Лабораторная работа №1

Цель - изучения принципа построения конвейера алгоритмов обработки данных, получение навыков работы со стендовой программой, изучения принципов создания блоков конвейера как функций DLL.

Часть I

Теоретическая часть

Современные алгоритмы обработки данных от систем технического зрения - это конвейеры, состоящие из последовательности более мелких (простых) алгоритмов. Новый алгоритм, как правило, является некоторым изменением ступеней конвейера, например, их модификация, оптимизация, или иной способ взаимодействия между ними.

С одной стороны, имеется множество алгоритмов обработки данных, предоставляемых различными средами разработки (например, Matlab), библиотекой OpenCV и прочими библиотеками, переписывание кода которых с целью исследования и проектирования новых алгоритмов нецелесообразно. С другой стороны, их возможностей не всегда достаточно для создания новых алгоритмов, к тому же формат входных данных в разных пакетах обработки изображений в большинстве случаев различен.

Следует отметить, что различные пакеты обработки изображений и библиотеки дополняют друг друга, но, как правило, не предусматривают между собой полноценного взаимодействия.

Для разработки и исследования новых алгоритмов обработки изображений в большинстве случаев используются уже существующие алгоритмы реализованные программно, а также вспомогательные алгоритмы читающие изображения с диска, захватывающие кадры из видео и т.п.

Часть функций по обработке изображений - захват кадров из видео, чтение запись на диск, передача данных между программными модулями, просмотр промежуточных результатов, подсчет статистических данных. берет на себя специальное программное обеспечение (стенд). Стенд позволяет производить разработку новых алгоритмов без наложения ограничений на язык программирования. Это дает творческую свободу а также позволяет использовать уже существующий программный код. Задачи решаемые стендом можно отразить на рисунке 1.

Рисунок 1 - Задачи решаемые стендовым ПО

Стенд позволяет собирать и исследовать программные конвейеры из отдельных алгоритмов обработки изображений.
Архитектура стенда представлена на рисунке 2.

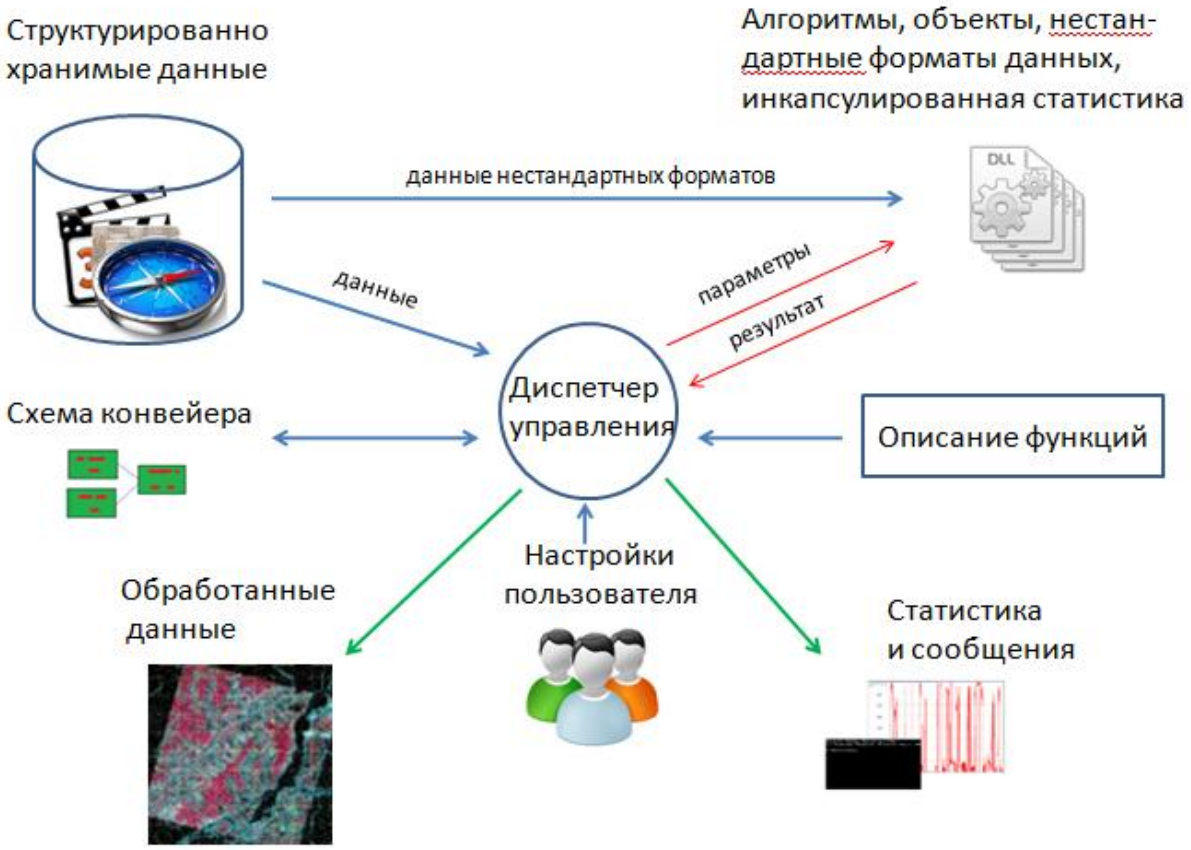


Рисунок 2 - архитектура стендового ПО

Основные определения применяемые при описании архитектуры стенда и концепции его функционирования.

Конвейер – совокупность последовательно выполняемых функций, в которой результат выполнения одной функции служит входными данными для другой.

Схема конвейера – совокупность блоков и связей между ними, представляющими собой каналы передачи данных (схема конвейера имеет графический интерфейс).

Функциональный блок - минимальный элемент схемы конвейера, реализующий одну закрепленную за этим блоком функцию обработки данных и является абстрактным представлением отдельной функции (алгоритма) для пользователя.

Функциональный блок описывает следующие характеристики выполняемой функции (процедуры, алгоритма):

- количество входов и их тип;
- тип выхода;
- значения на выходе и входах;
- имя DLL и имя функции, которую он представляет;
- указатели на соединенные с ним блоки.

Для работы с ранее не используемым алгоритмом добавляется новый блок (добавление происходит автоматически по данным конфигурационного файла). Для изменения конфигурационного файла можно воспользоваться интерфейсом стенда или изменить его вручную. Интерфейс стенда позволяет выбирать функцию из библиотеки

DLL, реализующую интересующий нас алгоритм и описать параметры этой функции и указать ее размещение в палитре функциональных блоков.

Часть графического интерфейса стенда представлено на рисунке 3.

