

# СЕМИНАР 5

---

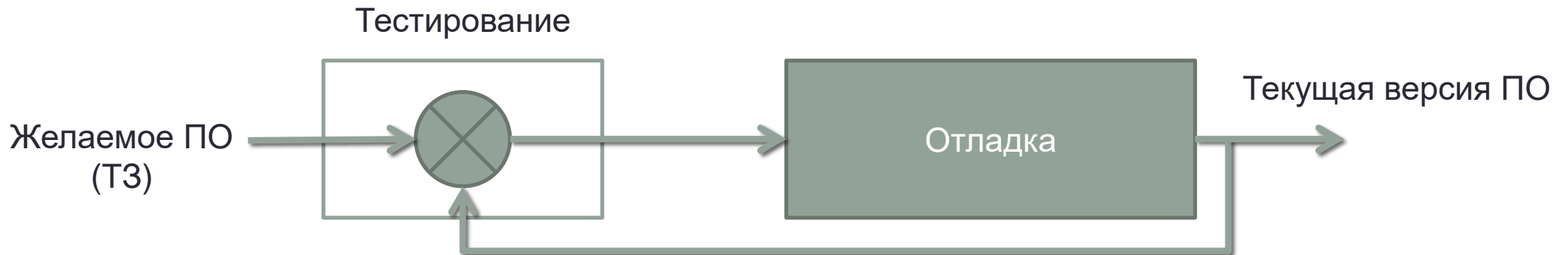
Немного про тестирование и отладку ПО подводных систем  
Подключение kx\_rult к проекту

*«Пишите код так, как будто сопровождать его будет склонный к насилию психопат, который знает, где вы живете»*

# Тестирование и отладка ПО

**Тестирование и отладка ПО** - процесс, позволяющий получить программное обеспечение, функционирующее с требуемыми характеристиками в заданной области входных данных.

- **тестирование** — деятельность, направленная на обнаружение ошибок;
- **отладка** – деятельность, направленная на установление точной природы известной ошибки, а затем - на исправление этой ошибки.



# Тестирование ПО

Уровни Тестирования:

1. Модульное тестирование (Unit Testing)
2. Интеграционное тестирование (Integration Testing)
3. Системное тестирование (System Testing)
4. Операционное тестирование (Release Testing).
5. Приемочное тестирование (Acceptance Testing)

# Инструменты тестирования ПО

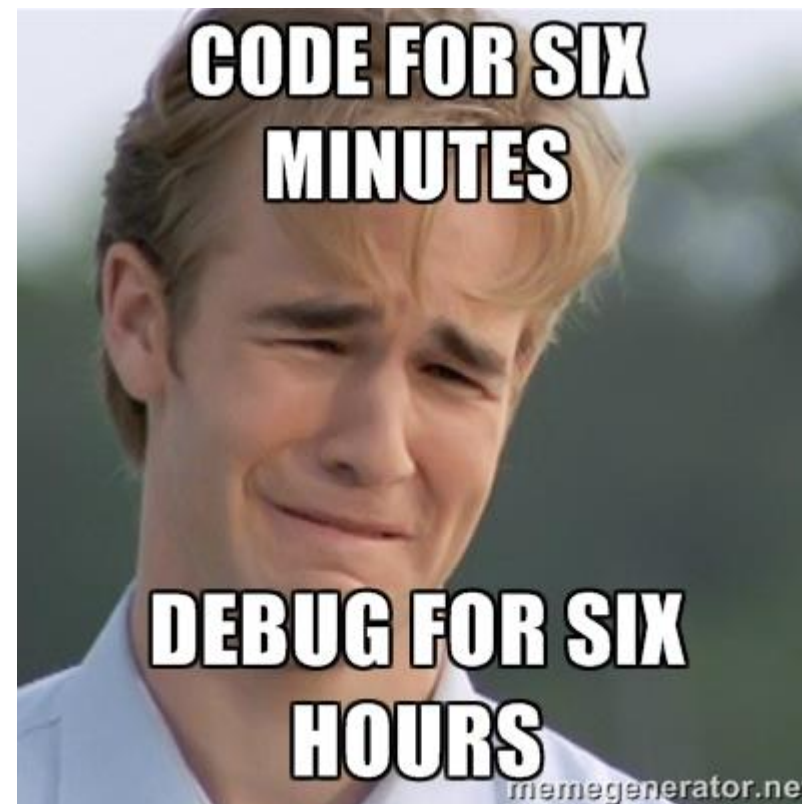
1. Автоматизированные средства (Google Tests и т.п.)
2. Средства Qt. QtTests
3. kx\_pult
4. и т.п.

# Отладка ПО

Отладка – это комплексный процесс по выявлению и исправлению дефектов в программном обеспечении

*Отлаживать код вдвое сложнее, чем писать. Поэтому, если при написании программы вы используете весь свой интеллект, вы по определению недостаточно умны, чтобы её отладить.*

Брайан Керниган



# Основные этапы отладки ПО

- воспроизведение дефекта (любым из доступных способов);
- анализ дефекта (поиск причины возникновения дефекта – root-cause);
- дизайн исправления дефекта (и возможно ревью, если есть альтернативы);
- кодирование исправления дефекта (и какие-либо активности связанные с кодированием);
- валидация исправления;
- интеграция исправления в кодовую базу или целевую систему;
- дополнительные валидации после интеграции (при необходимости).

