2. ВТОРОЙ ЭТАП

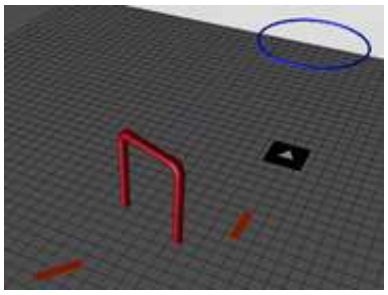
Задачи второго этапа

3.1. Задачи

Задания выполняются в симуляторе и среде программирования MUR_IDE.

Задача 3.1.1. (20 баллов)

Конфигурация первой сцены



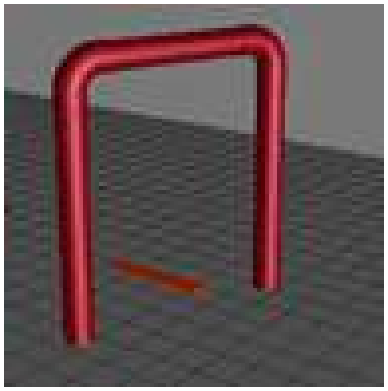
Для успешного прохождения задания аппарат в автономном режиме должен:

- Пройти по полоске – 3 балла.
- Пройти в ворота – 7 баллов.
- Пройти по полоске – 3 балла.
- Всплыть в обруче – 7 баллов (возможен штраф в 3 балла).

Полоска

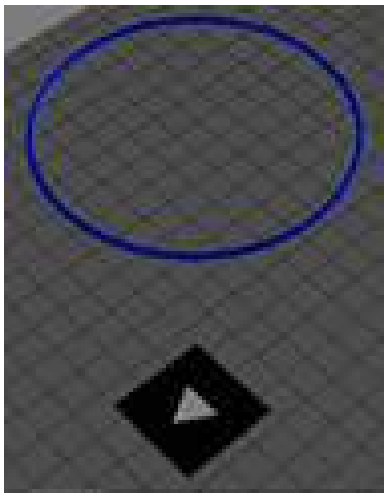
Пройти по полоске. Цвет полоски оранжевый. Полоска служит для определения направления на следующее задание. Направление полоски может варьироваться от −80 градусов до 80 градусов относительно начального курса аппарата. Расстояние от задания и направление полоски варьируется в зависимости от сцены. Полоска расположена на дне бассейна.

Ворота



Пройти сквозь ворота. Цвет столбов и перекладины – красный. Верхний край ворот совпадает с поверхностью воды. Расстояние от старта до ворот варьируется в зависимости от тестовой сцены. За выполнение этого задания начисляется 7 баллов.

Треугольник и обруч



Всплыть над треугольником в кольцо. Центр треугольника совпадает с центром обруча. Треугольник размещен в квадрате, цвет треугольника белый, квадрата черный. Цвет обруча синий. За всплытие в обруче начисляется 7. За касание обруча начисляется 3 штрафных балла.

Система оценки

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Ворота:

- Баллы: 7
- Штрафные баллы: -
- Описание штрафа: -

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Всплыть в обруче:

- Баллы: 7
- Штрафные баллы: 3
- Описание штрафа: Касание обруча

Решение

Для решения данных задач участнику необходимо реализовать:

- регулятор управления курсом и глубиной (подойдет релейный или простой пропорциональный);
- алгоритм поиска полосок и треугольника (подойдет как встроенный в murAPI, так и простая бинаризация);
- алгоритм определения наличия полоски в кадре;
- алгоритм определения угла полоски (подойдет встроенный в OpenCV или murAPI).

Начнем с реализации регуляторов глубины и курса.

```
1   void yaw_and_depth_regulator(float yaw, float depth, int power)
2   {
3       constexpr float k_yaw = 1.3f;
4       constexpr float k_depth = 6.0f;
5
6       float yaw_diff = mur.getYaw() - yaw;
7       if (yaw_diff < 0.0f) {
8           yaw_diff += 360.0f;
9       }
10
11      if (yaw_diff > 180.0f) {
12          yaw_diff -= 360.0f;
13      }
14
15      yaw_diff *= k_yaw;
16
17      float depth_diff = mur.getInputAOne() - depth;
18      depth_diff *= k_depth;
```

```
19
20          mur.setPorts(-power + yaw_diff, -power - yaw_diff, -depth_diff, 0);
21      }
```

Наши регуляторы готовы.

Теперь перейдем к поиску линии и треугольника.

Для решения данной задачи мы воспользуемся встроенными в murAPI функциями поиска прямоугольников и линий. Однако, нам необходимо проверять наличие полоски в кадре.

С помощью данной функции мы будем определять наличие или отсутствии оранжевых полосок в кадре.

```
1       bool is_line(cv::Mat image)
2       {
3           cv::Mat hsv_image;
4           // BGR -> HSV
5           cv::cvtColor(image, hsv_image, CV_BGR2HSV);
6
7           // Бинаризуем изображение по нижней и верхней границе красного
8           cv::Mat lower_red_hue_range;
9           cv::Mat upper_red_hue_range;
10          cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
11              cv::Scalar(10, 255, 255), lower_red_hue_range);
12          cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
13              cv::Scalar(179, 255, 255), upper_red_hue_range);
14          cv::Mat red_hue_image;
15          cv::addWeighted(lower_red_hue_range, 1.0,
16              upper_red_hue_range, 1.0, 0.0, red_hue_image);
17
18          std::vector<std::vector<cv::Point>> contours;
19
20          // Ищем контуры на бинаризованном изображениее
21          cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
22
23          for (std::size_t i = 0; i < contours.size(); i++) {
24              if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
25                  continue;
26              }
27              return true;
28          }
29          return false;
30      }
```

Алгоритм определения наличия линий готов. Для распознавания линий, их углов и треугольника будем использовать встроенные в murAPI функции.

