Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и механики

Кафедра прикладной математики

Документация программного комплекса

*«Там, где море»*

Выполнили: студенты групп 13641/1,3,4

|  |
| --- |
| Лысикова Наталья |
| Зефиров Артем |
| Пестова Наталья |
| Выменец Максим |
| Тхакушинова Рузанна |

Санкт-Петербург

2017

**Аннотация**

Настоящий документ является описанием программного комплекса для вычислений характеристик движений объектов, наблюдаемых при помощи радиолокатора. Указанный комплекс разработан в рамках учебного курса «Эффективная разработка программного обеспечения», проводимого рамках магистерской программы по направлению «прикладная математика» в Санкт-Петербургском политехническом университете им. Петра Великого и предназначен для использования в учебных целях.

Настоящий документ содержит необходимые сведения о назначении, основных характеристиках, входных и выходных данных, логической структуре программного комплекса.

**Оглавление**

[Словарь терминов 5](#_gjdgxs)

[1.](#_30j0zll) Общие сведения 6

[1.1.](#_1fob9te) Наименование и цель программного комплекса 6

[1.2.](#_3znysh7) ПО, необходимое для функционирования программы 6

[1.3.](#_2et92p0) Языки программирования, на которых написана программа 6

[2.](#_tyjcwt) Функциональное назначение 7

[2.1.](#_3dy6vkm) Решаемые задачи 7

[2.2.](#_1t3h5sf) Функциональные и количественные ограничения 7

[3.](#_4d34og8) Описание логической структуры 8

[3.1.](#_2s8eyo1) Общая структура программы 8

[3.2.](#_17dp8vu) Метод работы 9

[3.3.](#_3rdcrjn) Описание используемых структур данных 10

[3.4.](#_26in1rg) Описание используемых методов 11

[3.5.](#_lnxbz9) Алгоритмы 13

[3.5.1.](#_35nkun2) Алгоритмы, используемые в библиотеке Radar.lib 13

[3.5.2.](#_1ksv4uv) Алгоритмы, используемые в работе компонента GUI 13

[4.](#_2jxsxqh) Вызов и загрузка 16

[5.](#_z337ya) Входные данные 20

[6.](#_3j2qqm3) Выходные данные 21

# Словарь терминов

Радарное изображение – цифровое изображение круга, содержащее на нейтральном фоне контуры объектов и цифровой шум, получаемое в результате радиолокационных наблюдений

Цифровой шум – дефект радарного изображения

Объект на изображении – набор пикселей радарного изображения, соответствующий наблюдаемому физическому объекту

Относительная система координат – декартовая система координат, центром которой является центр радарного изображения, с направлением осей абсцисс и ординат в соответствии с горизонтальным и вертикальным направлением на радарном изображении.

Координаты объекта – среднее арифметическое координат его пикселей в относительной системе координат

Скорость объекта – скорость физического объекта в относительной системе координат

Размер объекта – длина и ширина физического объекта, соответствующая занимаемому на радарном изображении количеству пикселей

Характеристики движения объекта – координаты и скорость объекта

# Общие сведения

## Наименование и цель программного комплекса

Программный комплекс «Там, где море» предназначен для обработки радарных изображений и вычисления характеристик движения и физических параметров идентифицированных на изображениях объектов.

## ПО, необходимое для функционирования программы

Программный комплекс предназначен для программных платформ, имеющих возможность выполнения кода, написанного на языке C++. Это включает:

* выполнение на 32- и 64-битных ОС Linux и Windows для ПК и серверов;
* статическую и динамическую линковку с программными комплексами C++ и других компилируемых языках, предоставляющих возможность статической и динамической линковки с программными библиотеками, предоставляющими C++ интерфейс

## Языки программирования, на которых написана программа

Проект реализован на языке C++ с использованием фреймворка Qt5.

# Функциональное назначение

## Решаемые задачи

Программный комплекс «Там, где море» выполняет обработку радарных изображений, поступающих в файловую систему от радиолокатора.

В процессе первичной обработки устраняется цифровой шум, мешающий выявлению физических объектов на изображения. В результате идентификации физических объектов с учетом входной информации о времени наблюдения и масштабе изображения вычисляются характеристики движения объекта и размеры объекта.

## Функциональные и количественные ограничения

Вывод времени и даты осуществляется в шкале UTC.

Координаты и скорости объекта выводятся в относительной системе координат в (км/ч). Размеры объектов выводятся в км.

Масштаб радарного изображения задается параметром «масштаб» и устанавливается по умолчанию 100 м/пиксель. Данный параметр определяется характеристиками локатора и определяет нижнюю оценку точности, с которой определяются характеристики движения и размер объектов. Значения параметра ограничены диапазоном от 1 до 10000.

# Описание логической структуры

## Общая структура программы

Программный комплекс «Там, где море» состоит из следующих компонент:

1. Исполняемый модуль *Gui*, предоставляющий пользователю интерфейс взаимодействия с программой и обеспечивающего связь с остальными компонентами комплекса.
2. Библиотека *Radar.lib,* осуществляющая анализ радарных изображений и формирующая выходной файл программы.
3. Внешняя библиотека *EasyBMP.lib* для работы с файлами формата BMP.

Входными файлами являются файлы формата BMP, содержащие радарные изображения. Выходными – текстовые файлы, содержащие информацию о характеристиках движения объектах и их размерах, а также лог-файл, содержащий сведения о работе программы.

Компоненты программного комплекса отображены на диаграмме размещения (рисунок 1):

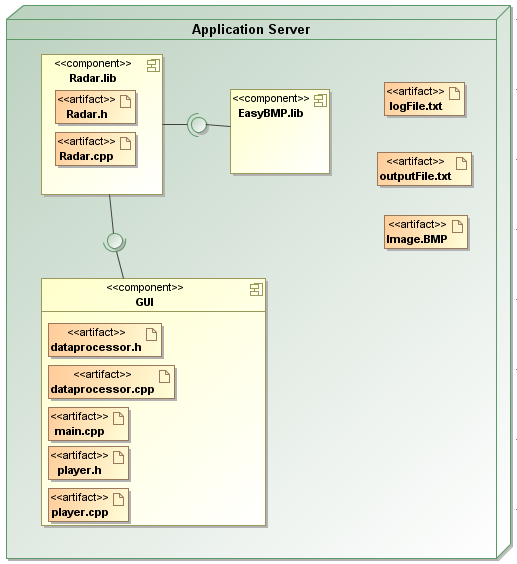


Рисунок 1 – Диаграмма размещения компонент программы

## Метод работы

Работа с библиотекой *Radar.lib* осуществляется по следующей схеме:

1. Инициализация объекта класса Radar.
2. Установка настроек, к которым относятся:

* путь к директории с файлами радарных изображений
* путь к выходному файлу
* указатель на лог-файл, созданный в компоненте GUI
* параметр минимального количества файлов для обработки
* параметр масштаба радарного изображения.

1. Запуск метода обработки файлов, находящихся в очереди в одном из потоков компонента GUI.
2. Уничтожение объекта класса и освобождение ресурсов.

*Примечание:* все указанные операции относятся к одному сеансу работы с *Radar.lib*. Пользователь имеет возможность работать с библиотекой в нескольких сеансах независимо, в этом случае шаги 1-4 должны выполняться независимо для каждого сеанса. Такой сценарий работы имеет смысл при использовании параллельных алгоритмов обработки радарных изображений.

## Описание используемых структур данных

Класс *Radar* является единственным классом библиотеки *Radar.lib* и осуществляет обработку входных радарных изображений и формирование выходного файла с результатами характеристиками движений объектов и их размеров.

Таблица 1. Поля класса *Radar*

|  |  |
| --- | --- |
| **Поле/Поля** | **Описание** |
| int freq | Минимальное количество файлов, обрабатываемых функцией run(). Значение по умолчанию: 1 |
| int YYYY, MM, DD, hh, mm, ss, cc | Год, месяц, день, час, минута, секунда и сотая часть секунды времени наблюдения |
| int oYYYY, oMM, oDD, ohh, omm, oss | Год, месяц, день, час, минута, секунда и сотая часть секунды времени наблюдения предыдущего изображения |
| int blackAndWhitePixels | Количество пикселей изображения, не являющихся пикселями рамки |
| int clearedPixels | Количество пикселей изображения. изменивших цвет при очистке изображения от шума |
| double scale = 100 | Масштаб изображения в метрах на пиксель |
| double noize | Оценка уровня зашумлённости изображения |
| bool firstImage = true | Переменная, в которой хранится, является ли изображение первым в серии. |
| BMP world | Текущее радарное изображение |
| int minMonitoredSize | Параметр, указывающий, насколько большие объекты не следует отбрасывать как ложные с учётом оценки шума |
| const double maxDelta = 10 | Параметр, указывающий, насколько большой допускается значение метрики delta между объектами на предыдущем и на текущем изображении для признания этих объектов одним реальным. |
| vector<double> ox, oy, os, ovx, ovy | Характеристики объектов, детектированных на предыдущем изображении. |
| vector<bool> ot | Характеристика объекта, указывающая, был ли он замечен на предыдущем изображении |
| const RGBApixel white = { 255,255,255,0 } | Белый цвет пикселя |
| const RGBApixel black = { 0,0,0,0 } | Чёрный цвет пикселя |
| const RGBApixel green = { 0,255,0,0 } | Зелёный цвет пикселя |
| const RGBApixel red = { 0,0,255,0 } | Красный цвет пикселя |
| string logFileName | Имя файла для вывода лога работы программы |
| string outputFileName | Имя файла для вывода результатов работы программы |
| FILE \* logFile | Файл для вывода лога работы программы |
| FILE \* outputFile | Файл для вывода рещультатов работы программы |
| int[4] dx = { 0, 1, 0, -1 } | Перемещения по первой координате при переборе соседних пикселей |
| int[4] dy = { 1, 0, -1, 0 } | Перемещения по второй координате при переборе соседних пикселей |

## Описание используемых методов

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **Методы видимости private** | |
| bool iw(int x, int y) | Проверка, что пиксель с данными координатами находится в границах изображения и не является пикселем рамки |
| void cleanPixel(int x, int y) | Локальная очистка шума, метод подробнее описан ниже |
| double clean(); | Глобальная очистка шума, возвращающая оценку уровня шума, метод подробнее описан ниже |
| vector<double> fillingPixel(RGBApixel from, RGBApixel to, int x, int y) | Заливка смежной одноцветной части изображения другим цветом, возвращающая количество и сумму координат залитых пикселей, метод подробнее описан ниже |
| void findObjects(); | Поиск объектов на изображении, метод подробнее описан ниже |
| double delta(double x1, double y1, double s1, double x2, double y2, double s2) | Метрика, определяющая непохожесть объектов. Равна 20 \* sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2)) / (s1 + s2) + max(s1,s2)/min(s1,s2) |
| BMP nextStep(BMP image, bool createRedArrows); | Метод, принимающий новое изображение и определяющий положение и скорости объектов на нём, подробнее описан ниже |
| double timeInterval(); | Метод, вычисляющий время, прошедшее между наблюдением предыдущего и текущего радарного изображения |
| void writeOutput(); | Метод, выводящий характеристики объектов в выходной файл |
| **Методы видимости public** | |
| Radar(double newScale) | Конструктор класса, принимает масштаб изображения |
| ~Radar() | Деструктор класса |
| void setFreq(int newFreq) | Метод изменения минимального количества входных изображений |
| void clearCashe(); | Метод, удаляющий информацию о предыдущих наблюдениях |
| void setOutputFile(const string newOutputFileName) | Метод для задания выходного файла |
| void setLogFile(const string newLogFileName) | Метод для задания файла логирования |
| void setLogFile(FILE \* newLogFile) | Метод для задания файла логирования |
| void setScale(int newScale) | Метод для изменения масштаба |
| void setIdentThreshold(double newThreshold) | Пустой метод, необходимый для взаимодействия с интерфейсом |
| int run(const list<string> inputFileNames, bool createOutputImage, const string outputImageFileName) | Метод для приёма новых изображений и вызова методов обработки. Эта версия метода выводит выходные изображения в целях тестирования |
| int run(const list<string> inputFileNames) | Метод для приёма новых изображений и вызова методов обработки. Эта версия метода не выводит выходные изображений |
| BMP getImage() | Метод, возвращающий текущее изображение в целях тестирования |

**Главный метод nextStep:**

1. Вызов метода clean.

2. Вызов метода findObjects.

3. Изменение длины вектора ot на равную количеству объектов и присваивание всем его элементам значения false.

3. Если изображение является первым в серии - перейти к шагу 10.

4. Если изображение не первое, то составить матрицу costs. В элементе costs[i][j] находится значение метрики delta различия i-го объекта, найденного при предыдущем вызове алгоритма (т.е. методом findObjects для предыдущего изображения) и j-го объекта, найденного при этом вызове. Также составить вектора costsI и costsJ, в которых costsI[i] = i и costsJ[j] = j.

5. Найти в матрице costs наименьший элемент costs[minI][minJ]

6. Если costs[minI][minJ] больше константы maxDelta (параметр задается вручную и характеризует ограничение на изменение объекта между радарными изображениями), перейти к шагу 10.

7. Покоординатно вычесть из расположения minJ-го объекта, найденного при этом вызове алгоритма, расположение minI-го объекта, найденного при предыдущем вызове, принять это в качестве скорости объекта по соответствующей координате. (при выводе результатов скорость будет разделена на время, прошедшее между изображениями)

8. Удалить из матрицы costs строку minI и столбец minJ, из вектора costsI - элемент с индексом minI, из вектора costsJ - элемент с индексом minJ. Таким образом, всегда получается, что costsI[i] - изначальный индекс в матрице costs строки, у которой сейчас индекс i, аналогично с столбцами и costsJ.

9. Вернуться к шагу 6.

10. Вернуть текущее изображение (в целях отладки, на практике оно в дальнейшем не используется).

**Метод clean:**

В данном методе происходит очистка изображения от шума.

1. Присвоить переменным blackAndWhitePixels и clearedPixels значение 0.

2. Вызвать метод cleanPixel поочерёдно для всех пикселей изображения.

3. Вернуть частное значений переменных clearedPixels и blackAndWhitePixel.

*Метод сleanPixel:*

В данном методе происходит локальная очистка изображения от шума.

1. Если пиксель изображения, для которого вызван метод, не является белым или чёрным, завершить выполнение метода.

2. Увеличить переменную blackAndWhitePixels на 1.

3. Перебрать соседние пиксели изображения, сохранить в переменную same число совпадающих с данным пикселем по цвету, а в переменную all количество соседних пикселей, не являющихся пикселями рамки.

4. Если same меньше 2, а all больше 1, обратить цвет пикселя, увеличить переменную clearedPixels на 1 и вызвать метод для соседних пикселей.

**Метод findObjects:**

1. Перейти к очередному пикселю изображения, если все пиксели уже перебраны, закончить выполнение метода.

2. Если пиксель не белый, вернуться к шагу 1.

3. Если пиксель белый, вызвать для него метод fillingPixel (см. ниже) с белым цветом в качестве исходного и зелёным в качестве конечного, и сохранить его выходные данные [x,y].

4. Если размер предполагаемого объекта не меньше параметра minMonitoredSize, зависящего от оценки уровня шума на основании статистических данных, то записать его характеристики в список объектов - в качестве каждой координаты следует записать вычисленную методом fillingPixel сумму соответствующих координат, разделённую на размер.

5. Если размер предполагаемого объекта меньше minMonitoredSize, вызвать для текущего пикселя метод fillingPixel с зелёным цветом в качестве исходного и чёрным в качестве конечного.

6. Вернуться к шагу 1.

**Метод fillingPixel:**

Метод вызывается для пикселя изображения p[x,y] и принимает два цвета в качестве исходного и конечного. Метод возвращает количество (оно же размер объекта) и сумму координат (для каждой координаты отдельно) перекрашенных пикселей в виде вектора из трёх значений.

1. Если цвет пикселя не совпадает с текущим, завершить выполнение метода. В качестве всех выходных значений вернуть нули.

2. Заменить цвет пикселя на конечный, рекурсивно вызвать метод для всех соседних пикселей. Вернувшиеся соответствующие значения сложить.

3. К сумме размеров прибавить единицу, к суммам сумм координат прибавить соответствующие координаты текущего пикселя.

4. Вернуть получившиеся значения.

## Алгоритмы

## Алгоритмы, используемые в библиотеке Radar.lib

Для очистки изображения используется вариация метода логического сглаживания/логического усреднения. Для идентификации на очередном изображении детектированных ранее объектов используется вариация приблизительного жадного алгоритма решения задачи о назначениях.

Подробно алгоритмы, использующиеся при обработке изображений, описаны в пункте 3.4.

## Алгоритмы, используемые в работе компонента GUI

Взаимодействие между компонентом GUI и библиотекой Radar.lib осуществляется при помощи сигнално-слотовых соединений, реализованных при помощи фреймворка Qt. Установка соединений производится в момент инициализации объектов. Большинство установленных соединений существует до уничтожения объектов интерфейса, то есть до момента завершения выполнения программы. Исключение составляет соединение *SIGNAL(QFileSystemModel::rowsInserted) –SLOT(dataProcessor::readImagesAndRun),* отвечающего за вызов метода обработки радарных изображений при их поступлении в файловую систему. Данное соединение может разрываться в случаях, указанных на диаграмме автоматов, описывающей состояния системы и переходы между ними.

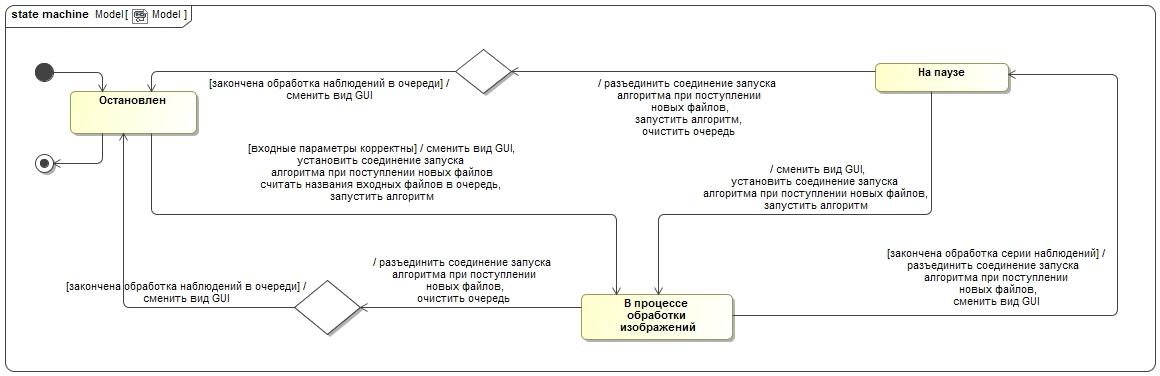


Рисунок 2 – Диаграмма автомата

Состояния системы («Остановлен», «На паузе», «В процессе обработки изображений») изображаются в прямоугольниках. Они отражают абстрактные состояния, в которых находится система в тот или иной момент времени. Указанные состояния интуитивно соответствуют ожиданиям пользователя, взаимодействующего с программой при помощи графического интерфейса. В каждый из этих моментов система полностью определяется своим контекстом.

Надписи на стрелках, соединяющих различные состояния, отражают операции, совершаемые компонентом GUI, необходимые для перехода из одного состояния в другое.

Программа *WhereTheSea* для своей работы три потока, которые обеспечивают:

* Работу интерфейса
* Работу класса *QFileSystemModel*
* Обработку изображений библиотекой *Radar.lib*

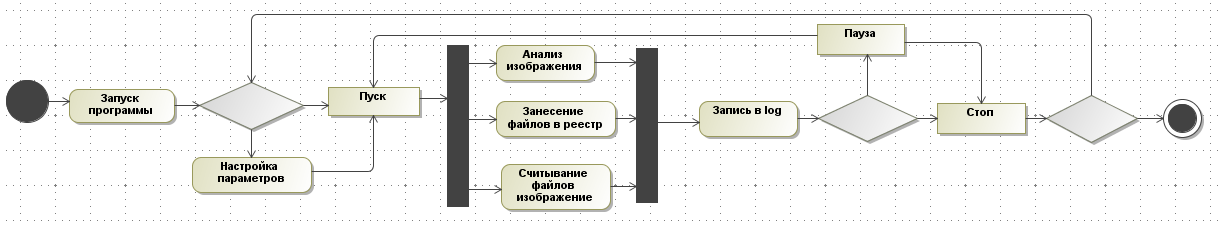


Рисунок 3 – Диаграмма деятельности

На рисунке 3 изображена диаграмма деятельности, отражающая последовательность действий, совершаемую программой.

Выполнение операций с интерфейсом возможно только в том случае, когда завершается обработка изображений, находящихся в очереди. Это обеспечивает надежную работу пользователя с программой, гарантирующую обработку занесенных в очередь изображений даже в случае некорректных действий пользователя во время их обработки.

Пользователь может выполнять лишь определенный набор операций, определяемый состояниям системы. Подобное ограничение повышает надежность системы не умаляя потребностей пользователя.

# Вызов и загрузка

На диаграмме вариантов использования можно увидеть возможные действия для пользователя с программой.

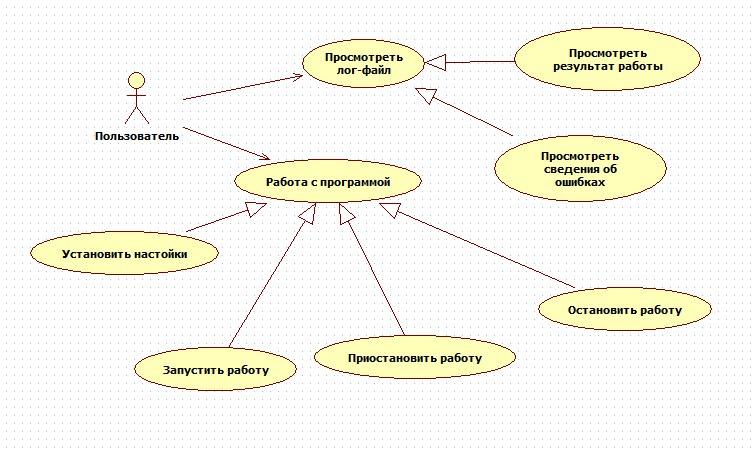


Рисунок – Варианты использования

В данном проекте задействовано одно действующее лицо. Пользователь может работать с программой, а именно установить настройки, запустить обработку изображений, поставить обработку на паузу или остановить обработку. В процессе обработки пользователь может посмотреть лог-файлы, на которых отображены ошибки обработки и характеристики объектов.

Само взаимодействие между объектами системы отображено на диаграмме взаимодействия (рисунок):

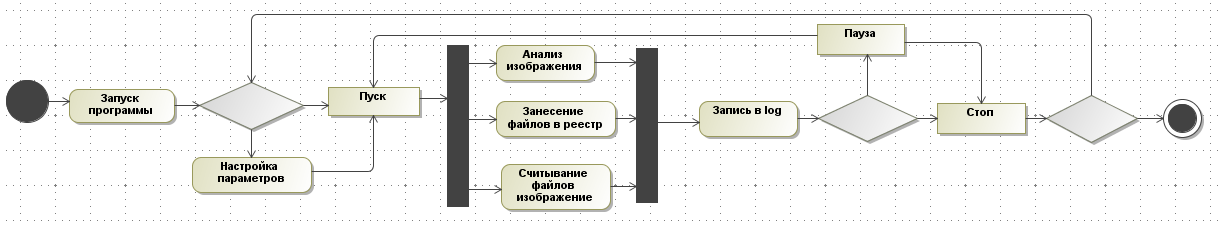


Рисунок – Диаграмма взаимодействия

Диаграмма взаимодействия связана с интерфейсом программы. После запуска программы, пользователь может установить настройки или сразу запустить обработку изображения. Программа одновременно выполняет 3 процесса: считывает изображение, добавляет его в очередь, анализирует изображения. Результат обработки записывается в log файлах. Также у пользователя имеется возможность поставить обработку изображения на паузу или остановить изображения.

Вызов программы осуществляется посредством использования интерфейса.

Запуск программы

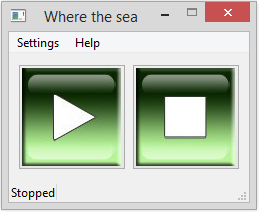


Рисунок – Запуск программы

Выбор настроек

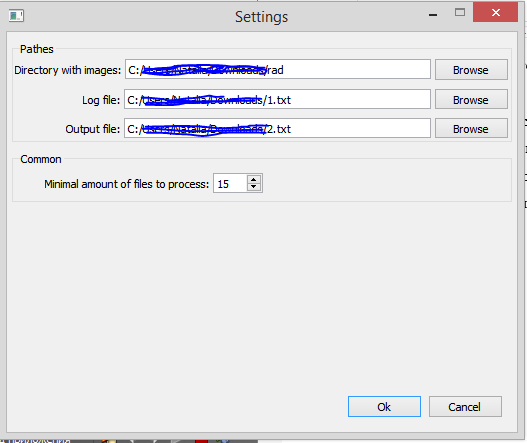


Рисунок – Выбор настроек программы

Запуск обработки изображения

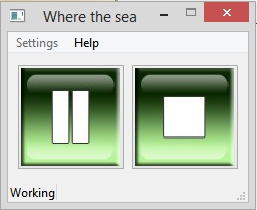


Рисунок – Начало обработки изображения

Приостановка обработки изображения

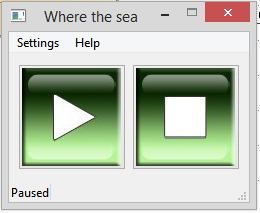


Рисунок – Пауза изображения

Выбор настроек при паузе

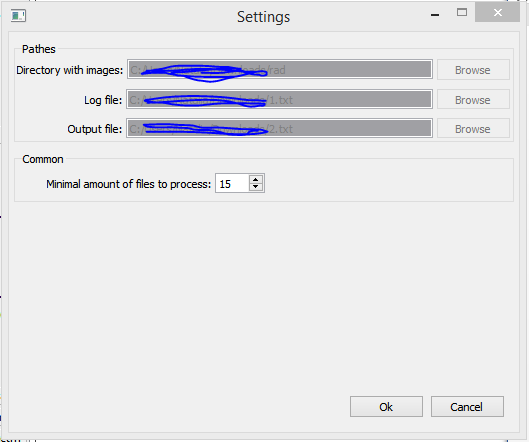


Рисунок – Выбор настроек при паузе

Остановка обработки изображения

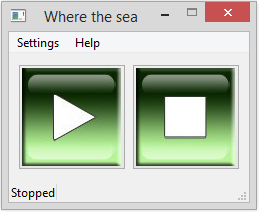


Рисунок – Остановка обработки изображения

Ошибка при указании расположения изображений

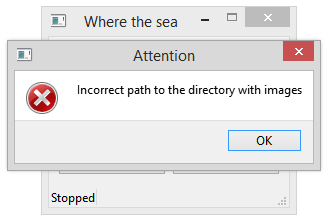


Рисунок – Ошибка указания расположения изображений

Ошибка в указании log и output файлов

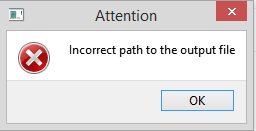
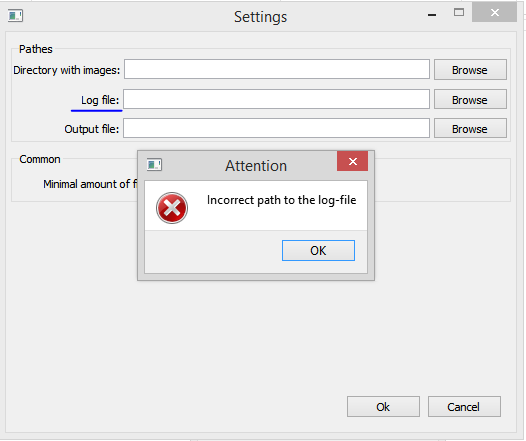


Рисунок - Ошибка в указании log и output файлов

# Входные данные

На входе программа получает радарное изображение формата BMP (рисунок №). В названии каждого файла изображения находится информация о дате изображения, его координата, название. На входном радарном изображении отображен белый шум, который может мешать анализу объектов, находящихся на изображении. В связи с этим одним из существенных этапов обработки изображения является устранение белого шума.

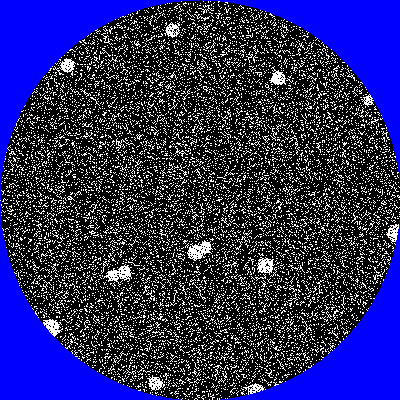


Рисунок № - Радарное изображение

# Выходные данные

На выходе пользователь получает 2 обработанных текстовых файла logfile.txt (рисунок ) с указанием информации по обработанным изображениям и outputFile.txt (рисунок ), где указана информация по объектам изображения.

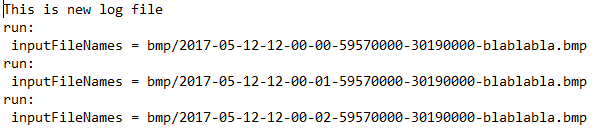


Рисунок – logfile

На рисунке отображена информация об обработанных файлов, которые программа поставила в очередь на обработку.

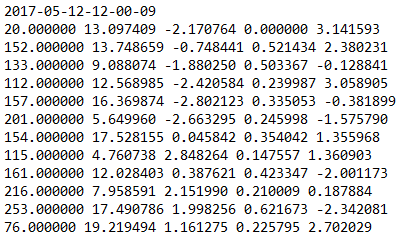


Рисунок – outputFile

В начале изображения отображена дата исходного изображения. После, в 1 столбце указан размер изображения в пикселях. Во 2 столбце – расстояние до объекта из центра изображения по выбранной шкале. В 3 – угол объекта из центра изображения, в радианах. В 4 – скорость объекта, по выбранной шкале. В 5 – Угол движения объекта по направлению вверх, в радианах.