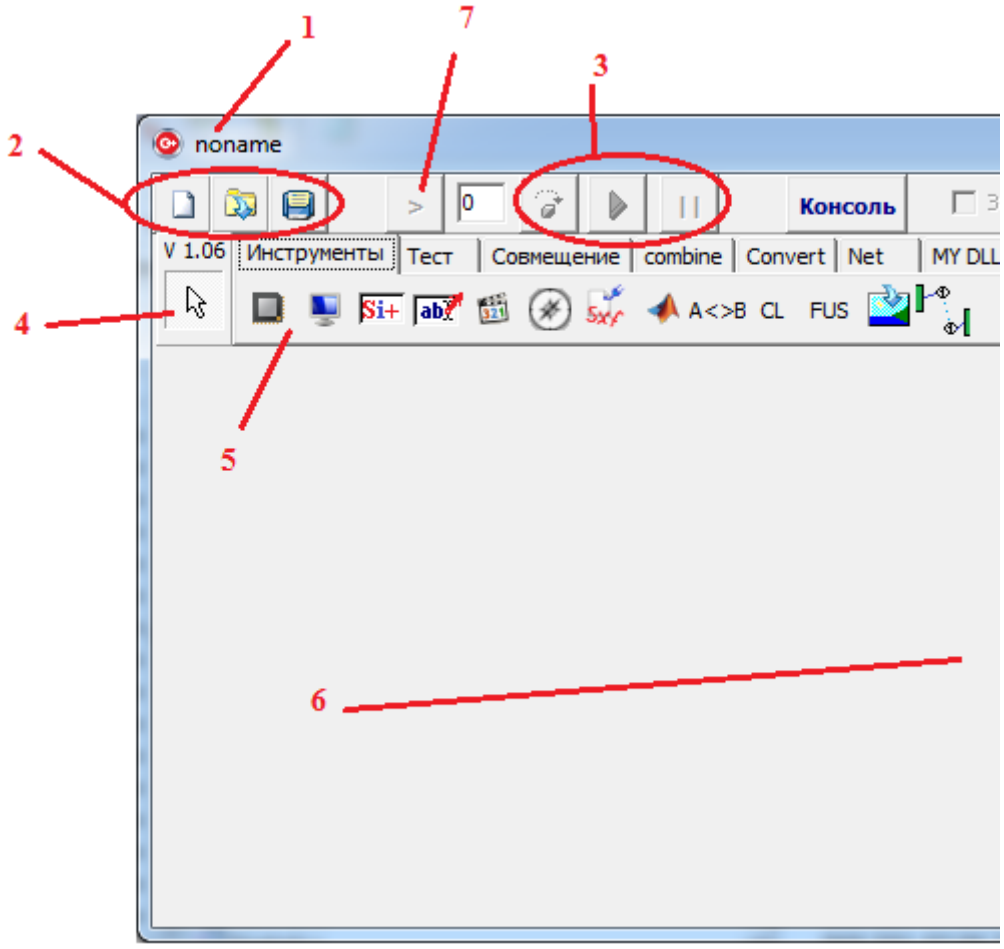


Рисунок 3 - Интерфейс стендового ПО

В заголовке окна 1 - отображается открытой схемы конвейера. По умолчанию это имя «noname».

Группа кнопок 2 позволяет создать чистый лист для сборки схемы, открыть схему или сохранить ее. В процессе работы с программой схемы автоматически не сохраняются на жестком диске. В программе доступна основная вкладка инструментов 5. На вкладке расположены фундаментальные блоки для работы с алгоритмами технического зрения. Пользователю доступны собственные вкладки, создаваемые динамически, на которых пользователь может разместить собственные блоки конвейера. Выбор блока осуществляется в режиме курсора 4. Блоки устанавливаются на листе схемы 6. Запуск схемы осуществляется кнопкой сборки схемы 7. Пошаговое и автоматическое управление процессом выполнения схемы осуществляется кнопками 3.

Блок алгоритма олицетворяет одну из выбранных функций в DLL созданных пользователем. Таким образов весь конвейер - это набор функций из различных, указанных пользователем, DLL и связи между функциями.

Установка блока на лист схемы осуществляется выбором этого блока на панели инструментов 5 и перенос его в поле 6. Блок может иметь один или несколько входов (на входы передаются параметры функции в виде указателей на требуемые структуры) и единственный выход (рисунок 4).

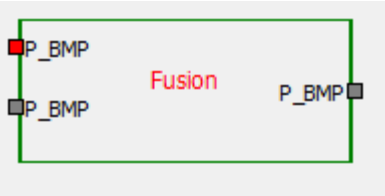


Рисунок 4 - внешний вид типового блока

Активный вход блока выделяется красным квадратом, не активные серыми. Соединение блоков между собой осуществляется протягиванием соединительной линии (связи) от активного входа к выходу соединяемого блока **(важно, именно в обратную сторону, от входа к выходу, по другому соединение выполнить невозможно).**
Выход, к которому подсоединили блок, отображается зеленым квадратом (рисунок 5).

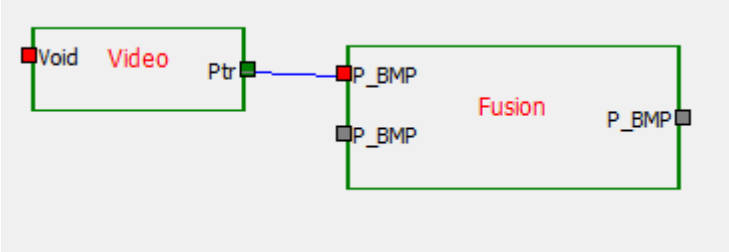


Рисунок 5 - Соединение блоков

Некоторые блоки имеют окно конфигурации параметров работы блока. Окно вызывается двойным щелчком мыши по блоку в режиме сборки схемы.
Любая схема должна иметь блок CPU, блок с которого осуществляется старт схемы (рисунок 6).

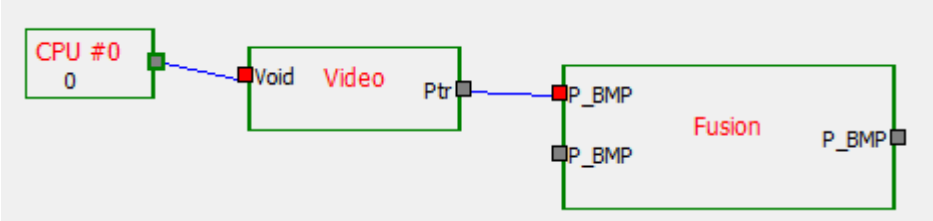


Рисунок 6 - Часть схемы конвейера с точкой старта

CPU использует заданное в блоке ядро физического процессора для обработки алгоритмов цепочки схемы присоединенной к нему. Номер ядра выбирается щелчком мыши по блоку и отображается в блоке в виде цифры после символа #.

Создание блока средствами IDE Visual Studio

Рассмотрим создание пользовательского блока с размещением его на новой пользовательской вкладке. Блок будем создавать в Visual Studio. Задача решаемая блоком - инверсия изображения.

