

СЕМИНАР 7

Qt + Matlab

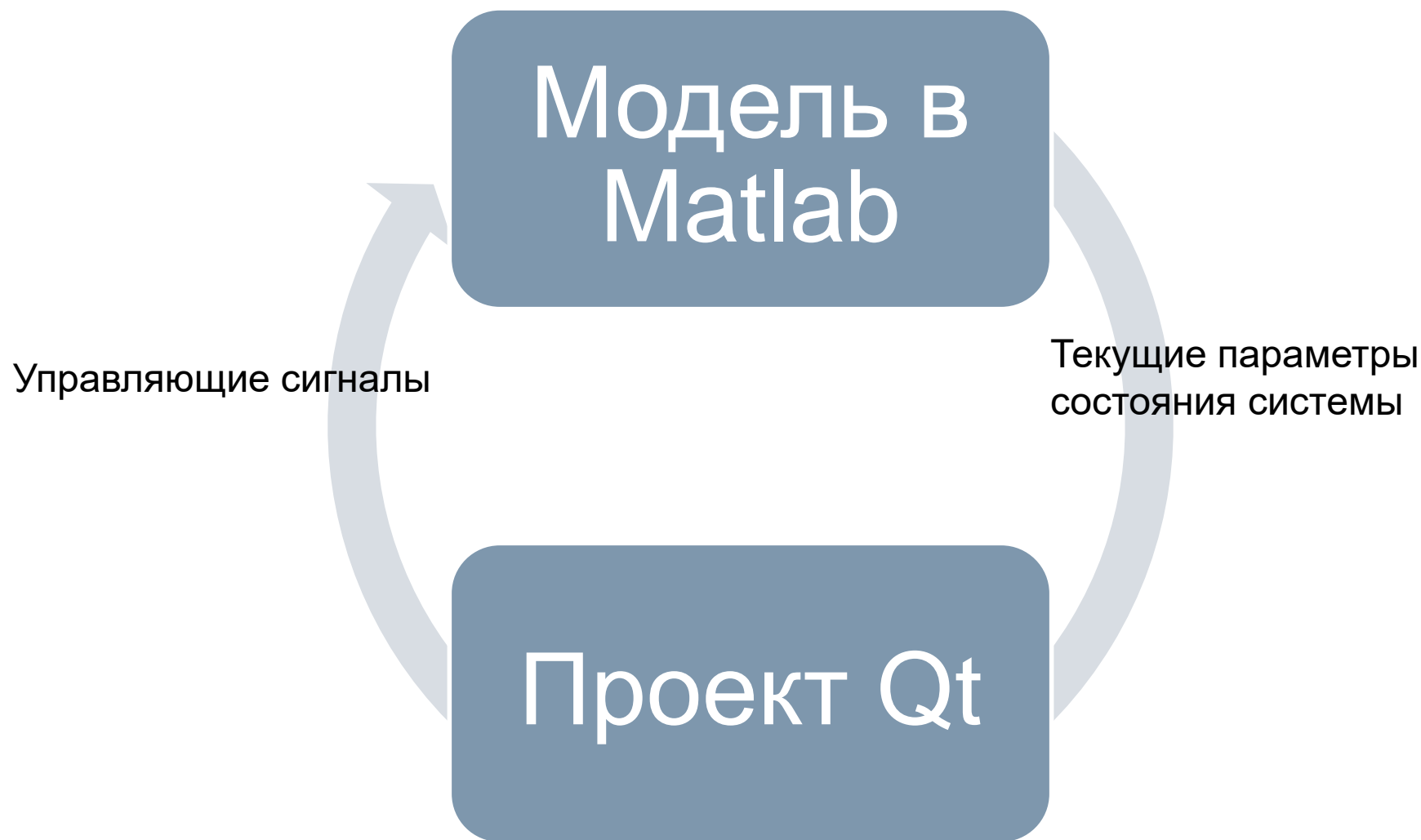
Мотивация

- При разработке необходима отладка алгоритмов на модели;
- Модель для исследования часто разрабатывается в Matlab;
- Подключение kx-pult – настоящий челендж.

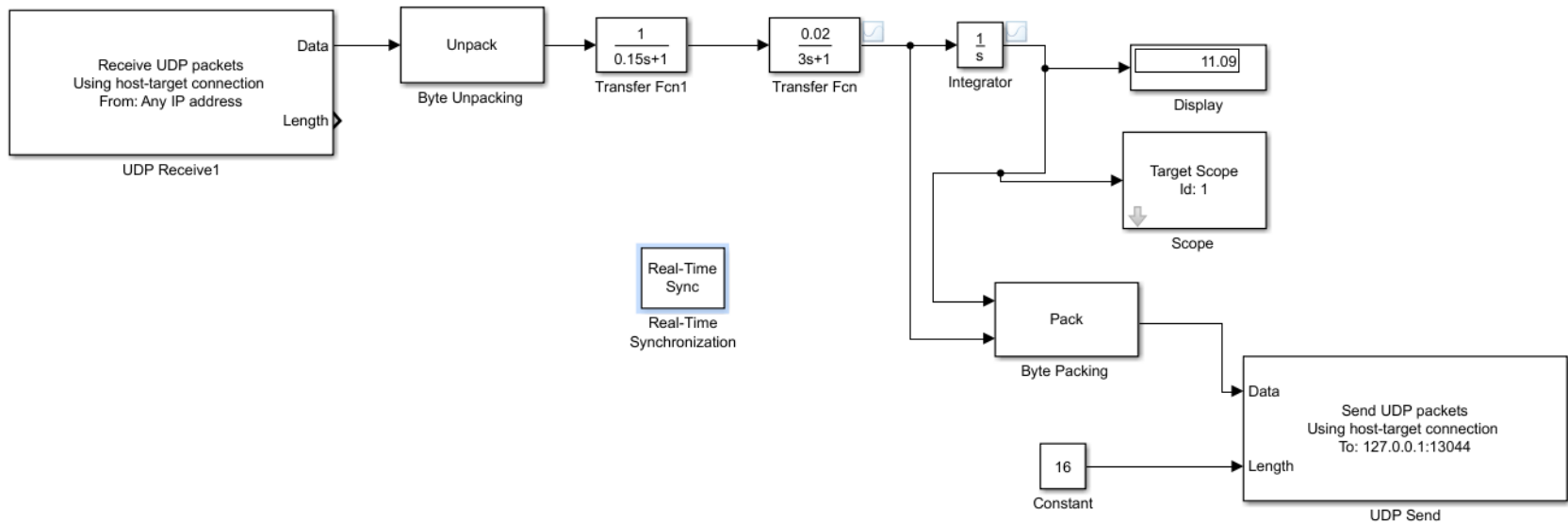
Что будем делать?

- Разрабатывать модель в matlab и подключать к модели обмен по Udp
- Разрабатывать проект в Qt с обменом по Udp

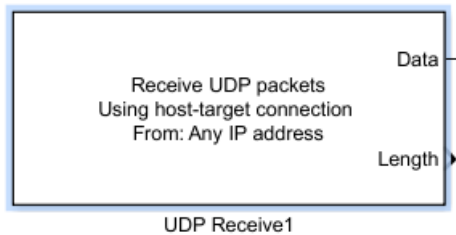
Схема работы модели и проекта Qt



Настройка приёма данных в Matlab



Udp Receive



Local port – порт, на котором будет «слушать» сокет Матлаба

Receive width – размер принимаемого сообщения в байтах

The dialog box is titled "Block Parameters: UDP Receive1". It contains the following sections and fields:

- UDP Receive**
Receive data over UDP network from a remote device.
'Local IP address' applies only when the block executes on a target computer.
- Parameters**
 - Local IP address:
Use host-target connection (dropdown menu)
 - Local port: 13042 (text field)
 - Receive width: 8 (text field)
 - ☒ Receive from any source
 - Sample time (-1 for inherited): -1 (text field)
- Buttons: OK, Cancel, Help, Apply

Udp Unpack



Output port data types:
Перечислить типы
выходных данных

Output port dimensions:
Перечислить
размерность данных

Block Parameters: Byte Unpacking

xpcbytepacking (mask) (link)

Unpack bytes from a single input vector into multiple output vectors.

Parameters

Output port (unpacked) data types (cell array):

{'double'}

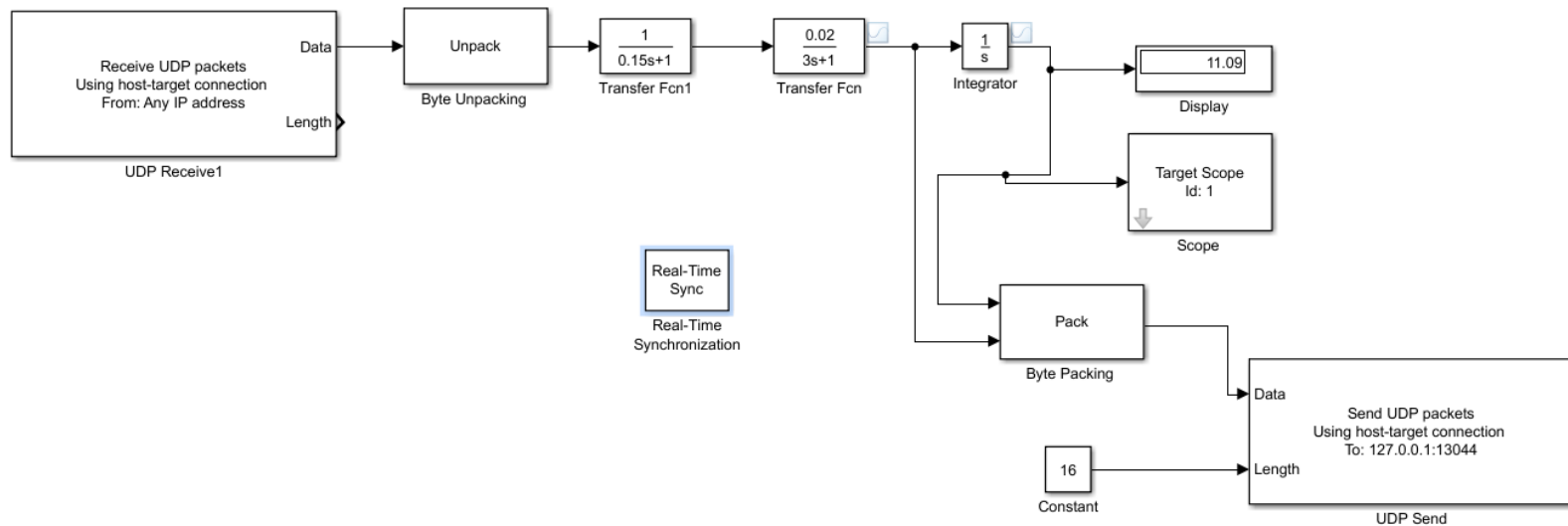
Output port (unpacked) dimensions (cell array):

{[1]}

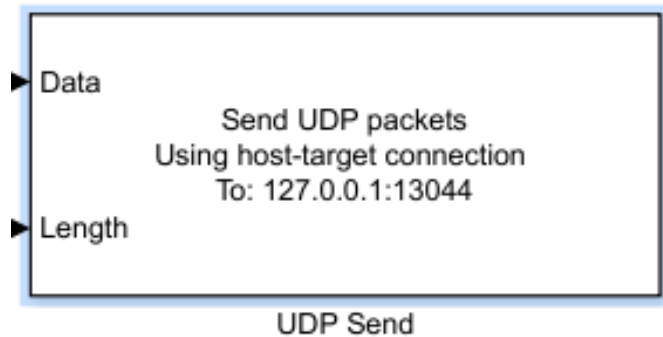
Byte Alignment: 1

OK Cancel Help Apply

Отправка данных из Matlab



Udp Send



Local port – порт, с которого будут отправляться данные

To IP address, To port: ip адрес и порт КУДА будут отправляться данные из модели matlab

Sample time – частота, с которой будут отправляться данные

The screenshot shows the "Block Parameters: UDP Send" dialog box. It has a title bar with a close button. The main area contains the following text: "UDP Send", "Send data over UDP network to a remote device. Send to 255.255.255.255 for broadcast.", and "'Local IP address' applies only when the block executes on a target computer." Below this is a "Parameters" section with the following fields: "Local IP address:" with a dropdown menu showing "Use host-target connection", "Local port:" with a text box containing "13043", "To IP address:" with a text box containing "127.0.0.1", "To port:" with a text box containing "13044", and "Sample time (-1 for inherited):" with a text box containing "1". At the bottom are four buttons: "OK", "Cancel", "Help", and "Apply".

Packing Bytes



Input port data types:
Приводится описание
типов входных данных

На вход Length блока
подается длина
сообщения в байтах

Block Parameters: Byte Packing

xpcbytepacking (mask) (link)
Pack bytes from multiple input vectors into a single output vector.

Parameters

Output port (packed) data type:

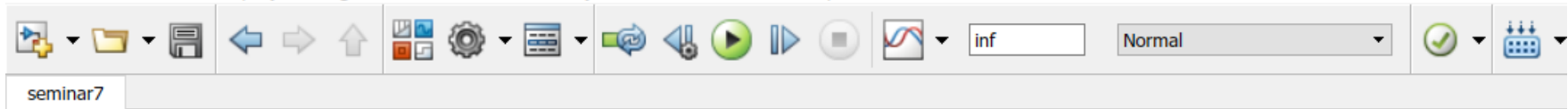
Input port (unpacked) data types (cell array):

Byte Alignment:

OK Cancel Help Apply

Синхронизация времени

File Edit View Display Diagram Simulation Analysis Code Tools Help



seminar7



Block Parameters: Real-Time Synchronization

Simulink Desktop Real-Time Synchronization (mask) (link)

Synchronize model execution to real time in Normal Mode.
This block performs no action in External Mode.

Parameters

Sample time

0.00001

Maximum missed ticks

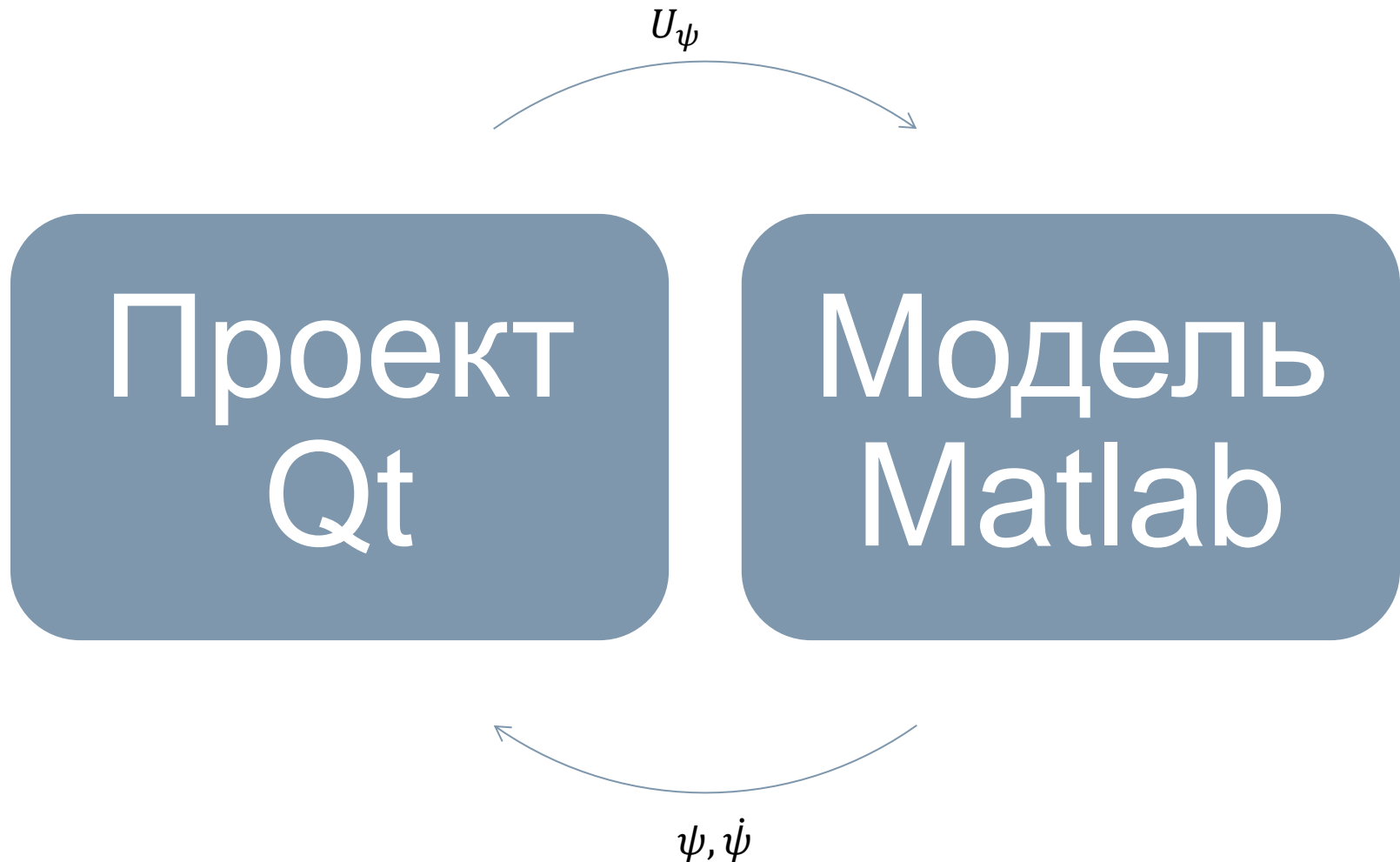
10

☐ Show missed ticks port

☐ Yield CPU when waiting

OK Cancel Help Apply

Работа программы в Qt



Логика работы программы

Считывание принятых по udr
данных $(\psi, \dot{\psi})$



Расчет управления U_{ψ}



Передача новых данных в модель

Практическая часть 1. Передача данных в модель

- Дописать методы для работы основного цикла класса SU_ROV
- В классе UDPSender:
 - Написать структуру под отправку данных в модель
 - Создать сокет под отправку данных
 - Создать метод, который отправляет заданные значения в модель

Практическая часть 1.

Передача данных в модель

Основной класс проекта –
SU_ROV :

- В нем происходит расчет управляющих значений;
- Создание объекта под приём и отправку данных из модели

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

Практическая часть 1. Класс SU_ROV

```
su_rov.h  Upsi: double

1  #ifndef SU_ROV_H
2  #define SU_ROV_H
3
4  #include <QObject>
5  #include <QTimer>
6  #include "udpsender.h"
7
8  class SU_ROV : public QObject
9  {
10     ... Q_OBJECT
11     public:
12     ... explicit SU_ROV(QObject *parent = 0);
13
14     signals:
15
16     public slots:
17     ... void tick();
18
19     private:
20     ... QTimer timer;
21     ... //класс под приём и отправку
22     ... UdpSender udp;
23     ... //заданный, текущий курс и угловая скорость по курсу
24     ... double psiDesired, psiCurrent, dPsi;
25     ... //коэффициенты K1, K2
26     ... double K1, K2,;
27     ... //управляющий сигнал
28     ... double Upsi;
29 };
```


Практическая часть 1. Класс SU_ROV

```
1  #include "su_rov.h"
2
3  ✓ SU_ROV::SU_ROV(QObject *parent) : QObject(parent)
4  {
5      ....psiDesired=10;
6      ....psiCurrent=0;
7      ....K1=2;
8      ....K2=1;
9      ....dPsi=0;
10     ....Upsi=0;
11     ....connect(&timer,SIGNAL(timeout()),SLOT(tick()));
12     ....timer.start(100);
13 }
14
15 ✓ void SU_ROV::tick()
16 {
17     // основной цикл
18 }
19 ◆
```

Передача данных

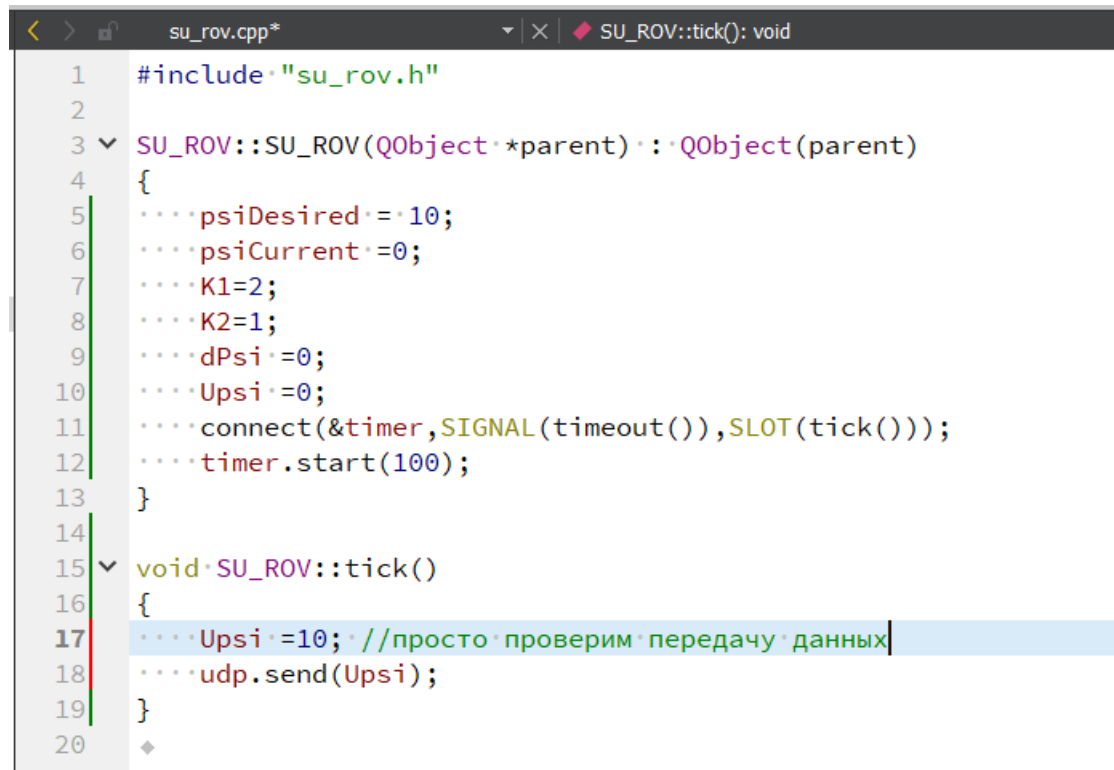
```
udpsender.h* send(double): void
1  #ifndef UDPSENDER_H
2  #define UDPSENDER_H
3  #include <QUdpSocket>
4
5  //структура данных, отправляемая в модель
6  struct ToMatlab{
7      ToMatlab():Upsi(0){}
8      double Upsi; //управляющий сигнал
9  };
10
11
12  class UdpSender:public QObject
13  {
14      Q_OBJECT
15  public:
16      UdpSender(QObject *prt=nullptr);
17  private:
18      //данные для отправки и приёма
19      ToMatlab sendData;
20
21      //сокет под отправку
22      QUdpSocket *m_socket;
23
24  public slots:
25      //метод, который отправляет сообщение с новым управляющим сигналом U
26      void send(double U);
27
28  };
29
30  #endif // UDPSENDER_H
31
```

Передача данных

```
udpsender.cpp*  UdpSender::send(double): void
1  #include "udpsender.h"
2
3
4  UdpSender::UdpSender(QObject *prt):QObject(prt)
5  {
6      ....//создаем сокеты
7      ....m_socket=new QUdpSocket(this);
8
9      ....//биндим сокет и выводим результат бинда
10     ....qDebug()<<m_socket->bind(QHostAddress::LocalHost,13041);
11     ....qDebug()<<m_socket->errorString();
12
13
14 }
15
16 void UdpSender::send(double U){
17     ....sendData.Upsi=U;
18     ....qDebug()<<m_socket->writeDatagram((char*)&sendData,sizeof(sendData),QHostAddress::LocalHost,13042);
19 }
20
```

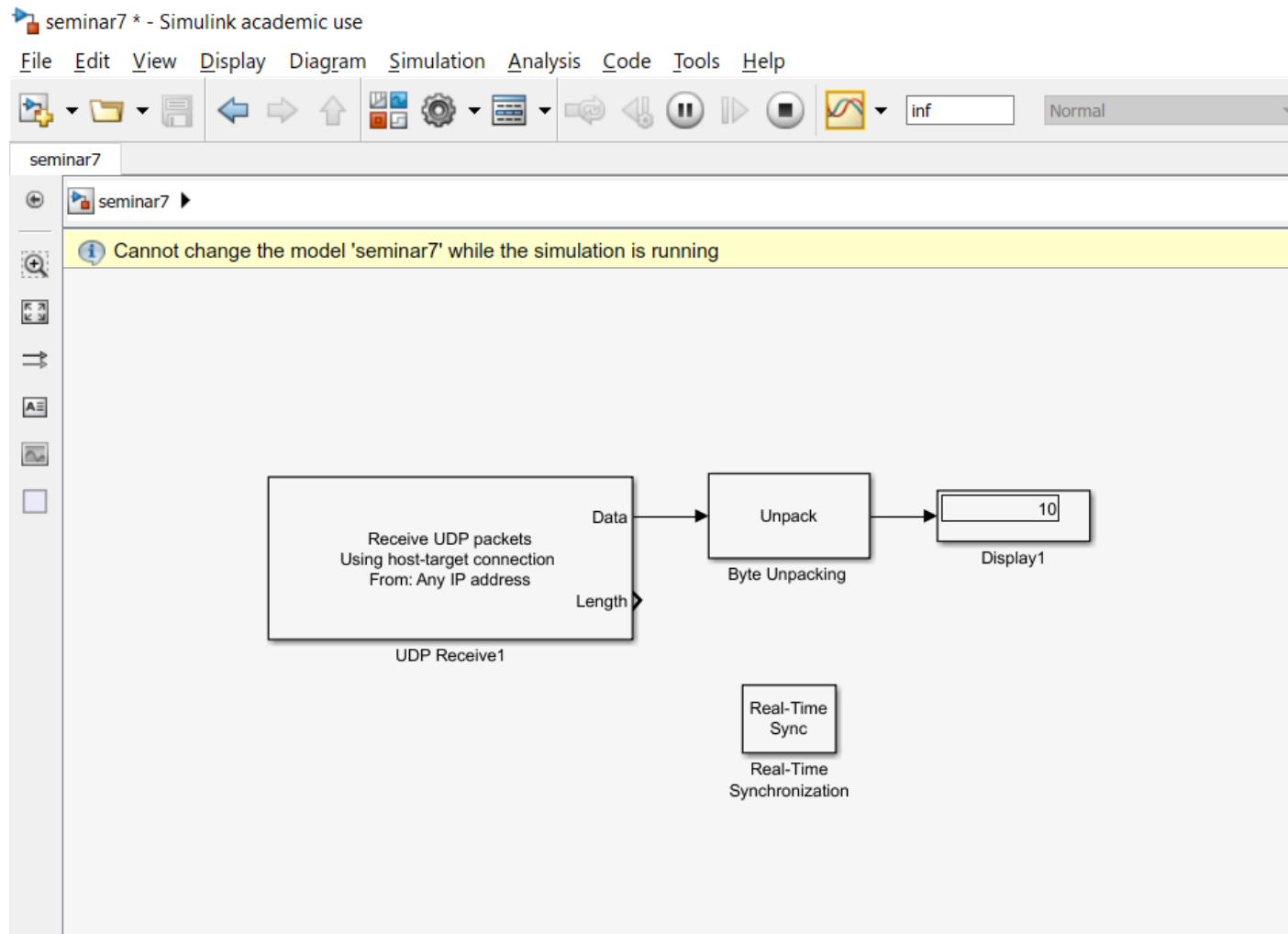
Передача данных

Допишем соответствующие строки под отправку в методе SU_ROV::tick()



```
1  #include "su_rov.h"
2
3  SU_ROV::SU_ROV(QObject *parent) : QObject(parent)
4  {
5      psiDesired = 10;
6      psiCurrent = 0;
7      K1 = 2;
8      K2 = 1;
9      dPsi = 0;
10     Upsi = 0;
11     connect(&timer, SIGNAL(timeout()), SLOT(tick()));
12     timer.start(100);
13 }
14
15 void SU_ROV::tick()
16 {
17     Upsi = 10; //просто проверим передачу данных
18     udp.send(Upsi);
19 }
20
```

Тестируем, что есть связь с моделью



Практическая часть 2.

Прием данных от модели

- Добавить структуру принимаемых данных
- Добавить сокет под приём данных в класс UdpSender;
- Добавить метод под обработку принятых данных;
- Добавить getter – под структуру принятых данных;
- Проверка работоспособности обмена.

Практическая часть

Добавляем структуру данных, принимаемых от модели

```
Seminar7/udpsender.h*  Upsi: double
4
5 //структура данных, отправляемая в модель
6 struct ToMatlab{
7     ToMatlab():Upsi(0){}
8     double Upsi; //управляющий сигнал
9 };
10
11 //[[1]] структура данных, принятых от модели в Matlab
12 struct FromMatlab{
13     FromMatlab(){Psi=0; dPsi=0;}
14     double Psi; //курс
15     double dPsi; //угловая скорость
16 };
17
18 class UdpSender:public QObject
19 {
20     Q_OBJECT
21 public:
22     UdpSender(QObject *prt=nullptr);
23 private:
24     //данные для отправки и приёма
25     ToMatlab sendData;
26     //[[1]] структура данных, принимаемых от модели
27     FromMatlab receivedData;
28     //сокет под отправку
29     QUdpSocket *m_socket;
30 }
```

Создание сокета под приём

```
17 |  
18 | class UdpSender: public QObject  
19 | {  
20 |     Q_OBJECT  
21 | public:  
22 |     UdpSender(QObject *prt = nullptr);  
23 | private:  
24 |     // данные для отправки и приёма  
25 |     ToMatlab sendData;  
26 |     // [1] = структура данных, принимаемых от модели  
27 |     FromMatlab receivedData;  
28 |     // сокет под отправку  
29 |     QUdpSocket *m_socket;  
30 |     // [2] сокет по приёму  
31 |     QUdpSocket *m_receiveSocket;
```


Создание сокета под приём

```
Seminar7/udpsender.cpp*  Udpsender::UdpSender(QObject *)
1  #include "udpsender.h"
2
3
4  Udpsender::UdpSender(QObject *prt):QObject(prt)
5  {
6      ....//создаем сокеты
7      ....m_socket=new QUdpSocket(this);
8      ....//биндим сокет и выводим результат бинда
9      ....qDebug()<<m_socket->bind(QHostAddress::LocalHost,13041);
10     ....qDebug()<<m_socket->errorString();
11     ....
12     ....//[2] -- добавляем сокет под приём
13     ....m_receiveSocket=new QUdpSocket(this);
14     ....//биндим сокет и выводим результат бинда
15     ....qDebug()<<m_receiveSocket->bind(QHostAddress::LocalHost,13044);
16     ....qDebug()<<m_receiveSocket->errorString();
17
18
```

Создание слота под чтение данных

```
Seminar7/udpsender.h*  readData(): void
16 };
17
18 class UdpSender: public QObject
19 {
20     Q_OBJECT
21 public:
22     UdpSender(QObject *prt = nullptr);
23 private:
24     // данные для отправки и приёма
25     ToMatlab sendData;
26     // [1] - структура данных, принимаемых от модели
27     FromMatlab receivedData;
28     // сокет под отправку
29     QUdpSocket *m_socket;
30     // [2] сокет по приёму
31     QUdpSocket *m_receiveSocket;
32
33 public slots:
34     // метод, который отправляет сообщение с новым управляющим сигналом U
35     void send(double U);
36
37     // [3] слот, который считывает принятые по udp сообщения и заполняет ими
38     // структуры receivedData
39     void readData() {
40         while(m_receiveSocket->hasPendingDatagrams()) {
41             m_receiveSocket->readDatagram((char*)&receivedData, sizeof(receivedData));
42             qDebug() << "received X:" << receivedData.Psi;
43             qDebug() << "received Y:" << receivedData.dPsi;
44         }
45     }
```

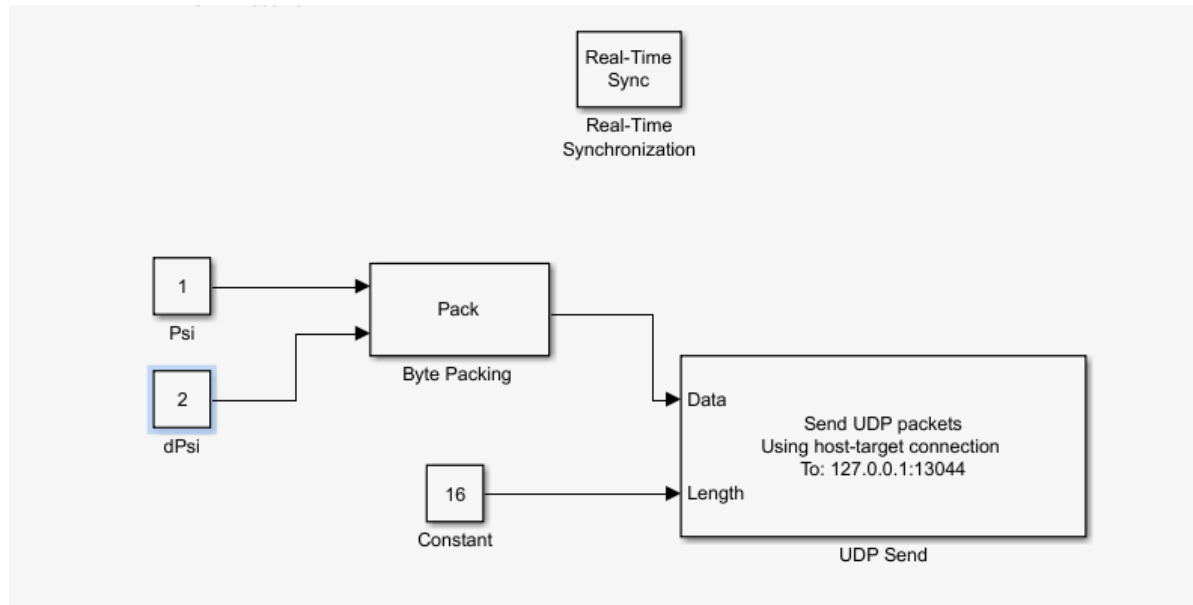
Коннект слота

```
Seminar7/udpsender.cpp  UdpSender::UdpSender(QObject *)
1  #include "udpsender.h"
2
3
4  UdpSender::UdpSender(QObject *prt):QObject(prt)
5  {
6      ....//создаем сокет
7      ....m_socket=new QUdpSocket(this);
8      ....//биндим сокет и выводим результат бинда
9      ....qDebug()<<m_socket->bind(QHostAddress::LocalHost,13041);
10     ....qDebug()<<m_socket->errorString();
11
12     ....//[2] -- добавляем сокет под приём
13     ....m_receiveSocket=new QUdpSocket(this);
14     ....//биндим сокет и выводим результат бинда
15     ....qDebug()<<m_receiveSocket->bind(QHostAddress::LocalHost,13044);
16     ....qDebug()<<m_receiveSocket->errorString();
17
18     ....//[3] соединяем слот, который считывает данные из порта с сигналом
19     ....//о том, что пришли новые сообщения readyRead()
20     ....connect(m_receiveSocket,SIGNAL(readyRead()),this,SLOT(readData()));
21
```

Добавление getter

```
Seminar7/udpsender.h*  readData(): void
25  ....ToMatlab* sendData;
26  ....//[1] *структура данных, *принимаемых *от *модели
27  ....FromMatlab* receivedData;
28  ....//сокет *под *отправку
29  ....QUdpSocket *m_socket;
30  ....//[2] *сокет *по *приём
31  ....QUdpSocket *m_receiveSocket;
32
33  public slots:
34  ....//метод, *который *отправляет *сообщение *с *новым *управляющим *сигналом *U
35  ....void send(double U);
36  ....
37  ....//[3] *слот, *который *считывает *принятые *по *udp *сообщения *и *заполняет *ими
38  ....//структуру *receivedData
39  ....void readData(){
40  ....    while(m_receiveSocket->hasPendingDatagrams()){
41  ....        m_receiveSocket->readDatagram((char*)&receivedData, sizeof(receivedData));
42  ....        qDebug() << "received X:" << receivedData.Psi;
43  ....        qDebug() << "received Y:" << receivedData.dPsi;
44  ....    }
45  ....}
46  public:
47  ....//[4] *метод, *который *возвращает *принятые *данные
48  ....FromMatlab* getData(){return receivedData;}
49  };
```

Проверка приёма данных



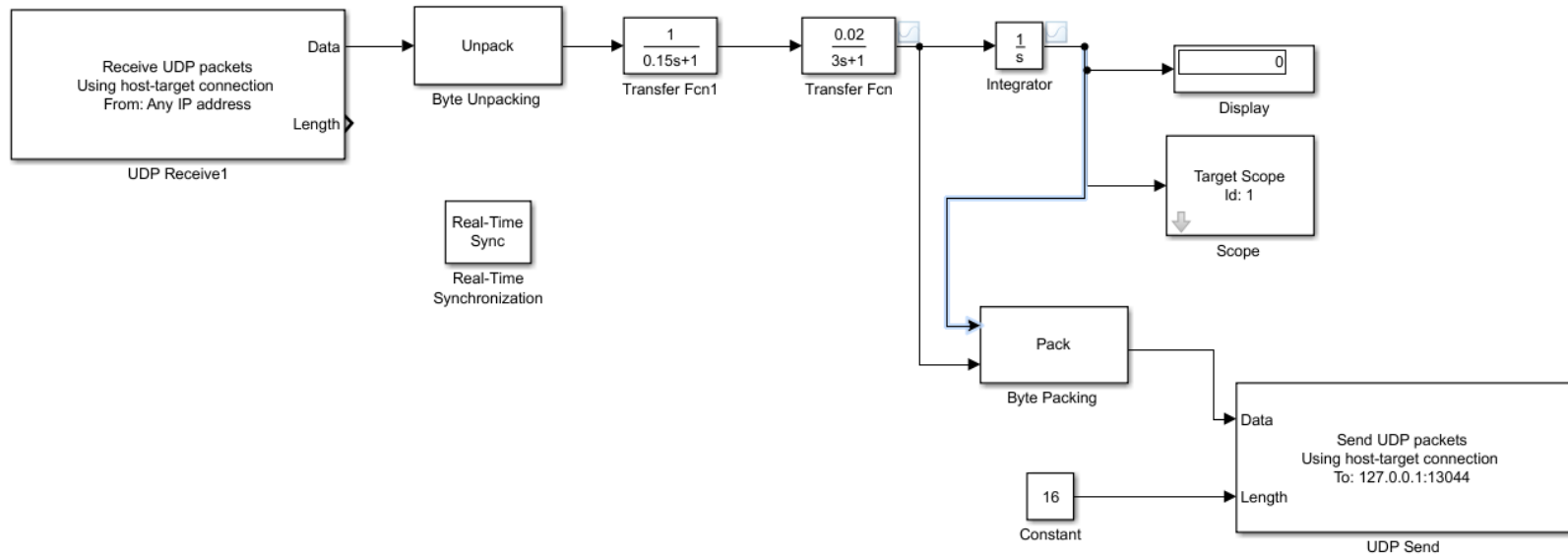
```
8
8
8
8
8
received X: 1
1679841594 2
8
8
8
```

Написать регулятор и посмотреть переходный процесс

- Переписываем метод SU_ROV::tick()

```
Seminar7/su_rov.cpp*  SU_ROV::tick(): void
1  #include "su_rov.h"
2
3  SU_ROV::SU_ROV(QObject* parent) : QObject(parent)
4  {
5      ...psiDesired = 10;
6      ...psiCurrent = 0;
7      ...K1 = 2;
8      ...K2 = 1;
9      ...dPsi = 0;
10     ...Upsi = 0;
11     ...connect(&timer, SIGNAL(timeout()), SLOT(tick()));
12     ...timer.start(100);
13 }
14
15 void SU_ROV::tick()
16 {
17     ...//считываем текущие данные
18     ...dPsi = udp.getData().dPsi;
19     ...psiCurrent = udp.getData().Psi;
20     ...//расчитываем управляющий сигнал регулятора
21     ...Upsi = (psiDesired - psiCurrent) * K1 - K2 * dPsi;
22     ...//отправляем данные
23     ...udp.send(Upsi);
24 }
```

Переделываем модель в Matlab



Вывод данных, проверка качества процессов