# Методики отладки

1. Запуск программы из под отладчика (Debug-mode)
2. Логирования кода – вывод в файл (или консоль и т.п.) (QDebug(), kx\_pult)
3. Анализ кода без исполнения программы
4. Анализ поведения системы или её части
5. Unit тестирование
6. Прототипирование
7. Отладка с помощью memory-dump-ов
8. Профилирование кода (если необходима оптимизация производительности)
9. и.т.п.

# Отработка ПО подводной системы

Отработка ПО – деятельность, направленная на то, чтобы качество ПО подводной системы соответствовало ТЗ.

Включает в себя:

- Тестирование,
- Настройку,
- Отладку ПО.





# Отработка ПО подводной системы

Этапы отработки ПО:

1. Отработка алгоритмов на моделях и имитаторах подсистем НПА;
2. Полунатурная отработка ПО;
3. Натурная отработка подводной системы.



# Отработка ПО подводной системы в натуральных условиях





# Полунатурная отработка ПО



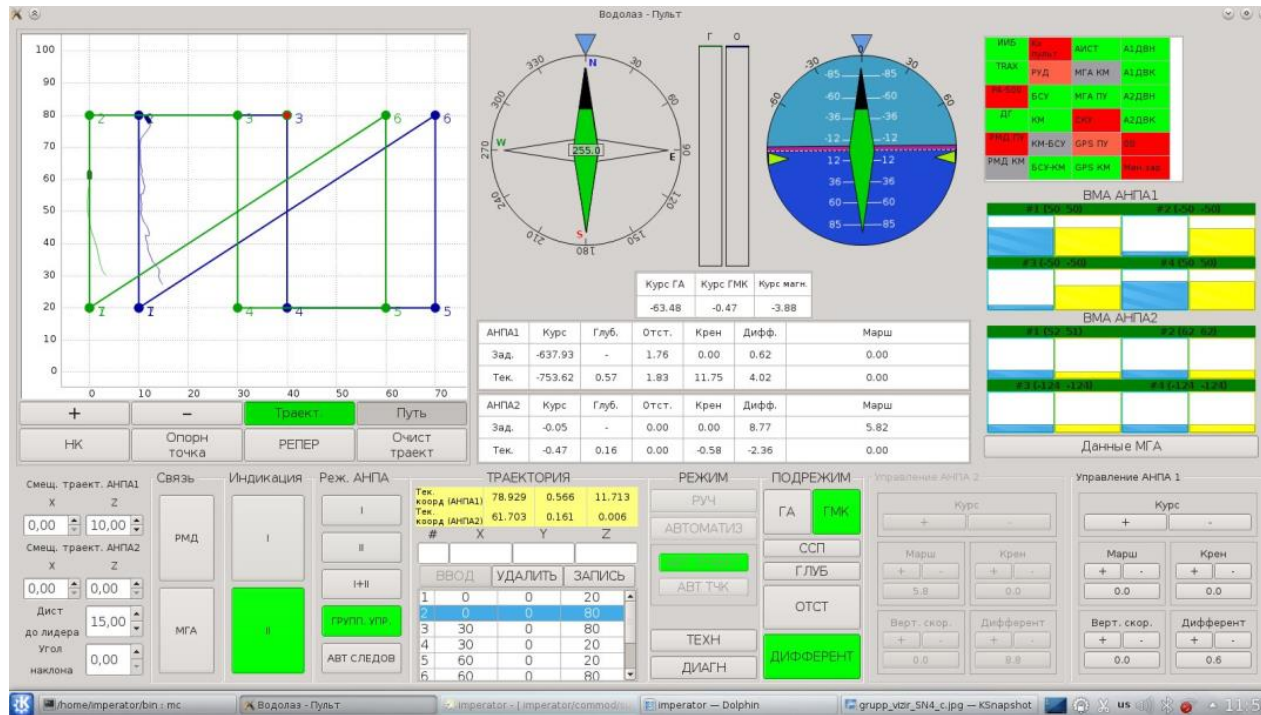


# Полунатурная отработка ПО



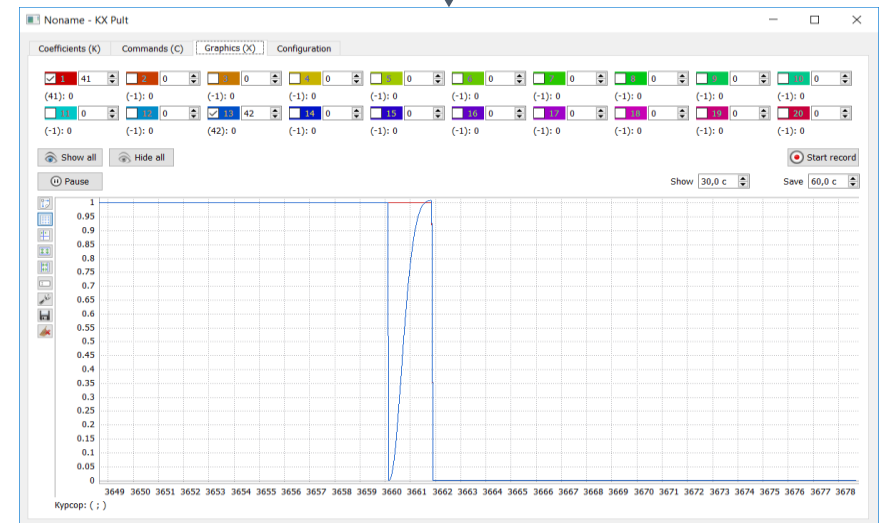
# Отработка алгоритмов на моделях и имитаторах

Пульт управления:



ПО ТНПА  
(СУ и математическая  
модель ТНПА)