Пример программы-решения

Ниже представлено решение на языке C++

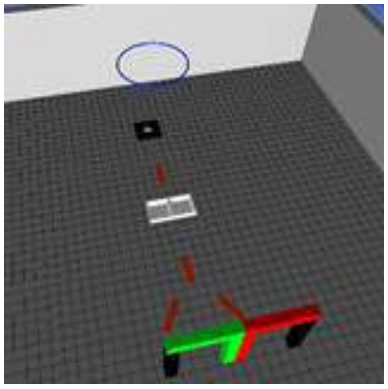
```
1   #include <murAPI.hpp>
2
3   bool is_line(cv::Mat image)
4   {
```

```
5       cv::Mat hsv_image;
6       // BGR -> HSV
7       cv::cvtColor(image, hsv_image, CV_BGR2HSV);
8
9       // Бинаризуем изображение по нижней и верхней границе красного
10      cv::Mat lower_red_hue_range;
11      cv::Mat upper_red_hue_range;
12      cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
13          cv::Scalar(10, 255, 255), lower_red_hue_range);
14      cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
15          cv::Scalar(179, 255, 255), upper_red_hue_range);
16      cv::Mat red_hue_image;
17      cv::addWeighted(lower_red_hue_range, 1.0,
18          upper_red_hue_range, 1.0, 0.0, red_hue_image);
19
20      std::vector<std::vector<cv::Point>> contours;
21
22      // Ищим контуры на бинаризованном изображениее
23      cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
24
25      for (std::size_t i = 0; i < contours.size(); i++) {
26          if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
27              continue;
28          }
29          return true;
30      }
31      return false;
32  }
33
34  float Yaw(float angle)
35  {
36      float angleNEW = angle;
37      if (angleNEW > 90)
38          angleNEW = (-1) * (180 - angleNEW);
39      return angleNEW;
40  }
41
42  void yaw_and_depth_regulator(float yaw, float depth, int power)
43  {
44      constexpr float k_yaw = 1.3;
45      constexpr float k_depth = 6;
46
47      float yaw_diff = mur.getYaw() - yaw;
48      if (yaw_diff < 0.0f) {
49          yaw_diff += 360.0f;
50      }
51
52      if (yaw_diff > 180.0f) {
53          yaw_diff -= 360.0f;
54      }
55
56      yaw_diff *= k_yaw;
57
58      float depth_diff = mur.getInputAOne() - depth;
59      depth_diff *= k_depth;
60
61      mur.setPorts(-power + yaw_diff, -power - yaw_diff, -depth_diff, 0);
62  }
63
64  int main()
```

```
{
    mur.addDetectorToList(Object::RECTANGLE, 0);
    mur.addDetectorToList(Object::TRIANGLE, 0);

    float yaw = 0.0f;
    int power = 10;
    float depth = 70.0f;

    auto triangle = [&yaw, &depth, &power]() {
        bool is_triangle_detected = false;

        while (!is_triangle_detected) {
            yaw_and_depth_regulator(yaw, depth, power);
            for (const auto& obj : mur.getDetectedObjectsList(0)) {
                if (obj.type == Object::TRIANGLE) {
                    yaw += Yaw(obj.angle);
                    is_triangle_detected = true;
                    break;
                }
            }
        }

        // Всплываем.
        mur.setPorts(0, 0, -55, 0);
        sleepFor(5000);
    };

    auto line = [&yaw, &depth, &power]() {
        bool is_line_detected = false;

        while (!is_line_detected) {
            yaw_and_depth_regulator(yaw, depth, power);
            for (const auto& obj : mur.getDetectedObjectsList(0)) {
                if (obj.type == Object::RECTANGLE) {
                    yaw += Yaw(obj.angle);
                    is_line_detected = true;
                    break;
                }
            }
        }
    };

    line();
    while (is_line(mur.getCameraOneFrame())) {
        yaw_and_depth_regulator(yaw, depth, power);
    }
    line();
    while (is_line(mur.getCameraOneFrame())) {
        yaw_and_depth_regulator(yaw, depth, power);
    }
    triangle();

    return 0;
}
```

Задача 3.1.2. (30 баллов)

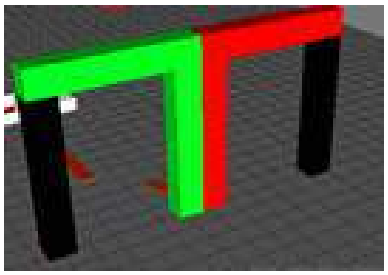
Конфигурация второй сцены



Для успешного прохождения задания аппарат в автономном режиме должен:

- Пройти двойные ворота – 7 (возможен штраф в 7 баллов).
- Пройти по полоске – 3 балла.
- Пройти по полоске – 3 балла.
- Сбросить маркер в корзину – 7 (возможен штраф в 7 баллов).
- Пройти по полоске – 3 балла.
- Всплыть в обруче – 7 баллов (возможен штраф в 3 балла).

Двойные ворота



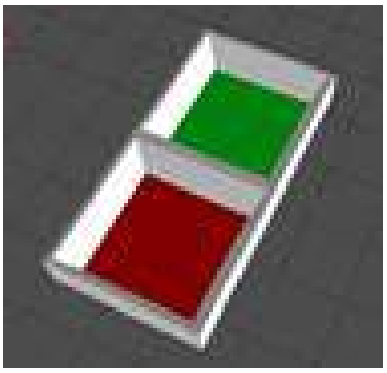
Пройти в зеленую часть ворот. Ворота разбиты на два сектора – красный и зеленый. Верхний край ворот совпадает с поверхностью воды. Расстояние от старта до ворот варьируется в зависимости от тестовой сцены. За прохождение красной части ворот начисляются 7 штрафных баллов, за прохождение зеленой части ворот начисляется 7 баллов.

- Взаимное расположение красной и зеленой части ворот может меняться.

Полоска

Пройти по полоске. Цвет полоски оранжевый. Полоска служит для определения направление на следующие задание. Направление полоски может варьироваться от -80 градусов до 80 градусов относительно начального курса аппарата. Расстояние от задания и направление полоски варьируется в зависимости от сцены.

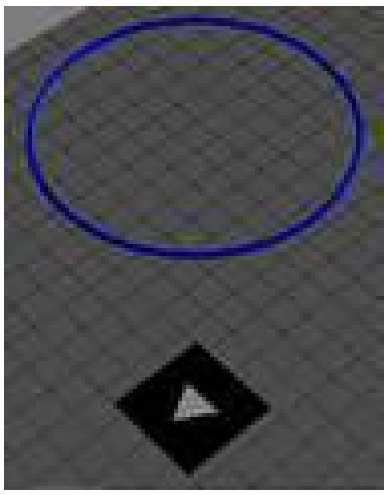
Корзины



Сбросить маркеры в зеленую корзину. Необходимо сбросить маркер синего цвета в зеленую корзину. Корзина имеет белые бортики и цветное дно. Корзины две – одна зеленого цвета, а вторая красного. Корзины расположены недалеко друг от друга. За сброс маркера в зеленую корзину начисляется 7 баллов. За сброс маркеров в красную корзину начисляется 7 штрафных баллов.

