**Введение**

Преимущества автоматизации технологического процесса неоспоримы. Она повышает объём производства и качество продукции. Задача автоматизации технологического процесса подразумевает автоматизацию его составных частей – технологических операций. В данной работе предложена система управления роботом, решающая задачу автоматизации перемещения. Ведь именно операция перемещения является неотъемлемой частью любого технологического процесса.

Наиболее универсальный путь автоматизация операции перемещения – использование промышленного робота-манипулятора, оснащённого системой машинного зрения. Такое решение позволяет сократить затраты на переналадку и переоборудование производственной линии при смене параметров продукции.

Внедрение данной технологии невозможно без разработки системы управления, позволяющей манипулятору работать автономно. Таким образом целью данной работы является разработка системы управления промышленным роботом на базе машинного зрения.

Далее представлен цикл перемещения объекта производства (далее просто объекта).



**Описание элементов, входящих в систему управления.**

**RoboDK –** среда оффлайн программирования, симуляции и онлайн программирования роботов.

**RoboDK C# API –** библиотека взаимодействия со средой RoboDK на языке программирования C#.

**Lotus** – программа, выполняющая следующие функции:

1. Взаимодействие с подключёнными устройствами видеосъёмки и настройка параметров видеопотока.
2. Распознавание объектов и определение их координат.
3. Взаимодействие со средой RoboDK.
4. Предоставление пользовательского графического интерфейса для регулировки параметров распознавания и параметров перемещения.
5. Отображение текущего процесса распознавания объектов.
6. Интеграция трёхмерной среды для симуляции перемещения робота.

**KUKAVarProxy –** программа, принимающая команды на чтение и запись системных переменных контроллера робота от удалённого компьютера.

**apikuka.exe** – драйвер робота служит интерфейсом взаимодействия RoboDK с программой KUKAVarProxy.

**Системные переменные –** список переменных разных типов, хранящийся в файле, расположенном на жёстком диске контроллера.

**Программа синхронизации ­-**  программа, написанная на языке KRLи запускаемая через человеко-машинный интерфейс контроллера. Предназначена для чтения системных переменных и взаимодействия с ядром контроллера**.**

**Робот –** промышленный манипулятор, обладающий шестью степенями свободы. При отладке системы управления использовалась модель KUKA KR6 R700. Подключён к контроллеру робота по интерфейсу передачи данных X21 и к порту питания приводов X20.

**Контроллер –** система непосредственного управления промышленным манипулятором. В работе была использована модель KUKA KR C4.

**Рабочая зона –** плоская поверхность, находящаяся в непосредственной близости с роботом. Имеет прямоугольную форму.

**Камера** – видеокамера, подключённая по интерфейсу USB2.0 к промышленному компьютеру.

**Оператор** – человек, производящий пуск и наладку системы.

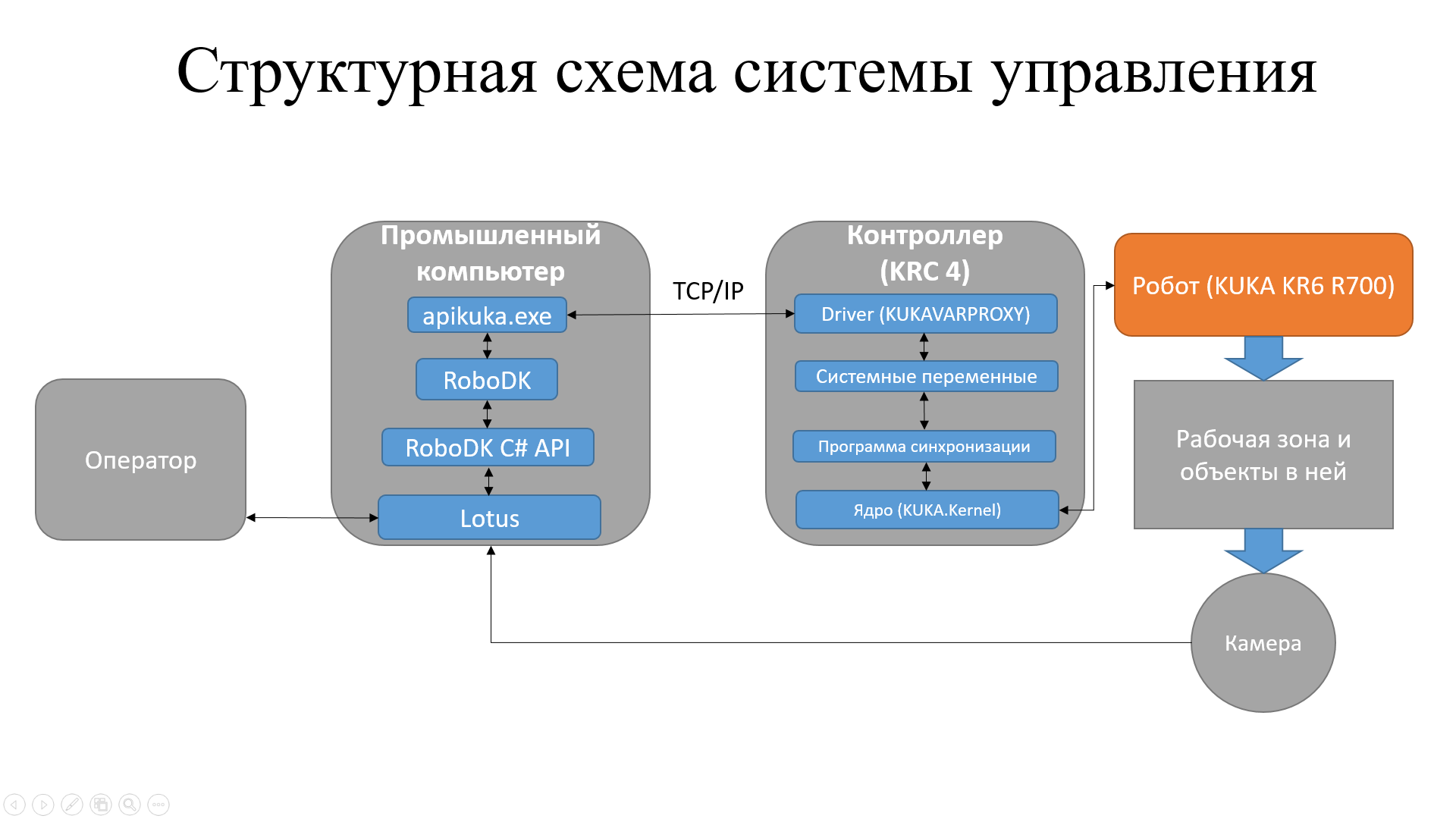
**Причины выбора среды RoboDK**

1. Наличие ограниченной бесплатной лицензии.
2. Возможность онлайн программирования роботов от большинства производителей.
3. Наличие программного интерфейса приложения (API).
4. Документация в доступном формате на английском языке.

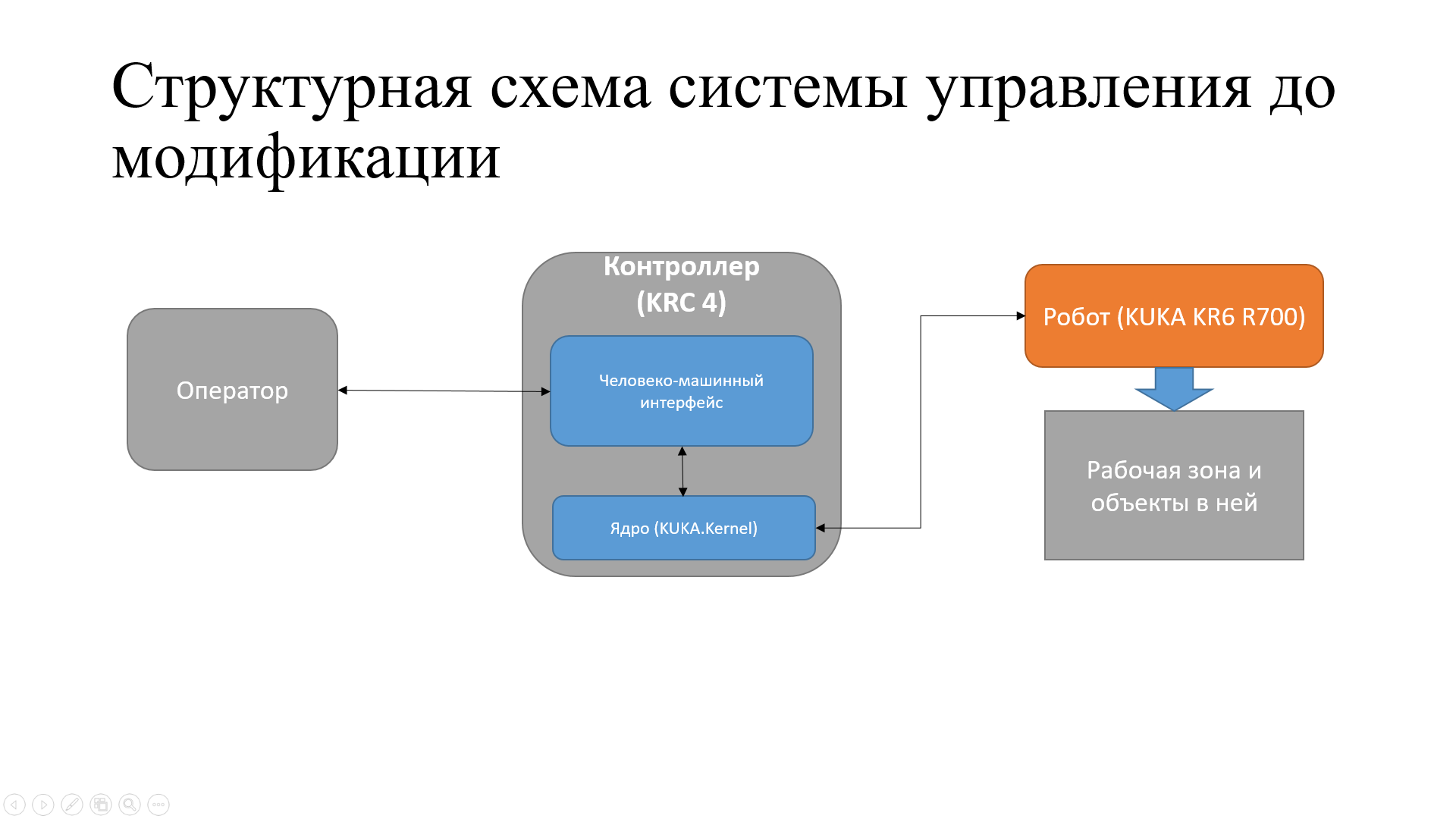
**Описание структурной схемы**

Структурная схема системы управления включает в себя контроллер, манипулятор, промышленный компьютер, рабочую зону с объектами манипулирования, камеру и оператора, управляющего системой.

Позиционирование робота начинается с захвата изображения рабочей зоны камерой с помощью программы Lotus, далее, посредством анализа изображения, распознаются координаты объектов манипуляции. После того, как координаты объектов были перенесены в систему координат робота, программа Lotus отправляет команду библиотеке RoboDK C# API на взаимодействие со средой RoboDK [2]. Далее среда RoboDK взаимодействует по интерфейсу Ethernet и протоколу TCP/IP [1] с сервером редактирования системных переменных контроллера KUKAVarProxy [3]. После того, как системные переменные контроллера были изменены извне, программа синхронизации переменных, запущенная через человеко-машинный интерфейс на контроллере, считывает значение этой переменной и отправляет команду в ядро контроллера KUKA Kernel на изменение положения манипулятора.



**Рис. . Структурная схема системы управления**



**Рис. . Структурная схема системы управления до модификации.**

**Преимущества разработанной схемы**

* Экономия вычислительных ресурсов контроллера.
* Неограниченные возможности разработки алгоритма распознавания образов.
* Выбор камеры зависит лишь от специфики автоматизируемой операции. Это позволяет использовать как более дешёвое оборудование, так и более совершенное.

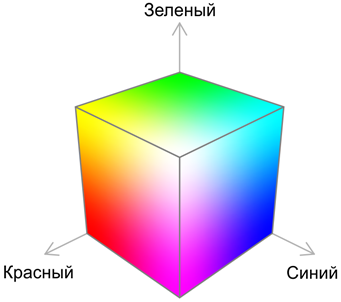
**Описание алгоритма распознавания положения объекта.**

**Этап калибровки.**

На этапе калибровки сохраняется информация о положении рабочей зоны в кадре. Оно задаётся четырьмя точками, которые соответствуют вершинам прямоугольной зоны, в которой позже будут появляться объекты и будут распознаваться их координаты. Во время калибровки в рабочей области должны отсутствовать объекты манипулирования.

Также на этом этапе происходит сохранение маски фона рабочей области и среднего цвета этой маски.

Если представить пиксели изображения, как множество точек, у которых есть цвет, полученный в результате сложения трёх цветовых компонент, то такое множество можно рассматривать в трёхмерном пространстве этих трёх признаков. Координату в данном пространстве определяет интенсивность каждой из трёх цветовых компонент. Расстоянием между точками, которые являются некоторыми цветами, в таком пространстве является отличие между этими цветами. Чем больше абсолютное значение расстояния от точки А до точки B в данном пространстве признаков, тем больше отличие цвета A от цвета B.

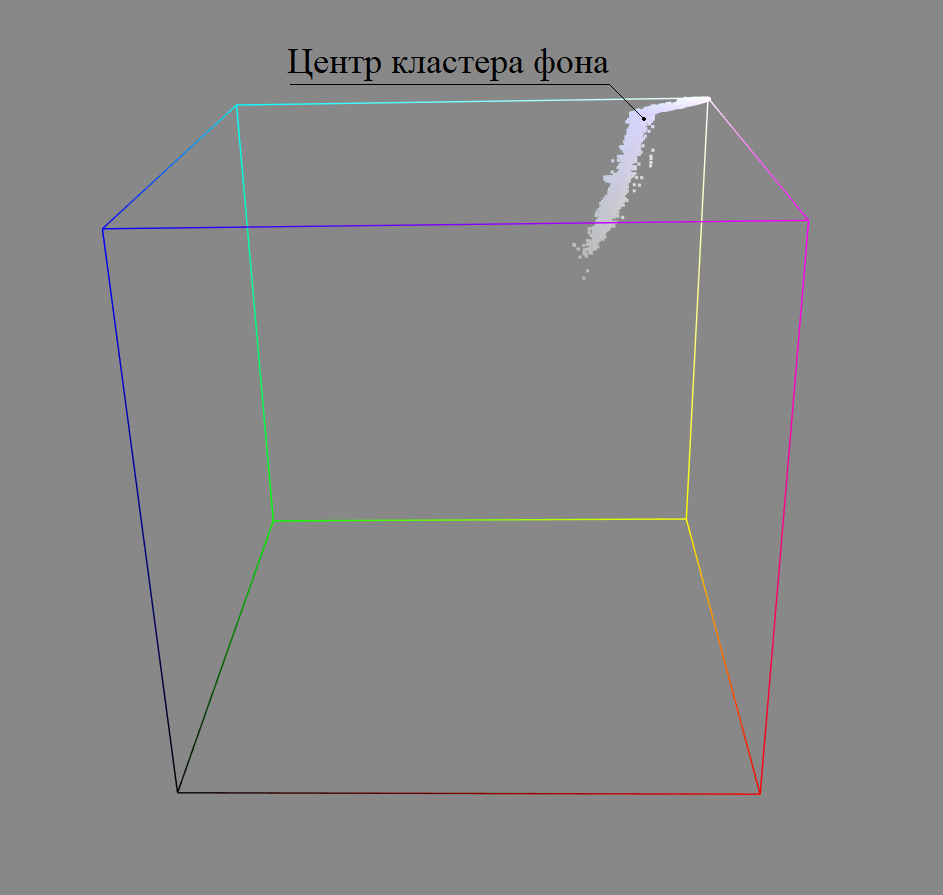


**Рис. . Модель цветового пространства.**

Таким образом средний цвет маски фона будет являться центром кластера его пикселей в пространстве признаков.



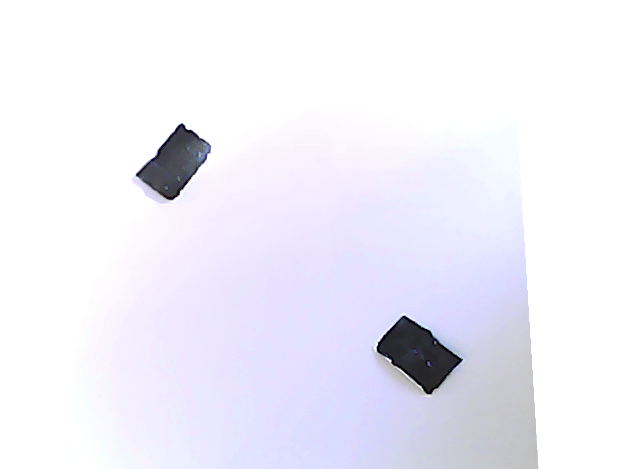
**Рис. . Маска фона рабочей области изображения.**



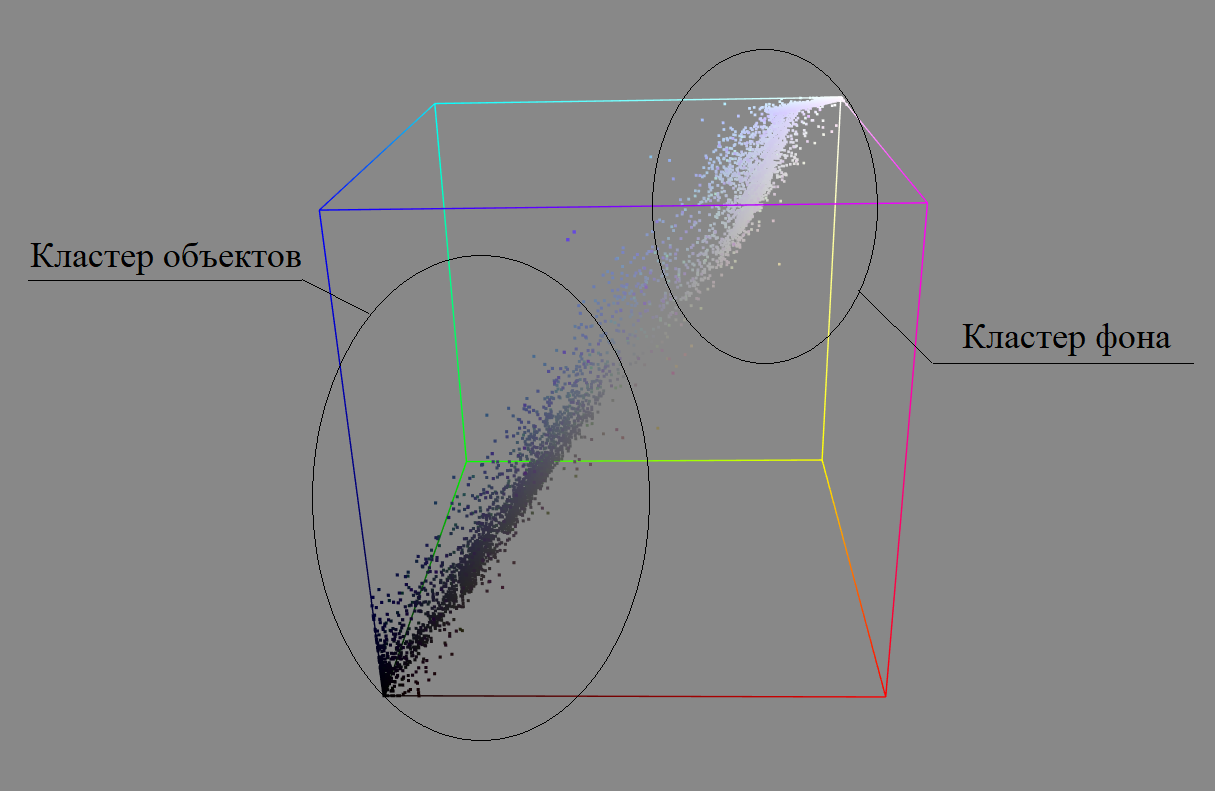
**Рис. . Визуализация маски фона в цветовом пространстве.**

**Этап распознавания.**

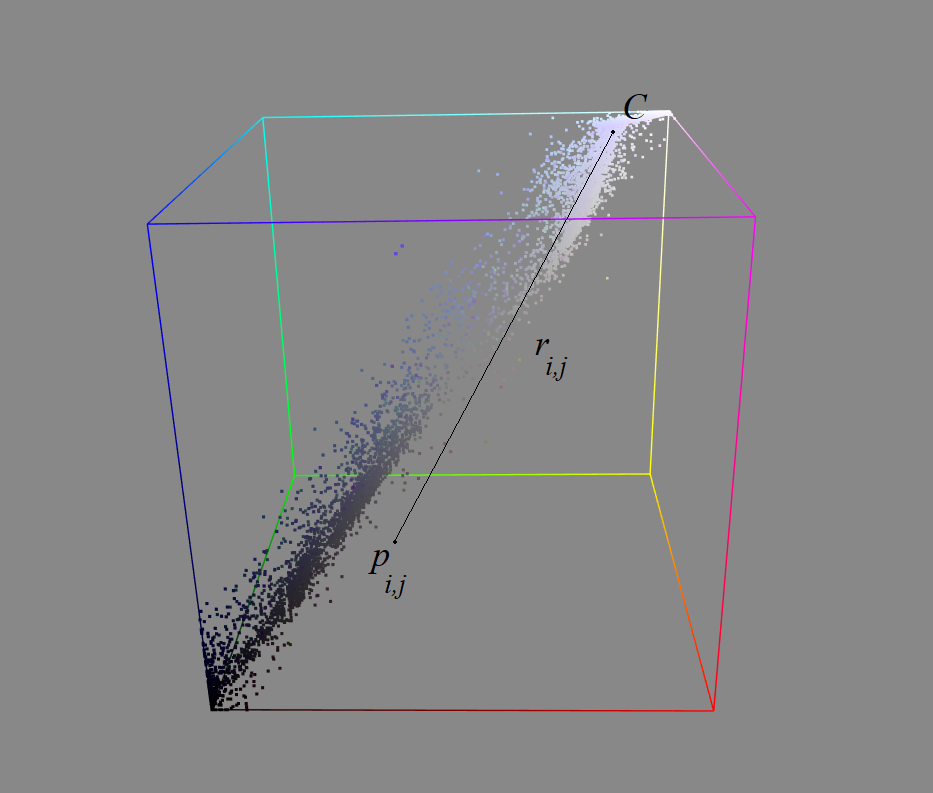
После того, как была получена информация о фоне рабочей области начинается распознавание положения объекта: программа Lotus получает видеопоток с камеры по последовательному интерфейсу USB 2.0. Далее из видеопотока захватывается изображение, которое сравнивается с маской.



**Рис. . Маска с объектами в ней.**



**Рис. . Визуализация маски с объектами в цветовом пространстве.**



**Рис. . Определение .**

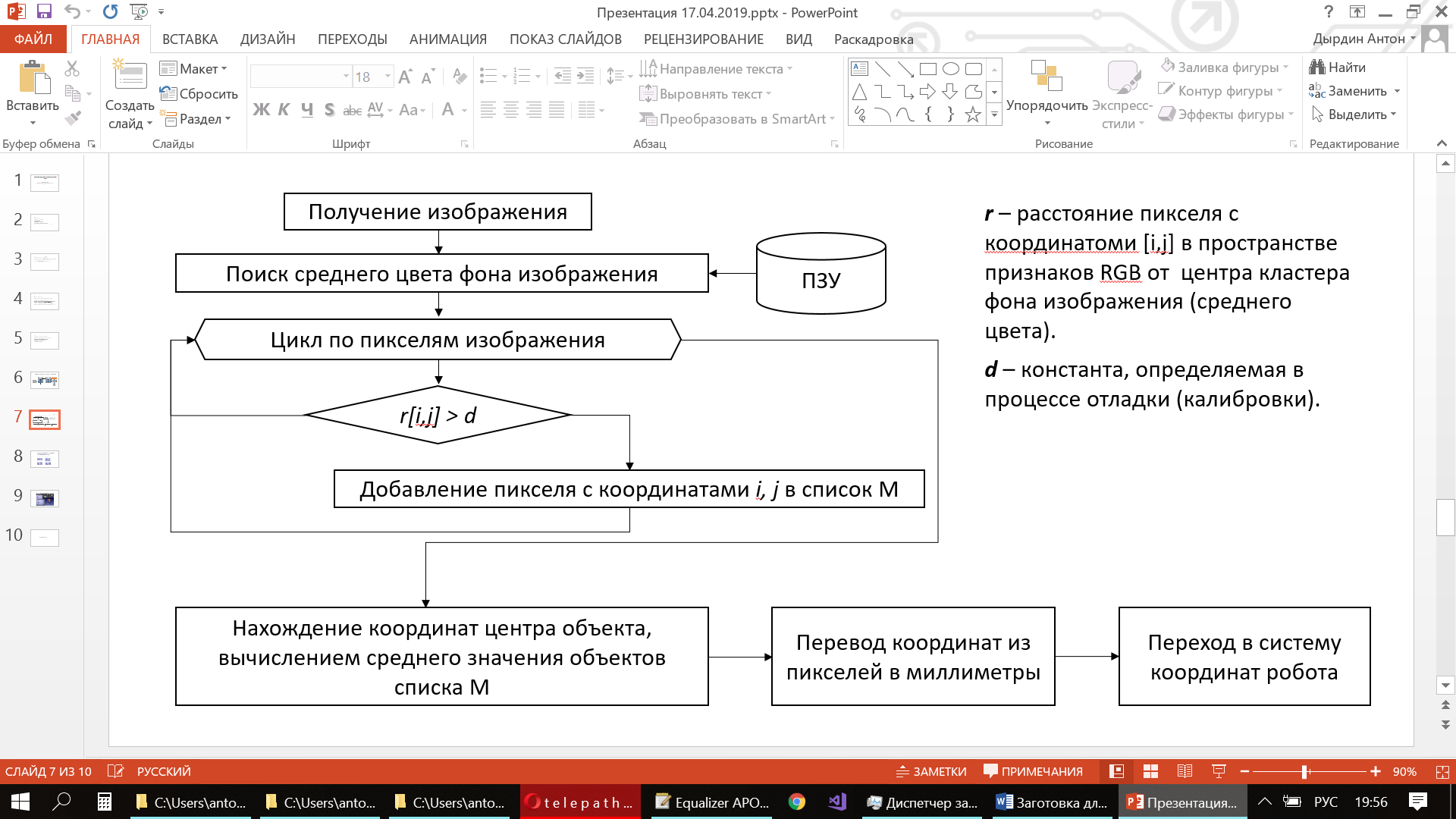
Для каждого пикселя , входящего в маску рабочей области выполняется проверка удалённости его цвета от цвета фона ***C***. Если больше, чем константа ***d*** , значение которой выбирается экспериментально (как правило оно находится в интервале [127;255]), то пиксель с индексами ***i,j*** указывается, как пиксель, принадлежащий множеству пикселей отображающих объект. После того, как множество всех пикселей маски разбито на два кластера («фон» и «объект»), определяются координаты (в пикселях) центра объекта вычислением среднего арифметического координат пикселей, принадлежащих кластеру объекта.

, где

- индекс столбца пикселей изображения, в котором находится центр объекта – координата ***y*** в пикселях,

- индекс строки пикселей изображения, в которой находится центр объекта – координата ***x*** в пикселях

N – количество пикселей, принадлежащих кластеру «объект».

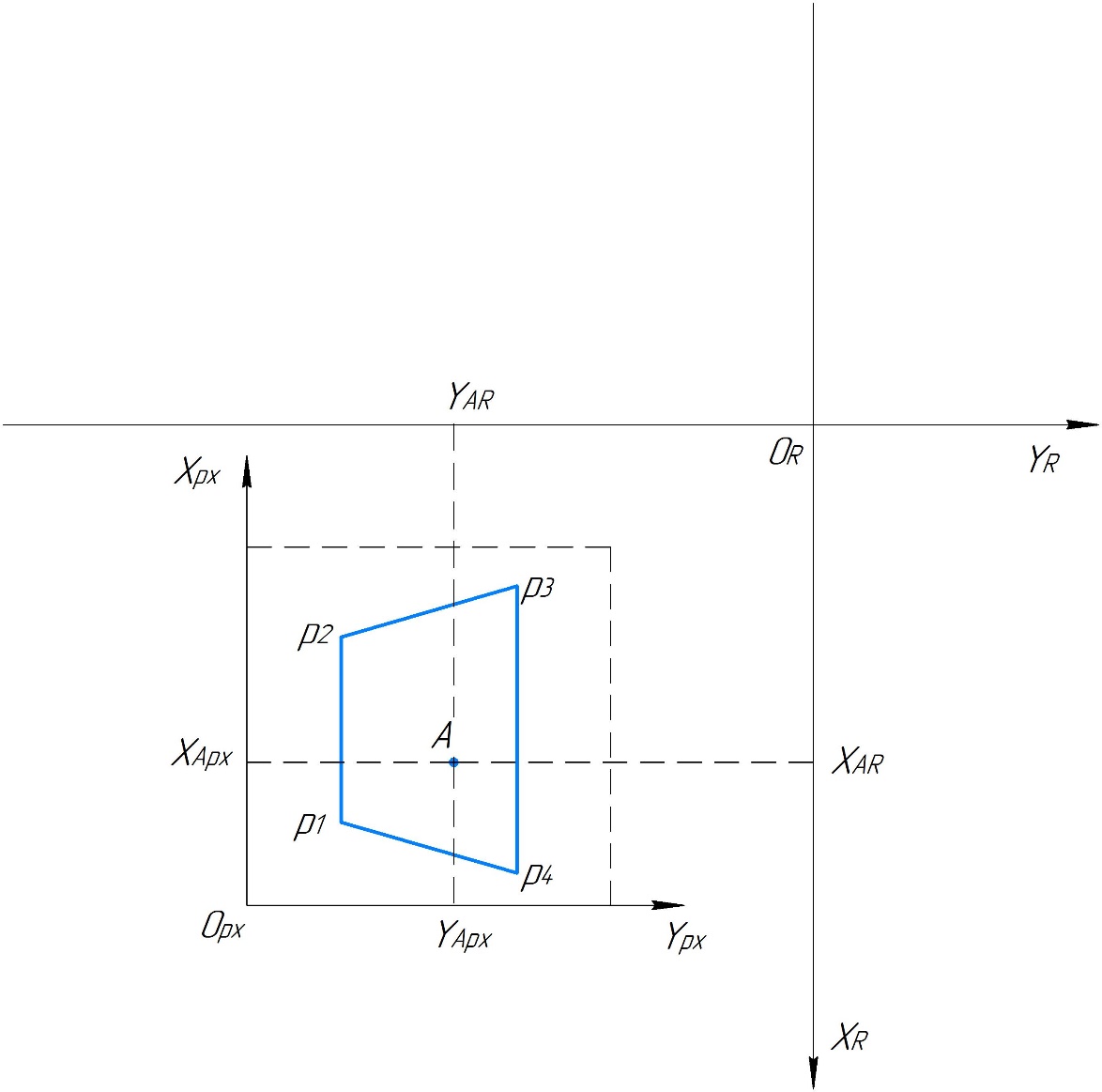


**Рис. . Блок-схема алгоритма распознавания координат одного объекта.**

**Модификация алгоритма для распознавания нескольких объектов, находящихся в рабочей области одновременно.**

Для распознавания координат нескольких объектов была введена константа objectSize, определяющая средний размер объектов. При этом слишком большое значение даст ошибку при распознавании координат близко расположенных объектов, а слишком малое приведёт к тому, что один объект будет определяться, как два или более. После кластеризации множества пикселей изображения, попавших в маску на два кластера, кластер «объект» делится методом ближайших соседей [5], но уже по координатам на кластеры «объект №1», «объект №2» … «объект n». Далее их координаты в пикселях определяются аналогично алгоритму для одного объекта.

**Переход к системе координат робота.**

****

**Рис. . Система координат изображения *px* в системе координат робота *R*.**

Так как камера установлена стационарно под углом к прямоугольной рабочей области, то её контур в перспективе имеет форму трапеции . Зная реальные высоту *Sh* и ширину *Sw* рабочей области, координаты точек в системе координат *px*, координаты точки *p2* в системе координат робота и координаты точки *A* в системе координат *px*, задачу переноса координат объекта *А* из системы координат изображения *px,* где единицей измерения является пиксель, в систему координат робота *r*, где единицей измерения является миллиметр,можно представить, как задачу нахождения функций *f* и *g:*

Найденное решение:

– коэффициент перевода пикселей в миллиметры по горизонтали

– коэффициент перевода пикселей в миллиметры по вертикали

Слагаемое служит для компенсации трапецевидности прямоугольной области в кадре.

**Описание разработанной программы Lotus**

**Системные требования.**

Целевая платформа - x86-64 (AMD64/Intel64/EM64T)

Совместимые операционные системы - Windows XP/ 7/ 8 /8.1 /10

Минимальная версия пакета библиотек - .NET Framework 4.

Центральный процессор Intel Pentium 4.

Объём ОЗУ – 2ГБ.

Видеокамера с интерфейсом USB 2.0 и разрешением не менее 320x240.

Адаптер интерфейса Ethernet.

**Описание файлов исходного кода.**

*App.config* – файл конфигурации приложения. Содержит XML текст.

*FastBitmap.cs* – класс, содержащий методы повышающий скорость взаимодействия с пикселями изображений.

*Form1.cs* – класс главной формы приложения. Содержит методы взаимодействия с видеокамерами, методы взаимодействия с RoboDK C# API, обработчики событий пользовательского графического интерфейса, цикл распознавания и методы, позволяющие запускать этот цикл в отдельном потоке, а также останавливать его.

*Form1.Designer.cs* – код, сгенерированный конструктором форм WindowsForms.

*Preview.cs* – форма, появляющаяся во время запуска главной формы и пропадающая после того, как главная форма была загружена.

*Preview.Designer.cs* - код, сгенерированный конструктором форм WindowsForms.

*Program.cs* – класс, содержащий точку входа в программу. В нём происходит вызов главной формы.

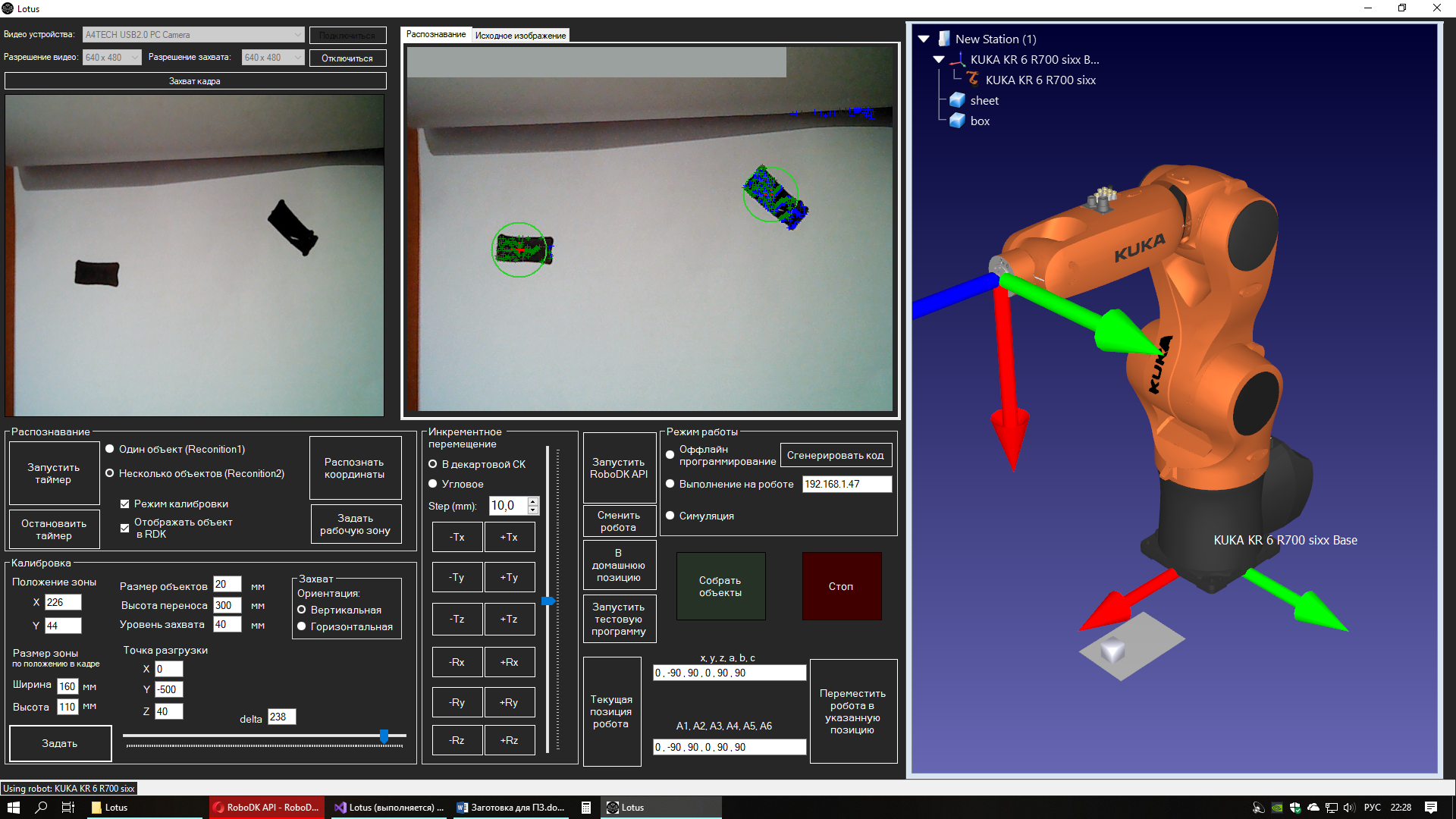
*Recognition1.cs* – класс, содержащий реализацию алгоритма распознавания одного объекта.

*Recognition2.cs* - класс, содержащий реализацию алгоритма распознавания одного и более объектов.

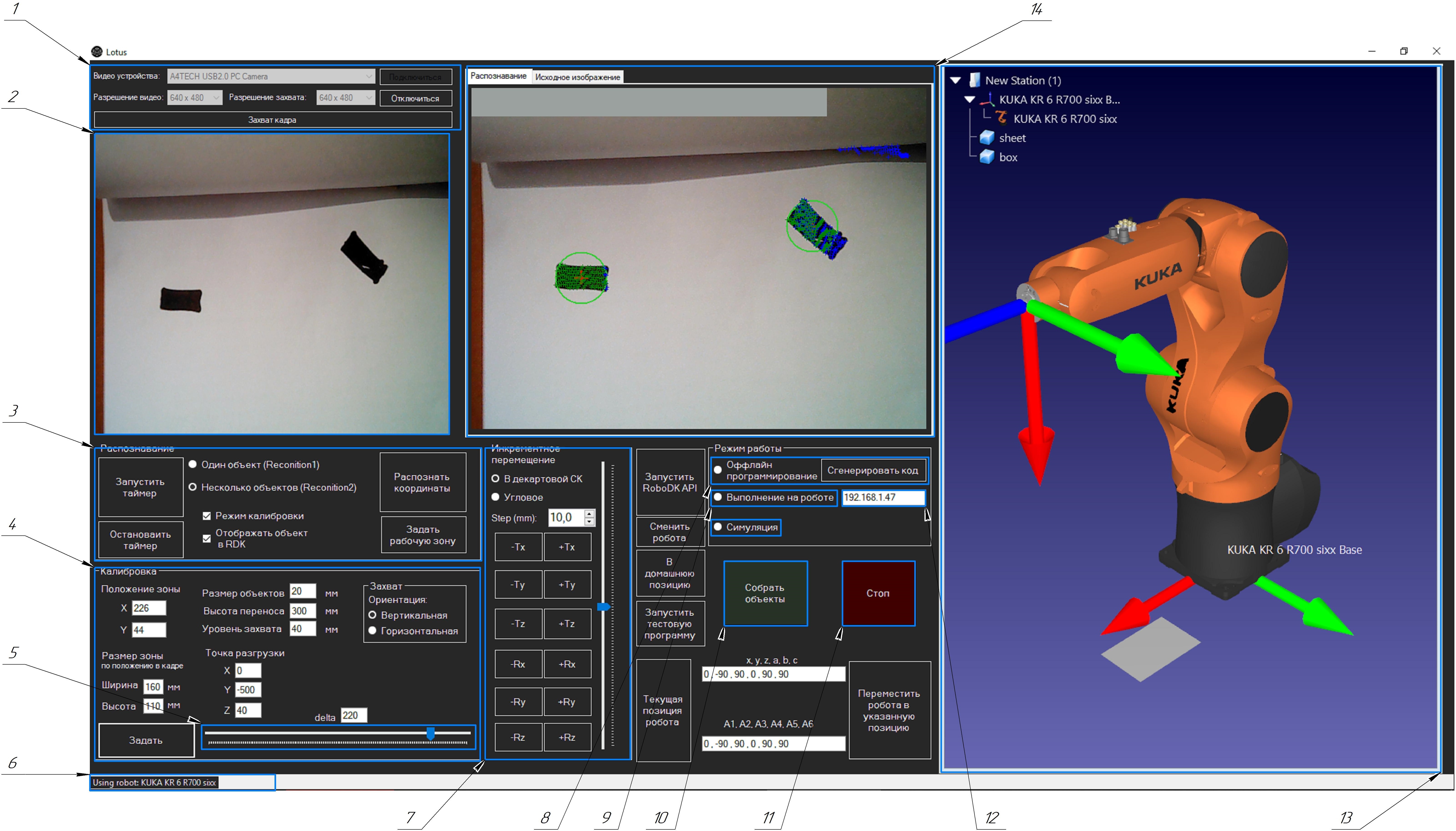
*RoboDK.cs* – исходный файл RoboDK C# API. Содержит класс RoboDK, класс RoboDK.Item и другие инструменты для робототехники, такие как Matrix для матричных операций для работы с преобразованиями позиций и координат.

*RobotControl.cs* – содержит код, определяющий логику перемещения объектов роботом.

**Описание графического интерфейса программы.**



**Рис. . Внешний вид пользовательского графического интерфейса.**



**Рис. . Графический интерфейс.**

1 – Элементы управления видеоустройствами,

2 – панель предпросмотра видеопотока,

3 – элементы управления процессом распознавания положения объектов,

4 – элементы управления параметрами захвата и перемещения объектов, координатами рабочей области относительно робота, размерами объектов и рабочей области,

5 – ползунок изменения параметра d, определяющего чувствительность алгоритма распознавания объектов,

6 – область уведомлений,

7 – элементы управления инкрементным перемещением,

8 – указатель перехода в режим оффлайн программирования и кнопка генерации кода.

9 – указатель перехода в режим онлайн программирования (альтернативного управления)

10 – кнопка, запускающая перемещение объектов в область выгрузки.

11 – кнопка остановки перемещения робота.

12 - поле ввода IP-адреса робота в сети предприятия.

13 – интегрированное окно, отображающее 3D модель робота, модели объектов и рабочей поверхности.

14 – панель, отображающая исходный видеопоток с наложенными на него эффектами, наглядно отображающими результаты распознавания.

**Список литературы**

1. Robot drivers [Online]// robodk [site] URL: https://robodk.com/doc/en/Robot-Drivers.html (date of request 10.04.2019).
2. RoboDK C# API [Online]// robodk [site] URL: <https://robodk.com/doc/en/RoboDK-API.html#CsAPI> (date of request 10.04.2019).
3. KUKAVarProxy open repository [Online]// github [site] URL: <https://github.com/ImtsSrl/KUKAVARPROXY> (date of request 10.04.2019).
4. AForgeNET framework documentation [Online]// aforgenet [site] URL: <http://www.aforgenet.com/framework/docs/> (date of request 10.04.2019).
5. Обзор алгоритмов кластеризации данных [Электронный ресурс]// habr [сайт] URL: <https://habr.com/ru/post/101338/>
6. Бабич, А. В. Промышленная робототехника / А.В. Бабич. - М.:, 2012. - 263 c.
7. Воскобойников, Б. С. Словарь по гибким производственным системам и робототехнике. Английский. Немецкий. Французский. Нидерландский / Б.С. Воскобойников, Б.И. Зайчик, С.М. Палей. - М.: Русский язык, 1991. - 392 c.
8. Иванов, А. А. Основы робототехники / А.А. Иванов. - М.: Форум, 2012. - 224 c.
9. Костров, Б. В. Искусственный интеллект и робототехника / Б.В. Костров, В.Н. Ручкин, В.А. Фулин. - М.: Диалог-Мифи, 2008. - 224 c.
10. Макаров, И. М. Робототехника. История и перспективы / И.М. Макаров, Ю.И. Топчеев. - М.: Наука, МАИ, 2003. - 352 c.
11. Петров, А. А. Англо-русский словарь по робототехнике / А.А. Петров, Е.К. Масловский.- М.: Русский язык, 1989. - 494 c.
12. Попов, Е.П. Робототехника и гибкие производственные системы / Е.П. Попов. - М.: ИЛ, 1987. - 192 c.
13. Робототехника и гибкие автоматизированные производства / ред. И.М. Макаров. - М.: Машиностроение, 1986. - 478 c.
14. Робототехника, прогноз, программирование. - М.: ЛКИ, 2008. - 208 c.
15. Юревич, Е. И. Основы робототехники / Е.И. Юревич. - Л.: Машиностроение, 1985. - 272c.
16. M. Schopfer, F. Schmidt, M. Pardowitz, and H. Ritter, “Open source real-time control software for the kuka light weight robot,” in 8th World Congress on Intelligent Control and Automation (WCICA). IEEE, 2010, pp. 444–449.
17. G. Schreiber, A. Stemmer, and R. Bischoff, “The fast research interface for the kuka lightweight robot,” in IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications How to Modify and Enhance Commercial Controllers (ICRA 2010), 2010.
18. H. M¨uhe, A. Angerer, A. Hoffmann, and W. Reif, “On reverseengineering the kuka robot language,” arXiv preprint arXiv:1009.5004, 2010.
19. F. Chinello, S. Scheggi, F. Morbidi, and D. Prattichizzo, “Kuka control toolbox,” Robotics & Automation Magazine, IEEE, vol. 18, no. 4, pp. 69–79, 2011.
20. R. Bischoff, U. Huggenberger, and E. Prassler, “Kuka youbot-a mobile manipulator for research and education,” in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2011, pp. 1–4.
21. B. Siciliano and O. Khatib, Springer handbook of robotics. Springer, 2008.
22. F. Sanfilippo, L. I. Hatledal, H. G. Schaathun, K. Y. Pettersen, and H. Zhang, “A universal control architecture for maritime cranes and robots using genetic algorithms as a possible mapping approach,” in Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, China. IEEE, 2013, pp. 322–327.
23. F. Sanfilippo, L. I. Hatledal, K. Y. Pettersen, and H. Zhang, “A mapping approach for controlling different maritime cranes and robots using ann,” in Proceeding of the 2014 IEEE International Conference on Mechatronics and Automation (ICMA 2014), in press.
24. L. I. Hatledal, F. Sanfilippo, and H. Zhang, “Jiop: a java intelligent optimisation and machine learning framework,” in Proceedings of the 28th European Conference on Modelling and Simulation (ECMS), Brescia, Italy. ECMS, 2014, pp. 101–107.