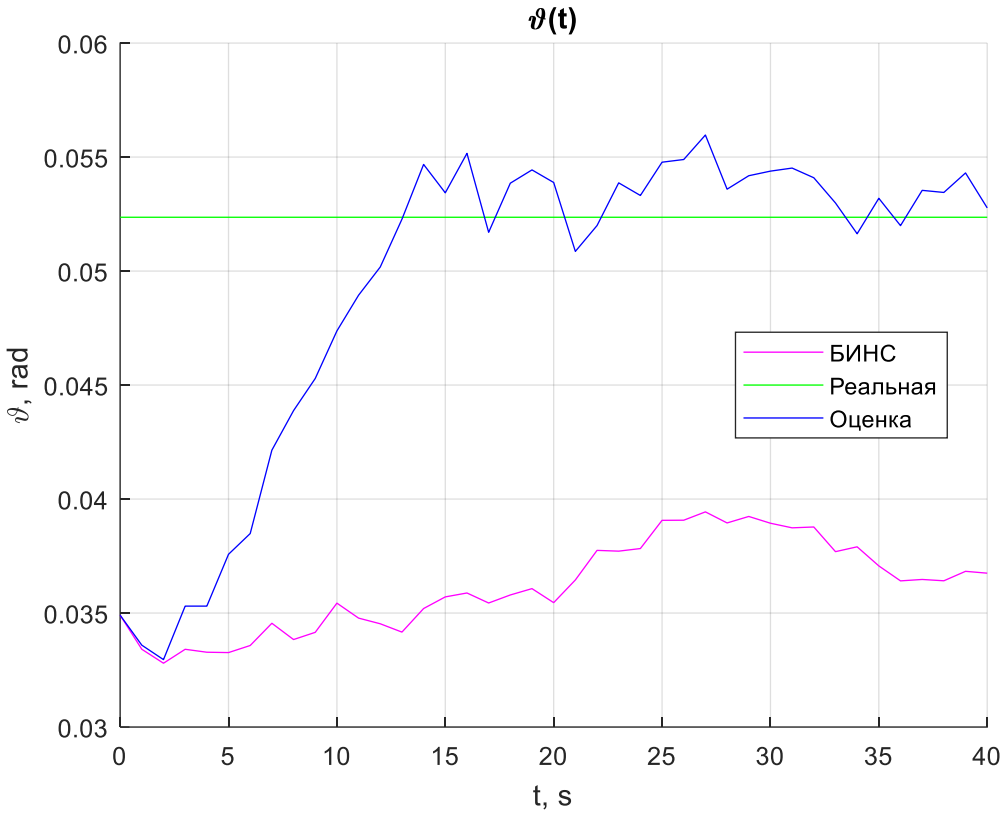

Рисунок В.9 – Угол тангажа

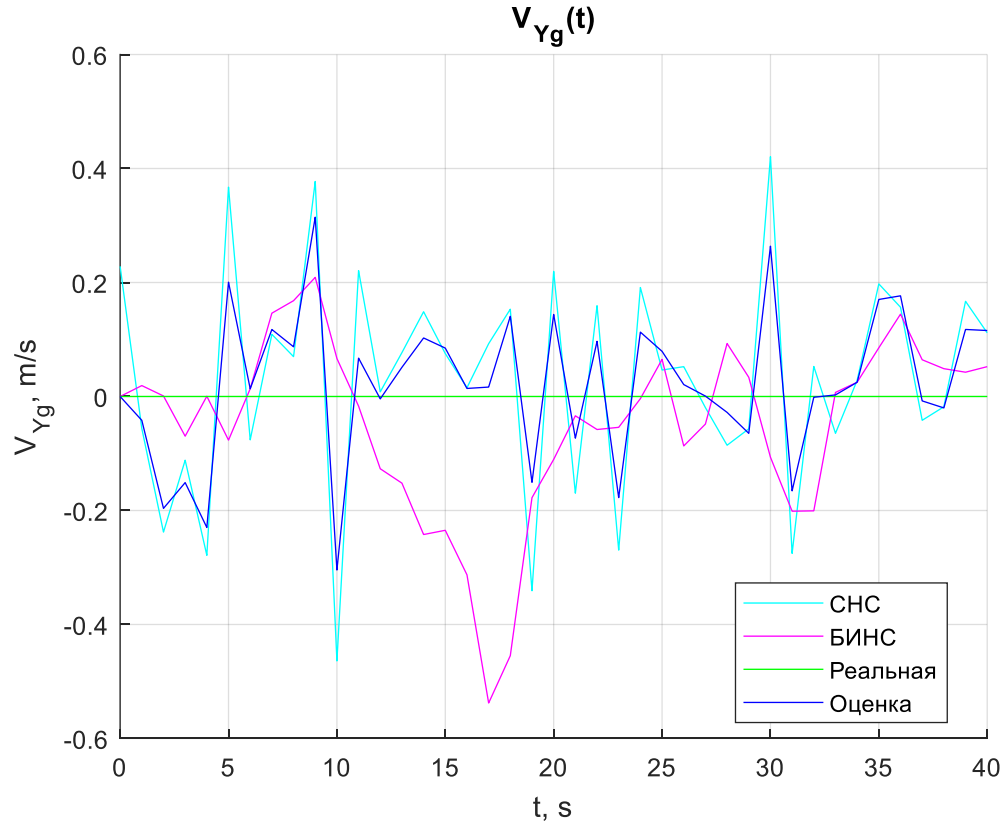

Рисунок В.10 – Вертикальная скорость

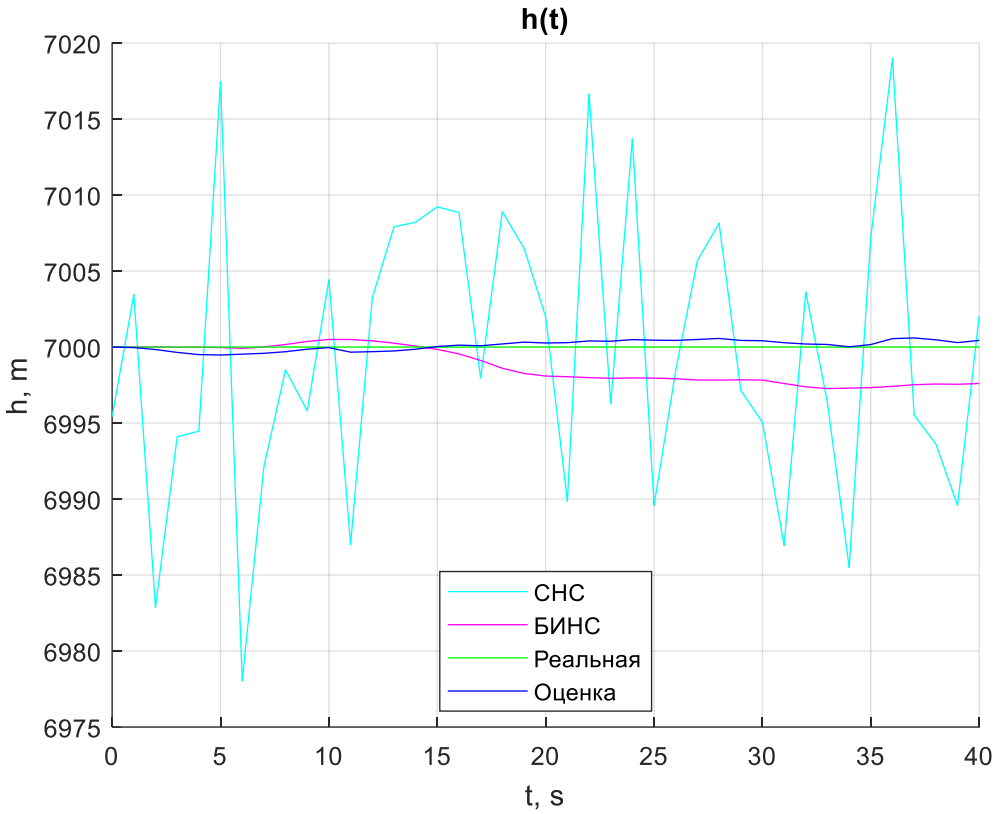

Рисунок В.11 – Высота

ПРИЛОЖЕНИЕ Г

Исходный код программного обеспечения в среде Qt Creator

```
// Библиотека NavLib:
// basepacket.h:
#pragma once

#include <QtCore>

class QByteArray;

//!
//! \brief The BasePacket class Базовый абстракный класс навигационных
пакетов
//!
class BasePacket
{
public:
    BasePacket() {}

    virtual ~BasePacket() {}

    virtual QByteArray encode();

    virtual void decode( QByteArray & ba ) = 0;

    virtual QByteArray toByteArray() const = 0;

    virtual QVector<double> toVector() const = 0;

    quint16 getChecksum() const { return checksum; }
    void setChecksum(const quint16 &value) { checksum = value; }

protected:
    quint8  packetId;
    quint16 checksum;
};

// basepacket.cpp:
#include "basepacket.h"

QByteArray BasePacket::encode()
{
    QByteArray result = this->toByteArray();
    checksum = qChecksum( result.data(), result.size() );

    QDataStream stream( &result, QIODevice::WriteOnly | QIODevice::Append );
    stream << checksum;

    return result;
}

// inspacket.h:
#pragma once

#include "basepacket.h"

//!
//! \brief The InsPacket class Пакет данных ИНС
//!
class InsPacket : public BasePacket
```

```
{
public:
    InsPacket();

    virtual void decode( QByteArray & ba );

    virtual QByteArray toByteArray() const;

    double getSecsElapsed() const;
    void setSecsElapsed(double newSecsElapsed);

    double getVxg() const;
    void setVxg(double newVxg);

    double getS() const;
    void setS(double newS);

    double getVyg() const;
    void setVyg(double newVyg);

    double getHeight() const;
    void setHeight(double newHeight);

    double getFi() const;
    void setFi(double newFi);

    double getTheta() const;
    void setTheta(double newTheta);

    double getDVxg() const;
    void setDVxg(double newDVxg);

    double getDS() const;
    void setDS(double newDS);

    double getDVyg() const;
    void setDVyg(double newDVyg);

    double getDh() const;
    void setDh(double newDh);

    double getBeta() const;
    void setBeta(double newBeta);

    virtual QVector<double> toVector() const;
    QVector<double> getErrorsVector() const;

private:
    double secsElapsed;
    double Vxg;
    double S;
    double Vyg;
    double height;
    double fi;
    double theta;
    double dVxg;
    double dS;
    double dVyg;
    double dh;
    double beta;
};

// inspacket.cpp:
```

```
#include "inspacket.h"

#include <QByteArray>
#include <QDataStream>

InsPacket::InsPacket() : BasePacket()
{
    packetId = 0;
}

void InsPacket::decode(QByteArray & ba)
{
     QDataStream stream( &ba, QIODevice::ReadOnly );

     stream >> packetId
            >> secsElapsed
            >> Vxg
            >> S
            >> Vyg
            >> height
            >> fi
            >> theta
            >> dVxg
            >> dS
            >> dVyg
            >> dh
            >> beta
            >> checksum;
}

QByteArray InsPacket::toByteArray() const
{
    QByteArray result;
    QDataStream stream( &result, QIODevice::WriteOnly );

    stream << packetId
           << secsElapsed
           << Vxg
           << S
           << Vyg
           << height
           << fi
           << theta
           << dVxg
           << dS
           << dVyg
           << dh
           << beta;

    return result;
}

double InsPacket::getSecsElapsed() const
{
    return secsElapsed;
}

void InsPacket::setSecsElapsed(double newSecsElapsed)
{
    secsElapsed = newSecsElapsed;
}

double InsPacket::getVxg() const
```

```
{
    return Vxg;
}

void InsPacket::setVxg(double newVxg)
{
    Vxg = newVxg;
}

double InsPacket::getS() const
{
    return S;
}

void InsPacket::setS(double newS)
{
    S = newS;
}

double InsPacket::getVyg() const
{
    return Vyg;
}

void InsPacket::setVyg(double newVyg)
{
    Vyg = newVyg;
}

double InsPacket::getHeight() const
{
    return height;
}

void InsPacket::setHeight(double newHeight)
{
    height = newHeight;
}

double InsPacket::getFi() const
{
    return fi;
}

void InsPacket::setFi(double newFi)
{
    fi = newFi;
}

double InsPacket::getTheta() const
{
    return theta;
}

void InsPacket::setTheta(double newTheta)
{
    theta = newTheta;
}

double InsPacket::getDVxg() const
{
    return dVxg;
}
```

```
void InsPacket::setDVxg(double newDVxg)
{
    dVxg = newDVxg;
}

double InsPacket::getDS() const
{
    return dS;
}

void InsPacket::setDS(double newDS)
{
    dS = newDS;
}

double InsPacket::getDVyg() const
{
    return dVyg;
}

void InsPacket::setDVyg(double newDVyg)
{
    dVyg = newDVyg;
}

double InsPacket::getDh() const
{
    return dh;
}

void InsPacket::setDh(double newDh)
{
    dh = newDh;
}

double InsPacket::getBeta() const
{
    return beta;
}

void InsPacket::setBeta(double newBeta)
{
    beta = newBeta;
}

QVector<double> InsPacket::toVector() const
{
    return QVector<double>{ secsElapsed,
                            Vxg,
                            S,
                            Vyg,
                            height,
                            fi,
                            theta };
}

QVector<double> InsPacket::getErrorsVector() const
{
    return QVector<double>{ secsElapsed,
                            dVxg,
                            dS,
                            dVyg,
```

```
                                dh,
                                beta };
}

// realpacket.h:
#pragma once

#include "basepacket.h"

//!
//! \brief The RealPacket class Пакет реальных данных
//!
class RealPacket : public BasePacket
{
public:
    RealPacket();

    virtual void decode( QByteArray & ba );

    virtual QByteArray toByteArray() const;

    double getSecsElapsed() const;
    void setSecsElapsed(double newSecsElapsed);

    double getVxg() const;
    void setVxg(double newVxg);

    double getS() const;
    void setS(double newS);

    double getVyg() const;
    void setVyg(double newVyg);

    double getHeight() const;
    void setHeight(double newHeight);

    double getFi() const;
    void setFi(double newFi);

    double getTheta() const;
    void setTheta(double newTheta);

    virtual QVector<double> toVector() const;

private:
    double secsElapsed;
    double Vxg;
    double S;
    double Vyg;
    double height;
    double fi;
    double theta;
};

// realpacket.cpp:
#include "realpacket.h"

#include <QByteArray>
#include <QDataStream>

RealPacket::RealPacket() : BasePacket()
{
    packetId = 2;
```

```
}

void RealPacket::decode(QByteArray & ba)
{
     QDataStream stream( &ba, QIODevice::ReadOnly );

     stream >> packetId
            >> secsElapsed
            >> Vxg
            >> S
            >> Vyg
            >> height
            >> fi
            >> theta
            >> checksum;
}

QByteArray RealPacket::toByteArray() const
{
    QByteArray result;
    QDataStream stream( &result, QIODevice::WriteOnly );

    stream << packetId
           << secsElapsed
           << Vxg
           << S
           << Vyg
           << height
           << fi
           << theta;

    return result;
}

double RealPacket::getSecsElapsed() const
{
    return secsElapsed;
}

void RealPacket::setSecsElapsed(double newSecsElapsed)
{
    secsElapsed = newSecsElapsed;
}

double RealPacket::getVxg() const
{
    return Vxg;
}

void RealPacket::setVxg(double newVxg)
{
    Vxg = newVxg;
}

double RealPacket::getS() const
{
    return S;
}

void RealPacket::setS(double newS)
{
    S = newS;
}
```

```
double RealPacket::getVyg() const
{
    return Vyg;
}

void RealPacket::setVyg(double newVyg)
{
    Vyg = newVyg;
}

double RealPacket::getHeight() const
{
    return height;
}

void RealPacket::setHeight(double newHeight)
{
    height = newHeight;
}

double RealPacket::getFi() const
{
    return fi;
}

void RealPacket::setFi(double newFi)
{
    fi = newFi;
}

double RealPacket::getTheta() const
{
    return theta;
}

void RealPacket::setTheta(double newTheta)
{
    theta = newTheta;
}

QVector<double> RealPacket::toVector() const
{
    return QVector<double>{ secsElapsed,
                            Vxg,
                            S,
                            Vyg,
                            height,
                            fi,
                            theta };
}

// snspacket.h:
#pragma once

#include "basepacket.h"

//!
//! \brief The SnsPacket class Пакет данных СНС
//!
class SnsPacket : public BasePacket
{
public:
```

```
    SnsPacket();

    virtual QByteArray toByteArray() const;

    virtual void decode( QByteArray & ba );

    double getSecsElapsed() const;
    void setSecsElapsed(double newSecsElapsed);

    double getVxg() const;
    void setVxg(double newVxg);

    double getS() const;
    void setS(double newS);

    double getVyg() const;
    void setVyg(double newVyg);

    double getHeight() const;
    void setHeight(double newHeight);

    virtual QVector<double> toVector() const;

private:
    double secsElapsed;
    double Vxg;
    double S;
    double Vyg;
    double height;
};

// snspacket.cpp:
#include "snspacket.h"

#include <QByteArray>
#include <QDataStream>

SnsPacket::SnsPacket() : BasePacket()
{
    packetId = 1;
}

QByteArray SnsPacket::toByteArray() const
{
    QByteArray result;
    QDataStream stream( &result, QIODevice::WriteOnly );

    stream << packetId
           << secsElapsed
           << Vxg
           << S
           << Vyg
           << height;

    return result;
}

void SnsPacket::decode(QByteArray & ba)
{
     QDataStream stream( &ba, QIODevice::ReadOnly );

     stream >> packetId
            >> secsElapsed
```

```
            >> Vxg
            >> S
            >> Vyg
            >> height
            >> checksum;
}

double SnsPacket::getSecsElapsed() const
{
    return secsElapsed;
}

void SnsPacket::setSecsElapsed(double newSecsElapsed)
{
    secsElapsed = newSecsElapsed;
}

double SnsPacket::getVxg() const
{
    return Vxg;
}

void SnsPacket::setVxg(double newVxg)
{
    Vxg = newVxg;
}

double SnsPacket::getS() const
{
    return S;
}

void SnsPacket::setS(double newS)
{
    S = newS;
}

double SnsPacket::getVyg() const
{
    return Vyg;
}

void SnsPacket::setVyg(double newVyg)
{
    Vyg = newVyg;
}

double SnsPacket::getHeight() const
{
    return height;
}

void SnsPacket::setHeight(double newHeight)
{
    height = newHeight;
}

QVector<double> SnsPacket::toVector() const
{
    return QVector<double>{ secsElapsed,
                            Vxg,
                            S,
                            Vyg,
```

```
                                height };
}
// mathematics.h:
#pragma once

#include <random>

namespace Math
{
    std::normal_distribution<double>::result_type normrnd(const double mean,
const double sigma);
}

// mathematics.cpp:
#include "mathematics.h"

namespace Math
{
    std::normal_distribution<double>::result_type normrnd(const double mean,
const double sigma)
    {
        static std::random_device rd;
        static std::mt19937 generator( rd() );

        std::normal_distribution<double> normrand( mean, sigma );
        return normrand( generator );
    }
}

// vector.h:
#pragma once

#include <QVector>

#include <vector>
#include <cassert>

namespace Math