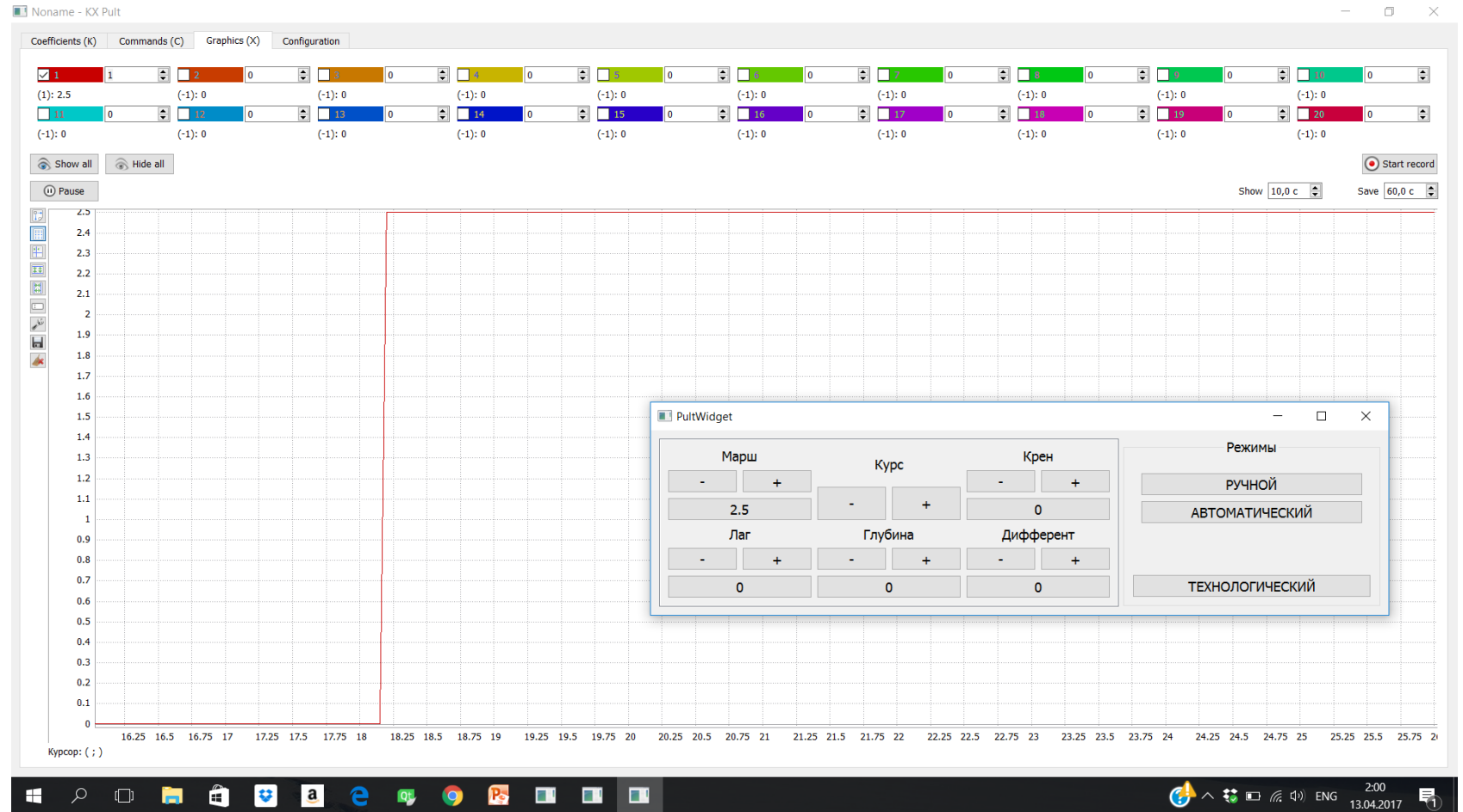
kx\_pult:



# Приложение.kx\_pult.

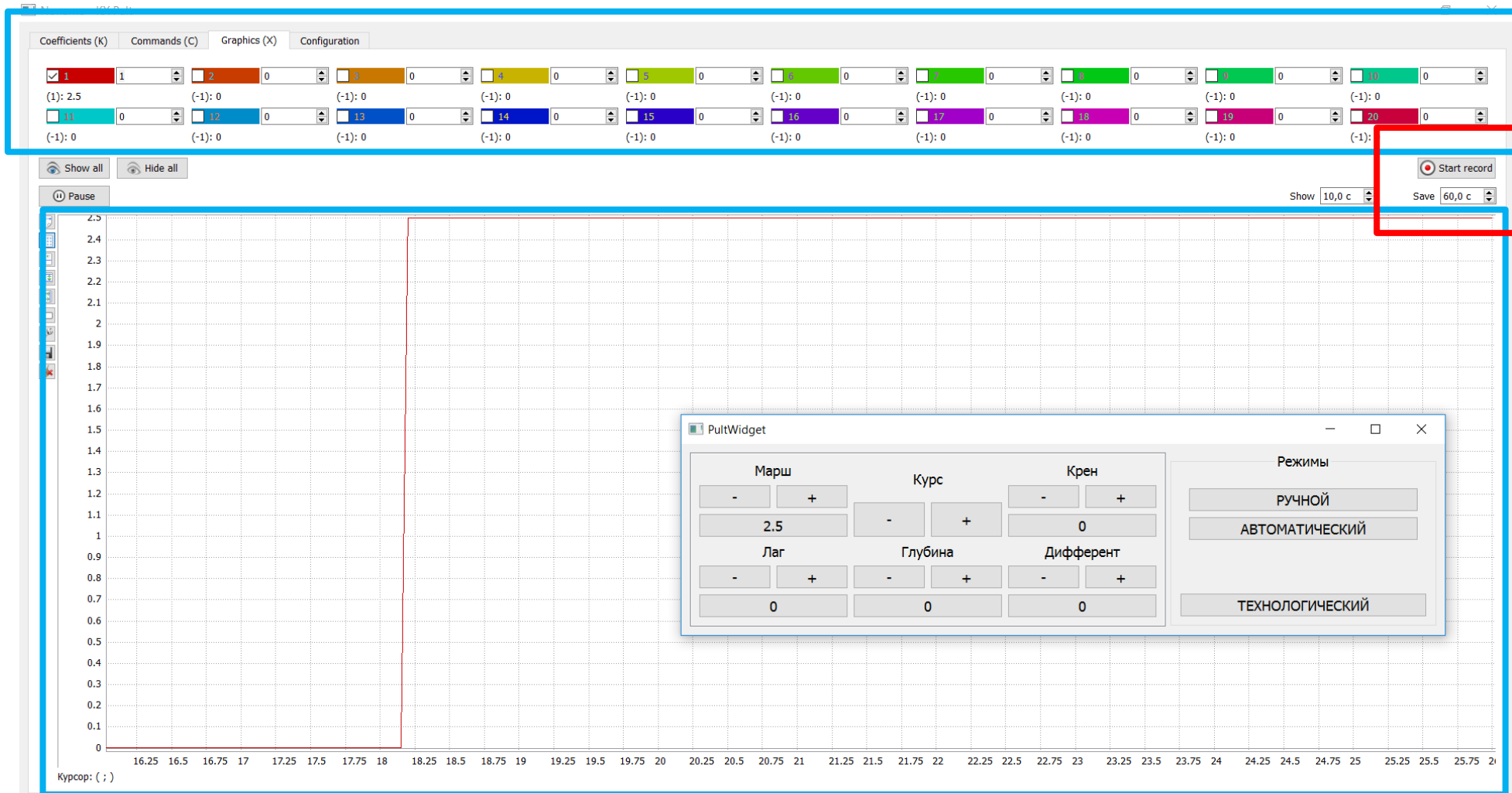
Kx\_pult:

- Выводит текущие переменные ПО;
- Предоставляет интерфейс для чтения/записи настроечных коэффициентов ПО ПРТС.





Поле выбора номеров переменных, которые будут приниматься от стороннего ПО для вывода на график или записи (видимость определяется флагом чекбокса)



Управление записью

Поле вывода графиков

# kx\_pult. Коэффициенты

Noname - KX Pult

Coefficients (K) Commands (C) Graphics (X) Configuration


Send Receive

Read file K Write file K

Set K desc file ... Reparse K desc

Resize 1000

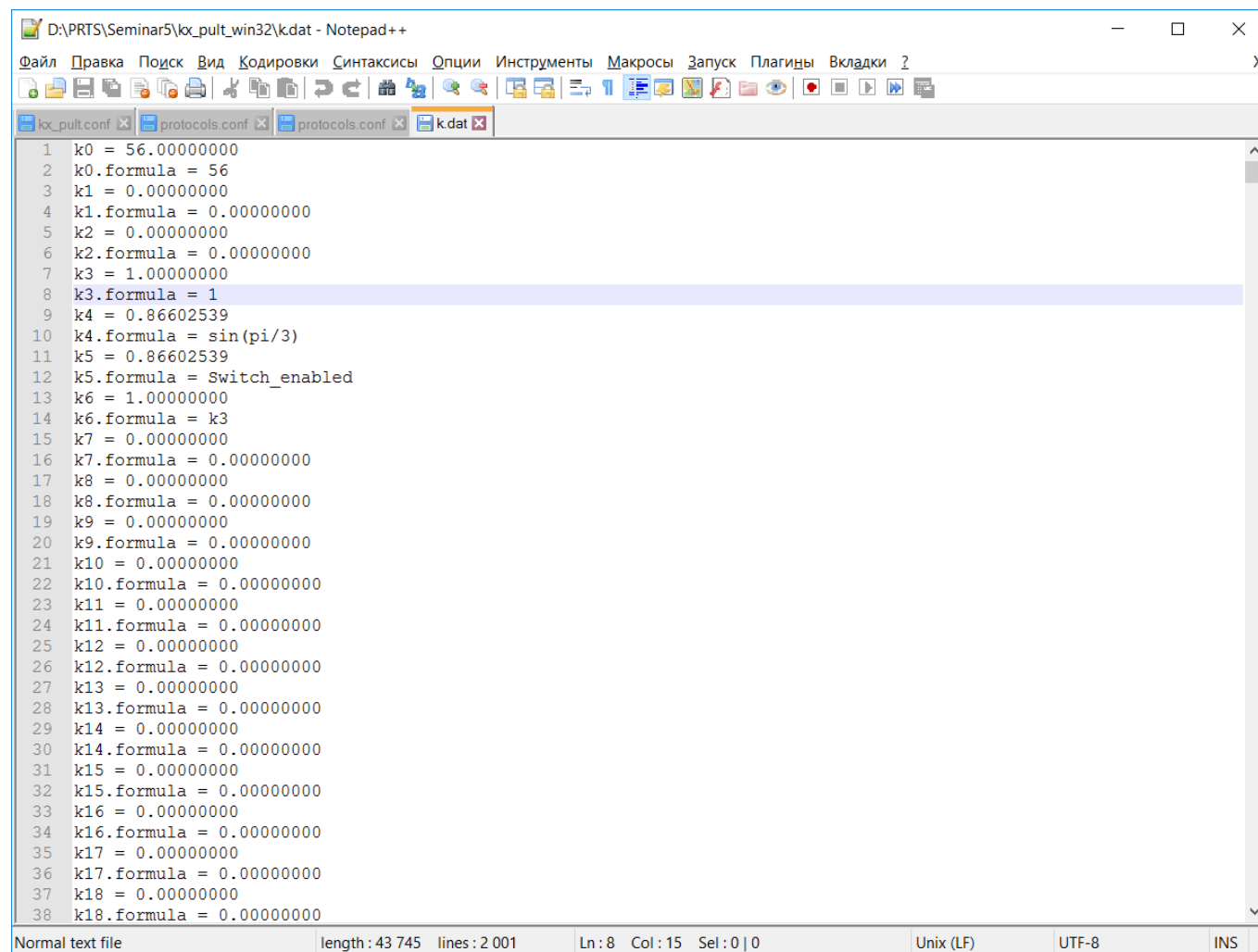
13/04/2017 02:12 - Update settings from "kx\_pult.conf"  
13/04/2017 02:12 - Read K file "k.dat": 1000 coeffs, 43745 bytes  
13/04/2017 02:12 - Update descriptions from "k\_description.h"  
13/04/2017 02:13 - Update descriptions from "k\_description.h"  
13/04/2017 02:20 - Write K file "k.dat": 1000 coeffs, 43745 bytes  
13/04/2017 02:20 - K sendend