- Взаимное расположение корзин может меняться.
- Линия перед корзинами указывает на «центр» между корзинами.
- Бортики корзин соприкасаются.

Тругольник и обруч



Всплыть над треугольником в обруче. Центр треугольника совпадает с центром обруча. Треугольник размещен в квадрате, цвет треугольника белый, квадрата черный. Цвет обруча синий. За всплытие в обруче начисляется 7 баллов. За касание обруча начисляется 3 штрафных балла.

Система оценки

Двойные ворота:

- Баллы: 7
- Штрафные баллы: 7
- Описание штрафа: проход сквозь красную часть ворот

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Корзины:

- Баллы: 7
- Штрафные баллы: 7
- Описание штрафа: сброс в красную корзину

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Всплыть в обруче:

- Баллы: 7
- Штрафные баллы: 3
- Описание штрафа: Касание обруча

Решение

Для решения данных задач участнику необходимо реализовать:

- регулятор управления курсом и глубиной (подойдет релейный или простой пропорциональный);
- алгоритмы поиска полосок, определения цвета ворот, определения цвета корзин и поиск треугольника (подойдет как встроенные в murAPI, так и простая бинаризация);
- алгоритм определения наличия полоски в кадре;
- алгоритм определения угла полоски (подойдет встроенный в OpenCV или murAPI).

Начнем с реализации регуляторов глубины и курса.

```
void yaw_and_depth_regulator(float yaw, float depth, int power)
{
    constexpr float k_yaw = 1.3f;
    constexpr float k_depth = 6.0f;

    float yaw_diff = mur.getYaw() - yaw;
    if (yaw_diff < 0.0f) {
        yaw_diff += 360.0f;
    }

    if (yaw_diff > 180.0f) {
        yaw_diff -= 360.0f;
    }

    yaw_diff *= k_yaw;

    float depth_diff = mur.getInputAOne() - depth;
    depth_diff *= k_depth;

    mur.setPorts(-power + yaw_diff, -power - yaw_diff, -depth_diff, 0);
}
```

Наши регуляторы готовы.

Теперь перейдем к поиску линии и треугольника.

Для решения данной задачи мы воспользуемся встроенными в murAPI функциями поиска прямоугольников и линий. Однако, нам необходимо проверять наличие полоски в кадре и количество полосок.

С помощью данной функции мы будем определять наличие или отсутствии оранжевых полосок в кадре и их количество.

```
int is_line(cv::Mat image)
{
```

```
    int line_count = 0;
    cv::Mat hsv_image;
    // BGR -> HSV
    cv::cvtColor(image, hsv_image, CV_BGR2HSV);

    // Бинаризуем изображение по нижней и верхней границе красного
    cv::Mat lower_red_hue_range;
    cv::Mat upper_red_hue_range;
    cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
        cv::Scalar(10, 255, 255), lower_red_hue_range);
    cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
        cv::Scalar(179, 255, 255), upper_red_hue_range);
    cv::Mat red_hue_image;
    cv::addWeighted(lower_red_hue_range, 1.0,
        upper_red_hue_range, 1.0, 0.0, red_hue_image);

    std::vector<std::vector<cv::Point>> contours;

    // Ищим контуры на бинаризованном изображениее
    cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
            continue;
        }
        line_count++;
    }
    return line_count;
}
```

Алгоритм определения наличия линий готов. Для распознавания линий их углов и треугольника будем использовать встроенные в murAPI функции.

Теперь перейдем к распознаванию зеленого цвета.

Данный детектор нам необходим для определения зеленого сектора ворот и зеленой корзины.

```
Object detect_green_object(cv::Mat img)
{
    int hmin = 50, hmax = 75;
    int smin = 105, smax = 255;
    int vmin = 0, vmax = 255;

    cv::Scalar lower(hmin, smin, vmin);
    cv::Scalar upper(hmax, smax, vmax);

    cv::Mat hsv;
    cv::cvtColor(img, hsv, CV_BGR2HSV);
    cv::inRange(hsv, lower, upper, hsv);

    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
    Object object_to_ret;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 5) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {
```

```
            continue;
        }
        cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
        object_to_ret.x = (int)b_ellipse.center.x;
        object_to_ret.y = (int)b_ellipse.center.y;
        object_to_ret.angle = b_ellipse.angle;
        object_to_ret.type = Object::RECTANGLE;
        return object_to_ret;
    }
    return object_to_ret;
}
```

Алгоритм распознавания зеленого готов, теперь добавим алгоритм движения к центру зеленой части ворот и корзины.

Для ворот нам достаточно смещаться влево или вправо относительно координаты X на кадре:

```
    bool move_to_v_center(int x_val)
    {
        constexpr auto center_v = 320 / 2;
        auto center_diff = x_val - center_v;

        if (std::abs(center_diff) < 25) {
            mur.setPortD(0);
            return true;
        }

        if (center_diff < 0) {
            mur.setPortD(-15);
        }

        if (center_diff > 0) {
            mur.setPortD(15);
        }
        return false;
    }
```

Для корзины нам необходимо смещаться по оси X и Y:

```
bool move_to_hv_center(int x, int y)
{
    constexpr auto center_v = 320 / 2;
    constexpr auto center_h = 240 / 2;

    auto center_x_diff = x - center_v;
    auto center_y_diff = y - center_h;
    if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
        mur.setPortD(0);
        return true;
    }

    if (center_x_diff < 0) {
        mur.setPortD(-15);
    }

    if (center_x_diff > 0) {
        mur.setPortD(15);
```

```
    }

    if (center_y_diff < 0) {
        mur.setPortA(-15);
        mur.setPortB(-15);
    }

    if (center_y_diff > 0) {
        mur.setPortA(15);
        mur.setPortB(15);
    }

    return false;
}
```

У нас готовы алгоритмы распознавания зеленого и движения к центру зеленых областей.

Пример программы-решения

Ниже представлено решение на языке C++

```
#include "murAPI.hpp"

Object detect_green_object(cv::Mat img)
{
    int hmin = 50, hmax = 75;
    int smin = 105, smax = 255;
    int vmin = 0, vmax = 255;

    cv::Scalar lower(hmin, smin, vmin);
    cv::Scalar upper(hmax, smax, vmax);

    cv::Mat hsv;
    cv::cvtColor(img, hsv, CV_BGR2HSV);
    cv::inRange(hsv, lower, upper, hsv);

    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
    Object object_to_ret;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 5) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {
            continue;
        }
        cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
        object_to_ret.x = (int)b_ellipse.center.x;
        object_to_ret.y = (int)b_ellipse.center.y;
        object_to_ret.angle = b_ellipse.angle;
        object_to_ret.type = Object::RECTANGLE;
        return object_to_ret;
    }
    return object_to_ret;
}

```

```
37  int is_line(cv::Mat image)
38  {
39      int line_count = 0;
40      cv::Mat hsv_image;
41      // BGR -> HSV
42      cv::cvtColor(image, hsv_image, CV_BGR2HSV);
43
44      // Бинаризуем изображение по нижней и верхней границе красного
45      cv::Mat lower_red_hue_range;
46      cv::Mat upper_red_hue_range;
47      cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
48          cv::Scalar(10, 255, 255), lower_red_hue_range);
49      cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
50          cv::Scalar(179, 255, 255), upper_red_hue_range);
51      cv::Mat red_hue_image;
52      cv::addWeighted(lower_red_hue_range, 1.0,
53          upper_red_hue_range, 1.0, 0.0, red_hue_image);
54
55      std::vector<std::vector<cv::Point>> contours;
56
57      // Ищим контуры на бинаризованном изображениее
58      cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
59
60      for (std::size_t i = 0; i < contours.size(); i++) {
61          if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
62              continue;
63          }
64          line_count++;
65      }
66      return line_count;
67  }
68
69  float yaw_from_angle(float angle)
70  {
71      float angle_new = angle;
72
73      if (angle_new > 90.0) {
74          angle_new = -1 * (180.0 - angle_new);
75      }
76
77      return angle_new;
78  }
79
80  void yaw_and_depth_regulator(float yaw, float depth, int power)
81  {
82      constexpr float k_yaw = 1.3;
83      constexpr float k_depth = 6;
84
85      float yaw_diff = mur.getYaw() - yaw;
86      if (yaw_diff < 0.0f) {
87          yaw_diff += 360.0f;
88      }
89
90      if (yaw_diff > 180.0f) {
91          yaw_diff -= 360.0f;
92      }
93
94      yaw_diff *= k_yaw;
95
96      float depth_diff = mur.getInputAOne() - depth;
```

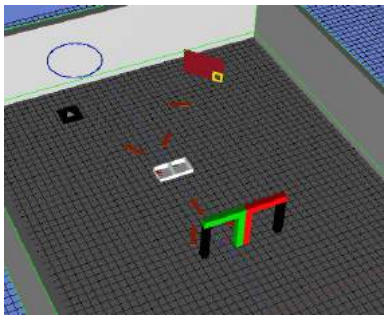
```
97      depth_diff *= k_depth;
98
99      mur.setPortA(-power + yaw_diff);
100     mur.setPortB(-power - yaw_diff);
101     mur.setPortC(-depth_diff);
102 }
103
104 bool move_to_v_center(int x_val)
105 {
106     constexpr auto center_v = 320 / 2;
107     auto center_diff = x_val - center_v;
108
109     if (std::abs(center_diff) < 45) {
110         mur.setPortD(0);
111         return true;
112     }
113
114     if (center_diff < 0) {
115         mur.setPortD(-15);
116     }
117
118     if (center_diff > 0) {
119         mur.setPortD(15);
120     }
121     return false;
122 }
123
124 bool move_to_hv_center(int x, int y)
125 {
126     constexpr auto center_v = 320 / 2;
127     constexpr auto center_h = 240 / 2;
128
129     auto center_x_diff = x - center_v;
130     auto center_y_diff = y - center_h;
131     if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
132         mur.setPortD(0);
133         return true;
134     }
135
136     if (center_x_diff < 0) {
137         mur.setPortD(-15);
138     }
139
140     if (center_x_diff > 0) {
141         mur.setPortD(15);
142     }
143
144     if (center_y_diff < 0) {
145         mur.setPortA(-15);
146         mur.setPortB(-15);
147     }
148
149     if (center_y_diff > 0) {
150         mur.setPortA(15);
151         mur.setPortB(15);
152     }
153
154     return false;
155 }
156
```

```
157 int main()
158 {
159     float yaw = 0.0f;
160     int power = 10;
161     float depth = 70.0f;
162 
163     mur.addDetectorToList(Object::RECTANGLE, 0);
164     mur.addDetectorToList(Object::TRIANGLE, 0);
165 
166     auto line = [&yaw, &depth, &power]() {
167         bool is_line_detected = false;
168 
169         while (!is_line_detected) {
170             yaw_and_depth_regulator(yaw, depth, power);
171             for (const auto& obj : mur.getDetectedObjectsList(0)) {
172                 if (obj.type == Object::RECTANGLE) {
173                     yaw += yaw_from_angle(obj.angle);
174                     is_line_detected = true;
175                     break;
176                 }
177             }
178         }
179     };
180 
181     auto gate_green = [&yaw, &depth, &power]() {
182         bool is_gate_centrated = false;
183         while (!is_gate_centrated) {
184             auto image = mur.getCameraTwoFrame();
185             auto result = detect_green_object(image);
186             if (move_to_v_center(result.x)) {
187                 is_gate_centrated = true;
188             }
189             yaw_and_depth_regulator(0.0, 80, 0);
190         }
191     };
192 
193     auto bin_green = [&yaw, &depth, &power]() {
194         bool is_bin_centrated = false;
195         while (!is_bin_centrated) {
196             auto image = mur.getCameraOneFrame();
197             auto result = detect_green_object(image);
198             yaw_and_depth_regulator(yaw, depth, power);
199             if (result.type == Object::NONE) {
200                 continue;
201             }
202             if (move_to_hv_center(result.x, result.y)) {
203                 mur.drop();
204                 is_bin_centrated = true;
205             }
206         }
207     };
208 
209     auto skip_lines = [&yaw, &depth, &power](int line_count) {
210         while (is_line(mur.getCameraOneFrame()) != line_count) {
211             yaw_and_depth_regulator(yaw, depth, power);
212         }
213     };
214 
215     auto triangle = [&yaw, &depth, &power]() {
216         bool is_triangle_detected = false;
```

```
217
218         while (!is_triangle_detected) {
219             yaw_and_depth_regulator(yaw, depth, power);
220             for (const auto& obj : mur.getDetectedObjectsList(0)) {
221                 if (obj.type == Object::TRIANGLE) {
222                     is_triangle_detected = true;
223                     break;
224                 }
225             }
226         }
227         if (is_triangle_detected) {
228             mur.setPorts(0, 0, -55, 0);
229             sleepFor(5000);
230         }
231     };
232
233     gate_green();
234
235     line();
236     depth = 40;
237
238     skip_lines(2);
239     skip_lines(1);
240
241     line();
242
243     bin_green();
244
245     skip_lines(2);
246     skip_lines(1);
247
248     line();
249
250     skip_lines(0);
251
252     triangle();
253
254     return 0;
255 }
```

Задача 3.1.3. (30 баллов)

Конфигурация третьей сцены

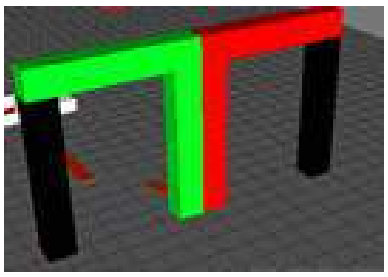


Для успешного прохождения задания аппарат в автономном режиме должен:

- Пройти двойные ворота – 7 (возможен штраф в 7 баллов).

- Пройти по полоске – 3 балла.
- Пройти по полоске – 3 балла.
- Сбросить маркер в корзину – 7 (возможен штраф в 7 баллов).
- Пройти по полоске – 3 балла.
- Выстрелить в мишень – 17 баллов.
- Пройти по полоске – 3 балла.
- Всплыть в обруче – 7 баллов (возможен штраф в 3 балла).

Двойные ворота



Пройти в зеленую часть ворот. Ворота разбиты на два сектора – красный и зеленый. Верхний край ворот совпадает с поверхностью воды. Расстояние от старта до ворот варьируется в зависимости от тестовой сцены. За прохождение красной части ворот начисляются 7 штрафных баллов, за прохождение зеленой части ворот начисляется 7 баллов.

- Взаимное расположение красной и зеленой части ворот может меняться.

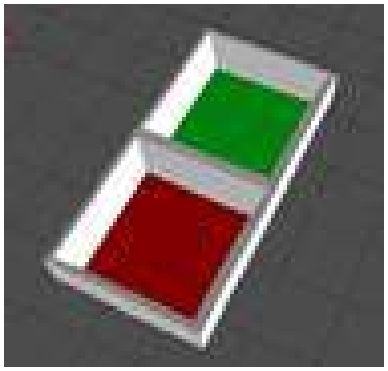
Полоска

Пройти по полоске. Цвет полоски оранжевый. Полоска служит для определения направление на следующие задание. Направление полоски может варьироваться от -80 градусов до 80 градусов относительно начального курса аппарата. Расстояние от задания и направление полоски варьируется в зависимости от сцены. Полоска расположена на дне бассейна.

- После ворот расположено две полоски, после красной и после зеленой части ворот, баллы начисляются только за одну полоску.

- После корзин расположено две полоски, после красной и после зеленой корзины. Баллы начисляются только за прохождение одной полоски.

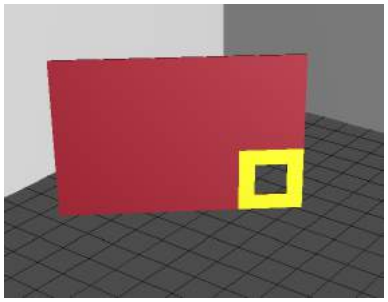
Корзины



Сбросить маркеры в зеленую корзину. Необходимо сбросить маркер синего цвета в зеленую корзину. Корзина имеет белые бортики и цветное дно. Корзины две – одна зеленого цвета, а вторая красного. Корзины расположены недалеко друг от друга. За сброс маркера в зеленую корзину начисляется 7 баллов. За сброс маркеров в красную корзину начисляется 7 штрафных баллов.

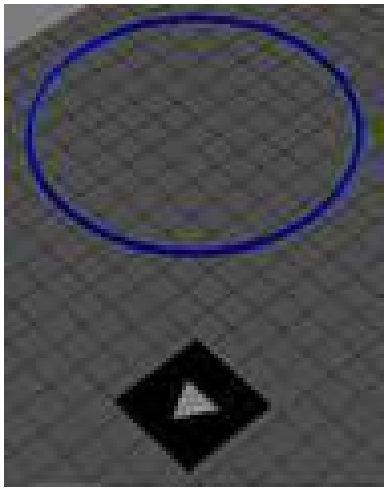
- Взаимное расположение корзин может меняться.
- Линия перед корзинами указывает на «центр» между корзинами.
- Бортики корзин соприкасаются.

Мишень



Выстрелить в мишень. Необходимо выстрелить маркером в отверстие мишени. Цвет мишени красный. Отверстие квадратной формы имеет желтое обрамление. Отверстие может находится в любом из четырех углов красного прямоугольника. За попадание в отверстие начисляется 17 баллов. Штрафные баллы в этом задании не начисляются. Перед мишенью находится полоска, указывающая на треугольник и обруч. Позиция треугольника и обруча может взаимно меняться с позицией мишени. Высота мишени относительно дна может меняться.

Тругольник и обруч



Всплыть над треугольником в обруче. Центр треугольника совпадает с центром обруча. Треугольник размещен в квадрате, цвет треугольника белый, квадрата черный. Цвет обруча синий. За всплытие в обруче начисляется 7 баллов. За касание обруча начисляется 3 штрафных балла. Позиция треугольника и обруча может взаимно меняться с позицией мишени.

Система оценки

Двойные ворота:

- Баллы: 7
- Штрафные баллы: 7
- Описание штрафа: проход сквозь красную часть ворот

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Корзины:

- Баллы: 7
- Штрафные баллы: 7
- Описание штрафа: сброс в красную корзину

Полоска:

- Баллы: 3
- Штрафные баллы: -

- Описание штрафа: -

Мишень:

- Баллы: 17
- Штрафные баллы: -
- Описание штрафа: -

Полоска:

- Баллы: 3
- Штрафные баллы: -
- Описание штрафа: -

Всплыть в обруче:

- Баллы: 7
- Штрафные баллы: 3
- Описание штрафа: Касание обруча

Решение

Для решения данных задач участнику необходимо реализовать:

- регулятор управления курсом и глубиной (подойдет релейный или простой пропорциональный);
- алгоритмы поиска полосок, определения цвета ворот, определения цвета корзин, поиск мишени и поиск треугольника (подойдет как встроенные в murAPI, так и простая бинаризация);
- алгоритм определения наличия полоски в кадре;
- алгоритм определения угла полоски (подойдет встроенный в OpenCV или murAPI).

Начнем с реализации регуляторов глубины и курса.

```
void cw(bool move = false)
{
    mur.setPortA(-30);
    mur.setPortB(0);

    if (move) {
        mur.setPortB(30);
    }
}

void ccw(bool move = false)
{
    mur.setPortA(0);
    mur.setPortB(-30);

    if (move) {
        mur.setPortA(30);
    }
}

```

```
void yaw_and_depth_regulator(float yaw, float depth, int power)
{
    auto c_yaw = mur.getYaw();
    constexpr float k_depth = 6;

    auto d = std::fmod((yaw - c_yaw + 540.0f), 360.0f) - 180.0f;
    bool move = false;
    if (power == 0) {
        move = true;
    }

    if (d > 0) {
        cw(move);
    } else {
        ccw(move);
    }

    float depth_diff = mur.getInputAOne() - depth;
    depth_diff *= k_depth;
    mur.setPortC(-depth_diff);
}
```

Наши регуляторы готовы.

Теперь перейдем к поиску линии и треугольника.

Для решения данной задачи мы воспользуемся встроенными в murAPI функциями поиска прямоугольников и линий. Однако, нам необходимо проверять наличие полоски в кадре и количество полосок.

С помощью данной функции мы будем определять наличие или отсутствие оранжевых полосок в кадре и их количество.

```
int is_line(cv::Mat image)
{
    int line_count = 0;
    cv::Mat hsv_image;
    // BGR -> HSV
    cv::cvtColor(image, hsv_image, CV_BGR2HSV);

    // Бинаризуем изображение по нижней и верхней границе красного
    cv::Mat lower_red_hue_range;
    cv::Mat upper_red_hue_range;
    cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
        cv::Scalar(10, 255, 255), lower_red_hue_range);
    cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
        cv::Scalar(179, 255, 255), upper_red_hue_range);
    cv::Mat red_hue_image;
    cv::addWeighted(lower_red_hue_range, 1.0,
        upper_red_hue_range, 1.0, 0.0, red_hue_image);

    std::vector<std::vector<cv::Point>> contours;

    // Ищим контуры на бинаризованном изображениее
    cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
            continue;
        }
```

```
28          line_count++;
29      }
30      return line_count;
31  }
```

Алгоритм определения наличия линий готов. Для распознавания линий их углов и треугольника будем использовать встроенные в murAPI функции.

Теперь перейдем к распознаванию зеленого цвета.

Данный детектор нам необходим для определения зеленого сектора ворот и зеленой корзины.

```
1   Object detect_green_object(cv::Mat img)
2   {
3       int hmin = 50, hmax = 75;
4       int smin = 105, smax = 255;
5       int vmin = 0, vmax = 255;
6
7       cv::Scalar lower(hmin, smin, vmin);
8       cv::Scalar upper(hmax, smax, vmax);
9
10      cv::Mat hsv;
11      cv::cvtColor(img, hsv, CV_BGR2HSV);
12      cv::inRange(hsv, lower, upper, hsv);
13
14      std::vector<std::vector<cv::Point>> contours;
15      cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
16      Object object_to_ret;
17
18      for (std::size_t i = 0; i < contours.size(); i++) {
19          if (contours.at(i).size() < 5) {
20              continue;
21          }
22          if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {
23              continue;
24          }
25          cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
26          object_to_ret.x = (int)b_ellipse.center.x;
27          object_to_ret.y = (int)b_ellipse.center.y;
28          object_to_ret.angle = b_ellipse.angle;
29          object_to_ret.type = Object::RECTANGLE;
30          return object_to_ret;
31      }
32      return object_to_ret;
33  }
```

Алгоритм распознавания зеленого готов, теперь добавим алгоритм движения к центру зеленой части ворот и корзины.

Для ворот нам достаточно смещаться влево или вправо относительно координаты X на кадре:

```
1   bool move_to_v_center(int x_val)
2   {
3       constexpr auto center_v = 320 / 2;
4       auto center_diff = x_val - center_v;
5
```

```
    if (std::abs(center_diff) < 25) {
        mur.setPortD(0);
        return true;
    }

    if (center_diff < 0) {
        mur.setPortD(-15);
    }

    if (center_diff > 0) {
        mur.setPortD(15);
    }
    return false;
}
```

Для корзины нам необходимо смещаться по оси X и Y:

```
bool move_to_hv_center(int x, int y)
{
    constexpr auto center_v = 320 / 2;
    constexpr auto center_h = 240 / 2;

    auto center_x_diff = x - center_v;
    auto center_y_diff = y - center_h;
    if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
        mur.setPortD(0);
        return true;
    }

    if (center_x_diff < 0) {
        mur.setPortD(-15);
    }

    if (center_x_diff > 0) {
        mur.setPortD(15);
    }

    if (center_y_diff < 0) {
        mur.setPortA(-15);
        mur.setPortB(-15);
    }

    if (center_y_diff > 0) {
        mur.setPortA(15);
        mur.setPortB(15);
    }

    return false;
}
```

У нас готовы алгоритмы распознавания зеленого и движения к центру зеленых областей.

Теперь перейдем к поиску мишени и алгоритму движения к ее центру.

Искать мишень мы будем при помощи бинаризации по желтому цвету.

```
Object detect_yellow_object(cv::Mat img)
{
```

```
    int hmin = 20, hmax = 30;
    int smin = 100, smax = 255;
    int vmin = 100, vmax = 255;

    cv::Scalar lower(hmin, smin, vmin);
    cv::Scalar upper(hmax, smax, vmax);

    cv::Mat hsv;
    cv::cvtColor(img, hsv, CV_BGR2HSV);
    cv::inRange(hsv, lower, upper, hsv);
    cv::imshow("W", hsv);
    cv::waitKey(1);
    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
    Object object_to_ret;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 5) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 150.0) {
            continue;
        }
        cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
        object_to_ret.x = (int)b_ellipse.center.x;
        object_to_ret.y = (int)b_ellipse.center.y;
        object_to_ret.angle = b_ellipse.angle;
        object_to_ret.type = Object::RECTANGLE;
        object_to_ret.size = cv::contourArea(contours.at(i));

        return object_to_ret;
    }
    return object_to_ret;
}
```

Алгоритм поиска желтого готов, теперь перейдем к алгоритму движения к центру мишени:

```
bool move_to_hv_center_front(int x, int y)
{
    constexpr auto center_v = 320 / 2;
    constexpr auto center_h = 240 / 2;

    mur.setPortA(0);
    mur.setPortB(0);

    auto center_x_diff = x - center_v;
    auto center_y_diff = y - center_h;
    if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
        mur.setPortD(0);
        return true;
    }

    if (center_x_diff < 0) {
        mur.setPortD(-15);
    }

    if (center_x_diff > 0) {
        mur.setPortD(15);
```

```
    }

    if (center_y_diff < 0) {
        mur.setPortC(-15);
    }

    if (center_y_diff > 0) {
        mur.setPortC(15);
    }

    return false;
}
```

Пример программы-решения

Ниже представлено решение на языке C++

```
#include "murAPI.hpp"

Object detect_yellow_object(cv::Mat img)
{
    int hmin = 20, hmax = 30;
    int smin = 100, smax = 255;
    int vmin = 100, vmax = 255;

    cv::Scalar lower(hmin, smin, vmin);
    cv::Scalar upper(hmax, smax, vmax);

    cv::Mat hsv;
    cv::cvtColor(img, hsv, CV_BGR2HSV);
    cv::inRange(hsv, lower, upper, hsv);
    cv::imshow("W", hsv);
    cv::waitKey(1);
    std::vector<std::vector<cv::Point>> contours;
    cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
    Object object_to_ret;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 5) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 150.0) {
            continue;
        }
        cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
        object_to_ret.x = (int)b_ellipse.center.x;
        object_to_ret.y = (int)b_ellipse.center.y;
        object_to_ret.angle = b_ellipse.angle;
        object_to_ret.type = Object::RECTANGLE;
        object_to_ret.size = cv::contourArea(contours.at(i));

        return object_to_ret;
    }
    return object_to_ret;
}

Object detect_green_object(cv::Mat img)
{
    int hmin = 50, hmax = 75;
```

```
43      int smin = 105, smax = 255;
44      int vmin = 0, vmax = 255;
45
46      cv::Scalar lower(hmin, smin, vmin);
47      cv::Scalar upper(hmax, smax, vmax);
48
49      cv::Mat hsv;
50      cv::cvtColor(img, hsv, CV_BGR2HSV);
51      cv::inRange(hsv, lower, upper, hsv);
52
53      std::vector<std::vector<cv::Point>> contours;
54      cv::findContours(hsv, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
55      Object object_to_ret;
56
57      for (std::size_t i = 0; i < contours.size(); i++) {
58          if (contours.at(i).size() < 5) {
59              continue;
60          }
61          if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {
62              continue;
63          }
64          cv::RotatedRect b_ellipse = cv::fitEllipse(contours.at(i));
65          object_to_ret.x = (int)b_ellipse.center.x;
66          object_to_ret.y = (int)b_ellipse.center.y;
67          object_to_ret.angle = b_ellipse.angle;
68          object_to_ret.type = Object::RECTANGLE;
69          return object_to_ret;
70      }
71      return object_to_ret;
72  }
73
74  int is_line(cv::Mat image)
75  {
76      int line_count = 0;
77      cv::Mat hsv_image;
78      // BGR -> HSV
79      cv::cvtColor(image, hsv_image, CV_BGR2HSV);
80
81      // Бинаризуем изображение по нижней и верхней границе красного
82      cv::Mat lower_red_hue_range;
83      cv::Mat upper_red_hue_range;
84      cv::inRange(hsv_image, cv::Scalar(0, 100, 100),
85          cv::Scalar(10, 255, 255), lower_red_hue_range);
86      cv::inRange(hsv_image, cv::Scalar(160, 100, 100),
87          cv::Scalar(179, 255, 255), upper_red_hue_range);
88      cv::Mat red_hue_image;
89      cv::addWeighted(lower_red_hue_range, 1.0,
90          upper_red_hue_range, 1.0, 0.0, red_hue_image);
91
92      std::vector<std::vector<cv::Point>> contours;
93
94      // Ищим контуры на бинаризованном изображениее
95      cv::findContours(red_hue_image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);
96
97      for (std::size_t i = 0; i < contours.size(); i++) {
98          if (std::fabs(cv::contourArea(contours.at(i))) < 800.0) {
99              continue;
100         }
101         line_count++;
102     }
```

```
103     return line_count;
104 }
105 
106 float yaw_from_angle(float angle)
107 {
108     float angle_new = angle;
109 
110     if (angle_new > 90.0) {
111         angle_new = -1 * (180.0 - angle_new);
112     }
113 
114     return angle_new;
115 }
116 
117 void cw(bool move = false)
118 {
119     mur.setPortA(-30);
120     mur.setPortB(0);
121 
122     if (move) {
123         mur.setPortB(30);
124     }
125 }
126 
127 void ccw(bool move = false)
128 {
129     mur.setPortA(0);
130     mur.setPortB(-30);
131 
132     if (move) {
133         mur.setPortA(30);
134     }
135 }
136 void yaw_and_depth_regulator(float yaw, float depth, int power)
137 {
138     auto c_yaw = mur.getYaw();
139     constexpr float k_depth = 6;
140 
141     auto d = std::fmod((yaw - c_yaw + 540.0f), 360.0f) - 180.0f;
142     bool move = false;
143     if (power == 0) {
144         move = true;
145     }
146 
147     if (d > 0) {
148         cw(move);
149     } else {
150         ccw(move);
151     }
152 
153     float depth_diff = mur.getInputAOne() - depth;
154     depth_diff *= k_depth;
155     mur.setPortC(-depth_diff);
156 }
157 
158 bool move_to_v_center(int x_val)
159 {
160     constexpr auto center_v = 320 / 2;
161     auto center_diff = x_val - center_v;
162 
```

```
163     if (std::abs(center_diff) < 25) {
164         mur.setPortD(0);
165         return true;
166     }
167 
168     if (center_diff < 0) {
169         mur.setPortD(-15);
170     }
171 
172     if (center_diff > 0) {
173         mur.setPortD(15);
174     }
175     return false;
176 }
177 
178 float yaw_to_180(float yaw)
179 {
180     if (yaw < 0.0f) {
181         yaw += 360.0f;
182     }
183 
184     if (yaw > 180.0f) {
185         yaw -= 360.0f;
186     }
187     return yaw;
188 }
189 
190 bool move_to_hv_center(int x, int y)
191 {
192     constexpr auto center_v = 320 / 2;
193     constexpr auto center_h = 240 / 2;
194 
195     auto center_x_diff = x - center_v;
196     auto center_y_diff = y - center_h;
197     if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
198         mur.setPortD(0);
199         return true;
200     }
201 
202     if (center_x_diff < 0) {
203         mur.setPortD(-15);
204     }
205 
206     if (center_x_diff > 0) {
207         mur.setPortD(15);
208     }
209 
210     if (center_y_diff < 0) {
211         mur.setPortA(-15);
212         mur.setPortB(-15);
213     }
214 
215     if (center_y_diff > 0) {
216         mur.setPortA(15);
217         mur.setPortB(15);
218     }
219 
220     return false;
221 }
222 
```

```
223 bool move_to_hv_center_front(int x, int y)
224 {
225 
226     constexpr auto center_v = 320 / 2;
227     constexpr auto center_h = 240 / 2;
228 
229     mur.setPortA(0);
230     mur.setPortB(0);
231 
232 
233     auto center_x_diff = x - center_v;
234     auto center_y_diff = y - center_h;
235     if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
236         mur.setPortD(0);
237         return true;
238     }
239 
240     if (center_x_diff < 0) {
241         mur.setPortD(-15);
242     }
243 
244     if (center_x_diff > 0) {
245         mur.setPortD(15);
246     }
247 
248     if (center_y_diff < 0) {
249         mur.setPortC(-15);
250     }
251 
252     if (center_y_diff > 0) {
253         mur.setPortC(15);
254     }
255 
256     return false;
257 }
258 
259 int main()
260 {
261     float yaw = 0.0f;
262     int power = 10;
263     float depth = 70.0f;
264 
265     mur.addDetectorToList(Object::RECTANGLE, 0);
266     mur.addDetectorToList(Object::TRIANGLE, 0);
267 
268     auto line = [&yaw, &depth, &power]() {
269         bool is_line_detected = false;
270 
271         while (!is_line_detected) {
272             yaw_and_depth_regulator(yaw, depth, power);
273             for (const auto& obj : mur.getDetectedObjectsList(0)) {
274                 if (obj.type == Object::RECTANGLE) {
275                     yaw += yaw_from_angle(obj.angle);
276                     is_line_detected = true;
277                     break;
278                 }
279             }
280         }
281     };
282 
```

```
283     auto gate_green = [&yaw, &depth, &power]() {
284         bool is_gate_centrated = false;
285         while (!is_gate_centrated) {
286             auto image = mur.getCameraTwoFrame();
287             auto result = detect_green_object(image);
288             if (move_to_v_center(result.x)) {
289                 is_gate_centrated = true;
290             }
291             yaw_and_depth_regulator(0.0, 80, 0);
292         }
293     };
294 
295     auto bin_green = [&yaw, &depth, &power]() {
296         bool is_bin_centrated = false;
297         while (!is_bin_centrated) {
298             auto image = mur.getCameraOneFrame();
299             auto result = detect_green_object(image);
300             yaw_and_depth_regulator(yaw, depth, power);
301             if (result.type == Object::NONE) {
302                 continue;
303             }
304             if (move_to_hv_center(result.x, result.y)) {
305                 mur.drop();
306                 is_bin_centrated = true;
307             }
308         }
309     };
310 
311     auto skip_lines = [&yaw, &depth, &power](int line_count) {
312         while (is_line(mur.getCameraOneFrame()) != line_count) {
313             yaw_and_depth_regulator(yaw, depth, power);
314         }
315     };
316 
317     auto triangle = [&yaw, &depth, &power]() {
318         bool is_triangle_detected = false;
319 
320         while (!is_triangle_detected) {
321             yaw_and_depth_regulator(yaw, depth, power);
322             for (auto&& obj : mur.getDetectedObjectsList(0)) {
323                 if (obj.type == Object::TRIANGLE) {
324                     is_triangle_detected = true;
325                     break;
326                 }
327             }
328         }
329         if (is_triangle_detected) {
330             mur.setPorts(0, 0, -55, 0);
331             sleepFor(5000);
332         }
333     };
334 
335     auto find_yellow = [&yaw, &depth, &power]() {
336         bool is_yellow_found = false;
337         yaw_and_depth_regulator(yaw, depth, 0);
338         while (!is_yellow_found) {
339             auto image = mur.getCameraTwoFrame();
340             auto result = detect_yellow_object(image);
341             if (result.type == Object::NONE) {
342                 yaw += 5.0f;
```

```
343
344                 if (yaw > 360.0f) {
345                     yaw -= 360;
346                 }
347
348                 while (std::fabs(yaw - mur.getYaw()) > 5) {
349                     yaw_and_depth_regulator(yaw, depth, 0);
350                 }
351
352                 continue;
353             } else if (result.type == Object::RECTANGLE) {
354                 is_yellow_found = true;
355                 break;
356             }
357             yaw_and_depth_regulator(yaw, depth, 0);
358         }
359     };
360
361     auto move_to_yellow = [&yaw, &depth, &power]() {
362         bool is_yellow_found = false;
363         while (!is_yellow_found) {
364             auto image = mur.getCameraTwoFrame();
365             auto result = detect_yellow_object(image);
366             if (result.type == Object::RECTANGLE) {
367                 if (result.size < 6500) {
368                     yaw_and_depth_regulator(yaw, depth, 10);
369                 } else {
370                     yaw_and_depth_regulator(yaw, depth, 0);
371                     break;
372                 }
373             }
374         }
375     };
376
377     auto centrate_on_yellow = [&yaw, &depth, &power]() {
378         bool is_yellow_found = false;
379         yaw_and_depth_regulator(yaw, depth, 0);
380         while (!is_yellow_found) {
381             auto image = mur.getCameraTwoFrame();
382             auto result = detect_yellow_object(image);
383             if (result.type == Object::NONE) {
384                 continue;
385             } else if (result.type == Object::RECTANGLE) {
386                 if (move_to_hv_center_front(result.x, result.y)) {
387                     depth = mur.getInputAOne();
388                     yaw = mur.getYaw();
389                     is_yellow_found = true;
390                     break;
391                 }
392             }
393         }
394     };
395
396     auto shoot_and_rotate = [&yaw, &depth, &power]() {
397         mur.shoot();
398         depth = 70;
399         yaw += 180;
400         if (yaw > 360) {
401             yaw -= 360;
402         }
```

```
403       };
404     gate_green();
405
406       line();
407       depth = 40;
408
409       skip_lines(2);
410       skip_lines(1);
411
412       line();
413       depth = 70;
414       bin_green();
415       find_yellow();
416       centrate_on_yellow();
417       move_to_yellow();
418       centrate_on_yellow();
419       shoot_and_rotate();
420       skip_lines(1);
421       line();
422       skip_lines(0);
423       triangle();
424
425       return 0;
426   }
```