Дополнительные сведения изображение передается в DLL как указатель на структуру:

```
struct TBmpbuf // буфер для передачи изображения
{
   int width; // ширина изображения
   int height; // высота изображения
   TPixelFormat pixelFormat;
   int reserve; // зарезервировано
   ptrPRGBTriple pixels; / указатель на массивы rgb-значений пикселей строк
};
```

Для решения поставленной задачи необходимо создать DLL с функцией инверсии изображения. Создание динамической библиотеки в Visual Studio. Выберите в меню File - New - Project. Выберите шаблон Win32 Project (последовательность действий проиллюстрирована серией рисунков).

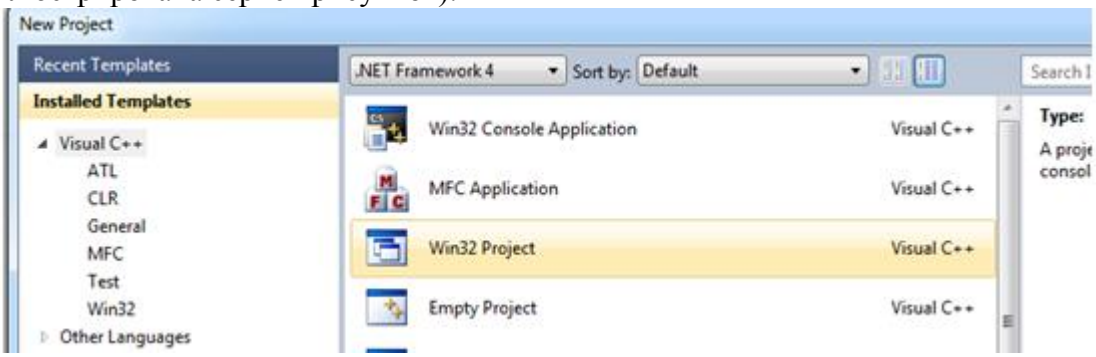


Рисунок 7 - выбор шаблона для создания пустого проекта

В мастере настройки шаблона укажите опции как показано на рисунке 8.

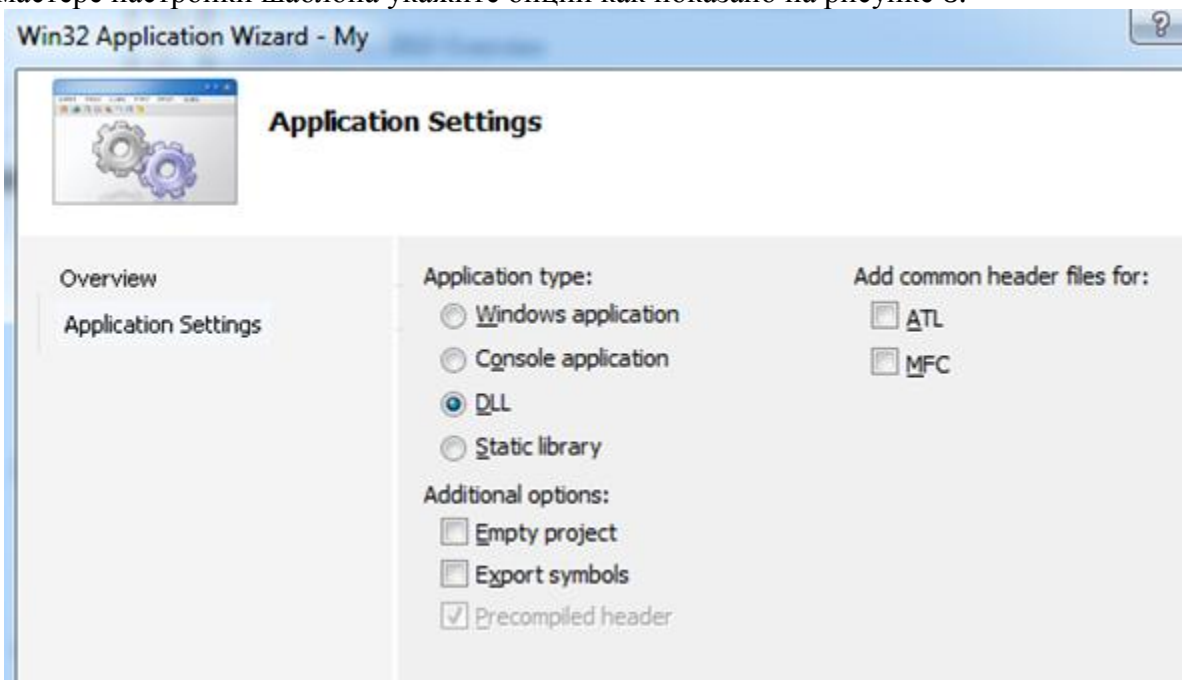


Рисунок 8 - Настройки для формирования проекта DLL

Настройте свойства проекта.

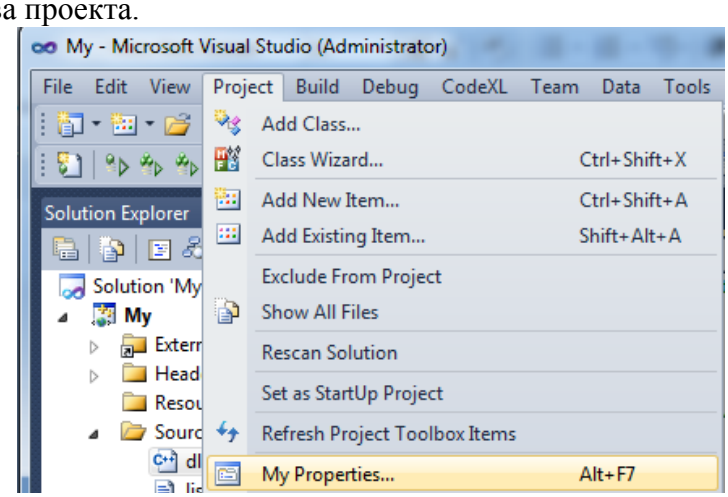


Рисунок 9 - Выбор свойств проекта

Укажите каталог, в который будет помещаться скомпилированная DLL. Этим каталогом служит каталог Plugins стенда.

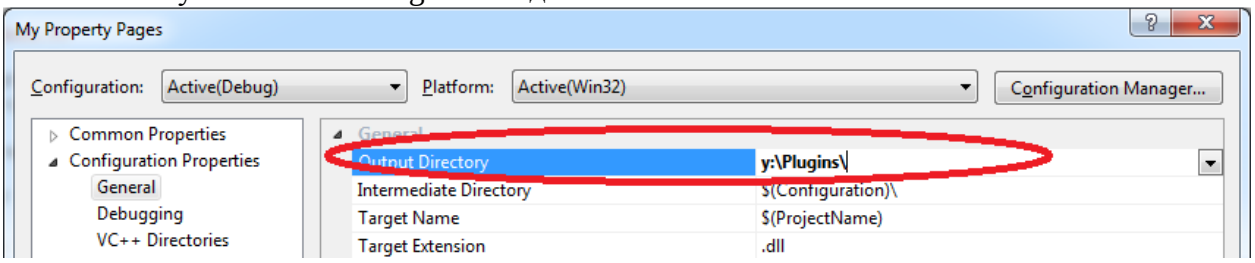


Рисунок 10 - Настройки проекта

Укажите стартовый процесс - exe файл запускающий стенд и директорию в которой он расположен как Working Directory.

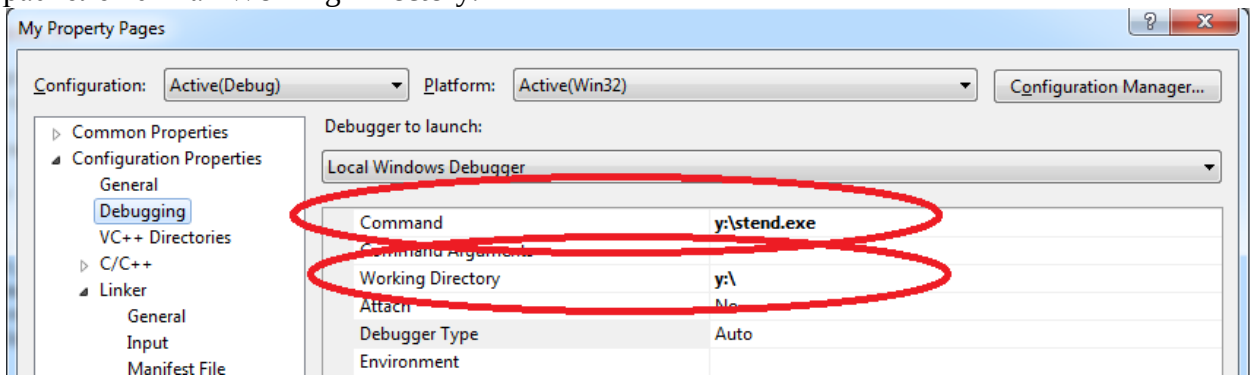


Рисунок 11 - Настройки проекта

В опциях компиляции установите режим предкомпиляции заголовков как показано на рисунке 12

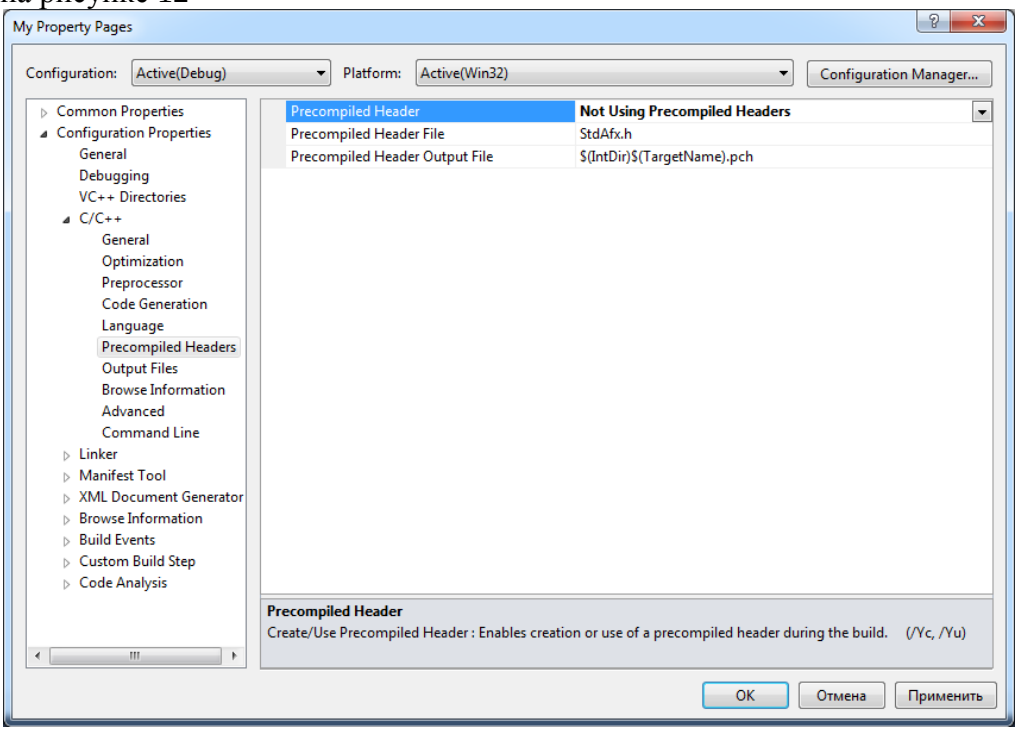


Рисунок 12 - Выбор свойств проекта

Добавьте в проект пустой текстовый файл с именем list.def и укажите его в настройках linkera.

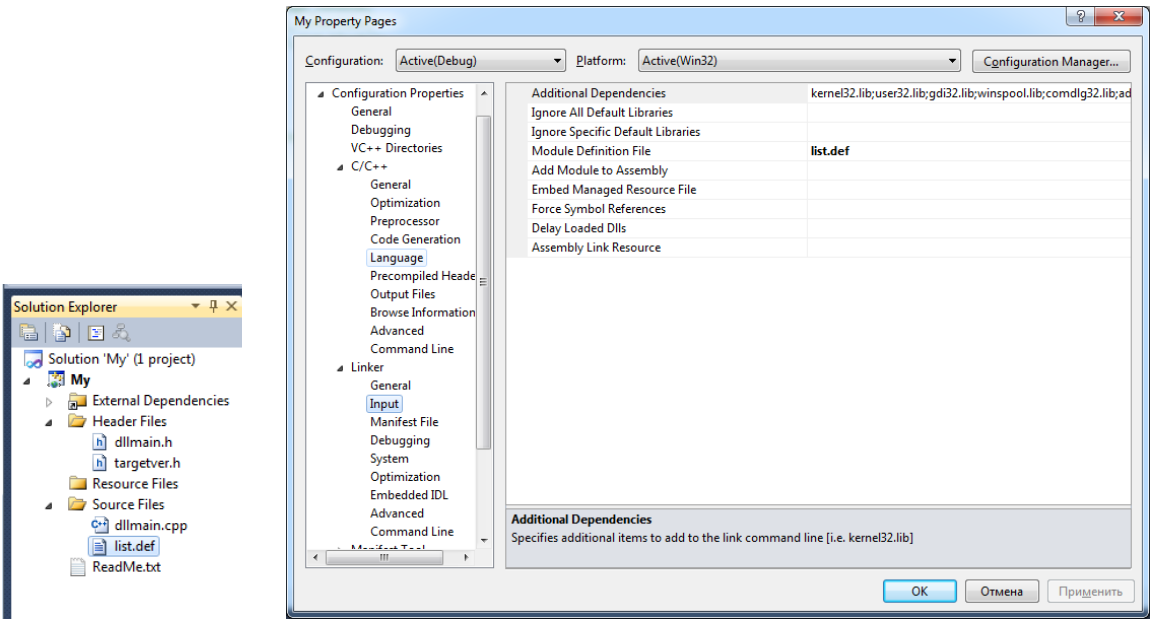


Рисунок 13 - Выбор свойств проекта

Отключите генерацию файла манифеста.

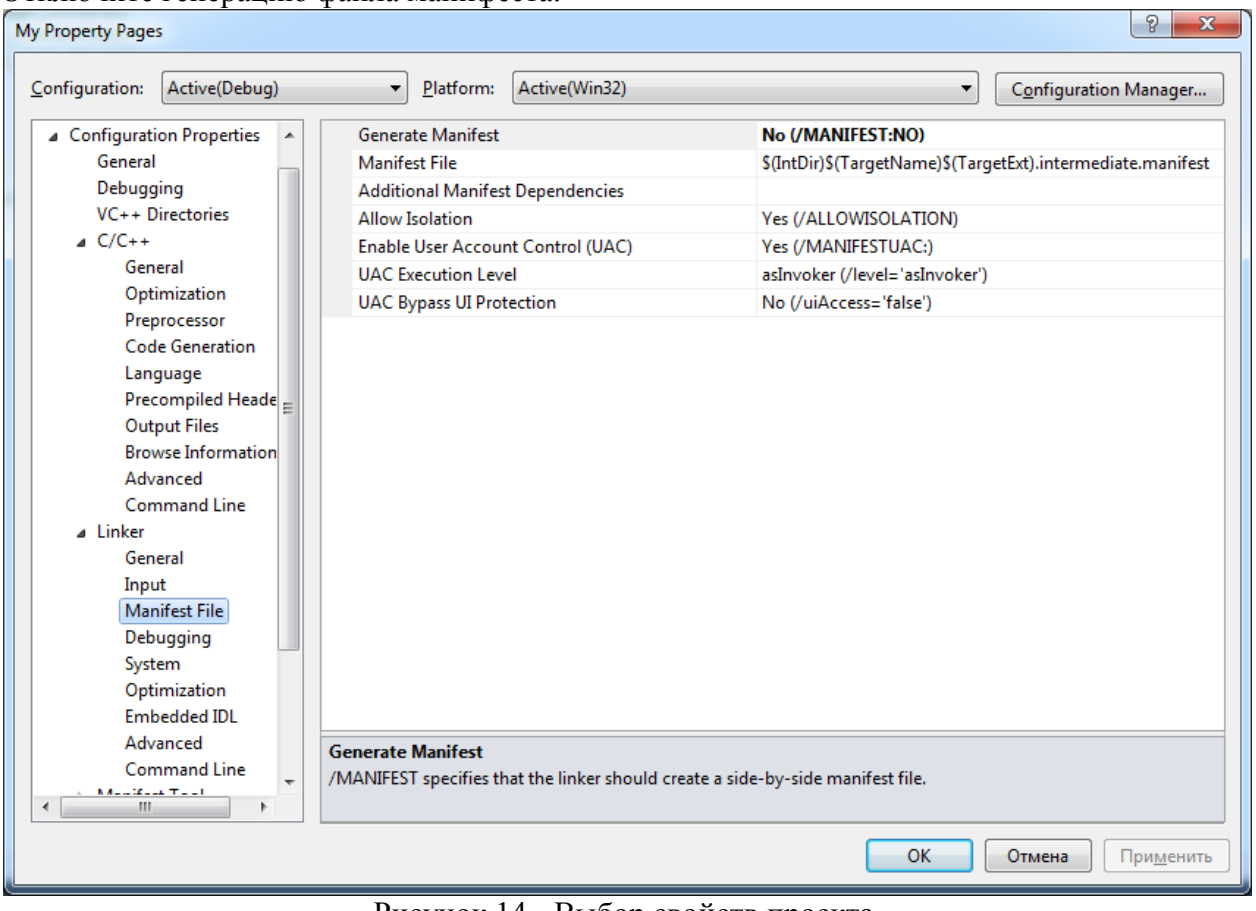


Рисунок 14 - Выбор свойств проекта

Программный код

Содержимое h файла

```
#ifdef _DLLEXPORT_
  #define _DECLARATOR_ __declspec(dllexport)
#else
  #define _DECLARATOR_ __declspec(dllimport)
#endif

extern "C"
{
  void*  _DECLARATOR_  __cdecl inversion(void* BufBmp);
}
```

Содержимое cpp файла

```
// dllmain.cpp : Defines the entry point for the DLL application.
//#include "stdafx.h"

#pragma once
//#include "targetver.h"
//#define WIN32_LEAN_AND_MEAN

#include <windows.h>

typedef tagRGBTRIPLE *PRGBTriple;
typedef PRGBTriple *ptrPRGBTriple;

enum TPixelFormat : unsigned char { pfDevice, pf1bit, pf4bit, pf8bit, pf15bit, pf16bit, pf24bit, pf32bit, pfCustom };

struct TBmpbuf // буфер для передачи изображения
{
   int width; // ширина изображения
   int height; // высота изображения
   TPixelFormat pixelFormat;
   int reserve; // зарезервировано
   ptrPRGBTriple pixels; // указатель на массивы rgb-значений пикселей
};

typedef TBmpbuf *PBmpBuf;
PRGBTriple rowSRC;

//#define _DECLARATOR_ __declspec(dllexport)

void* inversion(void* BufBmp)
{
          for (int j = 0; j < ((PBmpBuf) BufBmp)->height; j++)
   {
          rowSRC = ((PBmpBuf) BufBmp)->pixels[j];
    for (int i = 0; i < ((PBmpBuf) BufBmp)->width; i++)
      {
            rowSRC[i].rgbtBlue=255-rowSRC[i].rgbtBlue;
            rowSRC[i].rgbtGreen=255-rowSRC[i].rgbtGreen;
            rowSRC[i].rgbtRed=255-rowSRC[i].rgbtRed;
      }
    }
return BufBmp;
}


BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                          )
```

```
{
	switch (ul_reason_for_call)
	{
	case DLL_PROCESS_ATTACH:
	case DLL_THREAD_ATTACH:
	case DLL_THREAD_DETACH:
	case DLL_PROCESS_DETACH:
		break;
	}
	return TRUE;
}
```

Содержимое файла list.def

```
LIBRARY

EXPORTS
	inversion		@1   ; inversion

	inversion = _inversion
```

Добавление блока в стенд

Настраиваем конфигурацию пользовательского блока. Для этого необходимо внести изменения в файл config.txt расположенный в каталоге Plugins стенда.

Изначально файл содержит комментарий описывающий структуру записи для блока

//#File.dll,Label,Func_Name,Out_Types,OUT_label,Num_param,Types#1,Types#2,...,Types#n,File_ico_button

Необходимо внести следующее изменение: добавить новую вкладку на панель инструментов. Название вкладки задается в квадратных скобках в новой строке файла config.txt. Например так:

[My]

Затем делаем запись о блоке в виде параметров разделенных запятыми:
- имя dll;
- название, которое отобразится на блоке;
- имя функции dll которую будет олицетворять блок;
- тип выхода (указать pointer);
- метка выхода на блоке;
- количество входов блока;
-для каждого входа далее пара тип входа (pointer) и метка входа;
- спецсимвол # за которым пишется имя файла пиктограммы в находящегося в каталоге Icons в формате bmp (можно нарисовать свою или выбрать любую из существующих);
- комментарий который будет отображаться при наведении курсора на пиктограмму блока в панели управления.

В нашем примере файл конфигурации конфигурации может выглядеть так

//#File.dll,Label,Func_Name,Out_Types,OUT_label,Num_param,Types#1,Types#2,...,Types#n,File_ico_button
[My]
my.dll,"Inv",inversion,pointer,pbmp,1,pointer,pbmp,#b1.bmp,Инверсия изображения

Компилируем и запускаем проект.
В стенде собираем схему (как на рисунке 15).

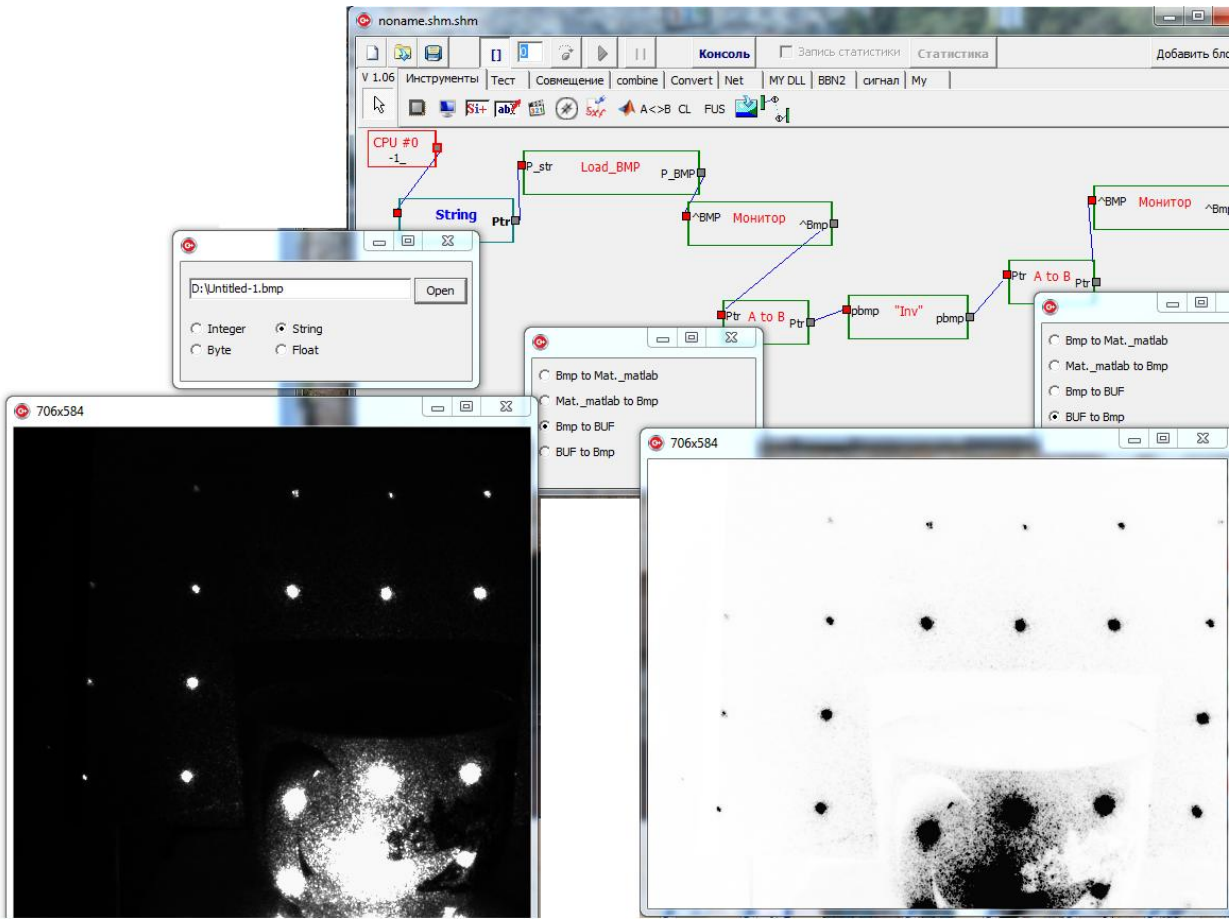


Рисунок 15 - Схема конвейера и окна конфигурации отдельных блоков в схеме.

Пояснения:

Первый блок CPU - начало конвейера. Завершив все ступени конвейера, управление сново передается блоку CPU который решает прекратить выполнение обработки или нет.

Блок «Константа» передает указатель на значение хранимое в нем. Значение, в зависимости от настроек блока, может трактоваться как int, byte, float, или строка (массив char).

Блок загрузки изображения, выполняет загрузку изображения. На вход подается указатель на строку содержащую имя файла, на выходе указатель на структуру Tbitmap (bmp с разрядностью 24 bit).

Блок монитор отображает в окне изображение полученное по указателю на входе. Блок отображает только bmp типа Tbitmap с разрядностью 24 bit.

Блок конвертирование типов, конвертирует Tbitmap в TBmpbuf, что позволяет отвязаться от borland vcl классов и типов.

Естественно после обработки требуется обратная конвертация, чтобы была возможность корректно отобразить результат блоком монитор.

Запуск схемы осуществляется нажатием на кнопку 7 (рисунок 1). Если схема собранна верно, то активируются кнопки пошагового и автоматического выполнения 3 (рисунок 1).

Все DLL используемые в конвейере будут загружены в адресное пространство стендового ПО и у каждой DLL выполнится код прописанной в точке старта DLL. Если стенд запущен как родительский процесс в результате компилирования и выполнения кода в среде разработки, то именно в этот момент станут доступны средства отладки в IDE (точки останова и т.п. если они были установлены).

Повторное нажатие на кнопку 7 (рисунок 1) вызовет процесс деактивации схемы. Все загруженные DLL будут выгружены из адресного пространства ПО стенда.

Задание
1 Создать проект с функцией инверсии изображения как DLL плагин.
2 Добавить функцию как блок в стенд и проверить ее работоспособность.
3 Создать функцию перевод изображения в серое. Формула для перевода каждого пикселя RGB в GRAY:

GRAY = 0.144*Blue +0.587*Green +0.299* Red,

проверить работоспособность.

Часть II

Исправление дисторсии от объектива

Фотограмметрическая калибровка цифровых съемочных камер выполняется с целью определения значений элементов внутреннего ориентирования съемочных камер, включая параметры фотограмметрической дисторсии объектива съемочной камеры.

Поправки d_x и d_y в координаты измеренных на снимке точек, компенсирующие влияние фотограмметрической дисторсии объектива съемочной камеры, в общем случае описываются различными уравнениями. Наиболее широко используются формулы Брауна.

Формулы Брауна

$$\left.\begin{aligned} d_x &= x(r^2k_1 + r^4k_2 + r^6k_3) + (r^2 + 2x^2)p_1 + 2xyp_2 \\ d_y &= y(r^2k_1 + r^4k_2 + r^6k_3) + (r^2 + 2y^2)p_2 + 2xyp_1 \end{aligned}\right\},$$

в которых:

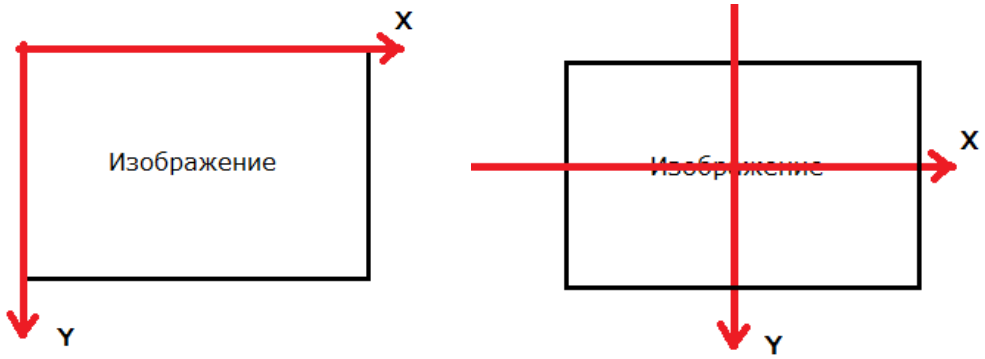
x, y — координаты точек снимка;
k_1, k_2, k_3 — коэффициенты радиальной дисторсии;
p_1, p_2 — коэффициенты тангенциальной дисторсии объектива;
$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$;
x_0, y_0 — координаты главной точки снимка.

Как показал практический опыт фотограмметрической калибровки цифровых фотокамер, для описания фотограмметрической дисторсии в подавляющем большинстве случаев достаточно ограничиться коэффициентами k1 и k2 вышеприведенной системы уравнений.

В нашей работе ограничимся только коэффициентом k1.
Будем считать, что каким то способом, нам удалось подобрать коэффициент k1.

Пусть есть изображение размером WxH пикселей. Центр изображения Cx,Cy находится по формулам:
Cx=(W- 1) / 2.0;
Cy=(H- 1) / 2.0.

Оси координат для изображения передаваемого в структуре TBmpbuf располагаются следующем образом (рисунок 16 а).



а) б)

Рисунок - 16 Системы координат для изображения а) основная, б) центрированная

Для расчета поправочных коэффициентов для дисторсии удобно перейти в центрированную систему координат (рисунок 16 б).

Координаты пикселя Pi,j изображения из системы координат (рисунок 16 а) в центрированную Px,y пересчитываются по формул формулам x=i-Cx, y=j-Cy.

Найдем r (радиус от центра изображения до текущего пикселя) из формул Брауна

```
r = sqrt(x * x + y * y);
```

и отношения для вычисления корректно направленного радиуса переноса пикселя для исправленной дисторсии.

```
if (r == 0)
{
  cosphi:= 1;
  sinphi:= 0;
}
else
{
  cosphi:= x / r;
  sinphi:= y / r;
}
```

Наконец радиус для откорректированного пикселя с учетом только значения коэффициента дисторсии первого порядка сводится к формуле (можно убедится в корректности подставив в формулу Брауна в качестве коэффициентов высших порядков нули)

```
distorsedR= r * (1 + d * r * r);
```

где d- коэффициент дисторсии первого порядка (задается пользователем).

Найдем новые координаты Px,y и добавим к нему центр изображения чтобы перейти к системе координат (рисунок 16 а)

ix=distorsedR * cosphi + cX;
iy=distorsedR * sinphi + cY;

Таким образом для изображения с исправленной дисторсией каждый пиксель
Pi,j следует заполнить цветом взятым из пикселя Pix,iy оригинального изображения.

Следует помнить, что перенос пикселей следует осуществлять в другой буфер изображения. Если читать пиксели из одного буфера и в него же писать результат - то получится «полная каша».
Однако в нашей работе не будем создавать дополнительный буфер (поступим хитрее). Так как изображения серое но при этом трехканальное, то все три канала содержат одинаковые значения. Поэтому следует читать цвет Pix,iy из любого цветового канала. а писать результат правки Pi,j в другой. В конце процедуры следует взять цвет всех пикселей из канала результата и перекопировать его в оставшиеся два канала.

Рекомендуемые типы для переменных в программном коде:

int i,j,ix,iy;
float cY,cX;
float r, distorsedR;
float cosphi, sinphi;
float x,y;

Следует заметить, что значения ix,iy на сомом деле должны быть не целыми числами, но в силу особенности изображения не хранит цвет между соседними пикселями.
Получить цвет между соседними пикселями можно интерполяцией (такой цвет назовем субпиксельным) а процедуру получения - субпиксельность.

Для получения субпиксельного цвета необходимо прочитать соседние пиксели и взять их с весами.

Несколько формул демонстрируют процедуру субпиксельности.

int id_x, id_y, id_x1, id_y1;

id_x= distorsedR * cosphi + cX;
id_y= distorsedR * sinphi + cY;
id_x1= distorsedR * cosphi + cX+1;
id_y1= distorsedR * sinphi + cY +1;
Обращаем внимание, что значения не целые но приводятся к целым путем отброса дробной части, так как переменные id_x и др. имеют тип int.

далее формулы для получения цвета (правильно их запрограммируйте), если изображение цветное должны применяться для каждого цветового канала по отдельности.
lefttop_color=P id_x , id_y
leftbot_color = P id_x , id_y1

righttop= P id_x1 , id_y
rightbot:= P id_x1 , id_y1

и веса
fractX=frac(distorsedR * cosphi + cX +1);
fractY= frac(distorsedR * sinphi + cY +1);
операция frac - получение дробной части.
Пример: frac(12,324)=0,324, frac(12,8)=0,8.

субкиксельный цвет sub_color=(lefttop * (1 - fractX) * (1 - fractY) +
leftbot * (1 - fractX) * fractY +righttop * fractX * (1 - fractY) +
rightbot * fractX * fractY);

Задание.

Исследовать влияния значения коэффициента дисторсии первого порядка на результат исправления дисторсии.

Для этого необходимо запустить программу pDistortion (рисунок 17). Загрузить изображения (понимает только формат BMP 24 бит). Изменяя значение коэффициента 2 получить результат преобразования нажав на кнопку 3.

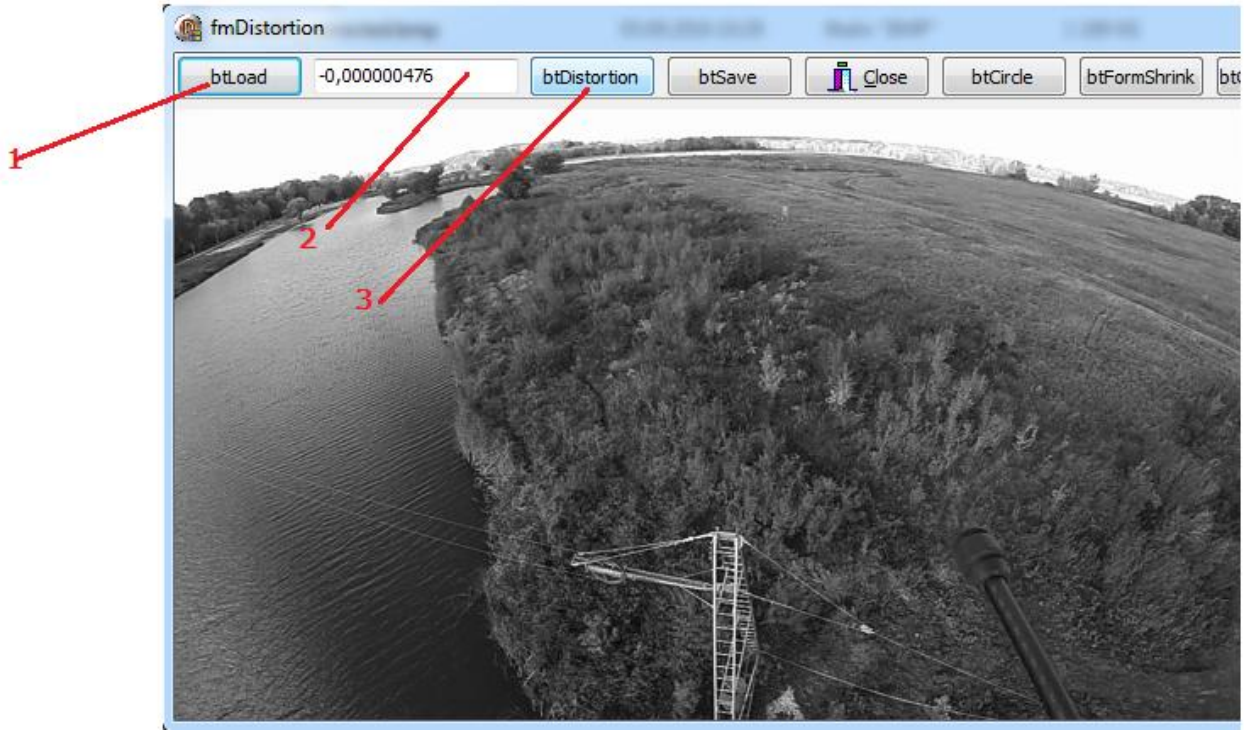


Рисунок 17 - Интерфейс для исследования влияния коэффициента дисторсии

Подберите коэффициент «на глаз» для изображений с сильными искажениями (результат должен быть правдоподобным).

2. Запрограммировать процедуру правки дисторсии дисторсии и получить результат применяя коэффициент первого порядка

-4,43770E-7 для изображения frame000.bmp,
-4,88016E-7 и изображения frame001.bmp.

В отчете должны присутствовать:

- изображение результата исправления дисторсии для любого изображения (до, после и коэффициент дисторсии;
- схема конвейера;
- текст функции исправления дисторсии.

Дополнительный бал за работу начисляется за самую быструю реализацию функции.

Пример схемы для исправления дисторсии на кадрах видеоряда (рисунок 18).

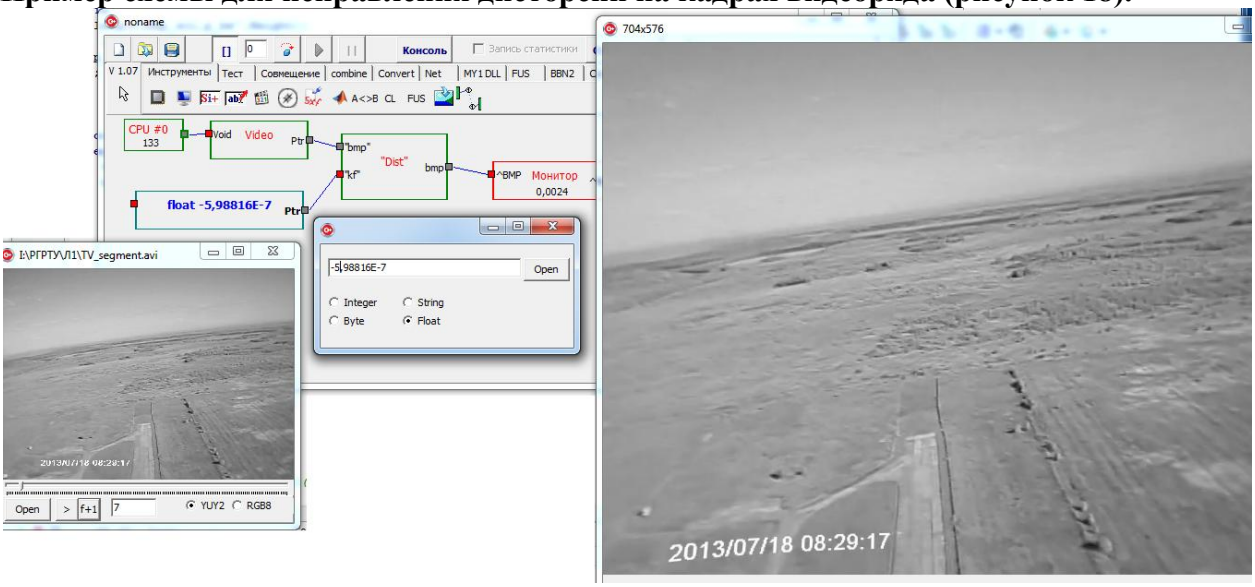


Рисунок 18 - Схема конвейера с модулем исправления дисторсии