☐ Hide empty ☐ Hide expressions ☐ Hide without errors 🔍 Search:  ☒ Auto calculate  Calculate 100%

Index	Name	Expression	Calculated	Type	Comment
0		56	56		
1		0.00000000	0		
2		0.00000000	0		
3	Gratings_history	1	1	int	Gratings peak values history, count
4	Switch_enabled	sin(pi/3)	0.866025	bool	0 or 1 Use optical switch or not
5	Switch_wait	Switch_enabled	0.866025	int	Delay after switching channel, ms
6	Fourier_size	k3	1	int	Size of fourier window, in counts
7	Fourier_enabled	0.00000000	0	bool	0 or 1, Global fourier enable flag
8	Player_mode	0.00000000	0	bool	0 or 1, Receive or not player packets from Gui
9	SynchronizationEnabled	0.00000000	0	bool	0 or 1, When enabled, start logs and reset time when sync signal received
10	LambdasAutoReset	0.00000000	0	bool	0 or 1, Set lambdas_0 to grating values on first data receive
11	Switch_autoscan	0.00000000	0	bool	0 or 1, 0 - scan with our forces, 1 - SDK autoscan
12	Switch_autoscan_offset	0.00000000	0	int	Offset for SDK autoscan
13		0.00000000	0		
14		0.00000000	0		
15		0.00000000	0		
16		0.00000000	0		
17		0.00000000	0		
18		0.00000000	0		
19		0.00000000	0		
20	Detector_Threshold_Min	0.00000000	0	double	
21	Detector_Threshold_Max	0.00000000	0	double	
22	Detector_SideSize	0.00000000	0	double	
23	Detector_SideOffset	0.00000000	0	double	
24	Gauss_size	0.00000000	0	int	
25	Peak_max_offset	0.00000000	0	double	in pixels
26	Peak_LF_coeff	0.00000000	0	double	1. - no filtering
27	Fourier_YScale	0.00000000	0	double	scale for fourier amplitude
28	Temperature_compensation	0.00000000	0	bool	0 or 1, Use temperature sensor or default temperature without compensation
29	Temperature_default	0.00000000	0	double	Default temperature using if compensation disabled
30	Fourier_Max_or_Density	0.00000000	0	int	0 - Maximum in fourier window, 1 - Density of fourier window
31	Startup_Logs_Data	0.00000000	0	bool	write data logs at program start