{
    //!
    //! \brief The Vector class Вектор
    //!
    template<typename T>
    class Vector : public std::vector<T>
    {
    public:
        Vector<T>() : std::vector<T>(){}

        Vector<T>(std::initializer_list<T> il) : std::vector<T>(il){}

        Vector<T>(std::initializer_list< Vector<T> > il) : std::vector<
std::vector<T> >(il){}

        Vector<T>(size_t size) : std::vector<T>(size){}

        Vector<T>( size_t rowSize,  const T & vector  ) : std::vector<T>(
rowSize, vector ){}

        Vector<T>& operator=( const QVector<T> & vector )
        {
            this->clear();
            this->resize( vector.size() );
```

```
    for (size_t i = 0; i < this->size(); ++i)
        this->data()[i] = vector[i];

    return *this;
}

Vector<T> operator*(const Vector<T> & right) const
{
    assert(this->size() == right.size());

    Vector<T> result(this->size());

    for (size_t i = 0; i < this->size(); ++i)
        result[i] = this->data()[i] * right[i];

    return result;
}

Vector<T> operator+(const Vector<T> & right) const
{
    assert(this->size() == right.size());

    Vector<T> result(this->size());

    for (size_t i = 0; i < this->size(); ++i)
        result[i] = this->data()[i] + right[i];

    return result;
}

Vector<T> operator-(const Vector<T> & right) const
{
    assert(this->size() == right.size());

    Vector<T> result(this->size());

    for (size_t i = 0; i < this->size(); ++i)
        result[i] = this->data()[i] - right[i];

    return result;
}

Vector<T> operator-(double num) const
{
    Vector<T> result(this->size());

    for (size_t i = 0; i < this->size(); ++i)
        result[i] = this->data()[i] - num;

    return result;
}

Vector<T> operator+(double num) const
{
    Vector<T> result(this->size());

    for (size_t i = 0; i < this->size(); ++i)
        result[i] = this->data()[i] + num;

    return result;
}

Vector<T> operator*(double num) const
```

```
        {
            Vector<T> result(this->size());

            for (size_t i = 0; i < this->size(); ++i)
                result[i] = this->data()[i] * num;

            return result;
        }

        Vector<T> operator/(double num) const
        {
            Vector<T> result(this->size());

            for (size_t i = 0; i < this->size(); ++i)
                result[i] = this->data()[i] / num;

            return result;
        }

        Vector<T> square()
        {
            return this->operator*( *this );
        }
    };

    template<typename T>
    double sum(const Vector<T> & v)
    {
        double sum = 0;
        for (size_t i = 0; i < v.size(); ++i)
            sum += v[i];

        return sum;
    }

    template<typename T>
    Vector<T> sqrt(const Vector<T> & v)
    {
        Vector<T> result( v.size() );

        for (size_t i = 0; i < v.size(); ++i)
            result[i] = sqrt( v[i] );

        return result;
    }
}

// matrix.h:
#pragma once

#include "vector.h"

#include <QDebug>

#include <iostream>
#include <cassert>

namespace Math
{
    //!
    //! \brief The Matrix class Матрица
    //!
    template <typename T>
```

```
    class Matrix
    {
    public:
        Matrix( size_t row, size_t col ) : row( row ), col( col ),
            data( row, Vector<T>(col) ) {}

        Matrix(std::initializer_list< Vector<T> > il)
        : row( il.size() )
        , col( (*il.begin()).size() )
        , data(il)
        {}

        T& operator()(size_t i, size_t j)
        {
            return data[i][j];
        }

        const T& operator()(size_t i, size_t j) const
        {
            return data[i][j];
        }

        Matrix<T> & operator=( const Matrix<T> & right )
        {
            assert( row == right.rows() && col == right.cols() && "dimen-
stions must agree" );

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < col; ++j)
                    data[i][j] = right(i, j);

            return *this;
        }

        Matrix<T> operator*( const Matrix<T> & right ) const
        {
            assert( col == right.rows() && "dimenstions must agree" );

            Matrix<T> result( row, right.cols() );

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < right.cols(); ++j)
                    result(i, j) = 0;

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < right.cols(); ++j)
                    for(size_t k = 0; k < col; ++k)
                    {
                        result(i, j) += data[i][k] * right(k, j);
                    }

            return result;
        }

        Matrix<T> operator*( const double right ) const
        {
            Matrix<T> result( row, col );

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < col; ++j)
                    result(i, j) = data[i][j] * right;

            return result;
```

```
        }

        Vector<T> operator*( const Vector<T> & right ) const
        {
            assert( col == right.size() && "dimenstions must agree" );

            Vector<T> result( row );

            for(size_t i = 0; i < row; ++i)
                result[i] = 0;

            for(size_t i = 0; i < row; ++i)
                for(size_t k = 0; k < col; ++k)
                {
                    result[i] += data[i][k] * right[k];
                }

            return result;
        }

        Matrix<T> operator+( const Matrix<T> & right )
        {
            assert( row == right.rows() && col == right.cols() && "dimen-
stions must agree" );

            Matrix<T> result(row, col);

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < col; ++j)
                {
                    result(i, j) = data[i][j] + right(i, j);
                }

            return result;
        }

        Matrix<T> operator-( const Matrix<T> & right )
        {
            assert( row == right.rows() && col == right.cols() && "dimen-
stions must agree" );

            Matrix<T> result(row, col);

            for(size_t i = 0; i < row; ++i)
                for(size_t j = 0; j < col; ++j)
                {
                    result(i, j) = data[i][j] - right(i, j);
                }

            return result;
        }

        Vector<T> getCol( size_t colIndex ) const
        {
            Vector<T> result( row );

            for(size_t i = 0; i < row; ++i)
                result[i] = data[i][colIndex];

            return result;
        }

        const Vector<T> & getRow( size_t rowIndex ) const
```

```
{
    return data[rowIndex];
}

void setCol( size_t colIndex, const Vector<T> & columnVector )
{
    assert( row == columnVector.size() && "dimenstions must agree" );

    for(size_t i = 0; i < row; ++i)
        data[i][colIndex] = columnVector[i];
}

void setRow( size_t rowIndex, const Vector<T> & rowVector )
{
    assert( col == rowVector.size() && "dimenstions must agree" );

    data[rowIndex] = rowVector;
}

static Matrix<T> eye(size_t n)
{
    Matrix<T> result(n, n);

    for(size_t i = 0; i < n; ++i)
        for(size_t j = 0; j < n; ++j)
            if ( i == j )
                result(i, j) = 1;

    return result;
}

Matrix<T> transpose() const
{
    Matrix<T> result(col, row);

    for(size_t i = 0; i < row; ++i)
        for(size_t j = 0; j < col; ++j)
        {
            result(j, i) = data[i][j];
        }

    return result;
}

Matrix<T> inv()
{
    assert( row == col && "matrix must be square" );

    Matrix<T> result = *this;


    double temp;

    Matrix<T> E(row, col);

    for(size_t i = 0; i < row; ++i)
        for(size_t j = 0; j < col; ++j)
        {
            if (i == j)
                E(i, j) = 1;
            else
                E(i, j) = 0;
        }
```

```
    for (size_t k = 0; k < row; k++)
    {
        temp = result(k, k);

        for (size_t j = 0; j < col; j++)
        {
            result(k, j) /= temp;
                 E(k, j) /= temp;
        }

        for (size_t i = k + 1; i < row; i++)
        {
            temp = result(i, k);

            for (size_t j = 0; j < col; j++)
            {
                result(i, j) -= result(k, j) * temp;
                     E(i, j) -=      E(k, j) * temp;
            }
        }
    }

    for (size_t k = col - 1; k > 0; k--)
    {
        for (int i = k - 1; i >= 0; i--)
        {
            temp = result(i, k);

            for (size_t j = 0; j < col; j++)
            {
                result(i, j) -= result(k, j) * temp;
                     E(i, j) -=      E(k, j) * temp;
            }
        }
    }

    result = E;

    return result;
}

void resize( size_t row, size_t col )
{
    data.resize( row );
    this->row = row;

    for ( size_t i = 0; i < row; ++i )
        data[i].resize( col );

    this->col = col;
}

void print() const
{
    for(size_t i = 0; i < row; ++i)
    {
        for(size_t j = 0; j < col; ++j)
            std::cout << data[i][j] << " ";
        std::cout << "\n";
    }
}
```

```
        size_t rows() const  { return row; }
        size_t cols() const  { return col; }

    private:
                        size_t row;
                        size_t col;
        Vector< Vector<T> > data;
    };
}

// settings.h:
#pragma once

#include "matrix.h"

class QSettings;

//!
//! \brief The Settings class Настройки приложения
//!
class Settings
{
public:
    Settings();
    ~Settings();

    void parse(const QString & fileName);

    double g;
    double R_earth;
    double r;

    double sig_gyro;
    double sig_acc_X;
    double sig_acc_Y;

    struct SigmaSNS
    {
        double Vxg;
        double S;
        double Vyg;
        double h;
    };

    SigmaSNS sigSns1;
    SigmaSNS sigSns2;

    double insMSecsInterval;
    double sns1MSecsInterval;
    double sns2MSecsInterval;
    double tEnd;

    double Vxg;
    double theta;
    double Vyg;
    double height;

    Math::Matrix<double> A;
    Math::Matrix<double> G;
    Math::Matrix<double> H;
    Math::Matrix<double> Q;
    Math::Matrix<double> R1;
    Math::Matrix<double> R2;
```

```
    Math::Matrix<double> F;
    Math::Matrix<double> gamma;

private:
    void calcMatrices();

    QSettings * settings;
};

// settings.cpp:
#include "settings.h"

#include <QSettings>
#include <QDebug>

#include <cmath>

Settings::Settings()
    : A(5, 5)
    , G(5, 3)
    , H(4, 5)
    , Q(3, 3)
    , R1(4, 4)
    , R2(4, 4)
    , F(5, 5)
    , gamma(5, 3)
    , settings( nullptr )
{

}

Settings::~Settings()
{
    delete settings;
}

void Settings::parse(const QString & fileName)
{
    settings = new QSettings(fileName, QSettings::IniFormat);

    g       = settings->value("g").toDouble();
    R_earth = settings->value("R_earth").toInt();
    r = R_earth + 7000;

    sig_gyro   = settings->value("sig_gyro").toDouble();
    sig_acc_X  = settings->value("sig_acc_X").toDouble();
    sig_acc_Y  = settings->value("sig_acc_Y").toDouble();

    sigSns1.Vxg = settings->value("sig_Vxg1").toDouble();
    sigSns1.S   = settings->value("sig_S1").toDouble();
    sigSns1.Vyg = settings->value("sig_Vyg1").toDouble();
    sigSns1.h   = settings->value("sig_h1").toDouble();

    double coef = settings->value("coef").toDouble();
    sigSns2.Vxg = sigSns1.Vxg * coef;
    sigSns2.S   = sigSns1.S   * coef;
    sigSns2.Vyg = sigSns1.Vyg * coef;
    sigSns2.h   = sigSns1.h   * coef;

    insMSecsInterval  = settings->value("insMSecsInterval").toDouble();
    sns1MSecsInterval = settings->value("sns1MSecsInterval").toDouble();
