

<b>Содержание</b>	
Введение.....	4
1. Техническое задание.....	5
1.1 Назначение разработки и область применения.....	5
1.2 Технические требования.....	5
2. Анализ технического задания .....	6
2.1 Выбор сенсора .....	6
2.2 Сравнение инструментов разработки .....	8
2.3 Выбор языка программирования .....	11
2.4 Выбор среды разработки .....	12
2.5 Обзор существующих методов обнаружения препятствий с помощью камеры. ....	13
2.6 Выбор подхода к созданию системы управления мобильным роботом. ..	15
3. Разработка структуры системы.....	19
3.1 Разработка общей структуры системы .....	19
3.2 Kinect .....	20
3.3 Разработка алгоритма приложения для обработки данных с Kinect .....	22
4. Разработка программных средств .....	27
4.1 Настройка и установка программных средств.....	27
4.2 Разработка приложения .....	28
5. Тестирование системы.....	34
5.1 Тестирование работы системы принятия решений. ....	34
Заключение .....	37
Список литературы .....	38

					ВКР-НГТУ-09.03.01-(15-В-1)-008-2019(ПЗ)						
Изм	Лист	№ докум	Подп.	Дата	Программная система управления движением мобильного робота				Литера	Лист	Листов
Разраб.		Ляляев Н.В.									
Пров.		Гай В.Е.								3	38
Т. контр.									НГТУ им. Р.Е. Алексеева		
Н. контр.											
Утв.		Мякинников А.В.									

## Введение

В наше время роботы помогают упростить жизнь человека, автоматизировать производство, проводить исследования в опасных для человека местах. Спектр занятий, где может применяться автоматизированный помощник, безграничен. И в зависимости от рода деятельности применяются различные требования к конструкции и возможностям.

Разрабатываемая мною программная система управления предназначена для мобильного робота телеприсутствия. Он позволяет своему оператору получать информацию различного рода, пока тот находится в совершенно ином месте. Также он может использоваться для взаимодействия с другими людьми или объектами на расстоянии. Высокий спрос на роботов подобного типа, а также огромный потенциал для развития их возможностей обеспечивает актуальность данной разработки.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						4
Изм	Лист	№ докум.	Подп.	Дата		

## 1. Техническое задание

### 1.1 Назначение разработки и область применения

Разрабатываемая программная система предназначена для управления роботом на основе данных получаемых с камеры в реальном времени. На основе полученных данных должны быть переданы команды движения. При этом необходимо реализовать два режима работы робота: следование за объектом и свободное перемещение.

### 1.2 Технические требования

Рассмотрим требования, предъявляемые аппаратной и программной частями робота к разрабатываемой системе управления:

- поддержка ARM архитектуры процессора Raspberry;
- обеспечение совместимости с Debian дистрибутивом Linux Raspbian;

Рассмотрим, какой функциональностью должна обладать проектируемая программная система управления мобильным роботом:

- 1) Пользователь робота имеет возможность выбрать режим работы .
- 2) Данные с сенсора обрабатываются в реальном времени и на их основе принимается решение о действиях робота в зависимости от выбранного режима.
- 3) В режиме автономного перемещения робот должен избегать столкновений.
- 4) В режиме следования сохранять определенную дистанцию от объекта.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						5
Изм	Лист	№ докум.	Подп.	Дата		

## 2. Анализ технического задания

### 2.1 Выбор сенсора

Одним из инструментов получения данных об окружающем пространстве является камера, однако для обработки изображения с неё необходимо учитывать качество изображения, освещенность, угол обзора. Для построения объемной карты местности обычно используется несколько камер и множество математических преобразований для обработки данных с них. Однако все эти проблемы решаются при помощи сенсора глубины. Он позволяет сразу получить информацию о местонахождении объекта, при этом практически не зависит от влияния окружающей среды. С выбора камер с сенсором глубины и стоит начать анализ технического задания.

#### 1.Kinect



Рисунок 1. Kinect

Выпущенный в 2010 году компанией Microsoft игровой контроллер Kinect[1] достаточно быстро был взломан любителями для использования в целях создания приложений на его основе. Спустя некоторое время был выпущен и официальный набор инструментов для разработчиков. В силу своей доступности, простоты в подключении и использовании обзавелся поддержкой нескольких open source проектов, адаптирован для использования под Linux, Mac OS, Windows. Kinect позволяет распознавать движения, голос, записывать RGB- видео, строить карту глубины. Инфракрасный CMOS – датчик позволяет с достаточно высокой точностью определить расстояние до объекта, при этом нет необходимости проводить сложные математические вычисления. Позже вместе с приставкой Xbox One была выпущена вторая версия контроллера. Цены на б/у версию устройства начинаются от 2000 рублей, новую – от 7000.

#### 2. Intel RealSense D400

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						6
Изм.	Лист	№ докум.	Подп.	Дата		



Рисунок 2. Intel RealSense D400

RealSense – это программно-аппаратная технология получения и обработки трехмерных изображений[4]. Состоит из связки камеры RealSense и набора программного обеспечения, драйверов и библиотек Intel RealSense SDK. Поддерживаются платформы Linux, Windows, MacOS. С недавних пор SDK приобрел статус open source проекта и появился в открытом доступе на GitHub. Камеры выпускаются как для мобильных, так и для стационарных устройств. Первые отличаются уменьшенным встраиваемым вариантом процессора с несколько урезанными характеристиками. При этом есть возможность приобрести отдельно процессоры и отдельно модули глубины, а не только «коробочные» решения. Камера позволяет снимать RGB-поток в Full HD разрешении и в HD поток глубины. Минимальное фокусное расстояние от 0,11м, а максимальное до 10м. На официальном сайте Intel можно заказать от 179 долларов.

### 3. 3D ZED Stereo camera

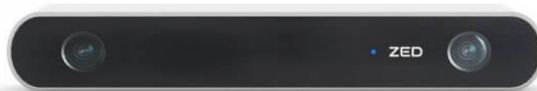


Рисунок 4. 3D ZED Stereo camera

Широкий угол обзора, съемка в разрешении 2K, высокое качество изображения даже при низкой освещенности. Дальность составления карты глубины составляет от 1,5 до 20 м. Интерфейс USB 3.0, совместимость с Windows, Linux, поддержка OpenCV. В зависимости от качества съемки меняется скорость кадров в секунду – от 15 до 100 при понижении разрешения до 1344x376. Множество настроек камеры: яркость, контрастность, насыщенность, гамма, чёткость, баланс белого. Достаточно впечатляющие возможности камеры соответствуют и ее цене – от 69 тысяч.

Для работы был выбран Kinect в силу своей доступности, широкого набора программных инструментов, поддерживающих в том числе совместимость с Linux и ARM процессорами. Также за счет огромного количества примеров и обсуждений на форумах сильно упрощается освоение данного сенсора и внедрение его в проект.

## 2.2 Сравнение инструментов разработки

Так как в условиях уже заданы программное и аппаратное обеспечения, то необходимо выбрать инструменты, которые будут совместимы с ними.

Для этого необходимо изучить решения, разработанные для работы с камерами с сенсором глубины и поддерживающими Kinect. Изучить требования, которые они предъявляют к системе, функционал, который они предоставляют, наличие документации для быстрого освоения. В результате этого исследования необходимо выбрать наиболее подходящие для разработки системы управления.

В сравнительном анализе будут участвовать следующие решения: libfreenect, OpenNI2, Microsoft Kinect SDK, PCL.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						8
Изм.	Лист	№ докум.	Подп.	Дата		

1) Libfreenect- открытое программное обеспечение, библиотека обеспечивающая использование Kinect с Windows, Linux и Mac. Распространяется по лицензии Apache 2.0. Существует также вторая версия для обеспечения поддержки Kinect второго поколения. Позволяет обнаружить Kinect, получить RGB-изображение и Depth Images, управлять мотором, акселерометром, записывать аудио. Для установки и работы необходимо иметь libusb, cmake. Написана библиотека на языке программирования C. Разработана сообществом OpenKinect, имеется подробная документация, реализованы обертки под различные языки – Python, C++, C#, Javascript и прочие.

2) OpenNI2 – открытое программное обеспечение, проект, направленный на сертификацию и улучшение функциональной совместимости естественных пользовательских интерфейсов и органических пользовательских интерфейсов для устройств NI(natural interactions), приложений использующих эти устройства, и промежуточного программного обеспечения, которое облегчает доступ и использование таких устройств. Разработано на C++, совместимо с Linux, Windows, Android. Реализована поддержка множества сенсоров – Kinect 1 и 2, Prime Sense и прочие. Обеспечивает возможность сбора и обработки RGB и Depth Image, подключение нескольких сенсоров, отслеживание движений тела, поддержку libfreenect и opencv, распознавание голосовых команд.

3) Microsoft Kinect SDK – официальный набор инструментов для разработки приложений, использующих возможности Kinect. Выпущен Microsoft для поддержки разработчиков приложений на Windows для коммерческого и некоммерческого использования. Позволяет легко начать разработку и поддерживает C#, C++, .NET-языки. На момент выхода предоставлял доступ к куда большему набору возможностей, нежели open source проекты. В комплекте шла демо- игра и набор приложений, демонстрирующих возможности сенсора, что позволяло легко освоиться и начать разработку собственного приложения. Для Kinect второго поколения также был выпущен подобный набор инструментов. Однако существовало одно большое ограничение – поддерживалась только ОС Windows.