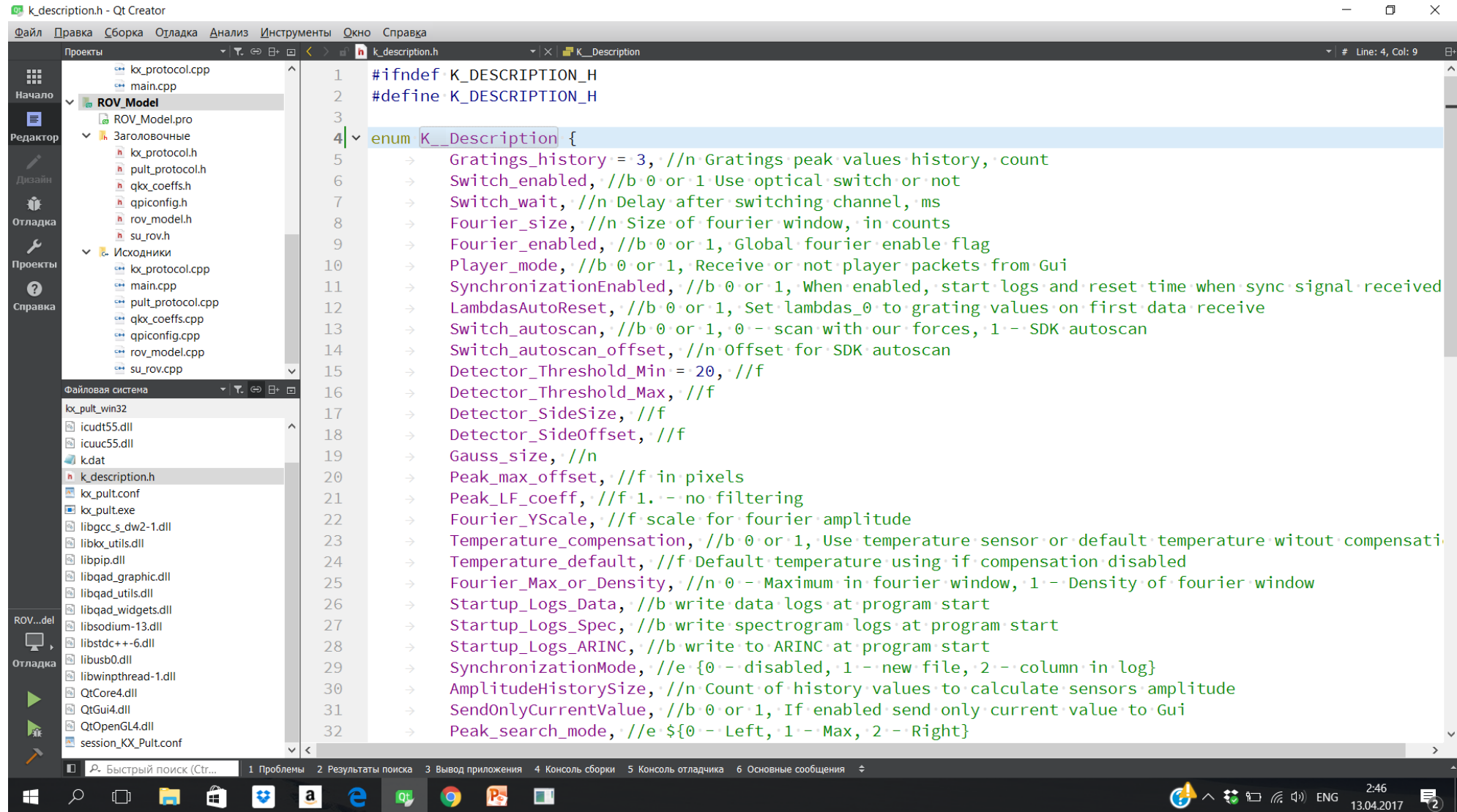


# kx\_pult. Коэффициенты



```
D:\PRTS\Seminar5\kx_pult_win32\k.dat - Notepad++
Файл Правка Поиск Вид Кодировки Синтаксисы Опции Инструменты Макросы Запуск Плагины Вкладки ?
kx_pult.conf protocols.conf protocols.conf k.dat
1 k0 = 56.00000000
2 k0.formula = 56
3 k1 = 0.00000000
4 k1.formula = 0.00000000
5 k2 = 0.00000000
6 k2.formula = 0.00000000
7 k3 = 1.00000000
8 k3.formula = 1
9 k4 = 0.86602539
10 k4.formula = sin(pi/3)
11 k5 = 0.86602539
12 k5.formula = Switch_enabled
13 k6 = 1.00000000
14 k6.formula = k3
15 k7 = 0.00000000
16 k7.formula = 0.00000000
17 k8 = 0.00000000
18 k8.formula = 0.00000000
19 k9 = 0.00000000
20 k9.formula = 0.00000000
21 k10 = 0.00000000
22 k10.formula = 0.00000000
23 k11 = 0.00000000
24 k11.formula = 0.00000000
25 k12 = 0.00000000
26 k12.formula = 0.00000000
27 k13 = 0.00000000
28 k13.formula = 0.00000000
29 k14 = 0.00000000
30 k14.formula = 0.00000000
31 k15 = 0.00000000
32 k15.formula = 0.00000000
33 k16 = 0.00000000
34 k16.formula = 0.00000000
35 k17 = 0.00000000
36 k17.formula = 0.00000000
37 k18 = 0.00000000
38 k18.formula = 0.00000000
Normal text file length : 43 745 lines : 2 001 Ln : 8 Col : 15 Sel : 0 | 0 Unix (LF) UTF-8 INS
```

# kx\_pult. Коэффициенты. k\_description.h



# kx\_pult. Подключение к проекту

Подключить к проекту исходники

Создать в проекте массивы под x и k

Создать объект класса передачи x-ов

Создать объект класса приема/передачи коэффициентов

Настроить обмен с kx\_pult

# kx\_pult. Подключение к проекту

- 1.Подключить к проекту исходники, в которых написана реализация механизма обмена данными с kx\_pult (передача x-ов и коэффициентов):
  1. kx\_protocol.h, kx\_protocol.cpp – классы передачи x-ов
  2. configdata.h, configdata.cpp - классы для чтения данных из config-файлов, в которых будут прописаны ip и порты для ethernet-соединения (нужно для класса x-ов)
  3. qkx\_coeffs.h, qkx\_coeffs.cpp – классы передачи k-тов
  4. qpriconfig.h, qpriconfig.cpp – классы для чтения данных из config-файлов, в которых будут прописаны ip и порты для ethernet-соединения
- \*кроме того, для корректной работы модулей в .pro файле проекта необходимо добавить модуль network

# kx\_pult. Подключение к проекту

## 2. Создать в проекте массивы под переменные и коэффициенты

main.cpp

```
1 #include <QCoreApplication>
2 #include "su_rov.h"
3
4 double X[2000][2];
5
6 int main(int argc, char *argv[]) {
7     QCoreApplication a(argc, argv);
8     SU_ROV su;
9     return a.exec();
10 }
11
```

Ваш\_Класс\_СУ.h

```
1 #ifndef SU_ROV_H
2 #define SU_ROV_H
3
4 #include <QObject>
5
6
7 #include "kx_protocol.h"
8 #include "qkx_coeffs.h"
9 #include "pult_protocol.h"
10 #include "rov_model.h"
11
12 extern double X[2000][2];
13 extern QVector<double> K;
14
```

qkx\_coeffs.h

```
1 #include "qkx_coeffs.h"
2 #include <QDataStream>
3 #include <QThread>
4
5 QVector<double> K;
```

# НЛО №0. Спецификатор extern

- Спецификатор extern сообщает компилятору, что следующие за ним типы и имена переменных объявляются где-то в другом месте.
- Если при объявлении выделяется память под переменную, то процесс называется определением. Использование extern приводит к объявлению, но не к определению. Оно просто говорит компилятору, что определение происходит где-то в другом месте программы.