    sns2MSecsInterval = settings->value("sns2MSecsInterval").toDouble();
```

```
    tEnd = settings->value("tEnd").toDouble();

    Vxg     = settings->value("Vxg").toDouble();
    theta   = settings->value("theta").toDouble() * M_PI/180.0;
    Vyg     = settings->value("Vyg").toDouble();
    height  = settings->value("height").toDouble();

    calcMatrices();
}

void Settings::calcMatrices()
{
    A = {
            { 0, -1/r, 0, 0, 0 },
            { g,   0,  0, 0, 0 },
            { 0,   1,  0, 0, 0 },
            { 0,   0,  0, 0, 0 },
            { 0,   0,  0, 1, 0 }
        };

    G = {
            {-1, 0, 0 },
            { 0, 1, 0 },
            { 0, 0, 0 },
            { 0, 0, 1 },
            { 0, 0, 0 }
        };

    H = {
            { 0, 1, 0, 0, 0 },
            { 0, 0, 1, 0, 0 },
            { 0, 0, 0, 1, 0 },
            { 0, 0, 0, 0, 1 }
        };


    Q = {
            { sig_gyro*sig_gyro,   0, 0 },
            { 0, sig_acc_X*sig_acc_X, 0 },
            { 0, 0, sig_acc_Y*sig_acc_Y }
        };

    R1 = {
            { sigSns1.Vxg*sigSns1.Vxg, 0, 0, 0 },
            { 0,  sigSns1.S*sigSns1.S,    0, 0 },
            { 0, 0, sigSns1.Vyg*sigSns1.Vyg, 0 },
            { 0, 0,     0, sigSns1.h*sigSns1.h }
        };

    R2 = {
            { sigSns2.Vxg*sigSns2.Vxg, 0, 0, 0 },
            { 0,  sigSns2.S*sigSns2.S,    0, 0 },
            { 0, 0, sigSns2.Vyg*sigSns2.Vyg, 0 },
            { 0, 0,     0, sigSns2.h*sigSns2.h }
        };

    double insSecsInterval = insMSecsInterval / 1000.0;
    F = Math::Matrix<double>::eye(5)  +  A * insSecsInterval;
    gamma = G * insSecsInterval;
}

// Приложение Imitators:
// udpsender.h:
```

```
#pragma once

#include <QHostAddress>
#include <QObject>

class QUdpSocket;

//!
//! \brief The UdpSender class Менеджер отправки навигационных пакетов по UDP
//!
class UdpSender : public QObject
{
    Q_OBJECT

public:
    explicit UdpSender(const QHostAddress & address, int outPort, QObject
*parent = nullptr);

    void sendData( const QByteArray & ba );

private:
    QUdpSocket * socket;
    QHostAddress address;
    int outPort;
};

// udpsender.cpp:
#include "udpsender.h"

#include <QUdpSocket>

UdpSender::UdpSender(const QHostAddress & address, int outPort, QObject
*parent) : QObject(parent)
  , socket( new QUdpSocket(this) )
  , address( address )
  , outPort( outPort )
{

}

void UdpSender::sendData(const QByteArray &ba)
{
    socket->writeDatagram( ba, address, outPort );
}

// baseimitator.h:
#pragma once

#include <QObject>

class UdpSender;
class QTimer;
class QHostAddress;

//!
//! \brief The BaseImitator class Базовый абстрактный класс имитаторов
//!
class BaseImitator : public QObject
{
    Q_OBJECT

public:
    explicit BaseImitator(const QHostAddress & address,
```

```
                            int outPort,
                            int msecsInterval,
                            QObject * parent = nullptr);

    virtual ~BaseImitator() {}

public slots:
    void start();
    void stop();

protected slots:
    virtual void onTimeout() = 0;

protected:
    QTimer    * timer;
    UdpSender * sender;

    double secsInterval;
    double secsElapsed;
};

// baseimitator.cpp:
#include "baseimitator.h"
#include "udpsender.h"

#include <QTimer>

BaseImitator::BaseImitator(const QHostAddress & address,
                           int outPort,
                           int msecsInterval,
                           QObject * parent)
    : QObject(parent)
    , timer ( new QTimer(this) )
    , sender( new UdpSender(address, outPort, this) )
    , secsInterval(msecsInterval / 1000.0)
    , secsElapsed(0)
{
    timer->setInterval( msecsInterval );
    timer->setTimerType( Qt::PreciseTimer );

    connect( timer, SIGNAL(timeout()),
             this,   SLOT(onTimeout()) );
}

void BaseImitator::start()
{
    timer->start();
}

void BaseImitator::stop()
{
    timer->stop();
}

// imitatorins.h:
#pragma once

#include "baseimitator.h"

#include <NavLib/Matrix>

class Settings;
class QTimer;
```

```
//!
//! \brief The ImitatorINS class Имитатор ИНС
//!
class ImitatorINS : public BaseImitator
{
    Q_OBJECT

public:
    explicit ImitatorINS(const QHostAddress & address,
                         int port,
                         Settings * settings,
                         int msecsInterval,
                         QObject  * parent = nullptr);

protected slots:
    virtual void onTimeout();

private:
    Settings * settings;

    Math::Vector<double> X;
    Math::Vector<double> wienerNoise;
};

// imitatorins.cpp:
#include "imitatorins.h"

#include "udpsender.h"
#include "pinknoise.h"

#include <NavLib/Math>
#include <NavLib/Settings>
#include <NavLib/Packets/InsPacket>

#include <QDebug>
#include <QDataStream>

ImitatorINS::ImitatorINS(const QHostAddress & address,
                         int port,
                         Settings * settings,
                         int msecsInterval,
                         QObject  * parent)
    : BaseImitator( address,
                    port,
                    msecsInterval,
                    parent )
    , settings( settings )
    , X { 1*M_PI/180, 0, 0, 0, 0 }
    , wienerNoise { 0, 0, 0 }
{

}

void ImitatorINS::onTimeout()
{
    secsElapsed += secsInterval;

    // params
    double S = settings->Vxg   * secsElapsed;
    double fi = S / settings->r;

    // errors
```

```
    Math::Vector<double> whiteNoise = {
        Math::normrnd(0, settings->sig_gyro),
        Math::normrnd(0, settings->sig_acc_X),
        Math::normrnd(0, settings->sig_acc_Y)
    };

    Math::Vector<double> w = {
        Math::normrnd(0, 0.0120),
        Math::normrnd(0, 0.0096),
        Math::normrnd(0, 0.0096)
    };
    wienerNoise = wienerNoise + w * secsInterval;

    static PinkNoise pinkGenerator;
    Math::Vector<double> pinkNoise = {
        pinkGenerator.tick(),
        pinkGenerator.tick(),
        pinkGenerator.tick()
    };

    X = settings->F * X  +  settings->gamma * ( whiteNoise + wienerNoise +
pinkNoise );

    // result
    InsPacket packet;
    packet.setSecsElapsed   ( secsElapsed );
    packet.setVxg           ( settings->Vxg    + X[1] );
    packet.setS             ( S                + X[2] );
    packet.setVyg           ( settings->Vyg    + X[3] );
    packet.setHeight        ( settings->height + X[4] );
    packet.setFi            ( fi               + X[2]/settings->r );
    packet.setTheta         ( settings->theta  - X[0] );
    packet.setDVxg          ( X[1] );
    packet.setDS            ( X[2] );
    packet.setDVyg          ( X[3] );
    packet.setDh            ( X[4] );
    packet.setBeta          ( X[0] );

    sender->sendData( packet.encode() );
}

// imitatorreal.h:
#pragma once

#include "baseimitator.h"

class Settings;
class QTimer;

//!
//! \brief The ImitatorReal class Имитатор реальных координат ЛА
//!
class ImitatorReal : public BaseImitator
{
    Q_OBJECT

public:
    explicit ImitatorReal(const QHostAddress & address,
                          int port,
                          Settings * settings,
                          int msecsInterval,
                          QObject  * parent = nullptr);
```

```
protected slots:
    virtual void onTimeout();

private:
    Settings * settings;
};

// imitatorreal.cpp:
#include "imitatorreal.h"

#include "udpsender.h"

#include <NavLib/Settings>
#include <NavLib/Packets/RealPacket>

#include <QDebug>
#include <QDataStream>

ImitatorReal::ImitatorReal(const QHostAddress & address,
                           int port,
                           Settings * settings,
                           int msecsInterval,
                           QObject  * parent)
    : BaseImitator( address,
                    port,
                    msecsInterval,
                    parent )
    , settings( settings )
{

}

void ImitatorReal::onTimeout()
{
    secsElapsed += secsInterval;

    // params
    double S  = settings->Vxg   * secsElapsed;
    double fi = S / settings->r;

    // result
    RealPacket packet;
    packet.setSecsElapsed   ( secsElapsed );
    packet.setVxg           ( settings->Vxg );
    packet.setS             ( S             );
    packet.setVyg           ( settings->Vyg );
    packet.setHeight        ( settings->height );
    packet.setFi            ( fi );
    packet.setTheta         ( settings->theta );

    sender->sendData( packet.encode() );
}

// imitatorsns.h:
#pragma once

#include "baseimitator.h"

#include <NavLib/Settings>

//!
//! \brief The ImitatorSNS class Имитатор СНС
//!
```

```
class ImitatorSNS : public BaseImitator
{
    Q_OBJECT

public:
    explicit ImitatorSNS( const QHostAddress & address,
                          int port,
                          int msecsInterval,
                          Settings * settings,
                          const Settings::SigmaSNS & sigma,
                          QObject  * parent = nullptr );

protected slots:
    virtual void onTimeout();

private:
    Settings * settings;
    Settings::SigmaSNS sigma;
};

// imitatorsns.cpp:
#include "imitatorsns.h"

#include "udpsender.h"

#include <NavLib/Math>
#include <NavLib/Settings>
#include <NavLib/Packets/SnsPacket>

#include <QDebug>
#include <QDataStream>

ImitatorSNS::ImitatorSNS(const QHostAddress & address,
                         int port,
                         int msecsInterval,
                         Settings * settings,
                         const Settings::SigmaSNS & sigma,
                         QObject  * parent)
    : BaseImitator( address,
                    port,
                    msecsInterval,
                    parent )
    , settings( settings )
    , sigma( sigma )
{

}

void ImitatorSNS::onTimeout()
{
    secsElapsed += secsInterval;

    // params
    double S = settings->Vxg   * secsElapsed;

    // errors
    QVector<double> delta = {
        Math::normrnd(0, sigma.Vxg),
        Math::normrnd(0, sigma.S),
        Math::normrnd(0, sigma.Vyg),
        Math::normrnd(0, sigma.h)
    };
```

```
    // result
    SnsPacket packet;
    packet.setSecsElapsed   ( secsElapsed );
    packet.setVxg           ( settings->Vxg    + delta[0] );
    packet.setS             ( S                + delta[1] );
    packet.setVyg           ( settings->Vyg    + delta[2] );
    packet.setHeight        ( settings->height + delta[3] );

    sender->sendData( packet.encode() );
}

// core.h:
#pragma once

#include <QObject>

class Settings;
class BaseImitator;

//!
//! \brief The Core class Ядро
//!
class Core : public QObject
{
    Q_OBJECT

public:
    explicit Core(Settings * settings, QObject *parent = nullptr);
    ~Core();

public slots:
    void start();

signals:
    void stopAll();

private:
    void startImitator(BaseImitator * imitator);

    Settings * settings;

    BaseImitator * ins;
    BaseImitator * sns1;
    BaseImitator * sns2;
    BaseImitator * real;
};

// core.cpp:
#include "core.h"

#include "imitatorins.h"
#include "imitatorsns.h"
#include "imitatorreal.h"

#include <NavLib/Settings>

#include <QHostAddress>
#include <QThread>

Core::Core(Settings * settings, QObject *parent) : QObject(parent)
  , settings( settings )
  , ins ( new ImitatorINS (QHostAddress::LocalHost, 4000, settings, settings-
>insMSecsInterval) )
```

```
  , sns1( new ImitatorSNS (QHostAddress::LocalHost, 4001, settings-
>sns1MSecsInterval, settings, settings->sigSns1) )
  , sns2( new ImitatorSNS (QHostAddress::LocalHost, 4002, settings-
>sns2MSecsInterval, settings, settings->sigSns2) )
  , real( new ImitatorReal(QHostAddress::LocalHost, 4003, settings, settings-
>insMSecsInterval) )
{

}

Core::~Core()
{
    emit stopAll();
}

void Core::start()
{
    startImitator(ins);
    startImitator(sns1);
    startImitator(sns2);
    startImitator(real);
}

void Core::startImitator(BaseImitator * imitator)
{
    QThread * thread = new QThread(this);

    imitator->moveToThread(thread);

    connect( thread, SIGNAL(started()),
             imitator,   SLOT(start()) );

    connect( this, SIGNAL(stopAll()),
             imitator, SLOT(stop()));

    connect( this, SIGNAL(stopAll()),
             thread, SLOT(quit()));

    connect( thread, SIGNAL(finished()),
             imitator,   SLOT(deleteLater()) );

    thread->start();
}

// main.cpp:
#include "core.h"

#include <NavLib/Settings>

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    Settings settings;
    settings.parse("../settings.ini");

    Core core(&settings);
    core.start();

    return app.exec();
}
```

```
// Приложение Estimation:
// common.h:
#pragma once

#include <QString>

inline QString toUni( const char * str )
{
    return QString::fromUtf8(str);
}

enum SnsId
{
    SNS1,
    SNS2
};

enum Switch
{
    OFF,
    ON
};

// datagramparser.h:
#pragma once

#include <QObject>

class QByteArray;

//!
//! \brief The DatagramParser class Парсер датаграмм
//!
class DatagramParser : public QObject
{
    Q_OBJECT

public:
    explicit DatagramParser(QObject *parent = nullptr);

    void parseData(QByteArray &ba );

signals:
    void dataReady  ( const QVector<double> & data);
    void errorsReady( const QVector<double> & errors);
};

// datagramparser.cpp:
#include "datagramparser.h"

#include <NavLib/Packets/BasePacket>
#include <NavLib/Packets/RealPacket>
#include <NavLib/Packets/InsPacket>
#include <NavLib/Packets/SnsPacket>

#include <QDataStream>
#include <QIODevice>

DatagramParser::DatagramParser(QObject *parent) : QObject(parent)
{

}
```

```
void DatagramParser::parseData(QByteArray & ba)
{
    BasePacket * packet = nullptr;

    quint8 packetId = ba.at(0);
    switch ( packetId )
    {
        case 0:
        {
            packet = new InsPacket();
            break;
        }

        case 1:
        {
            packet = new SnsPacket();
            break;
        }

        case 2:
        {
            packet = new RealPacket();
            break;
        }

        default:
        {
            qDebug() << "Error! Unknown Id received:" << packetId;
            return;
        }
    }

    packet->decode( ba );

    // checksum control
    QByteArray data = packet->toByteArray();

    if ( qChecksum( data.data(), data.size() ) != packet->getChecksum() )
    {
        qDebug() << "Error! Checksum mismatch";
        delete packet;
        return;
    }

    // data ready
    emit dataReady( packet->toVector() );

    if ( InsPacket * insPacket = dynamic_cast<InsPacket*>(packet) )
    {
        emit errorsReady( insPacket->getErrorsVector() );
    }

    delete packet;
}

// nsserver.h:
#pragma once

#include <QObject>

class QUdpSocket;
class DatagramParser;
```

```
//!
//! \brief The NSServer class Навигационный сервер
//!
class NSServer : public QObject
{
    Q_OBJECT

public:
    NSServer(int port, QObject *parent = nullptr);

public slots:
    void start();

signals:
    void dataReceived( const QVector<double> & data );
    void errorReceived( const QVector<double> & data );

private slots:
    void readData();

private:
    void connections();

    QUdpSocket     * socket;
    DatagramParser * parser;
    int port;
};

// nsserver.cpp:
#include "nsserver.h"

#include "datagramparser.h"

#include <QUdpSocket>

NSServer::NSServer(int port, QObject * parent)
    : QObject(parent)
    , socket( new QUdpSocket(this) )
    , parser( new DatagramParser(this) )
    , port( port )
{

}

void NSServer::start()
{
    connections();

    if ( !socket->bind(QHostAddress::LocalHost, port) )
    {
        qDebug() << "Error! Socket can not bind on port: "  << port;
        exit(-1);
    }
    else
        qDebug() << "Server started at port " << port;
}

void NSServer::readData()
{
    QByteArray received;

    received.resize( socket->pendingDatagramSize() );
```

```
    socket->readDatagram( received.data(), received.size() );

    parser->parseData( received );

}

void NSServer::connections()
{
    connect( socket, SIGNAL(readyRead()),
             this,     SLOT(readData()));

    connect( parser, SIGNAL(   dataReady(QVector<double>)),
               this, SIGNAL(dataReceived(QVector<double>)) );

    connect( parser, SIGNAL(  errorsReady(QVector<double>)),
               this, SIGNAL(errorReceived(QVector<double>)) );
}

// estimator.h:
#pragma once

#include "common.h"

#include <NavLib/Matrix>

#include <QObject>

class Settings;

//!
//! \brief The Estimator class Оценщик
//!
class Estimator : public QObject
{
    Q_OBJECT

public:
    explicit Estimator(Settings * settings, QObject *parent = nullptr);

public slots:
    void insReceived( const QVector<double> &dataIns );
    void sns1Received(const QVector<double> &dataSns );
    void sns2Received(const QVector<double> &dataSns );

signals:
    void estimationReady( const QVector<double> &dataEstimation );
    void errorEstimationReady( const QVector<double> &dataEstimation );
    void differenceWithSns1Ready( const QVector<double> &data );
    void differenceWithSns2Ready( const QVector<double> &data );

protected:
    virtual void timerEvent( QTimerEvent* );

private:
    inline void extrapolate();

    void estimate( const QVector<double> & dataSns );

    void init();

    void snsReceived(const QVector<double> &dataSns );
```

```
    void sendDifference(const QVector<double> &dataSns);

    SnsId snsIdReceived;

    Settings * settings;

    bool isInsReceived;
    bool isNoSnsSignals;
    int snsDataCounter;

    double currentEstimationTime;

    Math::Vector<double> dataSnsVector;
    Math::Vector<double> dataInsVector;

    Math::Vector<double> xEstimation;
    Math::Matrix<double> P;

    Math::Vector<double> xExtrap;
    Math::Matrix<double> pExtrap;

    const Math::Matrix<double> * R;
};

// estimator.cpp:
#include "estimator.h"

#include <NavLib/Settings>

Estimator::Estimator(Settings * settings, QObject *parent) : QObject(parent)
  , settings( settings )
  , isInsReceived( false )
  , isNoSnsSignals( false )
  , snsDataCounter(0)
  , P(5, 5)
  , pExtrap(5, 5)
  , R(nullptr)
{
    init();
    startTimer(1000);
}

void Estimator::insReceived(const QVector<double> & dataIns)
{
    isInsReceived = true;

    dataInsVector = dataIns;
    currentEstimationTime = dataIns.at(0);

    // extrapolate on 1 INS step
    extrapolate();

    if ( isNoSnsSignals )
    {
        emit estimationReady( QVector<double> { currentEstimationTime,
                                                dataIns.at(1) - xEx-
trap.at(1),
                                                dataIns.at(2) - xEx-
trap.at(2),
                                                dataIns.at(3) - xEx-
trap.at(3),
                                                dataIns.at(4) - xEx-
trap.at(4),
```

```
                                                    dataIns.at(5) - xEx-
trap.at(2)/settings->r,
                                                    dataIns.at(6) + xExtrap.at(0)
                                                   } );

        emit errorEstimationReady( QVector<double> { currentEstimationTime,
                                                     xExtrap.at(1),
                                                     xExtrap.at(2),
                                                     xExtrap.at(3),
                                                     xExtrap.at(4),
                                                     xExtrap.at(0)
                                                    } );
    }
}

void Estimator::sns1Received(const QVector<double> &dataSns)
{
    snsIdReceived = SNS1;
    R = &settings->R1;
    snsReceived( dataSns );
}

void Estimator::sns2Received(const QVector<double> &dataSns)
{
    snsIdReceived = SNS2;
    R = &settings->R2;
    snsReceived( dataSns );
}

void Estimator::timerEvent(QTimerEvent *)
{
    if ( snsDataCounter == 0 )
        isNoSnsSignals = true;
    else
        isNoSnsSignals = false;

    snsDataCounter = 0;
}

void Estimator::extrapolate()
{
    xExtrap = settings->F * xExtrap;
    pExtrap = settings->F * pExtrap * settings->F.transpose()  +
            settings->gamma * settings->Q * settings->gamma.transpose();
}

void Estimator::estimate(const QVector<double> & dataSns)
{
    this->dataSnsVector = dataSns;

    // Prepare INS data vector
    // save fi and theta
    double fi    = dataInsVector[5];
    double theta = dataInsVector[6];
    // then remove fi and theta
    dataInsVector.pop_back(); // remove theta
    dataInsVector.pop_back(); // remove fi

    // Update
    Math::Vector<double> z = dataInsVector - dataSnsVector;
    z.erase( z.begin() ); // erase first element - time (no need in further
equations)
```

```
    Math::Vector<double> v = z - settings->H * xExtrap;

    Math::Matrix<double> S = settings->H * pExtrap * settings->H.transpose()
+ *R;

    Math::Matrix<double> K = pExtrap * settings->H.transpose() * S.inv();

    // INS error optimal estimations, and matrix P
    xEstimation = xExtrap + K * v;
    P = pExtrap - K * settings->H * pExtrap;

    // Save optimal estimations to next extrapolation step
    xExtrap = xEstimation;
    pExtrap = P;

    // Send result
    emit estimationReady( QVector<double> { currentEstimationTime,
                                            dataInsVector.at(1) - xEstima-
tion.at(1),
                                            dataInsVector.at(2) - xEstima-
tion.at(2),
                                            dataInsVector.at(3) - xEstima-
tion.at(3),
                                            dataInsVector.at(4) - xEstima-
tion.at(4),
                                            fi - xEstimation.at(2)/settings-
>r,
                                            theta + xEstimation.at(0)
                                          });

    emit errorEstimationReady( QVector<double> { currentEstimationTime,
                                                 xEstimation.at(1),
                                                 xEstimation.at(2),
                                                 xEstimation.at(3),
                                                 xEstimation.at(4),
                                                 xEstimation.at(0)
                                               } );
}

void Estimator::init()
{
    xEstimation = Math::Vector<double>{ 0, 0, 0, 0, 0 };
    P = {
            { settings->sig_gyro*settings->sig_gyro,   0, 0, 0, 0 },
            { 0, settings->sig_acc_X*settings->sig_acc_X, 0, 0, 0 },
            { 0, 0,                                     2*2, 0, 0 },
            { 0, 0, 0, settings->sig_acc_Y*settings->sig_acc_Y, 0 },
            { 0, 0, 0, 0,                                  2*2 }
        };

    xExtrap = xEstimation;
    pExtrap = P;
}

void Estimator::snsReceived(const QVector<double> & dataSns)
{
    ++snsDataCounter;

    if ( isInsReceived )
    {
        sendDifference( dataSns );
        estimate( dataSns );
    }
```

```
    isInsReceived = false;
}

void Estimator::sendDifference(const QVector<double> &dataSns)
{
    QVector<double> diff = { currentEstimationTime,
                             dataInsVector.at(1) - dataSns.at(1),
                             dataInsVector.at(2) - dataSns.at(2),
                             dataInsVector.at(3) - dataSns.at(3),
                             dataInsVector.at(4) - dataSns.at(4)
                           };

    if      ( snsIdReceived == SNS1 )
        emit differenceWithSns1Ready( diff );

    else if ( snsIdReceived == SNS2 )
        emit differenceWithSns2Ready( diff );
}

// plotwidget.h:
#pragma once

#include <QWidget>

class QCustomPlot;

//!
//! \brief The PlotWidget class Виджет-график
//!
class PlotWidget : public QWidget
{
    Q_OBJECT

public:
    PlotWidget(QWidget * parent = nullptr);

    void addTitle(const QString & title);

    void setXRange(const double xMin, const double xMax);
    void setYRange(const double yMin, const double yMax);

    enum GraphId
    {
        INS,
        SNS1,
        SNS2,
        ESTIMATION,
        REAL
    };

    void addPoint(double x, double y, GraphId Id);

private:
    void setInteractions();
    void setGraphs();
    void setupLayout();

    QCustomPlot * plot;
};

// plotwidget.cpp:
#include "plotwidget.h"
```

```
#include "common.h"

#include "qcustomplot.h"

PlotWidget::PlotWidget(QWidget * parent) : QWidget(parent)
  , plot( new QCustomPlot(this) )
{
    setInteractions();
    setGraphs();
    setupLayout();

    plot->xAxis->setLabel(toUni("t, c"));
}

void PlotWidget::addTitle(const QString & title)
{
    plot->plotLayout()->insertRow(0);
    plot->plotLayout()->addElement(0, 0, new QCPTextElement(plot, title));
}

void PlotWidget::setXRange(const double xMin, const double xMax)
{
    plot->xAxis->setRange(xMin, xMax);
}

void PlotWidget::setYRange(const double yMin, const double yMax)
{
    plot->yAxis->setRange(yMin, yMax);
}

void PlotWidget::addPoint(double x, double y, GraphId Id)
{
    plot->graph( Id )->addData(x, y);
    plot->replot();
}

void PlotWidget::setInteractions()
{
    plot->setInteraction(QCP::iRangeZoom);
    plot->setInteraction(QCP::iRangeDrag);
}

void PlotWidget::setGraphs()
{
    plot->addGraph();
    plot->graph(INS)->setPen(QPen(Qt::magenta));

    plot->addGraph();
    plot->graph(SNS1)->setPen(QPen(Qt::cyan));

    plot->addGraph();
    plot->graph(SNS2)->setPen(QPen(Qt::darkYellow));

    plot->addGraph();
    plot->graph(ESTIMATION)->setPen(QPen(Qt::blue));

    plot->addGraph();
    plot->graph(REAL)->setPen(QPen(Qt::green));
}

void PlotWidget::setupLayout()
{
    QVBoxLayout * mainLO = new QVBoxLayout;
```

```
    mainLO->addWidget( plot );
    setLayout( mainLO );
}

// coordinateswidget.h:
#pragma once

#include <QWidget>

class PlotWidget;
class Settings;

//!
//! \brief The CoordinatesWidget class Виджет с графиками координат ЛА
//!
class CoordinatesWidget : public QWidget
{
    Q_OBJECT

public:
    explicit CoordinatesWidget(Settings * settings, QWidget *parent =
nullptr);

public slots:
    void addPointsIns(         const QVector<double> & points );
    void addPointsSns1(        const QVector<double> & points );
    void addPointsSns2(        const QVector<double> & points );
    void addPointsEstimation( const QVector<double> & points );
    void addPointsReal(        const QVector<double> & points );

private:
    void initWidgets();
    void initLayout();
    void setTitles();
    void setRanges();

    Settings * settings;

    const int rows;
    const int cols;
    QVector< PlotWidget* > widgets;
};

// coordinateswidget.cpp:
#include "coordinateswidget.h"

#include "plotwidget.h"
#include "common.h"

#include <NavLib/Settings>

#include <QGridLayout>

CoordinatesWidget::CoordinatesWidget(Settings * settings, QWidget *parent) :
QWidget(parent)
  , settings( settings )
  , rows(2)
  , cols(3)
  , widgets( rows*cols )
{
    initWidgets();
    initLayout();
}
```

```
void CoordinatesWidget::addPointsIns(const QVector<double> & points)
{
    for( int i = 0;  i < widgets.size();  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::INS );
}

void CoordinatesWidget::addPointsSns1(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size()-2;  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::SNS1 );
}

void CoordinatesWidget::addPointsSns2(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size()-2;  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::SNS2 );
}

void CoordinatesWidget::addPointsEstimation(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size();  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::ESTIMATION );
}

void CoordinatesWidget::addPointsReal(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size();  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::REAL );
}

void CoordinatesWidget::initWidgets()
{
    for( int i = 0; i < widgets.size(); ++i )
            widgets[i] = new PlotWidget();

    setTitles();
    setRanges();
}

void CoordinatesWidget::initLayout()
{
    QGridLayout * mainLO = new QGridLayout(this);

    // plots
    for( int i = 0, offset = 0;   i < rows;   ++i, offset += cols )
        for(    int j = 0;  j < cols;  ++j      )
            mainLO->addWidget( widgets[j+offset], i, j );

    setLayout(mainLO);
}

void CoordinatesWidget::setTitles()
{
    int i = 0;
    widgets.at(i)->addTitle("Vxg, " + toUni("м/с"));

    ++i;
```

```
    widgets.at(i)->addTitle("S, " + toUni("м"));

    ++i;
    widgets.at(i)->addTitle("Vyg, " + toUni("м/с"));

    ++i;
    widgets.at(i)->addTitle("h, " + toUni("м"));

    ++i;
    const QString FI_UNICODE("\u03c6");
    widgets.at(i)->addTitle(FI_UNICODE+", " + toUni("рад"));

    ++i;
    const QString TETA_UNICODE("\u03d1");
    widgets.at(i)->addTitle(TETA_UNICODE+", " + toUni("рад"));
}

void CoordinatesWidget::setRanges()
{
    // oX
    foreach( PlotWidget * w,  widgets )
        w->setXRange(0, settings->tEnd);

    // oY
    int i = 0;
    widgets.at(i)->setYRange(204, 213);

    ++i;
    widgets.at(i)->setYRange(0, 9000);

    ++i;
    widgets.at(i)->setYRange(-1, 1);

    ++i;
    widgets.at(i)->setYRange(6920, 7080);

    ++i;
    widgets.at(i)->setYRange(0, 1.4e-3);

    ++i;
    widgets.at(i)->setYRange(0.03, 0.055);
}

// errorswidget.h:
#pragma once

#include <QWidget>

class PlotWidget;
class Settings;

//!
//! \brief The ErrorsWidget class Виджет с графиками ошибок ИНС
//!
class ErrorsWidget : public QWidget
{
    Q_OBJECT

public:
    explicit ErrorsWidget(Settings * settings, QWidget *parent = nullptr);

public slots:
    void addPointsSns1(       const QVector<double> & points );
```

```
    void addPointsSns2(       const QVector<double> & points );
    void addPointsEstimation( const QVector<double> & points );
    void addPointsReal(       const QVector<double> & points );

private:
    void initWidgets();
    void initLayout();
    void setTitles();
    void setRanges();

    Settings * settings;

    const int rows;
    const int cols;
    QVector< PlotWidget* > widgets;
};

// errorswidget.cpp:
#include "errorswidget.h"

#include "plotwidget.h"
#include "common.h"

#include <NavLib/Settings>

#include <QGridLayout>

ErrorsWidget::ErrorsWidget(Settings * settings, QWidget *parent) :
QWidget(parent)
  , settings( settings )
  , rows(2)
  , cols(3)
  , widgets( rows*cols - 1 )
{
    initWidgets();
    initLayout();
}

void ErrorsWidget::addPointsSns1(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size()-1;  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::SNS1 );
}

void ErrorsWidget::addPointsSns2(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size()-1;  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::SNS2 );
}

void ErrorsWidget::addPointsEstimation(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size();  ++i )
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::ESTIMATION );
}

void ErrorsWidget::addPointsReal(const QVector<double> &points)
{
    for( int i = 0;  i < widgets.size();  ++i )
```

```
        widgets.at(i)->addPoint( points.at(0), points.at(i+1), Plot-
Widget::REAL );
}

void ErrorsWidget::initWidgets()
{
    for( int i = 0; i < widgets.size(); ++i )
            widgets[i] = new PlotWidget();

    setTitles();
    setRanges();
}

void ErrorsWidget::initLayout()
{
    QGridLayout * mainLO = new QGridLayout(this);

    // plots
    for( int i = 0, offset = 0;   i < rows;   ++i, offset += cols )
        for(    int j = 0;  j < cols;  ++j      )
            if ( j+offset != widgets.size() )
                mainLO->addWidget( widgets[j+offset], i, j );

    setLayout(mainLO);
}

void ErrorsWidget::setTitles()
{
    const QString DELTA_UNICODE("\u0394");

    int i = 0;
    widgets.at(i)->addTitle(DELTA_UNICODE+"Vxg, " + toUni("м/с"));

    ++i;
    widgets.at(i)->addTitle(DELTA_UNICODE+"S, " + toUni("м"));

    ++i;
    widgets.at(i)->addTitle(DELTA_UNICODE+"Vyg, " + toUni("м/с"));

    ++i;
    widgets.at(i)->addTitle(DELTA_UNICODE+"h, " + toUni("м"));

    ++i;
    const QString BETA_UNICODE("\u03b2");
    widgets.at(i)->addTitle(BETA_UNICODE+", " + toUni("рад"));
}

void ErrorsWidget::setRanges()
{
    // oX
    foreach( PlotWidget * w,  widgets )
        w->setXRange(0, settings->tEnd);

    // oY
    int i = 0;
    widgets.at(i)->setYRange(0, 7);          // Vxg

    ++i;
    widgets.at(i)->setYRange(-40, 140);      // S

    ++i;
    widgets.at(i)->setYRange(-1, 1);         // Vyg
```

```
    ++i;
    widgets.at(i)->setYRange(-40, 50);       // h

    ++i;
    widgets.at(i)->setYRange(0, 0.018);      // beta
}

// mainwidget.h:
#pragma once

#include "common.h"

#include <QWidget>

class Settings;
class CoordinatesWidget;
class ErrorsWidget;
class QTabWidget;
class QPushButton;

//!
//! \brief The MainWidget class Виджет главного окна
//!
class MainWidget : public QWidget
{
    Q_OBJECT

public:
    explicit MainWidget(Settings *settings, QWidget *parent = nullptr);

public slots:
    void addPointsIns(        const QVector<double> & points );
    void addPointsErrorIns( const QVector<double> & points );

    void addPointsSns1(       const QVector<double> & points );
    void addPointsDifferenceWithSns1( const QVector<double> & points );

    void addPointsSns2(       const QVector<double> & points );
    void addPointsDifferenceWithSns2( const QVector<double> & points );

    void addPointsEstimation( const QVector<double> & points );
    void addPointsErrorEstimation( const QVector<double> & points );

    void addPointsReal(       const QVector<double> & points );

signals:
    void switchSns(SnsId id, Switch s);

private slots:
    void handleBtn1Click();
    void handleBtn2Click();

private:
    void connections();
    void initButtons();
    void initLayout();
    void handleClick( QPushButton * btn, SnsId id );

    Settings * settings;
    QTabWidget * tabWidget;
    CoordinatesWidget * coordsWidget;
    ErrorsWidget * errWidget;
```

```
    QString onStr;
    QString offStr;
    QString onStyleSheet;
    QString offStyleSheet;
    QPushButton * switchSns1Btn;
    QPushButton * switchSns2Btn;
};

// mainwidget.cpp:
#include "mainwidget.h"

#include "coordinateswidget.h"
#include "errorswidget.h"

#include <QTabWidget>
#include <QVBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QGroupBox>

MainWidget::MainWidget(Settings * settings, QWidget *parent) :
QWidget(parent)
  , settings( settings )
  , tabWidget( new QTabWidget(this) )
  , coordsWidget( new CoordinatesWidget(settings) )
  , errWidget( new ErrorsWidget(settings) )
  , onStr(  toUni("Включить") )
  , offStr( toUni("Выключить") )
  , onStyleSheet("color: green; font-size: 12px; font-weight: bold")
  , offStyleSheet("color: red; font-size: 12px; font-weight: bold")
  , switchSns1Btn( new QPushButton() )
  , switchSns2Btn( new QPushButton() )
{
    connections();
    initButtons();
    initLayout();

    setMinimumSize(1280, 720);
    showMaximized();
}

void MainWidget::addPointsIns(const QVector<double> &points)
{
    coordsWidget->addPointsIns(points);
}

void MainWidget::addPointsErrorIns(const QVector<double> &points)
{
    errWidget->addPointsReal(points);
}

void MainWidget::addPointsSns1(const QVector<double> &points)
{
    coordsWidget->addPointsSns1(points);
}

void MainWidget::addPointsDifferenceWithSns1(const QVector<double> &points)
{
    errWidget->addPointsSns1(points);
}

void MainWidget::addPointsSns2(const QVector<double> &points)
{
```

```
    coordsWidget->addPointsSns2(points);
}

void MainWidget::addPointsDifferenceWithSns2(const QVector<double> &points)
{
    errWidget->addPointsSns2(points);
}

void MainWidget::addPointsEstimation(const QVector<double> &points)
{
    coordsWidget->addPointsEstimation(points);
}

void MainWidget::addPointsErrorEstimation(const QVector<double> &points)
{
    errWidget->addPointsEstimation(points);
}

void MainWidget::addPointsReal(const QVector<double> &points)
{
    coordsWidget->addPointsReal(points);
}

void MainWidget::handleBtn1Click()
{
    handleClick( switchSns1Btn, SNS1 );
}

void MainWidget::handleBtn2Click()
{
    handleClick( switchSns2Btn, SNS2 );
}

void MainWidget::connections()
{
    connect( switchSns1Btn, SIGNAL(clicked()),
             this, SLOT(handleBtn1Click()));

    connect( switchSns2Btn, SIGNAL(clicked()),
             this, SLOT(handleBtn2Click()));
}

void MainWidget::initButtons()
{
    switchSns1Btn->setText(offStr);
    switchSns2Btn->setText(offStr);

    switchSns1Btn->setStyleSheet(offStyleSheet);
    switchSns2Btn->setStyleSheet(offStyleSheet);

    switchSns1Btn->setFixedWidth(100);
    switchSns2Btn->setFixedWidth(100);

    switchSns1Btn->setFixedHeight(35);
    switchSns2Btn->setFixedHeight(35);
}

void MainWidget::initLayout()
{
    QVBoxLayout * mainLO = new QVBoxLayout();

    // tabs
    tabWidget->addTab( coordsWidget, toUni("Координаты") );
```

```
    tabWidget->addTab( errWidget, toUni("Ошибки") );
    mainLO->addWidget( tabWidget );

    // legend
    QHBoxLayout * bottomLO = new QHBoxLayout();

    QGroupBox * legendBox = new QGroupBox( toUni("Обозначения") );
    legendBox->setFixedHeight(70);
    legendBox->setFont( QFont("Courier New", 10, QFont::Bold) );
    QHBoxLayout * legendBoxLO = new QHBoxLayout();

    QVBoxLayout * legendLO1 = new QVBoxLayout();
    QLabel * real = new QLabel("<font color=#00FF00>\u2014</font>" +
toUni("Реальный параметр"));
    QLabel * estimation = new QLabel("<font color=blue>\u2014</font>" +
toUni("Оценка"));
    legendLO1->addWidget(real);
    legendLO1->addWidget(estimation);
    legendBoxLO->addLayout(legendLO1);

    QVBoxLayout * legendLO2 = new QVBoxLayout();
    QLabel * sns1 = new QLabel("<font color=cyan>\u2014</font>" +
toUni("Измерение СНС1"));
    QLabel * sns2 = new QLabel("<font color=#9b870c>\u2014</font>" +
toUni("Измерение СНС2"));
    legendLO2->addWidget(sns1);
    legendLO2->addWidget(sns2);
    legendBoxLO->addLayout(legendLO2);

    QLabel * ins = new QLabel("<font color=#FF00FF>\u2014</font>" +
toUni("Измерение ИНС"));
    legendBoxLO->addWidget(ins);
    legendBox->setLayout(legendBoxLO);

    bottomLO->addWidget(legendBox);

    bottomLO->addSpacerItem( new QSpacerItem(100, 20) );

    // buttons
    QGroupBox * sns1Box = new QGroupBox( toUni("СНС1") );
    sns1Box->setFixedHeight(70);
    sns1Box->setFont( QFont("Courier New", 10, QFont::Bold) );
    QHBoxLayout * sns1BoxLO = new QHBoxLayout();
    sns1BoxLO->addWidget( switchSns1Btn );
    sns1Box->setLayout(sns1BoxLO);
    bottomLO->addWidget(sns1Box);

    QGroupBox * sns2Box = new QGroupBox( toUni("СНС2") );
    sns2Box->setFixedHeight(70);
    sns2Box->setFont( QFont("Courier New", 10, QFont::Bold) );
    QHBoxLayout * sns2BoxLO = new QHBoxLayout();
    sns2BoxLO->addWidget( switchSns2Btn );
    sns2Box->setLayout(sns2BoxLO);
    bottomLO->addWidget(sns2Box);

    bottomLO->addStretch();

    mainLO->addLayout( bottomLO );
    setLayout(mainLO);
}

void MainWidget::handleClick(QPushButton *btn, SnsId id)
{
```

```
    if      ( btn->text() == QString(offStr) )
    {
        emit switchSns(id, OFF);
        btn->setText(onStr);
        btn->setStyleSheet(onStyleSheet);
    }

    else if ( btn->text() == QString(onStr) )
    {
        emit switchSns(id, ON);
        btn->setText(offStr);
        btn->setStyleSheet(offStyleSheet);
    }
}

// core.h:
#pragma once

#include "common.h"

#include <QObject>

class NSServer;
class Estimator;
class Settings;
class MainWidget;

//!
//! \brief The Core class Ядро
//!
class Core : public QObject
{
    Q_OBJECT

public:
    explicit Core(Settings * settings, QObject *parent = nullptr);
    ~Core();

public slots:
    void start();

signals:
    void stopAll();

private slots:
    void switchSns(SnsId id, Switch s);

private:
    void makeConnections();
    void startServers();
    void startServer(NSServer * server);
    void startEstimator();

    Settings * settings;

    NSServer * serverIns;
    NSServer * serverSns1;
    NSServer * serverSns2;
    NSServer * serverReal;

    Estimator * estimator;

    MainWidget * widget;
```

```
};

// core.cpp:
#include "core.h"

#include "nsserver.h"
#include "estimator.h"
#include "mainwidget.h"

#include <NavLib/Settings>

#include <QThread>

Core::Core(Settings * settings, QObject *parent) : QObject(parent)
  , settings  ( settings )
  , serverIns ( new NSServer(4000) )
  , serverSns1( new NSServer(4001) )
  , serverSns2( new NSServer(4002) )
  , serverReal( new NSServer(4003) )
  , estimator ( new Estimator (settings) )
  , widget    ( new MainWidget(settings) )
{

}

Core::~Core()
{
    delete widget;
    emit stopAll();
}

void Core::start()
{
    makeConnections();
    startEstimator();
    startServers();
}

void Core::switchSns(SnsId id, Switch s)
{
    // prepare signal to switch
    const char * serverSignal = SIGNAL(dataReceived(QVector<double>));

    // prepare server and slots to switch
    NSServer   * server        = nullptr;
    const char * estimatorSlot = nullptr;
    const char * widgetSlot    = nullptr;

    if      ( id == SNS1 )
    {
        server = serverSns1;
        estimatorSlot = SLOT(sns1Received(QVector<double>));
        widgetSlot    = SLOT(addPointsSns1(QVector<double>));
    }
    else if ( id == SNS2 )
    {
        server = serverSns2;
        estimatorSlot = SLOT(sns2Received(QVector<double>));
        widgetSlot    = SLOT(addPointsSns2(QVector<double>));
    }

    // nullptr check
    if ( !server || !estimatorSlot || !widgetSlot )
```

```
        return;

    // switch
    if      ( s == OFF )
    {
        disconnect( server,    serverSignal,
                    estimator, estimatorSlot);

        disconnect( server,    serverSignal,
                    widget,    widgetSlot );
    }
    else if ( s == ON )
    {
        connect( server,    serverSignal,
                 estimator, estimatorSlot);

        connect( server,    serverSignal,
                 widget,    widgetSlot );
    }
}

void Core::makeConnections()
{
    // make type be able to cross threads boundaries
    qRegisterMetaType<QVector<double>>("QVector<double>");

    // coordinates connections
    connect( serverIns, SIGNAL(dataReceived(QVector<double>)),
             estimator,   SLOT(insReceived(QVector<double>)) );

    connect( serverIns, SIGNAL(dataReceived(QVector<double>)),
              widget,     SLOT(addPointsIns(QVector<double>)) );


    connect( serverSns1, SIGNAL(dataReceived(QVector<double>)),
             estimator,    SLOT(sns1Received(QVector<double>)) );

    connect( serverSns1, SIGNAL(dataReceived(QVector<double>)),
             widget,       SLOT(addPointsSns1(QVector<double>)) );


    connect(serverSns2, SIGNAL(dataReceived(QVector<double>)),
            estimator,    SLOT(sns2Received(QVector<double>)) );

    connect( serverSns2, SIGNAL(dataReceived(QVector<double>)),
             widget,       SLOT(addPointsSns2(QVector<double>)) );


    connect( serverReal, SIGNAL(dataReceived(QVector<double>)),
             widget,       SLOT(addPointsReal(QVector<double>)) );


    connect(estimator, SIGNAL(estimationReady(QVector<double>)),
            widget,      SLOT(addPointsEstimation(QVector<double>)) );

    // errors to widget connections
    connect( serverIns, SIGNAL(errorReceived(QVector<double>)),
              widget,     SLOT(addPointsErrorIns(QVector<double>)) );

    connect( estimator, SIGNAL(differenceWithSns1Ready(QVector<double>)),
             widget,      SLOT(addPointsDifferenceWithSns1(QVector<double>))
);
```

```
    connect( estimator, SIGNAL(differenceWithSns2Ready(QVector<double>)),
             widget,      SLOT(addPointsDifferenceWithSns2(QVector<double>))
);

    connect( estimator, SIGNAL(errorEstimationReady(QVector<double>)),
             widget,      SLOT(addPointsErrorEstimation(QVector<double>)) );

    // switch SNS connection
    connect(widget, SIGNAL(switchSns(SnsId,Switch)),
            this,     SLOT(switchSns(SnsId,Switch)));
}

void Core::startServers()
{
    startServer(serverIns);
    startServer(serverSns1);
    startServer(serverSns2);
    startServer(serverReal);
}

void Core::startServer(NSServer * server)
{
    QThread * thread = new QThread(this);

    server->moveToThread(thread);

    connect( thread, SIGNAL(started()),
             server,   SLOT(start()) );

    connect( this, SIGNAL(stopAll()),
             thread, SLOT(quit()));

    connect( thread, SIGNAL(finished()),
             server,   SLOT(deleteLater()) );

    thread->start();
}

void Core::startEstimator()
{
    QThread * thread = new QThread(this);

    estimator->moveToThread(thread);

    connect( this, SIGNAL(stopAll()),
             thread, SLOT(quit()));

    connect( thread,    SIGNAL(finished()),
             estimator,   SLOT(deleteLater()) );

    thread->start();
}

// main.cpp:
#include "core.h"

#include <NavLib/Settings>

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
```

```
    Settings settings;
    settings.parse("../settings.ini");

    Core core(&settings);
    core.start();

    return app.exec();
}

; Файл конфигурации settings.ini
g       = 9.8
R_earth = 6371000

sig_gyro   = 0.02
sig_acc_X  = 0.1
sig_acc_Y  = 0.1

sig_Vxg1 = 0.2
sig_S1   = 6
sig_Vyg1 = 0.3
sig_h1   = 10

coef = 1.3

insMsecsInterval  = 100
sns1MsecsInterval = 200
sns2MsecsInterval = 250
tEnd = 40

Vxg     = 205
theta   = 3
Vyg     = 0
height  = 7000
```

ПРИЛОЖЕНИЕ Д

Результаты моделирования при увеличении СКО на главной диагонали матрицы Q в 100 раз

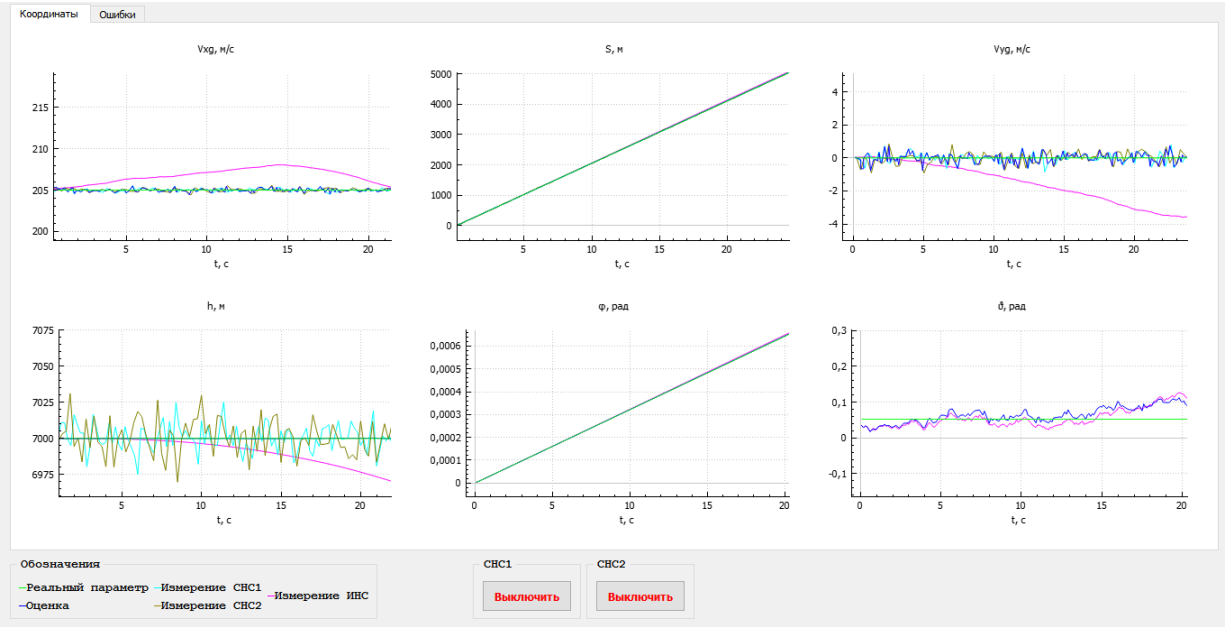

Рисунок Д.1 – Координаты ЛА

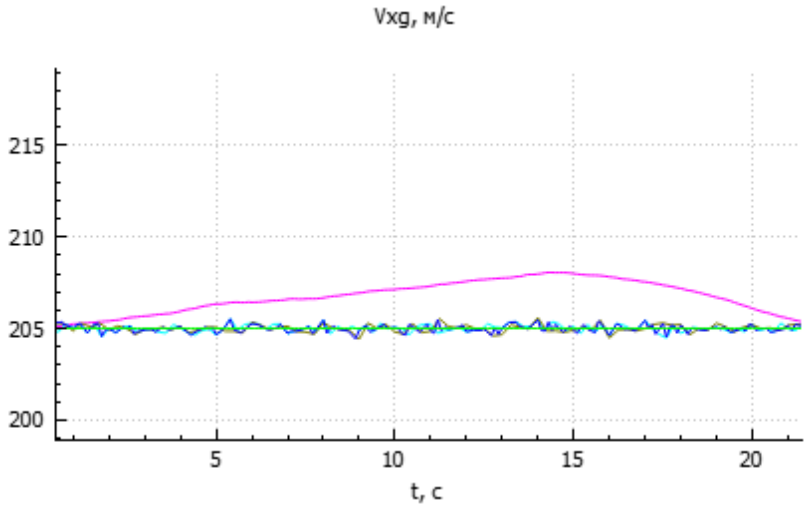

Рисунок Д.2 – Горизонтальная скорость

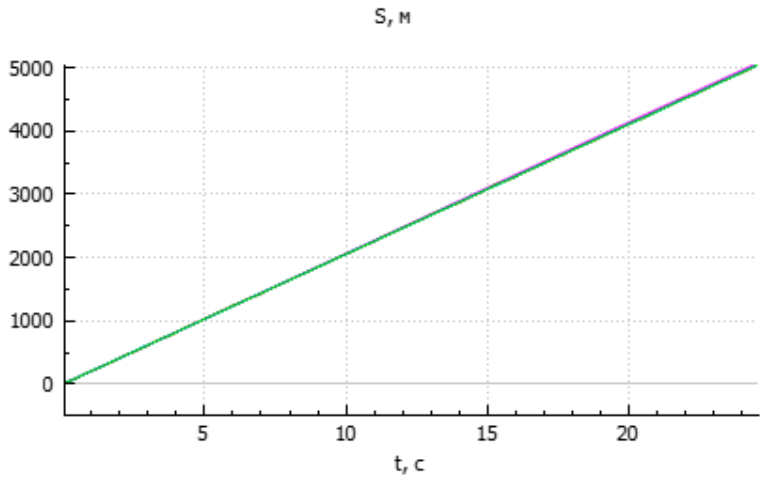

Рисунок Д.3 – Пройденный путь

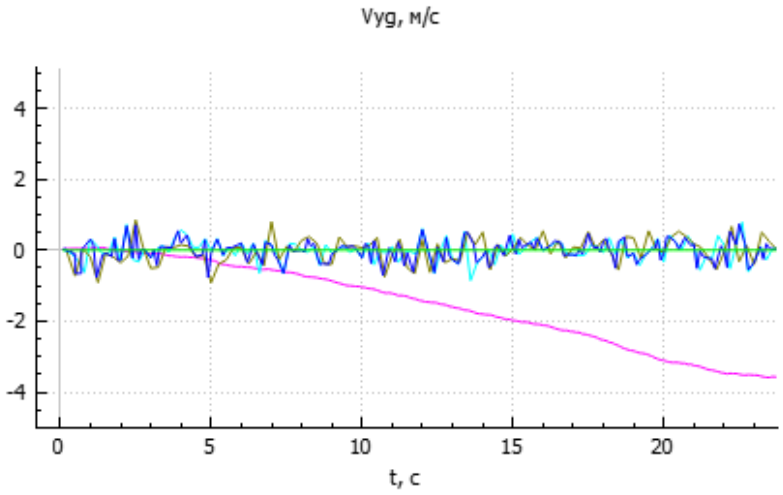

Рисунок Д.4 – Вертикальная скорость

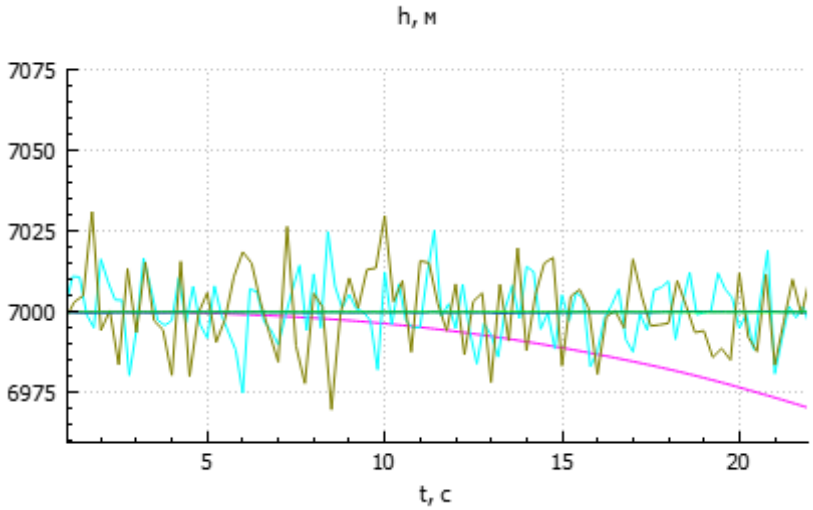

Рисунок Д.5 – Высота

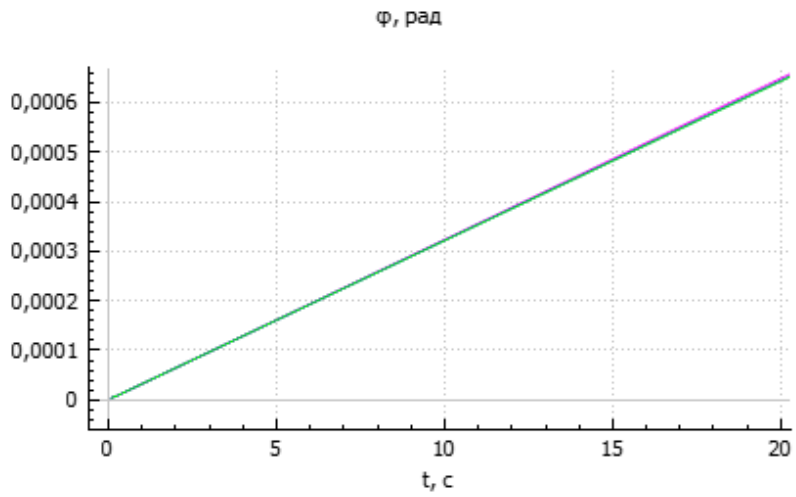

Рисунок Д.6 – Широта

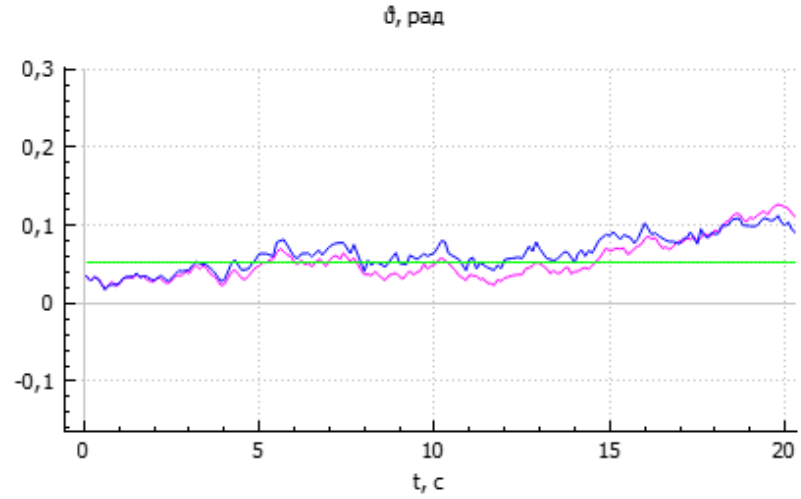

Рисунок Д.7 – Угол тангажа

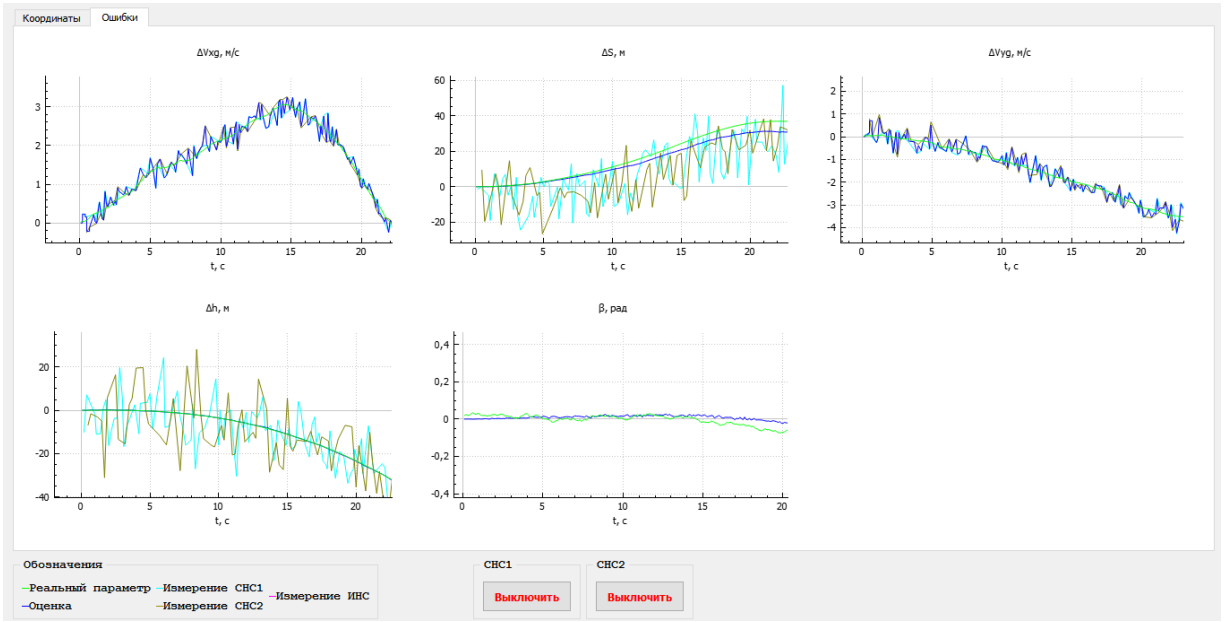

Рисунок Д.8 – Ошибки ИНС

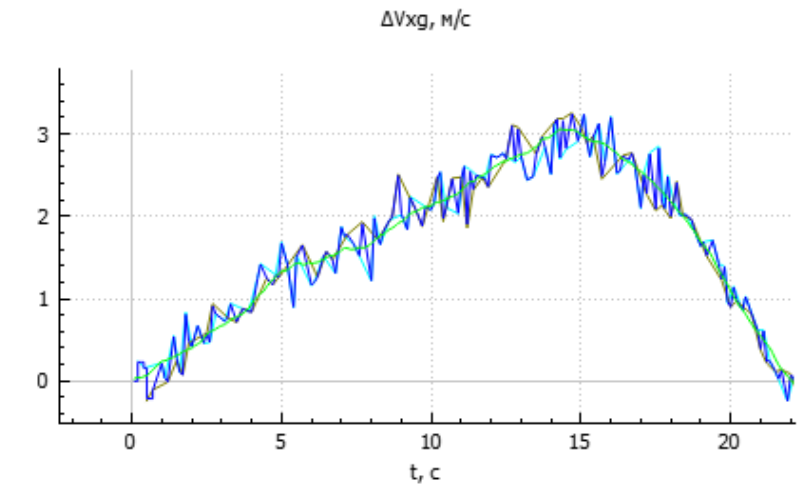

Рисунок Д.9 – Ошибка выработки горизонтальной скорости

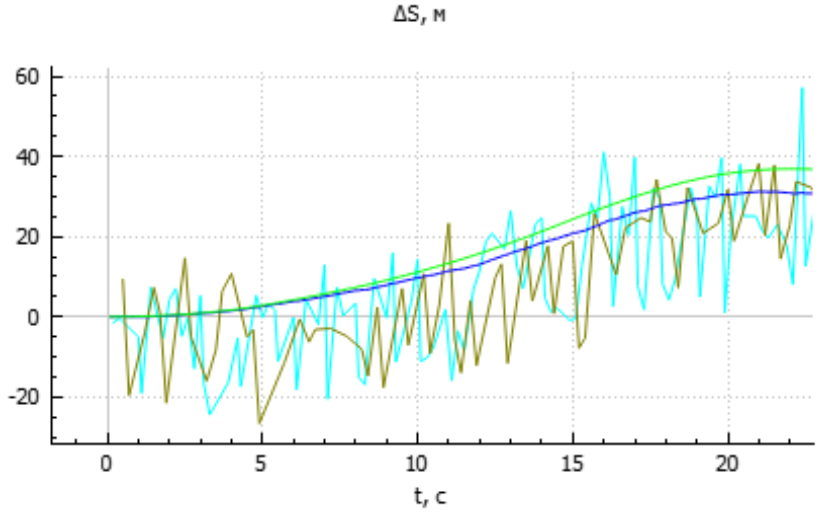

Рисунок Д.10 – Ошибка выработки пройденного расстояния

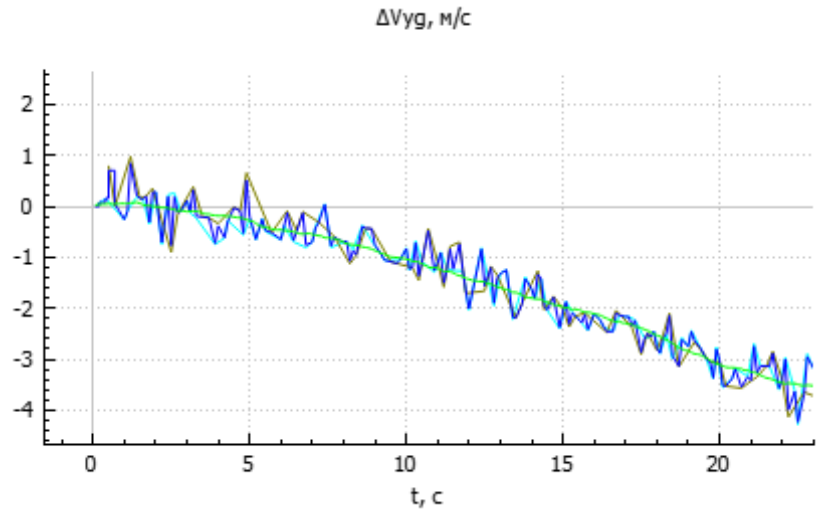

Рисунок Д.11 – Ошибка выработки вертикальной скорости

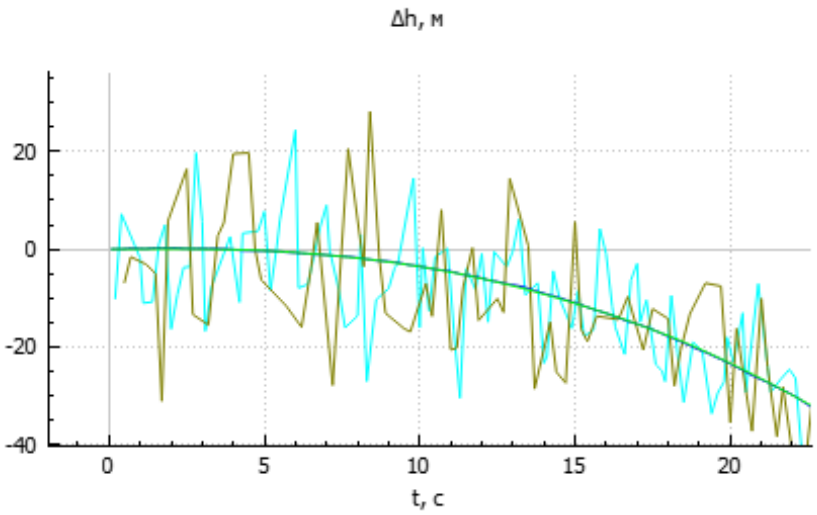

Рисунок Д.12 – Ошибка выработки высоты

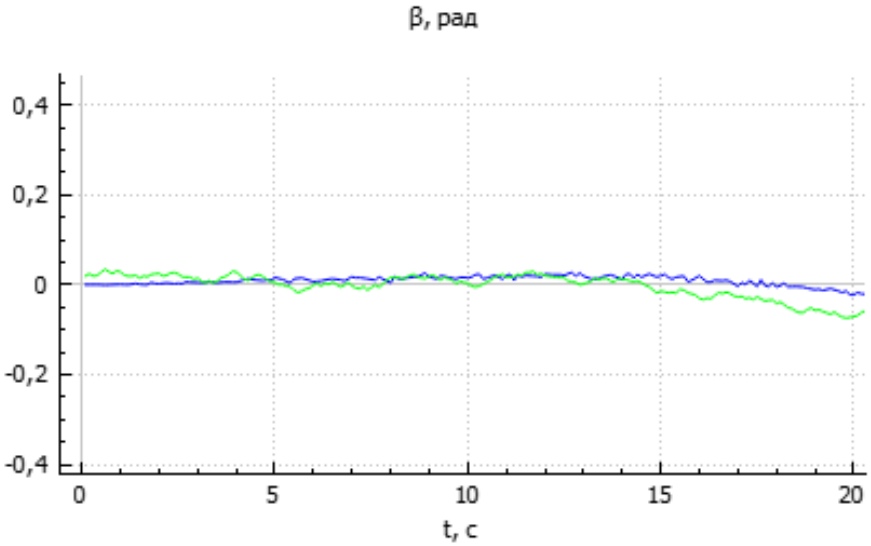

Рисунок Д.13 – Ошибка построения вертикали