4) PCL – это библиотека алгоритмов с открытым исходным кодом для задач обработки облаков точек и обработки трехмерной геометрии, например, для трехмерного компьютерного зрения. Библиотека написана на C++, распространяется по лицензии BSD. Алгоритмы позволяют фильтровать зашумленные данные, сегментировать соответствующие части сцены, извлекать ключевые точки, дескрипторы для распознавания объектов в мире на основе их геометрического вида, создавать поверхности из облаков точек и визуализировать их. Для хранения и обработки данных, полученных с сенсоров используется специальный формат PCD. Устанавливается с драйверами нужного сенсора для обеспечения совместимости.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						9
Изм	Лист	№ докум.	Подп.	Дата		

Перейдем к оценке и сравнению выбранных решений. Необходимо учесть системные требования этих продуктов, возможности, предоставляемые для разработки, требования к аппаратной части системы, наличие подробной документации.

1) Требования к системному обеспечению.

На этом этапе полностью проигрывает Microsoft Kinect SDK, так как не поддерживает Linux. Остальные же инструменты вполне совместимы с Linux, как и сопровождающее их ПО, необходимое для установки.

2) Требования к аппаратному обеспечению.

Так как Raspberry PI построен на ARM архитектуре, то необходимо проверить совместимость. К сожалению, PCL и Microsoft Kinect SDK не поддерживают подобные процессоры. К тому же PCL крайне требовательна по отношению к оперативной памяти, что также сильно затрудняет возможности ее использования.

3) Наличие подробной документации.

Абсолютно все представленные решения имеют подробную документацию, которая описывает все функции, классы, переменные, есть примеры их использования, шаблоны для компиляции собственных проектов, образцы, демонстрирующие возможности, реализующие базовый функционал. Так что с ознакомлением с тонкостями и возможностями данных библиотек нет проблем.

4) Возможности для разработки.

Microsoft Kinect SDK, очевидно на голову выше, ведь создан и заточен непосредственно под разработку приложений для Kinect. Он позволяет использовать возможности сенсора на максимум. PCL для разработки приложений под Kinect необходимо связать с OpenNI, для сбора данных. Но зато он предоставляет широкий набор для распознавания жестов, алгоритмов, основанных на машинном обучении, построении облака точек. OpenNI2 и libfreenect используются для более простых приложений, но также позволяют использовать все возможности Kinect.

Для разработки программной системы управления роботом был выбран OpenNI2, так как он совместим и с программной, и с аппаратной частью робота, предлагает инструментарий, достаточный для реализации требований ТЗ. Однако для поддержки первого Kinect необходимо также наличие драйвера libfreenect.



## 2.3 Выбор языка программирования

Следующая по значимости задача, которая встает перед нами - это выбор языка программирования, на котором будет производиться разработка всего приложения. Выбор конкретного языка должен зависеть от специфики разработки и навыков разработчика. Рассмотрим наиболее популярные языки программирования для разработки Windows-приложений:

1) Язык ассемблера - машинно-ориентированный язык программирования низкого уровня. Его команды прямо соответствуют отдельным командам машины или их последовательностям, также он может предоставлять дополнительные возможности облегчения программирования, такие как макрокоманды, выражения, средства обеспечения модульности программ. Может рассматриваться как автокод, расширенный конструкциями языков программирования высокого уровня. Является существенно платформо-зависимым. Языки ассемблера для различных аппаратных платформ несовместимы, хотя могут быть в целом подобны.

2) C++ – компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

3) Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

4) Java – сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						11
Изм.	Лист	№ докум.	Подп.	Дата		

транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. На 2019 год Java — один из самых популярных языков программирования.

Для разработки системы был выбран язык программирования C++, так как данный язык показывает высокую скорость вычислений, является основой OpenNI2, позволяет глубже работать с памятью, оптимизируя ее выделение, что играет большую роль в ускорении обработки данных при использовании маломощных систем.

## 2.4 Выбор среды разработки

Очень важно подобрать удобную и функциональную среду разработки. Верный выбор может сильно ускорить процесс создания проекта. Рассмотрим наиболее популярные среды разработки для Linux на языке C++:

1) Code::Blocks- среда разработки с открытым исходным кодом. Используется архитектура плагинов. Code::Blocks также ориентирован на C и C++. Поддерживает большинство компиляторов, содержит интегрированный список задач, дебаггер, GUI. Распространяется бесплатно, кроссплатформенная.

2) QtCreator - кроссплатформенная свободная IDE для разработки на C, C++ и QML. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCCE, RVCT, WINSCW. Среди возможностей, присущих любой среде разработки, есть и специфичные, такие как отладка приложений на QML и отображение в отладчике данных из контейнеров Qt, встроенный дизайнер интерфейсов как на QML, так и на QtWidgets. Qt Creator поддерживает системы сборки qmake, cmake, autotools, с версии 2.7 qbs. Для проектов, созданных под другими системами, может использоваться в качестве редактора исходных кодов. Есть возможность редактирования этапов сборки проекта. Также IDE нативно поддерживает системы контроля версии, такие как Subversion, Mercurial, Git, CVS, Bazaar, Perforce.

3) Eclipse - свободная интегрированная среда разработки модульных кроссплатформенных приложений. Наиболее известные приложения на основе Eclipse Platform — различные «Eclipse IDE» для разработки ПО на множестве языков. Eclipse служит в первую очередь платформой для разработки расширений, чем он и завоевал популярность: любой разработчик может расширить Eclipse своими модулями. Eclipse написана на Java, потому является платформо-независимым продуктом.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						12
Изм	Лист	№ докум.	Подп.	Дата		

4)NetBeans - имеет дружелюбный к пользователю интерфейс, множество полезных шаблонов проектов предоставляет функционал drag-and-drop. Netbeans написана на Java, но предоставляет полную поддержку и набор инструментов, необходимых для разработчиков, пишущих на C и C++. Одним из важнейших достоинств являются простые и эффективные инструменты для управления проектами. Поддерживаются плагины для расширения функционала и возможность удаленного мониторинга проекта. Среда доступна для Windows, Mac OS X, Linux и поддерживает языки C++, C, Java, HTML, HTML5.

Для разработки была выбрана QtCreator – за счет удобного интерфейса, поддержки cmake для сборки проектов, так же он идет предустановленным в Raspbian, что позволяет не перегружать систему лишним ПО.

## **2.5 Обзор существующих методов обнаружения препятствий с помощью камеры.**

Обнаружение препятствий на пути – крайне важная и актуальная проблема робототехники, интеллектуальных систем, помогающих людям в управлении транспортом, разработке автопилотов.

В зависимости от условий и поставленных задач необходим уникальный подход к каждому частному случаю. Допустим, для создания беспилотного автомобиля крайне важна скорость и точность определения препятствий и принятие решений. Для этого его оснащают различными датчиками, камерами, дополняют системами машинного обучения и высокопроизводительной аппаратной частью. С другой стороны, небольшому мобильный робот нуждается скорее в мобильности и продолжительной автономности работы, так что не может позволить себе экипироваться всеми существующими датчиками и небольшим сервером для обработки видеопотока в реальном времени.

Заданных условиях для обнаружения препятствий роботом предпочтительнее всего рассмотреть именно алгоритмы, основанные на обработке данных с камеры, ведь это позволит снизить экономические затраты на его разработку и избежать дополнительных проблем с установкой и обработкой данных с иных сенсоров.

В работе [6] предлагается использовать одну видеокамеру и систему распознавания препятствий. Этот алгоритм принимает ширину препятствия и ширину кадра в пикселях, угол обзора камеры в градусах. Затем

определяется угловой размер препятствия. После чего робот продвигается на определенное расстояние и повторяет вычисления, затем решает систему уравнений для определения расстояния до препятствий. Плюсами данного алгоритма являются низкая энергоемкость и отсутствие необходимости в высоких вычислительных мощностях. Однако данный зависит от качества изображения, не способен реагировать на перемещение препятствий, замедляет движение робота.

В работе [7] учтена проблема описанная в прошлом алгоритме – зависимость от качества полученного изображения. Для ее решения применяется сенсор глубины. Он позволяет получить трехмерное изображение при любом освещении. Полученные данные затем преобразуются в трехмерное облако точек, после чего понижается разрешение изображения и преобразование в двумерное облако. И уже на основе полученных данных определяются препятствия и расстояния до них. Это один из самых эффективных способов, так как позволяет достаточно быстро обрабатывать изображение, используя не самые большие вычислительные мощности. При этом сам алгоритм зависит от выбранного сенсора – его точности и программной поддержки. В данном случае в работе был использован Kinect, OpenNI и PCL.

Еще один вариант [8] разработки подобных алгоритмов основан на связке вебкамеры и ультразвукового датчика. Данный алгоритм основан на появлении помех в изображении при понижении освещения на низкой частоте кадров. Область помех отмечается как препятствие и дополнительно подтверждается данными с ультразвукового датчика. Данный алгоритм не обладает высокой точностью и требует дополнительного оборудования, однако может быть реализован при использовании малых вычислительных мощностей и в условиях низкого энергопотребления.

Из приведенных выше примеров можно сделать вывод, что выбор реализации алгоритма сильно зависит от аппаратной части и условий, в которых придется работать роботу. При этом использование одной или нескольких обычных камер всегда будет упираться в зависимость от условий освещения, помех, угла обзора. Данные проблемы успешно решаются применением датчиков глубины, которые позволяют сильно снизить требуемые вычислительные мощности для обработки видеопотока, а также получить данные непосредственно о расстоянии до объекта без дополнительных подсчетов и вне зависимости от окружающих условий.

## 2.6 Выбор подхода к созданию системы управления мобильным роботом.

Для создания системы управления необходимо решить несколько задач:

- 1) Получение изображения
- 2) Обработка полученных данных
- 3) Принятие решений на их основе

Так как в качестве источника изображения выступает сенсор Kinect, появляется возможность сильно снизить сложность алгоритмов обработки изображения и не учитывать уровень освещения, угол под которым установлена камера и различные помехи. Kinect оснащен RGB-камерой и дальномером, состоящим из инфракрасного лазерного проектора и инфракрасного CMOS- датчика. ИК-камера, как и RGB имеют разрешение 640x480 пикселей. Подобное устройство сенсора позволяет нам в реальном времени получать данные о расстоянии до каждого объекта в области видимости камеры. Каждый пиксель Depth Image хранит информацию о координатах своего места в изображении и глубине. Таким образом необходимо обрабатывать каждый кадр изображения в реальном времени и определять наличие ближайшего препятствия на пути робота(рис.1) или же наличие объекта в определенном диапазоне(рис.2), когда робот находится в режиме следования.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						15
Изм	Лист	№ докум.	Подп.	Дата		

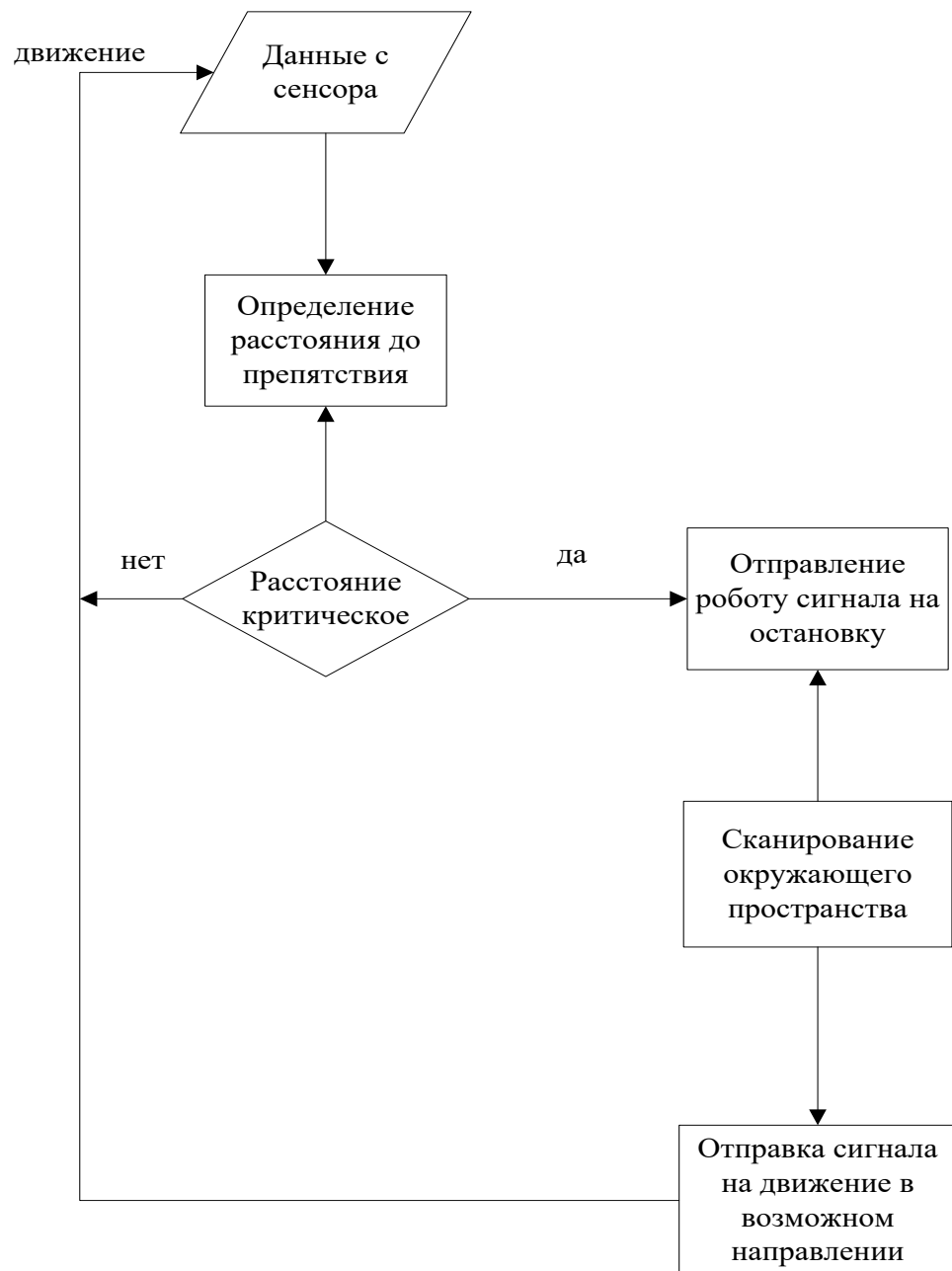


Рисунок 5. Алгоритм автономного перемещения

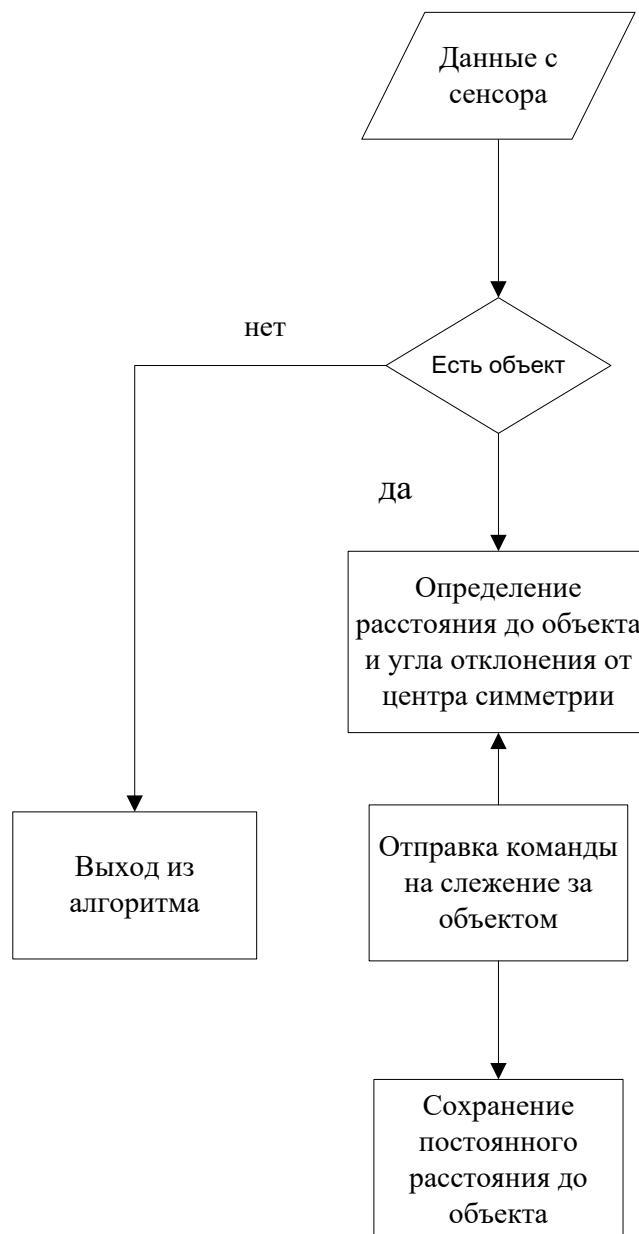


Рисунок 6. Алгоритм следования

Для сбора данных и подключения сенсора Kinect используется OpenNI2 с драйвером libfreenect для обеспечения совместимости. Полученное изображение обрабатывается при помощи инструментов библиотеки, находится ближайшая к камере точка, проверяется ее соответствие условиям, чтобы отбросить лишние помехи или объекты, находящиеся не на пути робота. Данный подход позволяет мгновенно реагировать даже на подвижные объекты, возникающие на пути. После этого система передает команды, зависящие от выбранного режима работы робота.

Отсутствие необходимости опознавать объекты целиком, используя машинное обучение, перевода изображения в реальные координаты или построение облака точек позволяет снизить необходимые вычислительные

мощности до минимума. Такой подход позволяет расширить список задач, которые сможет решать робот одновременно. При этом ввод определенных условий для отбора точек на проверку позволяет принимать роботу достаточно точные решения, игнорировать объекты, находящиеся слишком далеко или в стороне от его маршрута. Подобный подход позволяют применить и условия эксплуатации.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						18
Изм	Лист	№ докум.	Подп.	Дата		



### 3. Разработка структуры системы

#### 3.1 Разработка общей структуры системы



Рисунок 7. Структурная схема программной системы управления роботом

Система состоит из 3 основных составляющих- сенсора, приложения для обработки данных и принятия решений, приложения для передачи команд роботу. После включения робота и выбора режима начинается сканирование сенсором Kinect окружающей обстановки. Каждый кадр в сыром виде передается на обработку. На основе анализа кадра проводится проверка на соответствие определенным условиям, в результате чего приложение возвращает определенный параметр. Этот параметр передается в систему управления движением робота и приводит к определенным

действиям – движению, повороту или остановке. Сканирование местности происходит постоянно в реальном времени для своевременного принятия решений. Формат изображения, получаемый с Kinect позволяет обрабатывать их почти мгновенно, что также позволяет повысить точность обнаружения препятствий. Рассмотрим подробнее части схемы.

### 3.2 Kinect

Рассмотрим подробнее устройство сенсора Kinect и формат данных, которые он возвращает.



Рисунок 8. Устройство Kinect

Сенсор [2] оснащен дальномером, состоящим из инфракрасного лазерного проектора и инфракрасного CMOS- датчика, RGB-камеры, и специализированного микрофона с шумоподавлением. Также установлен светодиод, трехосный акселерометр и сервопривод. Рассмотрим подробнее некоторые части. RGB-камера представляет собой VGA-камеру поддерживающую съёмку в разрешении 640 x 480 пикселей с 8-битным разрешением. Для воссоздания 3D-захвата движения применяется чип от PrimeSense. Его используют ИК-проектор и ИК-камера, которая имеет такое же разрешение, как и RGB. Рассмотрим подробнее устройство CMOS-датчика (ИК-камеры).

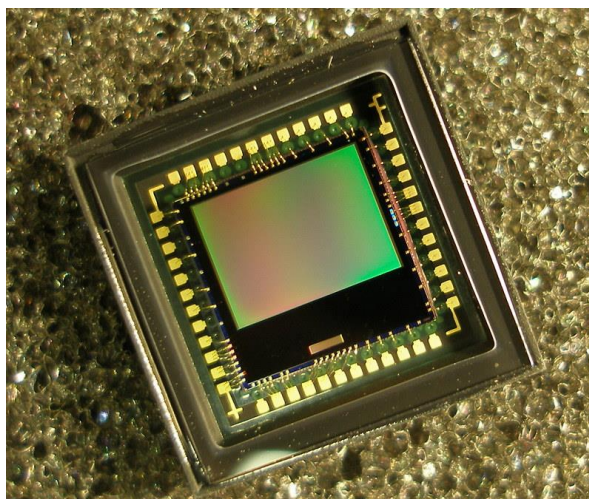


Рисунок 9. КМОП-матрица

КМОП-матрица (CMOS) – светочувствительная матрица на основе КМОП-технологии. То есть основанная на наборе полупроводниковых технологий построения интегральных микросхем и соответствующая ей схемотехника микросхем. В этой технологии применяются полевые транзисторы с изолированным затвором с каналами разной проводимости. Основным преимуществом таких схем является низкое энергопотребление в статическом режиме.

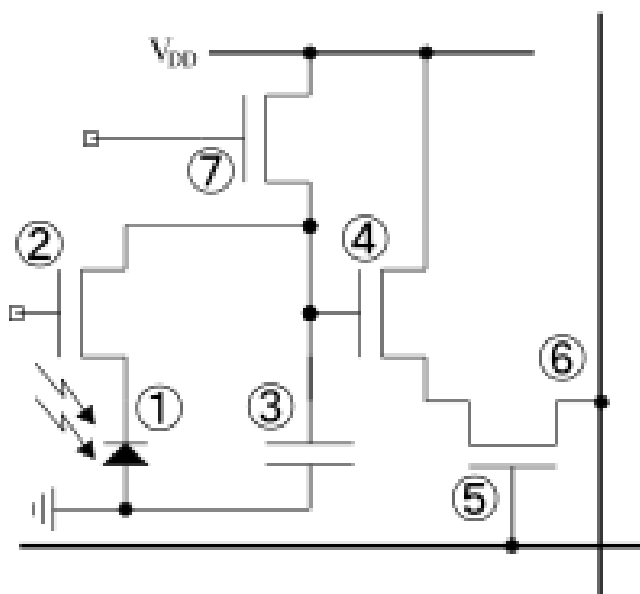


Рисунок 10 Эквивалентная схема ячейки КМОП-матрицы

- 1 – светочувствительный элемент
- 2 – затвор
- 3 - конденсатор, сохраняющий заряд с диода
- 4 – усилитель

- 5 – шина выбора строки
- 6 – вертикальная шина, передающая сигнал процессору
- 7 – сигнал сброса

Инфракрасный сенсор накладывает на пространство сетку из точек, а монохромный CMOS- сенсор отвечает за преобразование данных в 3D- проекцию. Для каждой точки при этом определяется расстояние от нее до сенсора.

Kinect возвращает два вида изображения[3] – RGB и Depth Image. В первом случае в информации о каждом пикселе описаны его координаты и цвет, а вот во втором – координаты на изображении и глубина. То есть расстояние от сенсора до этой точки в пространстве. Подобная особенность сенсора позволяет нам не тратить вычислительные мощности на обсчет расстояния до объектов, а получить их сразу. А дальше проводить нужные действия – находить ближайшую точку, определять расстояние до интересующих объектов, строить 3D- облака точек и тд.



Рисунок 11. Depth Image и RGB- изображение

### 3.3 Разработка алгоритма приложения для обработки данных с Kinect

Каждый пиксель в кадрах, полученных с Kinect содержит информацию о своих координатах на изображении и расстоянии до объекта, находящегося в данной точке, записанном в миллиметрах. Для обнаружения препятствий на пути робота необходимо искать точку, находящуюся ближе всего к сенсору. Следующим шагом необходимо проверять, где находится данная точка. При этом объекты также могут оказаться слишком близко к сенсору в его «слепой» зоне. В этом случае необходимо остановить движение. Технологии Kinect и программные инструменты OpenNI2 позволяют очень быстро обработать изображение и найти ближайшую точку. Однако аппаратные ограничения самого робота не позволяют использовать алгоритмы машинного обучения и компьютерного зрения для опознавания объектов на

изображении. Подобные ограничения вынуждают строить логику алгоритма вокруг обнаружения ближайшей точки.

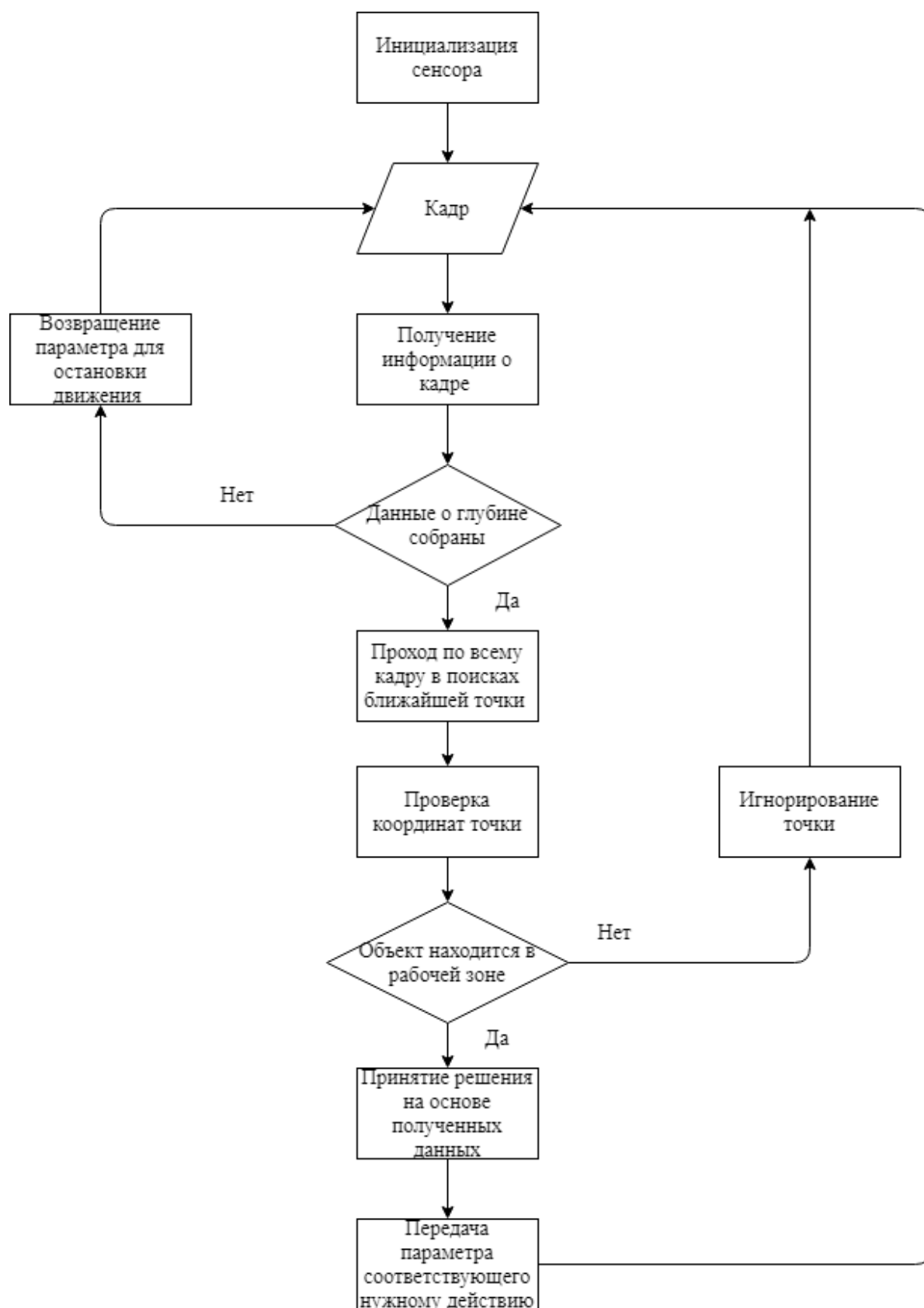


Рисунок 12. Общий алгоритм работы приложения

Так как в наборе точек изображения всегда существует ближайшая, необходимо определить какие точки стоит учитывать при принятии решений, а какие нет.

Для этого можно спроецировать точки на координаты реального мира и проверять их не только по степени удаленности от камеры, но и месту в пространстве. Однако в этом случае возникают проблемы с установкой жесткой системы координат для определения условий отбора точек и также добавляются дополнительные математические вычисления, сильно замедляющие работу алгоритма.

В следствии это было принято решение применять координаты точки на самом изображении. Для этого отбрасываются края изображения, чтобы не ловить ложные точки. А рабочая зона изображения размечается для принятия решения в зависимости от режима работы алгоритма. В случае следования за объектом происходит отслеживание смещения от центральной зоны для корректировки курса и поворота робота. Если же объект находится в центральной зоне, робот ожидает пока тот начнет двигаться и при достижении определенного расстояния начинает преследование на определенном расстоянии. В режиме автономного перемещения робот всегда поворачивает направо для того чтобы иметь возможность обойти все помещение. Любой объект, появившийся слишком близко к сенсору приводит к его остановке, так как перестают поступать данные о глубине.

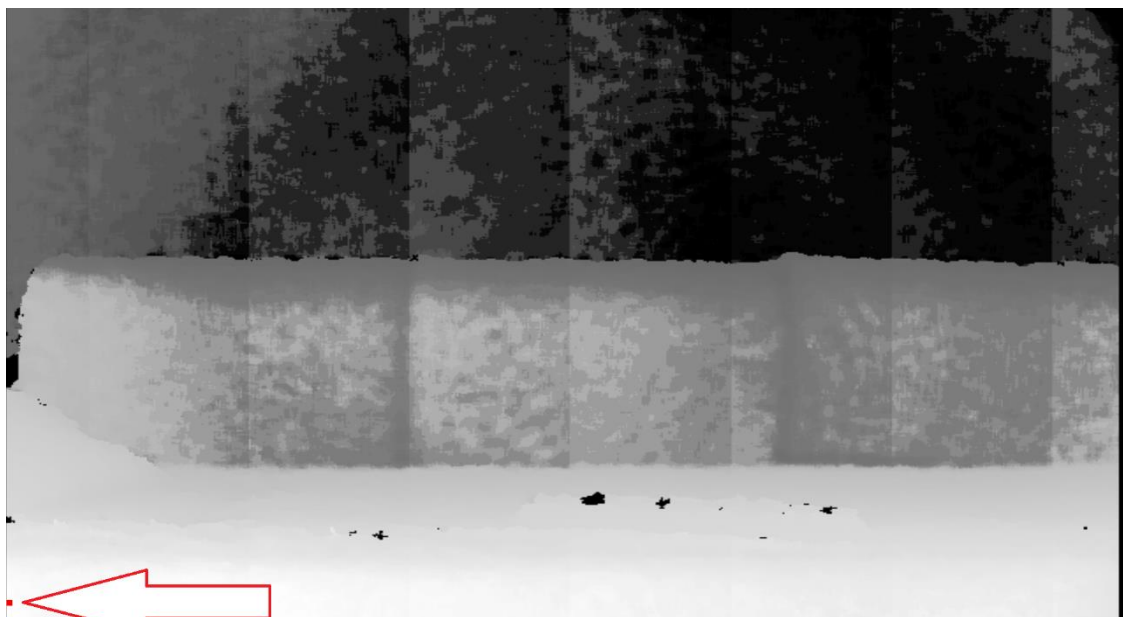


Рисунок 13. Ложная точка

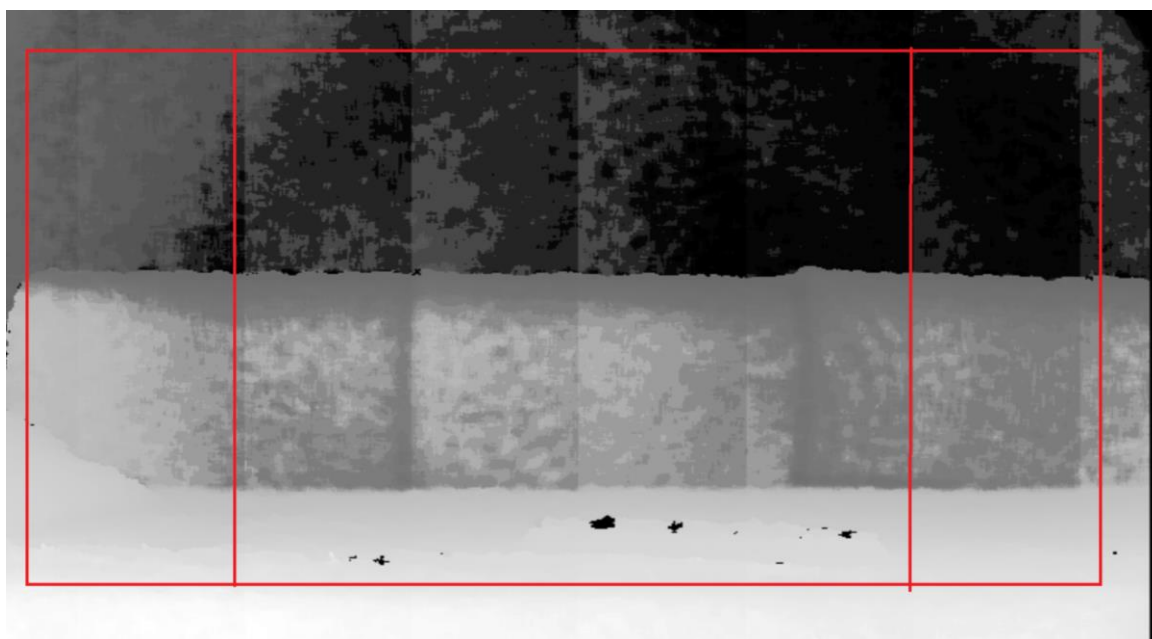


Рисунок 14. Разметка изображения для алгоритма следования

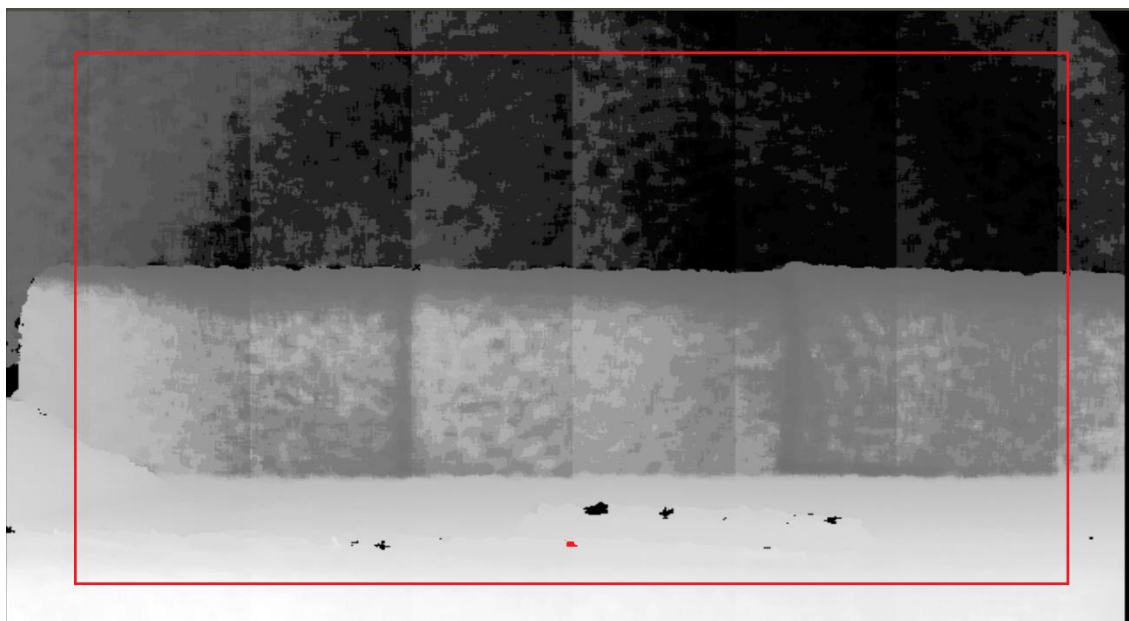


Рисунок 15. Разметка изображения для алгоритма автономного перемещения



## 4. Разработка программных средств

### 4.1 Настройка и установка программных средств

Для работы с сенсором Kinect и разработки приложений, использующих его возможности в первую очередь необходимо настроить и установить выбранную раньше связку OpenNI2 с поддержкой драйвера libfreenect.

В первую очередь необходимо установить все сопутствующие программные средства:

Для libfreenect:

- Libusb  $\geq 1.0.18$
- CMake  $\geq 3.1.0$

Для OpenNI2:

- GCC 4.x
- Python 2.6+/3.x
- LibUDEV
- JDK 6.0
- FreeGLUT3
- Doxygen
- GraphViz

Для установки данных средств можно воспользоваться командой apt-get install:

```
sudo apt-get install g++ python libusb-1.0-0-dev libudev-dev openjdk-6-jdk  
freeglut3-dev doxygen graphviz cmake
```

Или же скачать исходные файлы с официальных сайтов и репозиториев для ручной установки.

Так как работать предстоит с ARM системой, то необходимо выбрать специальную версию OpenNI2. Данная версия специально облегчена, также есть возможность воспользоваться параметром для кросс-компиляции. Сборка происходит при помощи “make”. Для кросс-компиляции[10] под ARM необходимо определить следующие переменные для окружения:

- ARM\_CXX=path-to-cross-compilation-g++
- ARM\_STAGING=path-to-cross-compilation-staging-dir

После этого выполнить:

```
PLATFORM=Arm make
```

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						27
Изм.	Лист	№ докум.	Подп.	Дата		

После установки и настройки OpenNI2 необходимо скачать с официального репозитория libfreenect. И выполнить сборку с драйвером для OpenNI2.

```
cmake.. -DBUILD_OPENNI2_DRIVER=ON
```

```
make
```

После этого необходимо скопировать файл драйвера в директорию драйверов OpenNI2. Драйвер устанавливается вместе с libfreenect, так что нет необходимости дополнительно подключать его к проекту.

Драйвер позволяет использовать связать интерфейс OpenNI2 и libfreenect используя множественное наследование[10]. Это позволяет использовать API libfreenect, получать информацию об устройстве, кадрах, видеопотоке, состоянии устройства, и прочим возможностям.

## 4.2 Разработка приложения

Основные этапы, на которые стоит обратить внимание при разработке – выбор режима работы, инициализация сенсора, проверка его статуса, прием кадра из видеопотока, его анализ, принятие решения.

Для передачи команд управления на модуль Arduino был разработан скрипт на языке Python. Данный скрипт работал для режима ручного управления. Данный скрипт проверял нажатые клавиши интерпретировал их и отправлял. Для автономного режима робота необходимо, чтобы скрипт проверял команды, поступающие от приложения, принимающего решения.

Эффективнее всего для этого вызвать наше приложение как subprocess и связать стандартные потоки ввода-вывода. Также необходимо учесть, что между передачей команд должно пройти время, для того, чтобы скрипт успел обработать команду и отправить на модуль управления Arduino. Для этого необходимо импортировать модуль subprocess.

```
p = Popen(['/home/nikita/OpenNI2-master/Bin/x64-Release/Ride'], shell=True, stdout=PIPE, stdin=PIPE)
while True:
    #currcomm = input()
    currcomm = p.stdout.readline().strip()
    # тут идёт обработка команды
```

Рисунок 16. Открытие приложение в subprocessе и считывание данных.

В результате обработки данных в приложении в стандартный поток вывода будут отправлены команды для робота. Эти команды будут считаны скриптом из этого потока.

```
std::cout << "D" << std::endl;
std::cout.flush();
Sleep (1000);
```

Рисунок 17. Пример отправки команды в поток вывода.

Так как в стандартном потоке используется буферизация, то для мгновенной отправки сообщения необходимо форсировать вывод. А после применить функцию Sleep для паузы.

Инициализация и проверка статуса сканера реализуются при помощи OpenNI2.

```
OpenNI::initialize();
Status rc = m_pInternal->m_pDevice->open(uri);
if (rc != STATUS_OK)
{
    printf("Open device failed:\n%s\n", OpenNI::getExtendedError());
    return;
}
initialize();
```

Рисунок 18. Открытие сенсора.

```
void ClosestPoint::initialize()
{
    m_pInternal->m_pStreamListener = NULL;
    m_pInternal->m_pListener = NULL;

    m_pInternal->m_pDepthStream = new VideoStream;
    Status rc = m_pInternal->m_pDepthStream->create(*m_pInternal->m_pDevice, SENSOR_DEPTH);
    if (rc != STATUS_OK)
    {
        printf("Created failed\n%s\n", OpenNI::getExtendedError());
        return;
    }

    m_pInternal->m_pStreamListener = new StreamListener(m_pInternal);

    rc = m_pInternal->m_pDepthStream->start();
    if (rc != STATUS_OK)
    {
        printf("Start failed:\n%s\n", OpenNI::getExtendedError());
    }

    m_pInternal->m_pDepthStream->addNewFrameListener(m_pInternal->m_pStreamListener);
}
```

Рисунок 19. Инициализация

Происходит инициализация библиотеки, сбор информации о драйверах, портах, состоянии устройства. Методы реализации описаны в заголовочном файле библиотеки OpenNI.h. При этом Kinect будет

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						29
Изм.	Лист	№ докум.	Подп.	Дата		

инициализироваться с помощью драйвера libfreenect, установленного в режиме совместимости с библиотекой OpenNI2. Он будет собирать все необходимые данные и передавать их по запросу в нужные функции.

```
Status ClosestPoint::getNextData(IntPoint3D& closestPoint, VideoFrameRef& rawFrame)
{
    Status rc = m_pInternal->m_pDepthStream->readFrame(&rawFrame);
    if (rc != STATUS_OK)
    {
        printf("readFrame failed\n%s\n", OpenNI::getExtendedError());
    }
}
```

Рисунок 20. Прием новых кадров

Принимает новый кадр, для поиска ближайшей точки на изображении.

```
struct IntPoint3D
{
    int X;
    int Y;
    int Z;
};
```

Рисунок 21. Хранение координат точки

Для хранения точек используется структура, хранящая координаты и значение глубины каждой точки изображения.

```

DepthPixel* pDepth = (DepthPixel*)rawFrame.getData();
bool found = false;
closestPoint.Z = 0xffff;
int width = rawFrame.getWidth();
int height = rawFrame.getHeight();

for (int y = 0; y < height; ++y)
    for (int x = 0; x < width; ++x, ++pDepth)
    {
        if (*pDepth < closestPoint.Z && *pDepth != 0)
        {
            closestPoint.X = x;
            closestPoint.Y = y;
            closestPoint.Z = *pDepth;
            found = true;
        }
    }

if (!found)
{
    return STATUS_ERROR;
}

```

Рисунок 22. Проход по массиву.

Для определения ближайшей точки необходимо определить формат глубины, размеры кадра. Если точки не были найдены, возвращается статус ошибки. Это значит, что объект находится слишком близко к сенсору и вышел за пределы его рабочей зоны. Ближайшая точка находится методом прохода по всему значению массива точек. В случае обнаружения точки выставляется значение found и статус Ok.

```

class MyMwListener : public closest_point::ClosestPoint::Listener
{
public:
    void readyForNextData(closest_point::ClosestPoint* pClosestPoint)
    {
        openni::VideoFrameRef frame;
        closest_point::IntPoint3D closest;
        openni::Status rc = pClosestPoint->getNextData(closest, frame);
    }
}

```

Рисунок 23. Проверка статуса.

Принимает значение ближайшей точки на данном кадре, и если был возвращен статус –ок , то переходит к проверке данной точки на соответствие условиям, если же нет, то возвращается сообщение об ошибке обновления и команда об остановке.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						31
Изм.	Лист	№ докум.	Подп.	Дата		

Варианты условий зависят от режима работы робота. Если это слежение, то проходит проверка на местоположение объекта.

Если объект достаточно близко, то робот ожидает на месте, если объекта вообще нет в зоне видимости, то выводится сообщение об отсутствии объекта.

Если же объект начинает удаляться, то робот начинает следование, поворачивая, если объект смещается из середины изображения.

```

if (40 <= closest.X && closest.X <= 600 && 60<= closest.Y && closest.Y <= 440 && closest.Z<=1050 && 750< closest.Z){
    if (closest.X <= 150){
        printf ("stop\n");
        printf ("turn left\n");
        Sleep(5000);
    }
    if (490 <= closest.X){
        printf ("stop\n");
        printf ("turn right\n");
        Sleep(5000);
    }
    printf("follow\n");
    Sleep (3000);
}

```

Рисунок 24. Проверка точки

В функции main происходит инициализация, вызов всех методов.

```

int main()
{
    |
    closest_point::ClosestPoint closestPoint;

    if (!closestPoint.isValid())
    {
        printf("ClosestPoint: error in initialization\n");
        return 1;
    }
    MyMwListener myListener;
    closestPoint.setListener(myListener);

    while (!wasKeyboardHit())
    {
        Sleep(1000);
    }
    closestPoint.resetListener();

    return 0;
}

```

Рисунок 25. Main

В режиме избегания препятствий точки проходят проверку на иное условие.

```
if (40 <= closest.X && closest.X <= 600 && 40<= closest.Y && closest.Y <= 440 && closest.Z<=750)
/*printf("stop\n");
printf("turn right\n");*/
std::cout << ' ' << std::endl;
std::cout.flush();
Sleep (1000);
std::cout << "D" << std::endl;
std::cout.flush();
Sleep (1000);
}
else{
//printf("go\n");
std::cout << "W" << std::endl;
std::cout.flush();
Sleep (1000);
}
```

Рисунок 26. Проверка точки

Робот останавливается и производит поворот, если рядом был найден объект, иначе он продолжает движение вперед. После передачи каждой команды, как было отмечено выше, происходит пауза, чтобы команда была обработана.

```
else
{
/*printf("Update failed\n");
printf("stop\n");*/
std::cout << ' ' << std::endl;
std::cout.flush();
Sleep (1000);
std::cout << "S" << std::endl;
std::cout.flush();
Sleep (1000);
std::cout << "D" << std::endl;
std::cout.flush();
Sleep (1000);
}
```

Рисунок 27. Близкий объект

Если же была получено сообщение об ошибке и невозможности получить новые кадры, то происходит остановка робота. Данное событие означает, что объект слишком близко к сенсору, поэтому отправляется команда на задний ход и разворот, чтобы избежать столкновения.