# kx\_rult. Подключение к проекту

## 3. В проекте создать

3.1. объект класса Qkx\_coeffs (класса приема/передачи коэффициентов)

- K\_Protocol = new Qkx\_coeffs(*“название\_конфиг\_файла”*,  
*“индекс\_к\_в\_конфиге”*);

3.2. объект класса x\_protocol (класса передачи x-ов):

X\_Protocol = new x\_protocol (*“название\_конфиг\_файла”*,  
*индекс\_x\_в\_конфиге*,X);

В проекте: *«название\_конфиг\_файла»*=*«protocols.conf»*

*«индекс\_к\_в\_конфиге»*=*«ki»*

X – указатель на массив x-ов;

# kx\_pult. Настройка обмена.

kx\_pult.conf

```
9
10 [x]
11 receiver.ip = 127.0.0.1 #i
12 receiver.port = 40012 #n
13 receiver.frequency = 20 #f
14 sender.ip = 127.0.0.1 #i
15 sender.port = 40013 #n
16 sender.frequency = 20 #f
17 type = 0xAA #t
18 addr_x = 0x0A #a
19 addr_pult = 0x0B #p
20 count = 2000 #n
21
22 [k]
23 receiver.ip = 127.0.0.1 #i
24 receiver.port = 4015 #n
25 sender.ip = 127.0.0.1 #i
26 sender.port = 4014 #n
27 type = 0xBB #t
28 addr_k = 0x1A #a
29 addr_pult = 0x1B #p
30 count = 1000 #n
31 file = k.dat #f
```

Для настройки kx\_pult'a:

x.receiver.\* - параметры kx\_pult'a для передачи x-ов  
x.receiver.ip - ip-адрес по которому доступен kx\_pult  
x.receiver.port – порт, на котором «слушает» kx\_pult  
x.receiver.frequency – частота обмена kx\_pult'a с вашим ПО

x.sender.\* - параметры приложения, которое «общается» с kx\_pult'ом  
x.sender.ip – ip-адрес приложения  
x.sender.port – порт, на котором приложение слушает  
x.count - количество x-ов

Аналогичные параметры необходимо задать для передачи коэффициентов:

k.receiver.\*

Кроме того, для коэффициентов необходимо задать название файла  
k.file = k.dat



# Настройки x\_protocol. Пример.

Kx_pult.conf (КХ-пульт)	Protocols.conf (в вашем ПО)
[x]	
receiver.ip = 127.0.0.1 #i	xi.receiver.ip = 127.0.0.1 #i
receiver.port = 40012 #n	xi.receiver.port = 40013 #n
receiver.frequency = 20 #f	xi.receiver.frequency = 20 #f
sender.ip = 127.0.0.1 #i	xi.sender.ip = 127.0.0.1 #i
sender.port = 40013 #n	xi.sender.port = 40012 #n
sender.frequency = 20 #f	xi.sender.frequency = 20 #f
type = 0xAA #t	xi.type = 0xAA #t
addr_x = 0x0A #a	xi.addr_x = 0x0A #a
addr_pult = 0x0B #p	xi.addr_pult = 0x0B #p
count = 2000 #n	xi.count = 2000 #n

# Практическая часть 1.

- Встроить kx\_pult в ваш проект:
  1. Создать проект
  2. Добавить в проект необходимые файлы и подключить в .pro необходимые модули
  3. Создать в проекте класс под СУ
  4. Создать переменные под x-ы и ккоэффициенты
  5. Создать объекты для обмена с kx\_pult
  6. Настроить обмен с kx\_pult
  7. Запустить приложения и удостовериться, что они работают!

# Практическая часть.

- Шаг 2.
- QT += core gui network
- SOURCES += main.cpp \
- widget.cpp\
- configdata.cpp\
- kx\_protocol.cpp\
- qkx\_coeffs.cpp\
- qpiconfig.cpp \
- su\_rov.cpp
- HEADERS += widget.h\
- configdata.h\
- kx\_protocol.h\
- qkx\_coeffs.h\
- qpiconfig.h \
- su\_rov.h

Создать класс под математическую модель и СУ:

Класс C++

Определить класс

Имя класса:

Базовый класс:

☒ Подключить QObject

☐ Подключить QWidget

☐ Подключить QMainWindow

☐ Подключить QDeclarativeItem - Qt Quick 1

☐ Подключить QQuickItem - Qt Quick 2

☐ Подключить QSharedData

Заголовочный файл:

Файл исходных текстов:

Путь:

# Практическая часть.

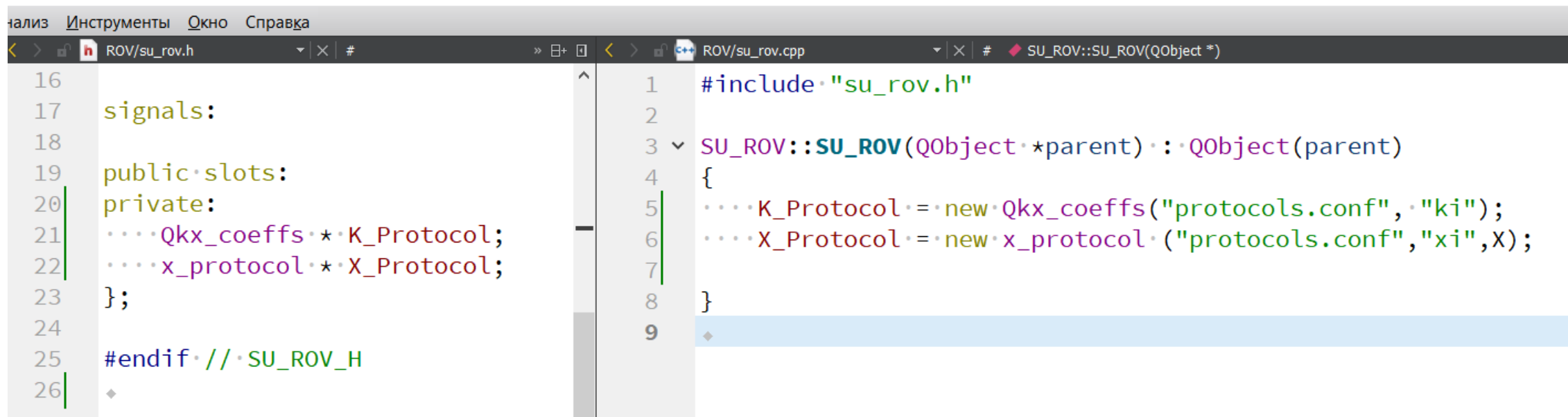
- Создаем переменные:

```
main.cpp
1  #include "widget.h"
2  #include <QApplication>
3  #include "su_rov.h"
4
5  double X[2000][2];
6
7  int main(int argc, char *argv[])
8  {
9      QApplication a(argc, argv);
10     Widget w;
11     w.show();
12
13     return a.exec();
14 }
15
```

```
ROV/su_rov.h
1  #ifndef SU_ROV_H
2  #define SU_ROV_H
3
4  #include <QObject>
5  #include "kx_protocol.h"
6  #include "qkx_coeffs.h"
7
8  extern double X[2000][2];
9  extern QVector<double> K;
10
```

# Практическая часть.

- Создаем объекты для обмена:



The screenshot shows the Qt Creator IDE with two files open: `ROV/su_rov.h` and `ROV/su_rov.cpp`. The `su_rov.h` file (left) contains a header for the `SU_ROV` class, including a `signals:` section, a `public slots:` section, and a `private:` section with two pointers: `Qkx_coeffs *K_Protocol;` and `x_protocol *X_Protocol;`. The `su_rov.cpp` file (right) shows the implementation of the `SU_ROV` constructor, which initializes the `K_Protocol` and `X_Protocol` pointers with new `Qkx_coeffs` and `x_protocol` objects, respectively, using configuration files `protocols.conf`.

```
16 signals:
17
18
19 public slots:
20 private:
21     ....Qkx_coeffs *K_Protocol;
22     ....x_protocol *X_Protocol;
23 };
24
25 #endif // SU_ROV_H
26
1 #include "su_rov.h"
2
3 SU_ROV::SU_ROV(QObject *parent) : QObject(parent)
4 {
5     ....K_Protocol = new Qkx_coeffs("protocols.conf", "ki");
6     ....X_Protocol = new x_protocol("protocols.conf", "xi", X);
7
8 }
9
```

# Практическая часть.

- Добавляем конфигурационный файл `protocols.conf` в директорию, где запускается ваше ПО

# Практическая часть.

Настраиваем обмен с kx\_pult:

```
lx_pult.conf* protocols.conf

1 [x]
2 receiver.ip = 127.0.0.1 #i
3 receiver.port = 13040 #n
4 receiver.frequency = 20 #f
5 sender.ip = 127.0.0.1 #i
6 sender.port = 13041 #n
7 sender.frequency = 20 #f
8 type = 0xAA #t
9 addr_x = 0x0A #a
10 addr_pult = 0x0B #p
11 count = 2000 #n
12
13 [k]
14 receiver.ip = 127.0.0.1 #i
15 receiver.port = 13043 #n
16 sender.ip = 127.0.0.1 #i
17 sender.port = 13042 #n
18 type = 0xBB #t
19 addr_k = 0x1A #a
20 addr_pult = 0x1B #p
21 count = 1000 #n
22 file = k.dat #f

1 xi.receiver.ip = 127.0.0.1 #i
2 xi.receiver.port = 13041 #n
3 xi.receiver.frequency = 20 #f
4 xi.sender.ip = 127.0.0.1 #i
5 xi.sender.port = 13040 #n
6 xi.sender.frequency = 20 #f
7 xi.type = 0xAA #t
8 xi.addr_x = 0x0A #a
9 xi.addr_pult = 0x0B #p
10 xi.count = 2000 #n
11
12 ki.receiver.ip = 127.0.0.1 #i
13 ki.receiver.port = 13042 #n
14 ki.sender.ip = 127.0.0.1 #i
15 ki.sender.port = 13043 #n
16 ki.type = 0xBB #t
17 ki.addr_k = 0x1A #a
18 ki.addr_pult = 0x1B #p
19 ki.count = 1000 #n
20 ki.file = k.dat #c
21
```

# Практическая часть.

Запускаем kx\_pult и ваше приложение и наблюдаем за диагностикой соединения и выводом переменных:

