# ПРОЕКТ «ВИЗУАЛИЗАЦИЯ ДВИЖЕНИЯ АНПА В ПЛОСКОСТИ ГОРИЗОНТА V2»

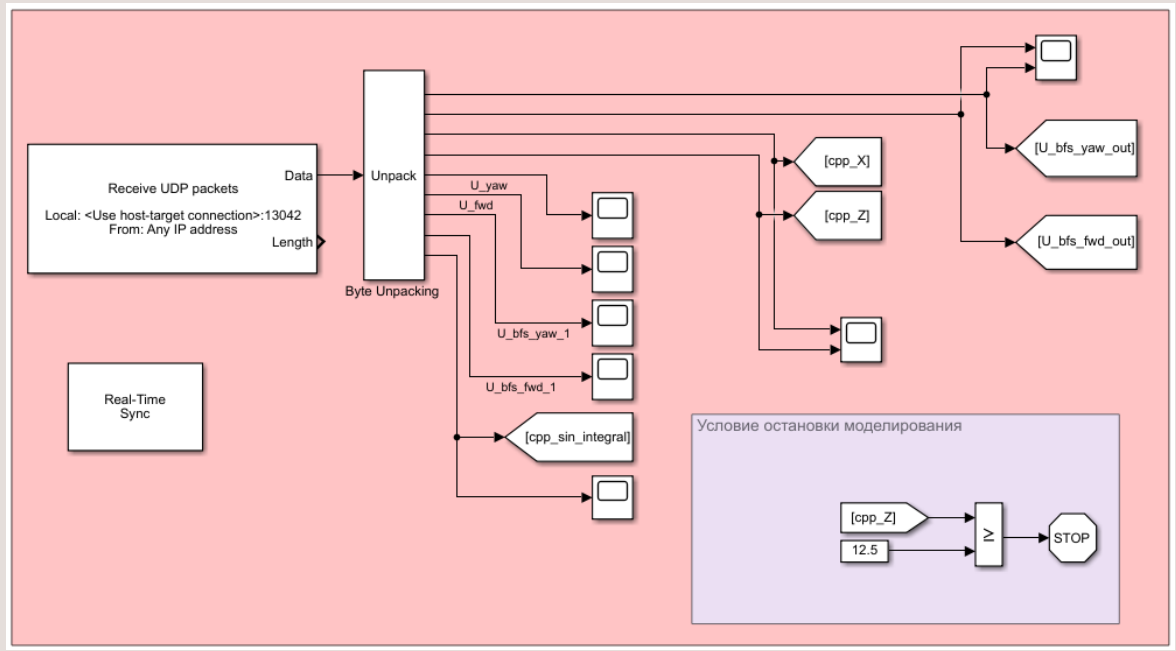
Андреев Е.В., СМ11-41М

# Цели и задачи

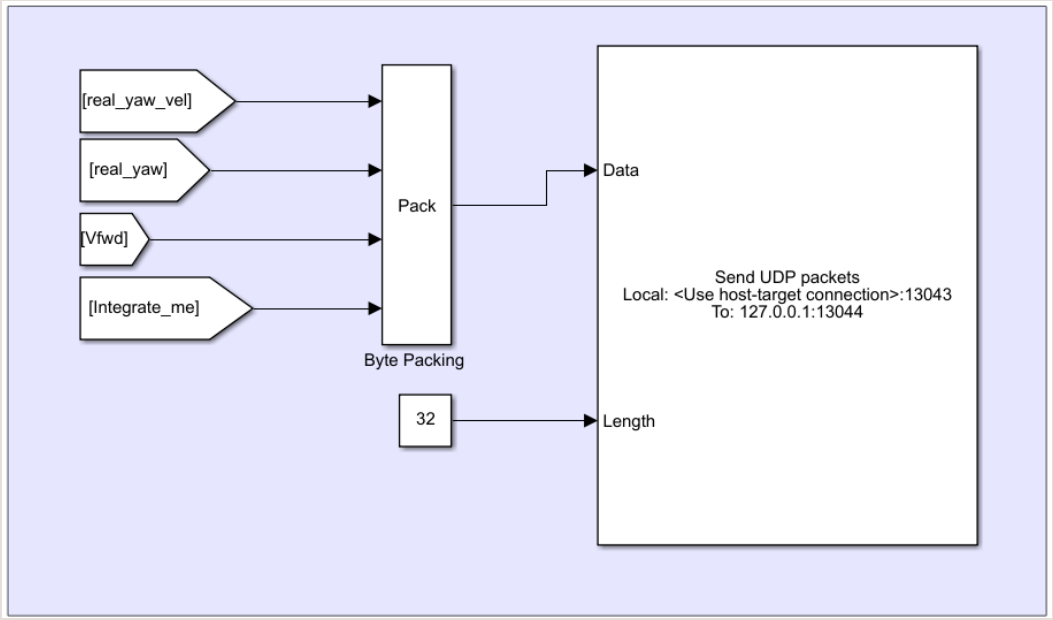
- Получить зачёт по курсу «Проектирование ПО ПРТС»
- Попутно узнать что-нибудь новенькое для себя

# Избранный путь

- Обмен по UDP с периодом в 10 мс
- Получаем в модели
  - напряжение движителей по маршу и курсу
  - Вычисленные параметры положения АНПА для отладки
- Отправляем из модели
  - значение угла курса,
  - скорость поворота по курсу и
  - маршевую скорость

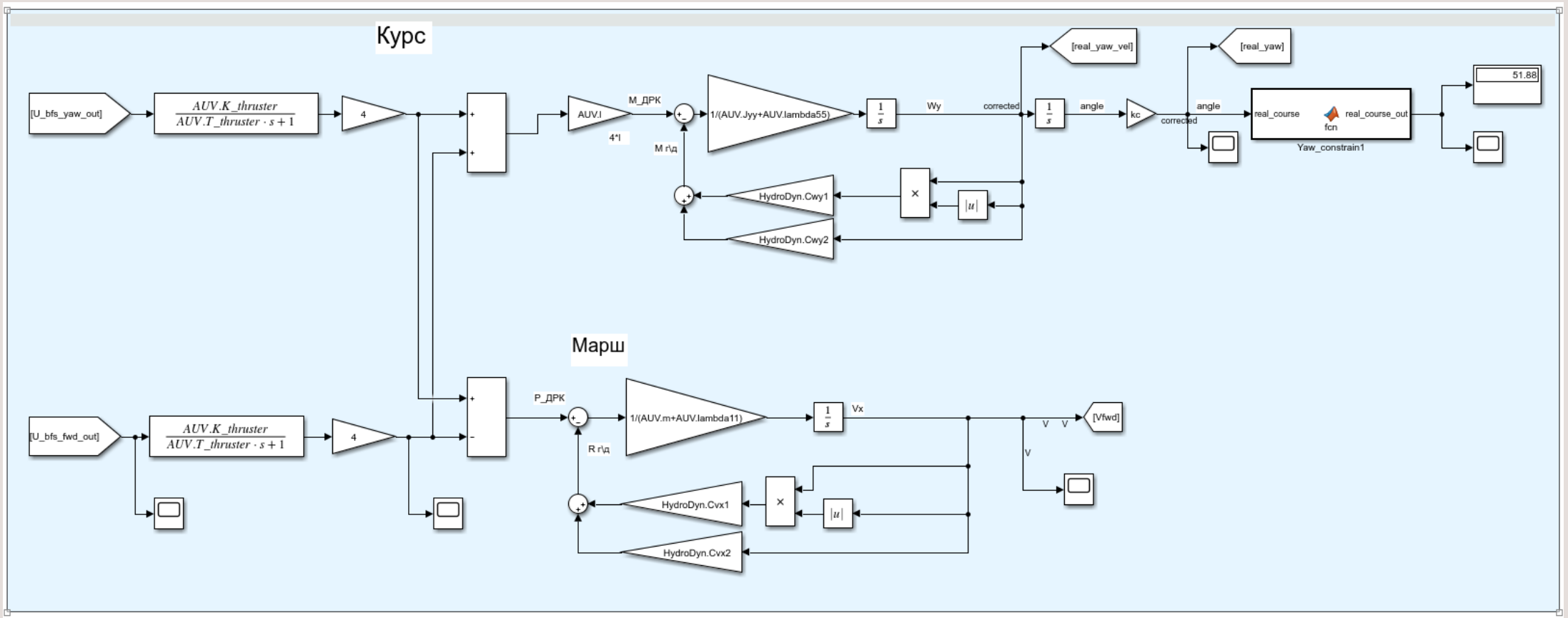






# Общий вид модели

- Контур марша разомкнут, контур курса – замкнут
- Учтено взаимовлияние каналов
- Регуляторы **k1** && **k2** реализованы в коде
- Порядок модели – 3-й
- Параметры мат. модели заботливо спрятаны с matlab-скрипте

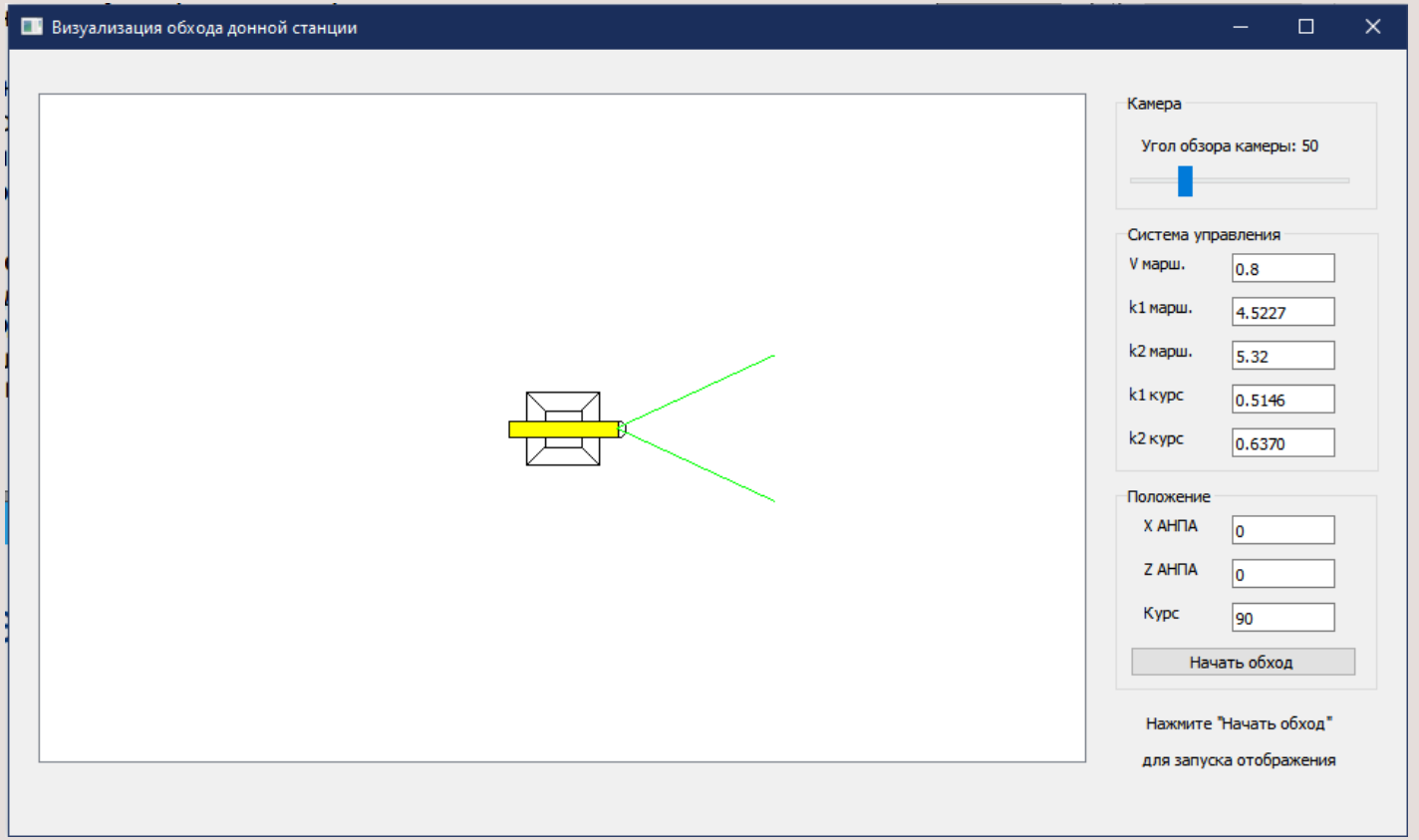


# Что делает код

| | |
|---|---|
| Генерирует точки траектории (заранее) | `generate_path_points()` |
| Получает данные по UDP | `this->real_yaw = udp.getData().real_yaw;` |
| Вычисляет текущее положение АНПА | `calc_position();` |
| Проверяет не достигнута ли текущая траекторная точка | `check_distance();` |
| Осуществляет выборку следующей траекторной точки | `this->X1 = x_final[dot_number];`<br>`this->Z1 = z_final[dot_number]` |
| Определяет направление движения по маршу | `calc_dir();` |
| Рассчитывает желаемый курс | `calc_desired_yaw();`<br>`constrain_yaw();` |
| Рассчитывает напряжения движителей с учётом насыщений и взаимовлияния каналов | <много кода> |
| Отправляет данные по UDP | |
| Отрисовывает вычисленную траекторию на главном экране | |

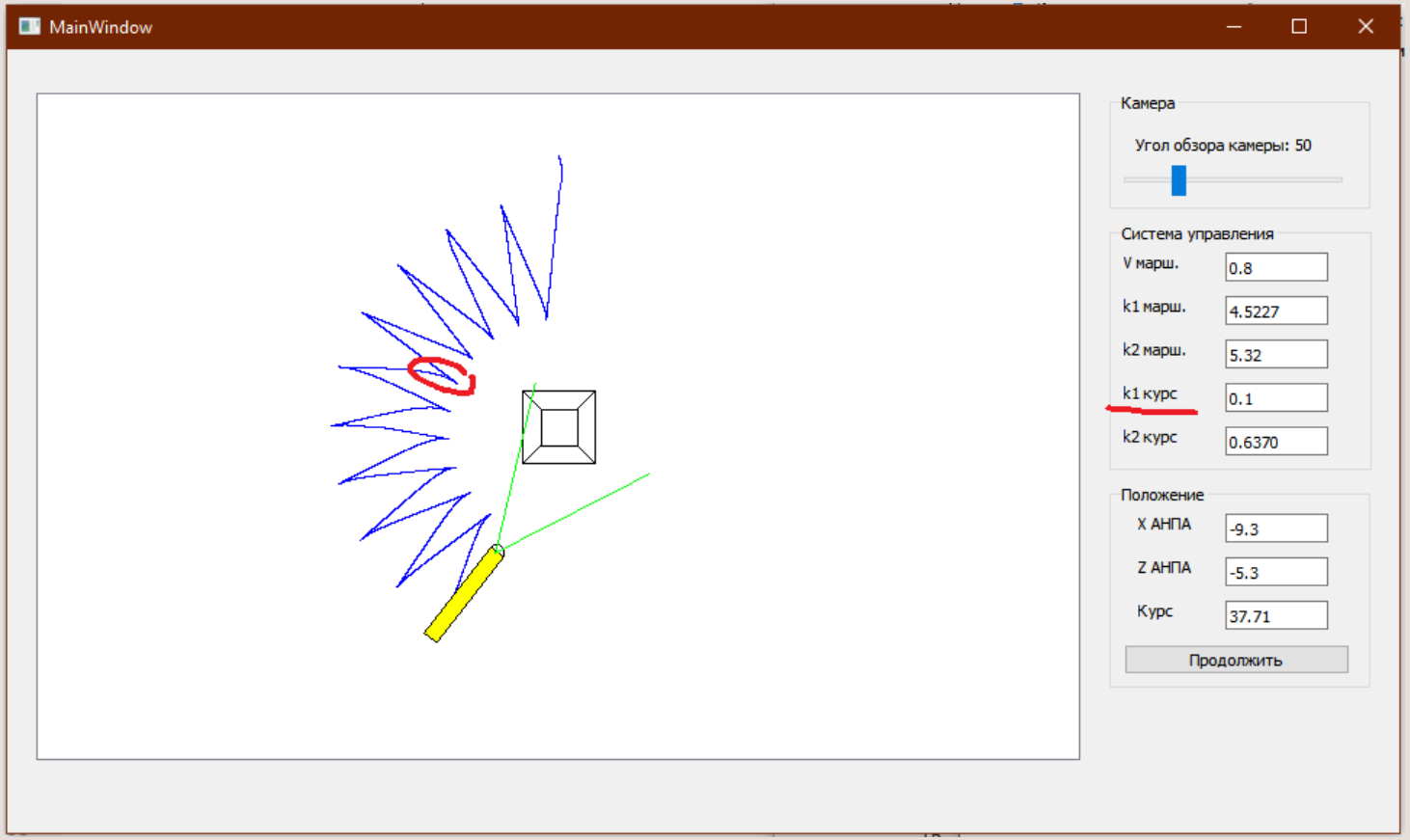
# Графический интерфейс

Начальное состояние GUI программы



# Процесс визуализации

Был произведён останов моделирования, уменьшение k1 курса и возобновление расчёта



# Было, 1

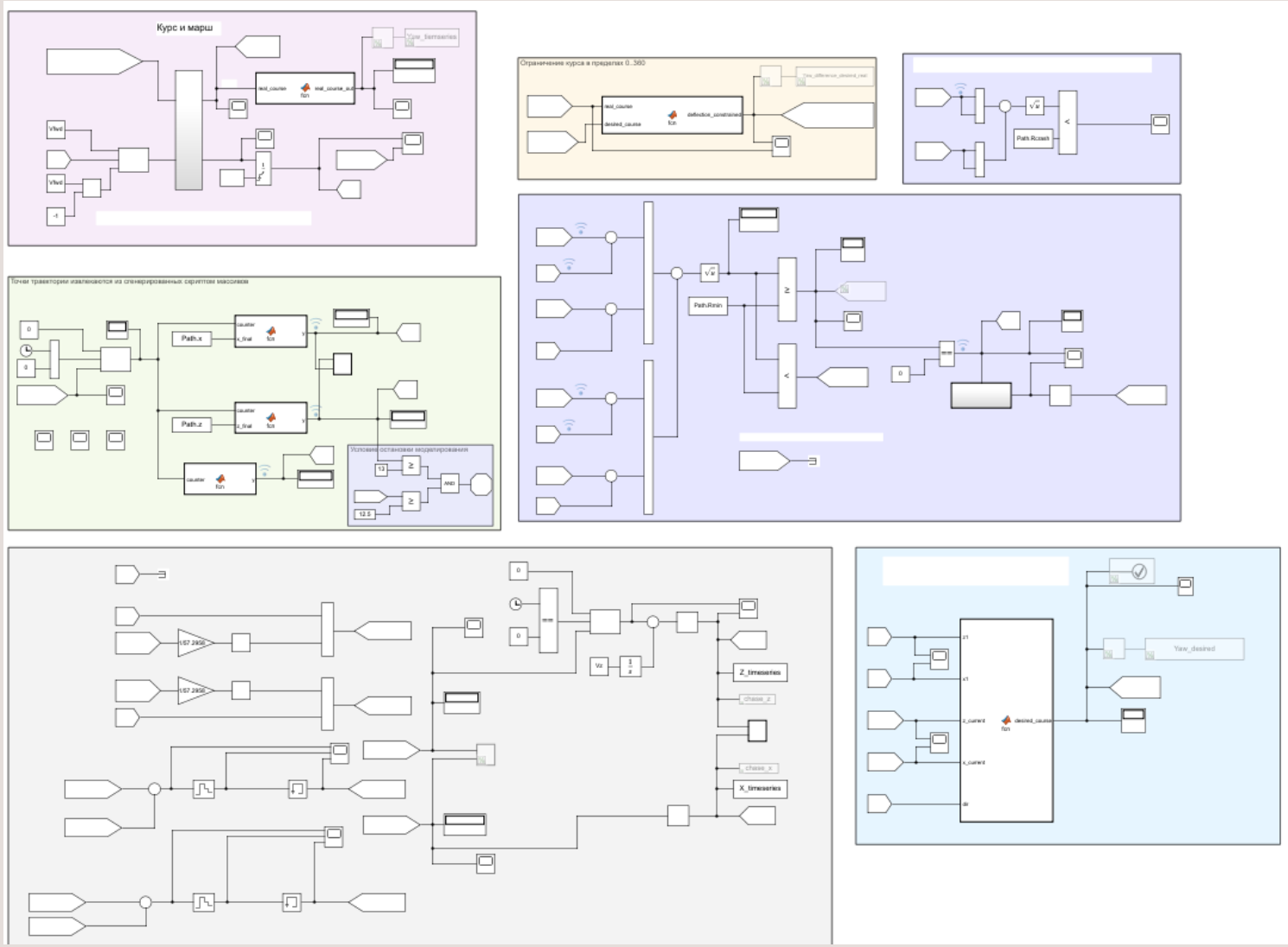
Calculations.m | Parameters_2.m | Yaw_constrain

```
function real_course_out    = fcn(real_course)

    % Приводим к стандарту
    if real_course < 0
        real_course_out = 360 - abs(real_course);
    elseif real_course >= 360
        real_course_out = real_course - 360;
    else
        real_course_out = real_course;
    end

end
```

```
function deflection_constrained    = fcn(real_course, desired_course)

    % предстоит повернуться более чем на 180 градусов
    % текущее положение курса не имеет никакого значения,
    %важно лишь конечное положение.
    % Па должен поворачивать менее чем на 180 градусов

    if abs(desired_course - real_course) > 180
    % выполняем ограничение
        if desired_course > 180
            deflection_constrained = desired_course - 360;
        %elseif desired_course < -180
        %    deflection_constrained = 360 + desired_course;
        else
            deflection_constrained = desired_course;
        end
    % иначе просто приравниваем
    else
        deflection_constrained = desired_course;
    end
end
```

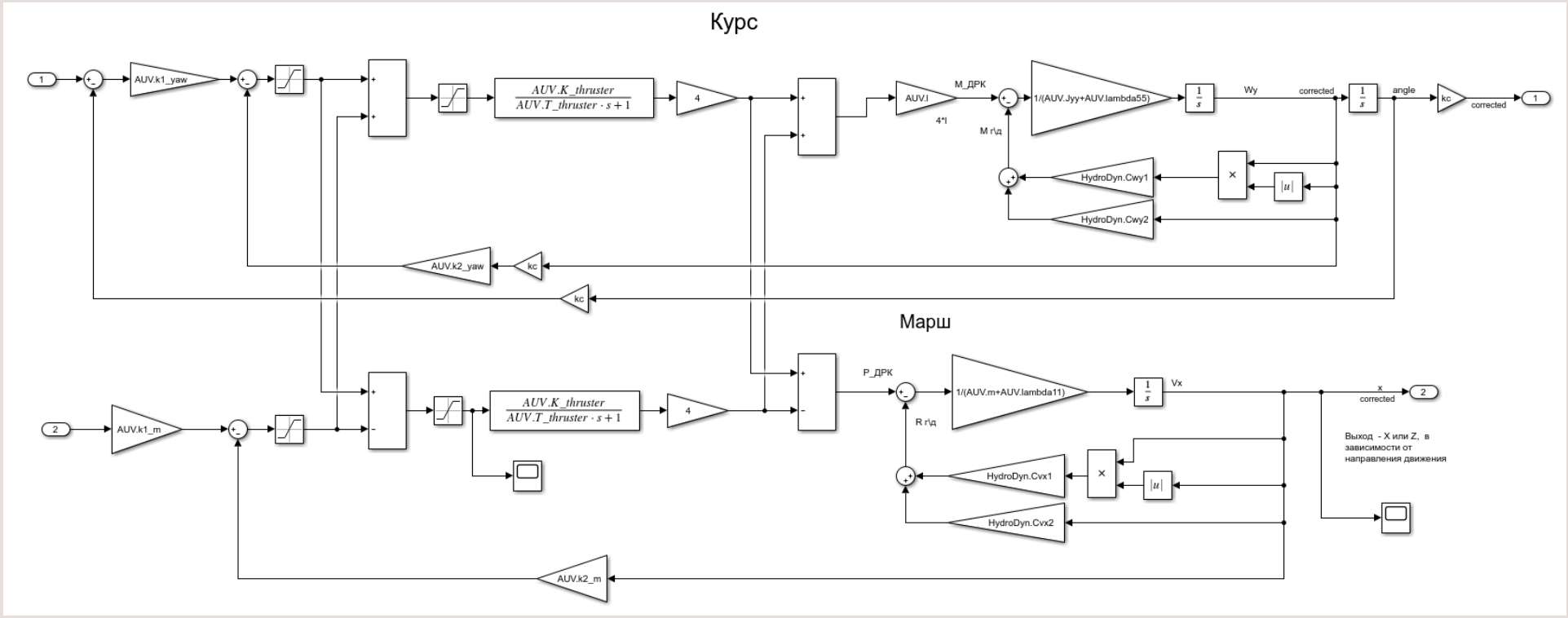
Calculations.m | Parameters_2.m | MATLAB Function2

```
function desired_course = fcn(z1, x1, z_current, x_current, dir)

delta_x = x1 - x_current;
delta_z = z1 - z_current;

%1
if (delta_x >= 0) && (delta_z >= 0)
    desired_course = 57.2958*atan(delta_z / delta_x);
%2
elseif (delta_x >= 0) && (delta_z < 0)
    desired_course = 360 - 57.2958*atan( abs(delta_z) / delta_x);
%3
elseif (delta_x < 0) && (delta_z < 0)
    desired_course = 180 + 57.2958*atan(abs(delta_z) / abs(delta_x));
%4
elseif (delta_x < 0) && (delta_z >= 0)
    desired_course = 180 - 57.2958*atan( delta_z / abs(delta_x));

else
    desired_course = -1;
end


% При движении назад отнимаем от 360-и
if dir == -1
    desired_course = desired_course - 180;
end

end
```

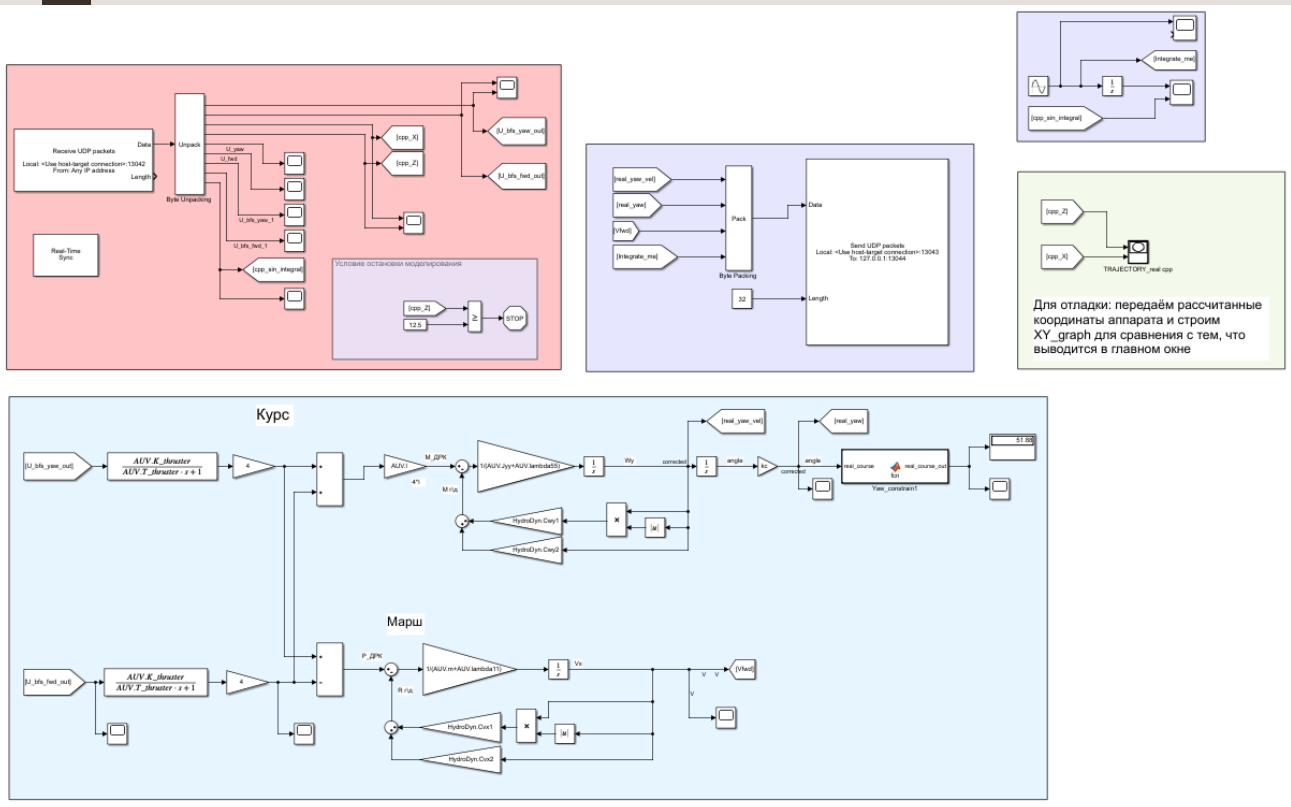



```
    z_outer = zeros(1, int32(Path.N_points));
    x_outer = zeros(1, int32(Path.N_points));
    z_inner = zeros(1, int32(Path.N_points));
    x_inner = zeros(1, int32(Path.N_points));

for i=1:Path.N_points
    %disp( class(i) ); disp( i );

    [p1_z, p1_x] = pol2cart(deg2rad(Path.angle), Path.D_mass_center_max);
    [p2_z, p2_x] = pol2cart(deg2rad(Path.angle + Path.angular_delta/2), Pa

    %disp(p1_z); disp(p1_x);

    z_outer(int32(i)) = p1_z;
    x_outer(int32(i)) = p1_x;

    z_inner(int32(i)) = p2_z;
    x_inner(int32(i)) = p2_x;

    Path.angle = Path.angle + Path.angular_delta;

end

% Инициализируем нулями массивы для точек итоговой траектории
Path.z = zeros(1, int32(Path.N_points*2));
Path.x = zeros(1, int32(Path.N_points*2));
Path.direction = zeros(1, int32(Path.N_points*2));

j = 1;
for i=1:Path.N_points
    Path.z(j) = z_outer(i);
    Path.x(j) = x_outer(i);
    Path.direction(j) = 1;
    j = j + 1;

    Path.z(j) = z_inner(i);
    Path.x(j) = x_inner(i);
    Path.direction(j) = -1;
    j = j + 1;
end

clear p1_z p1_x p2_z p2_x i j z_inner x_inner z_outer x_outer;
```

# Было, 2


Курс
1
AUV.k1_yaw
AUV.K_thruster
AUV.T_thruster · s + 1
4
AUV.I
4*I
M_ДРК
M r\д
1/(AUV.Jyy+AUV.lambda55)
Wy
corrected
angle
kc
corrected
1
HydroDyn.Cwy1
HydroDyn.Cwy2
AUV.k2_yaw
kc
kc
Марш
2
AUV.k1_m
AUV.K_thruster
AUV.T_thruster · s + 1
4
P_ДРК
R r\д
1/(AUV.m+AUV.lambda11)
Vx
x
corrected
2
Выход - X или Z, в зависимости от направления движения
HydroDyn.Cvx1
HydroDyn.Cvx2
AUV.k2_m

# Стало




```
182 void SU_ROV::tick()
183 {
184     // Получаем данные
185     this->real_yaw_vel = udp.getData().real_yaw_vel;
186     this->real_yaw = udp.getData().real_yaw;
187     this->real_V_fwd = udp.getData().real_V;
188
189     this->sin_curr = udp.getData().real_sin;
190     qDebug() << "sin_curr: " << sin_curr << " prev:" << sin_prev << " sin_summ: " << sin_summ <<  "\n";
191     // Вычисляем
192
193     //double temp =
194     this->sin_summ += ( (sin_prev + sin_curr)/2 * ( (double)timer_period / 1000) );
195     this->sin_prev = sin_curr;
196
197     calc_position();
198     check_distance();
199
200     assert(dot_number < x_final.size());
201
202     this->X1 = x_final[dot_number];
203     this->Z1 = z_final[dot_number];
204
205     calc_dir();
206     check_end_simulation();
207     calc_desired_yaw();
208     constrain_yaw();
209
210     this->U_fwd = this->dir * this->modelParams.Vfwd * this->k1_m;
211     this->U_yaw = (this->deflection_yaw_constrained - this->real_yaw) * this->k1_yaw;
212
213     this->U_bfs_yaw_1 = saturation_block(this->U_yaw - this->k2_yaw * qRadiansToDegrees(this->real_yaw_vel));
214     this->U_bfs_fwd_1 = saturation_block(U_fwd - this->k2_m * this->real_V_fwd);
215
216     this->U_bfs_yaw_out = saturation_block(this->U_bfs_yaw_1 + U_bfs_fwd_1, 12, -12);
217     this->U_bfs_fwd_out = saturation_block(this->U_bfs_yaw_1 - U_bfs_fwd_1, 12, -12);
218
219     emit sendComputedCoords(X_current, Z_current, real_yaw); // to Trajectory class
220
221     udp.send(this->U_bfs_yaw_out, U_bfs_fwd_out, this->X_current, this->Z_current,
222              this->U_yaw,
223              this->U_fwd,
224              this->U_bfs_yaw_1,
225              this->U_bfs_fwd_1,
226              this->sin_summ);
227 }
```

# Итого

- Обмен по UDP
- Регуляторы **k1** & **k2** марша и курса в коде
- Контур курса замкнут
- Реализовано следование заданной траектории: выборка путевой точки, расчёт желаемого угла курса, определение выхода в точку
- Формирование управляющих сигналов в коде
- Реализована система счисления пути
- GUI позволяет изменить скорость маршевого движения и ~~скорректировать~~ сломать регуляторы СУ
- Сигналы и слоты соединены

# Пути улучшения ПО

- Добавить поля для задания IP-адреса и номера порта

**С прошлого семестра:**

- Добавить оси с подписями и отсчётами и координатную сетку.
- В случае перемещения в пространстве использовать модуль Qt Data Visualization.
- Встроить текстуру морского дна в виде фона.
- Добавить возможность масштабирования и перемещения по получившейся карте.
- Возможность запуска моделирования напрямую из C++-кода (сделано)

Спасибо за внимание)