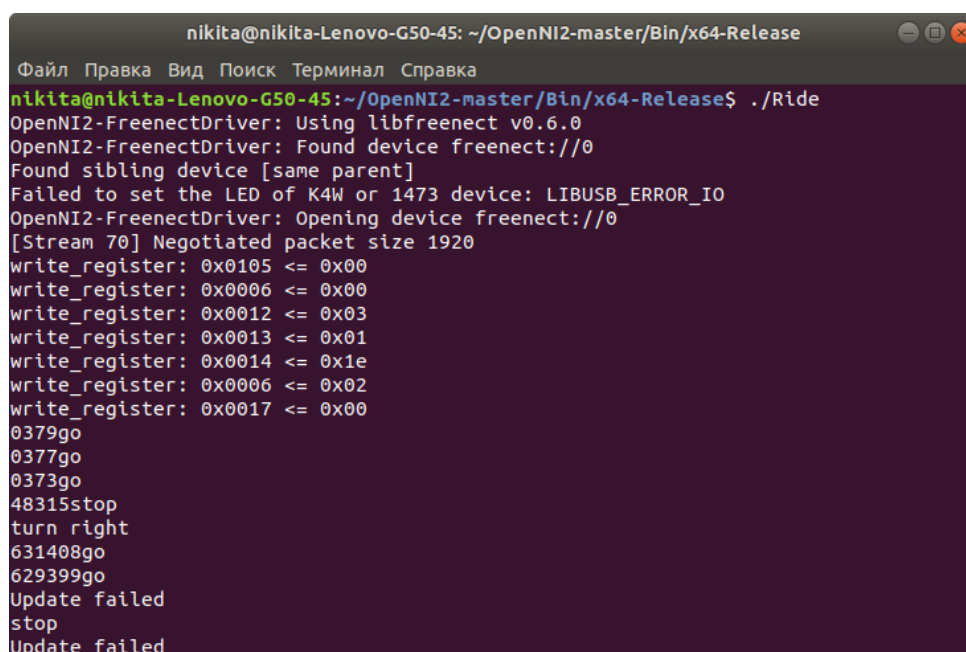
## 5. Тестирование системы.

### 5.1 Тестирование работы системы принятия решений.

Прежде чем интегрировать систему в робота, проводилось отдельное тестирование системы.

Первым тестировался режим автономного перемещения. Для наглядной демонстрации в консоль выводились принятые решения. В комнате со слабым освещением сенсор Kinect приближался к различным объектам для проверки реакции. Первый тест проводился при срезе рабочей зоны до разрешения 320x240, однако его результаты оказались неудовлетворительными. Связано это было с тем, что при приближении к объектам ближайшая точка чаще всего находилась за пределами этой зоны. И если объект находился не по центру, то система не воспринимала его до критического сближения. После расширения размеров рабочей зоны качество работы алгоритма сильно увеличилось.

Если перед Kinect в указанной близости не было объектов в консоль выводилась команда «go». Если же в зоне появлялся объект, то выводились команды на остановку и разворот. В нескольких случаях из-за задержки между командами сенсору не удавалось уловить препятствие. Однако в этом случае, как только объект заполнял все поле обзора Kinect возвращалось сообщение об ошибке и сигнал на остановку, так как переставало поступать изображение.



```
nikita@nikita-Lenovo-G50-45: ~/OpenNI2-master/Bin/x64-Release
Файл Правка Вид Поиск Терминал Справка
nikita@nikita-Lenovo-G50-45:~/OpenNI2-master/Bin/x64-Release$ ./Ride
OpenNI2-FreenectDriver: Using libfreenect v0.6.0
OpenNI2-FreenectDriver: Found device freenect://0
Found sibling device [same parent]
Failed to set the LED of K4W or 1473 device: LIBUSB_ERROR_IO
OpenNI2-FreenectDriver: Opening device freenect://0
[Stream 70] Negotiated packet size 1920
write_register: 0x0105 <= 0x00
write_register: 0x0006 <= 0x00
write_register: 0x0012 <= 0x03
write_register: 0x0013 <= 0x01
write_register: 0x0014 <= 0x1e
write_register: 0x0006 <= 0x02
write_register: 0x0017 <= 0x00
0379go
0377go
0373go
48315stop
turn right
631408go
629399go
Update failed
stop
Update failed
```

Рисунок 28. Тест реакции



```

nikita@nikita-Lenovo-G50-45: ~/OpenNI2-master/Bin/x64-Release
Файл Правка Вид Поиск Терминал Справка
Update failed
stop
Update failed
stop
Update failed
stop
Update failed
stop
Update failed
stop
Update failed
stop
62405stop
turn right
396475go
541475go
511468go
1473go
58466go
465302stop
turn right
57322stop
turn right
^[write_register: 0x0006 <= 0x00
OpenNI2-FreenectDriver: Closing device freenect://0
nikita@nikita-Lenovo-G50-45:~/OpenNI2-master/Bin/x64-Release$

```

Рисунок 29. Вывод решений

Следующее тестирование проходила система слежения. В консоль выводилось сообщение об отсутствии объекта, стоп, если объект появлялся в ближней зоне робота, follow если объект уходил в дальнюю зону, и сообщения о поворотах, если объект смещался относительно камеры.

```

nikita@nikita-Lenovo-G50-45: ~/OpenNI2-master/Bin/x64-Release
Файл Правка Вид Поиск Терминал Справка
nikita@nikita-Lenovo-G50-45:~/OpenNI2-master/Bin/x64-Release$ ./Follow
OpenNI2-FreenectDriver: Using libfreenect v0.6.0
OpenNI2-FreenectDriver: Found device freenect://0
Found sibling device [same parent]
Failed to set the LED of K4W or 1473 device: LIBUSB_ERROR_IO
OpenNI2-FreenectDriver: Opening device freenect://0
[Stream 70] Negotiated packet size 1920
write_register: 0x0105 <= 0x00
write_register: 0x0006 <= 0x00
write_register: 0x0012 <= 0x03
write_register: 0x0013 <= 0x01
write_register: 0x0014 <= 0x1e
write_register: 0x0006 <= 0x02
write_register: 0x0017 <= 0x00
nikogo netnikogo netnikogo netnikogo netfollow
follow
stop
nikogo netstop
stop
stop
follow
nikogo netfollow
nikogo netfollow
follow

```

Рисунок 30. Тест слежения

```

nikita@nikita-Lenovo-G50-45: ~/OpenNI2-master/Bin/x64-Release
Файл Правка Вид Поиск Терминал Справка
nikogo netstop
stop
stop
follow
nikogo netfollow
nikogo netfollow
follow
nikogo netstop
turn left
follow
nikogo netfollow
stop
stop
stop
stop
turn right
follow
stop
turn right
follow
nikogo netnikogo netnikogo netnikogo netnikogo net^[write_register: 0x0006 <= 0x
00
OpenNI2-FreenectDriver: Closing device freenect://0
nikita@nikita-Lenovo-G50-45:~/OpenNI2-master/Bin/x64-Release$

```

Рисунок 31. Тест слежения

В результате тестов были выявлены проблемы связанные, с потерей объекта, когда тот смещался относительно центра. К сожалению, данную проблему реально решить только при переходе на алгоритм распознавания объектов. Из-за задержки в обработке команд также пришлось снизить частоту обновления данных о препятствиях. Из-за чего некоторые не отмечались, однако в этом случае при критическом сближении сенсор перестает передавать новые кадры, что приводит к остановке. Так что на частоту столкновений это влияния не оказало. В целом данный алгоритм показал приемлемые результаты в обоих режимах.

## Заключение

В процессе выполнения выпускной работы была разработана программная система управления движением мобильного робота. Протестирована ее работа в двух режимах. Изучены возможности обработки данных с изображений и алгоритмов, основанных на сенсорах глубины. В дальнейшем планируется разработка подобной системы уже на основе алгоритмов компьютерного зрения, для повышения точности работы. Однако алгоритм, разработанный в заданных аппаратных условиях, показал себя также достаточно успешно.

					<b>ВКР-НГТУ-09.03.01-(15-В-1)-008-2019 (ПЗ)</b>	Лист
						37
Изм	Лист	№ докум.	Подп.	Дата		

## Список литературы

1. Jarrett Webb, James Ashley Beginning Kinect Programming with the Microsoft Kinect SDK (Expert's Voice in Microsoft) 1st ed. Edition (February 2012)
2. Clemente Giorio Kinect in Motion - Audio and Visual Tracking by Example (April 2013)
3. Sean Kean, Jonathan Hall, Phoenix Perry Meet the Kinect: An Introduction to Programming Natural User Interfaces (Technology in Action) 1st ed. Edition (December 2011)
4. Intel [электронный ресурс] Режим доступа: <https://www.intel.ru/content/www/ru/ru/architecture-and-technology/realSense-overview.html>
5. Control Engineering Россия – Февраль 2014 “Системы компьютерного зрения: современные задачи и методы” – Алексей Потапов
6. Методы определения расстояния до препятствия при движении мобильного робота – О.В. стрельцов, А.О.Даниленко
7. Алгоритмы ускоренной обработки изображений препятствий в системе технического зрения робота – Туан Зунг Нгуен
8. Использование веб-камеры для обнаружения препятствия на пути движения мобильного робота – Ю.В.Литвинов, В.В.Мазулина, С.Н.Фролов
9. GitHub [электронный ресурс] Режим доступа: <https://github.com/OpenNI/OpenNI2>
10. GitHub [электронный ресурс] Режим доступа: <https://github.com/OpenKinect/libfreenect>

## Приложение А

### Main.cpp

```
#include <MWClosestPoint.h>
#include <OpenNI.h>
#include <iostream>
#include <unistd.h> // STDIN_FILENO
#include <stdlib.h>
#include <termios.h> // заголовочный файл для перевода терминала
в неканонический режим
#include <fcntl.h>
using namespace std;
int wasKeyboardHit()
{
    struct termios oldt, newt;
    int ch;
    int oldf;

    // don't echo and don't wait for ENTER
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);

    // make it non-blocking (so we can check without
waiting)
    if (0 != fcntl(STDIN_FILENO, F_SETFL, oldf |
O_NONBLOCK))
    {
        return 0;
    }

    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    if (0 != fcntl(STDIN_FILENO, F_SETFL, oldf))
    {
        return 0;
    }

    if(ch != EOF)
    {
        ungetc(ch, stdin);
        return 1;
    }
}
```

```

        return 0;
    }

void Sleep(int ms)
{
    usleep(ms*1000);
}

class MyMwListener : public
closest_point::ClosestPoint::Listener
{
public:
    void readyForNextData(closest_point::ClosestPoint*
pClosestPoint)
    {
        openni::VideoFrameRef frame;
        closest_point::IntPoint3D closest;
        openni::Status rc = pClosestPoint->getNextData(closest,
frame);

        if (rc == openni::STATUS_OK)
        {

            if (40 <= closest.X && closest.X <= 600 && 40<=
closest.Y && closest.Y <= 440 && closest.Z<=750){

                std::cout << ' ' << std::endl;
                std::cout.flush();
                Sleep (1000);
                std::cout << "D" << std::endl;
                std::cout.flush();
                Sleep (1000);
            }
            else{
                //printf("go\n");
                std::cout << "W" << std::endl;
                std::cout.flush();
                Sleep (1000);
            }

        }

        else
        {

```

```

        std::cout << ' ' << std::endl;
        std::cout.flush();
        Sleep (1000);
        std::cout << "S" << std::endl;
        std::cout.flush();
        Sleep (1000);
        std::cout << "D" << std::endl;
        std::cout.flush();
        Sleep (1000);
    }
}

};

int main()
{
    int x;
    closest_point::ClosestPoint closestPoint;

    if (!closestPoint.isValid())
    {
        printf("ClosestPoint: error in initialization\n");
        return 1;
    }
    MyMwListener myListener;
    closestPoint.setListener(myListener);

    while (!wasKeyboardHit())
    {
        Sleep(1000);
    }
    closestPoint.resetListener();

    return 0;
}

```

## ClosestPoint.cpp

```

#include <OpenNI.h>
#include "MWClosestPoint.h"

using namespace openni;

namespace closest_point
{
    class StreamListener;

    struct ClosestPointInternal
    {
        ClosestPointInternal(ClosestPoint* pClosestPoint) :

```

```

        m_pDevice(NULL), m_pDepthStream(NULL),
m_pListener(NULL), m_pStreamListener(NULL),
m_pClosesPoint(pClosestPoint)
        {}

void Raise()
{
    if (m_pListener != NULL)
        m_pListener->readyForNextData(m_pClosesPoint);
}
bool m_oniOwner;
Device* m_pDevice;
VideoStream* m_pDepthStream;

ClosestPoint::Listener* m_pListener;

StreamListener* m_pStreamListener;

ClosestPoint* m_pClosesPoint;
};

class StreamListener : public VideoStream::NewFrameListener
{
public:
    StreamListener(ClosestPointInternal* pClosestPoint) :
m_pClosestPoint(pClosestPoint)
    {}
    virtual void onNewFrame(VideoStream& stream)
    {
        m_pClosestPoint->Raise();
    }
private:
    ClosestPointInternal* m_pClosestPoint;
};

ClosestPoint::ClosestPoint(const char* uri)
{
    m_pInternal = new ClosestPointInternal(this);

    m_pInternal->m_pDevice = new Device;
    m_pInternal->m_oniOwner = true;

    OpenNI::initialize();
    Status rc = m_pInternal->m_pDevice->open(uri);
    if (rc != STATUS_OK)
    {
        printf("Open device failed:\n%s\n",
OpenNI::getExtendedError());
        return;
    }
    initialize();
}

```



```

ClosestPoint::ClosestPoint(openni::Device* pDevice)
{
    m_pInternal = new ClosestPointInternal(this);

    m_pInternal->m_pDevice = pDevice;
    m_pInternal->m_oniOwner = false;

    OpenNI::initialize();

    if (pDevice != NULL)
    {
        initialize();
    }
}

void ClosestPoint::initialize()
{
    m_pInternal->m_pStreamListener = NULL;
    m_pInternal->m_pListener = NULL;

    m_pInternal->m_pDepthStream = new VideoStream;
    Status rc = m_pInternal->m_pDepthStream-
>create(*m_pInternal->m_pDevice, SENSOR_DEPTH);
    if (rc != STATUS_OK)
    {
        printf("Created failed\n%s\n",
OpenNI::getExtendedError());
        return;
    }

    m_pInternal->m_pStreamListener = new
StreamListener(m_pInternal);

    rc = m_pInternal->m_pDepthStream->start();
    if (rc != STATUS_OK)
    {
        printf("Start failed:\n%s\n",
OpenNI::getExtendedError());
    }

    m_pInternal->m_pDepthStream-
>addNewFrameListener(m_pInternal->m_pStreamListener);
}

ClosestPoint::~~ClosestPoint()
{
    if (m_pInternal->m_pDepthStream != NULL)
    {
        m_pInternal->m_pDepthStream-
>removeNewFrameListener(m_pInternal->m_pStreamListener);
    }
}

```

```

        m_pInternal->m_pDepthStream->stop();
        m_pInternal->m_pDepthStream->destroy();

        delete m_pInternal->m_pDepthStream;
    }

    if (m_pInternal->m_pStreamListener != NULL)
    {
        delete m_pInternal->m_pStreamListener;
    }

    if (m_pInternal->m_oniOwner)
    {
        if (m_pInternal->m_pDevice != NULL)
        {
            m_pInternal->m_pDevice->close();

            delete m_pInternal->m_pDevice;
        }
    }

    OpenNI::shutdown();

    delete m_pInternal;
}

bool ClosestPoint::isValid() const
{
    if (m_pInternal == NULL)
        return false;
    if (m_pInternal->m_pDevice == NULL)
        return false;
    if (m_pInternal->m_pDepthStream == NULL)
        return false;
    if (!m_pInternal->m_pDepthStream->isValid())
        return false;

    return true;
}

Status ClosestPoint::setListener(Listener& listener)
{
    m_pInternal->m_pListener = &listener;
    return STATUS_OK;
}

void ClosestPoint::resetListener()
{
    m_pInternal->m_pListener = NULL;
}

```

```

Status ClosestPoint::getNextData(IntPoint3D& closestPoint,
VideoFrameRef& rawFrame)
{
    Status rc = m_pInternal->m_pDepthStream-
>readFrame(&rawFrame);
    if (rc != STATUS_OK)
    {
        printf("readFrame failed\n%s\n",
OpenNI::getExtendedError());
    }

    DepthPixel* pDepth = (DepthPixel*)rawFrame.getData();
    bool found = false;
    closestPoint.Z = 0xffff;
    int width = rawFrame.getWidth();
    int height = rawFrame.getHeight();

    for (int y = 0; y < height; ++y)
        for (int x = 0; x < width; ++x, ++pDepth)
        {
            if (*pDepth < closestPoint.Z && *pDepth != 0)
            {
                closestPoint.X = x;
                closestPoint.Y = y;
                closestPoint.Z = *pDepth;
                found = true;
            }
        }

    if (!found)
    {
        return STATUS_ERROR;
    }

    return STATUS_OK;
}
}

```

## ClosestPoint.h

```

#ifndef _MW_CLOSEST_POINT_H_

#define _MW_CLOSEST_POINT_H_

#include <OpenNI.h>

#ifdef _CLOSEST_POINT

```

```
#define MW_CP_API ONI_API_EXPORT  
  
#else  
  
#define MW_CP_API ONI_API_IMPORT  
  
#endif
```

```
namespace openni  
{  
  
    class Device;  
  
}
```

```
namespace closest_point  
{
```

```
struct IntPoint3D  
{  
  
    int X;  
  
    int Y;  
  
    int Z;  
  
};
```

```
struct ClosestPointInternal;
```

```
class MW_CP_API ClosestPoint  
{  
  
public:  
  
    class Listener  
    {
```

```

public:
    virtual void readyForNextData(ClosestPoint*) = 0;
};

ClosestPoint(const char* uri = NULL);
ClosestPoint(openni::Device* pDevice);
~ClosestPoint();

bool isValid() const;

openni::Status setListener(Listener& listener);
void resetListener();

openni::Status getNextData(IntPoint3D& closestPoint,
openni::VideoFrameRef& rawFrame);
private:
    void initialize();

    ClosestPointInternal* m_pInternal;
};

}

#endif // _MW_CLOSEST_POINT_H_

```