Командный тур

5.1. Задачи

Финальный командный этап включает в себя три задачи:

- разработка устройства для запуска торпед, совместимого с MUR – *30 баллов*
- решение задачи в симуляторе MUR IDE – *30 баллов*
- выполнение задач в бассейне – *40 баллов*

Все задачи выдаются одновременно в первый час соревнований. Проверка решений производится на протяжении всех дней заключительного этапа олимпиады. Победитель будет определен по сумме баллов за три задачи.

Задача 5.1.1. (30 баллов)

Каждая команда Олимпиады НТИ по профилю Водные робототехнические системы будет иметь в своем распоряжении набор элементов, из которых в течении 4 дней необходимо будет разработать и изготовить устройство для запуска торпед, совместимое с MUR.

Требования к устройству:

- Размеры устройства: не регламентируются
- Размеры торпеды (диаметр $\times$ длина), мм: не менее 10×50
- Вес устройства: не регламентируется
- Плавучесть: нейтральная (± 10 грамм)
- Количество выстрелов без перезарядки: не менее 1

Задача предусматривает выполнение командами следующих подзадач:

- Разработка конструкции
- Электромонтаж
- Программирование устройства, его тестирование и отладка

Список комплектующих и материалов, которые будут выданы каждой команде, приведен в таблице 1.

Таблица 1. Список комплектующих и материалов

Наименование	**Кол-во**
Перчатки	1
Холодная сварка	1
Кабель, 50 см	1
Мотор коллекторный	2
Разъем четырехпиновый	1

Команды будут иметь доступ к паяльным станциям, мультиметрам, кусачкам, пинцетам, источникам питаниям, 3D-принтеру и другим электромонтажным и форомообразующим инструментам.

В таблице 2 приведена распиновка разъема четырехконтактного, подключаемого к MUR.

ВНИМАНИЕ: при распайке разъема необходимо проявить максимальное внимание. При неправильной распайке может выйти из строя Модуль бортового компьютера MUR.

Таблица 2. Распиновка четырехконтактного разъема

1	-
2	-
3	Контакт двигателя
4	Контакт двигателя

Информация для программирования устройства.

Конструктор подводного робота MUR на портах Thrust имеет 2 PWM вывода напряжением 12 В, которые используются для управления коллекторными моторами. Управление скважностью PWM происходит с помощью функции из MurAPI – mur.setPortA(B,C,D)(int power), где A, B, C, D – это индекс порта Thrust, а power параметр, принимающий значение от -100 до 100, означающий скважность на одном из выводов PWM.

Примеры:

1. Если вызвать функцию mur.setPortA(0), то это будет означать что на порту Thrust с индексом A будет выставлен уровень PWM в 0, т.е. оба вывода PWM будут держать низкий уровень напряжения и мотор двигаться не будет.
2. Если вызвать функцию mur.setPortB(50), то это будет означать что на порту Thrust с индексом B будет выставлен уровень PWM со скважностью 50% на выводе №1, а №2 будет держать низкий уровень. Последствием вызова этой функции будет, то, что двигатель начнет вращение в определённом направление с 50% скоростью. Для большего понимая, будет считать, что направление движения мотора будет по часовой стрелки. Хотя оно может и отличаться в зависимости от того, как были припаяны контакты на мотор.
3. Если вызвать функцию mur.setPortC(-50), то это будет означать что на порту Thrust с индексом C будет выставлен уровень PWM со скважность 50% на выводе №2, а №1 будет держать низкий уровень. Последствием вызова этой функции будет, то, что двигатель начнет вращение в противоположном направлении (в условиях примера, против часовой стрелки) со скоростью в 50%.

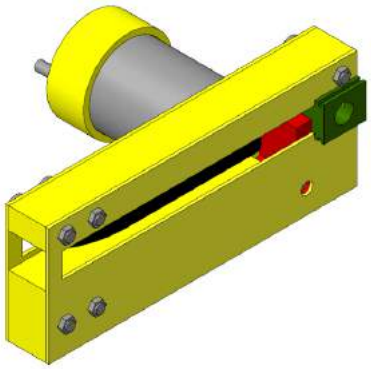
Таким образом, для того чтобы управлять коллекторным двигателем с помощью конструктора MUR, необходимо подключить его в соответствующий порт Thrust A, B, C или D. Далее на этот порт подать необходимую тягу, где знак тяги определяет направление вращения двигателя, а значение определяет силу с которой вращается двигатель, если подать 0 то мотор двигаться не будет, если подать +
- 100 мотор будет вращаться с максимальной скоростью в направлении заданном знаком тяги.

Система оценки

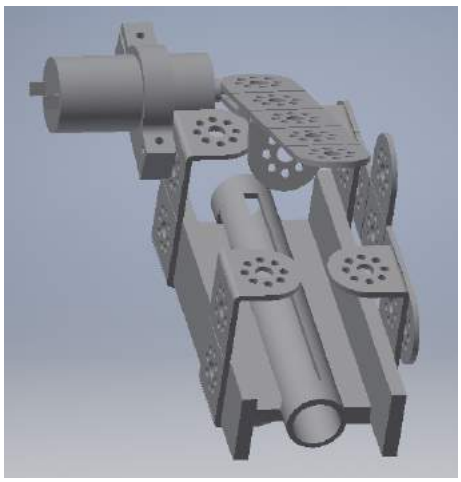
№	Критери	Баллы
	Электроника	
1	Качество пайки разъема (аккуратность, использование термоусадки и др.)	2
2	Качество пайки контактов мотора (аккуратность, использование термоусадки и др.)	2
	Конструирование	
3	Качество изготовления, обработки деталей конструкци	6
4	Наличие и соответствие 3D-модели реальному устройств	4
	Работоспособность	
5	Устройство «выстреливает» торпедой на воздухе минимум на 50 см. При этом устройство установлено и подключено к аппарату, робот стоит на столе или полу, запуск производит конкурсант.	10
6	Надежность (экспериментатор может повторить подряд «выстрел» не менее 3 раз)	6

Решение

Так как данную задачу можно решить различными способами, то в качестве решения мы приведем фотографии и 3D-модели нескольких успешных устройств, удовлетворяющих всем критериям оценки.

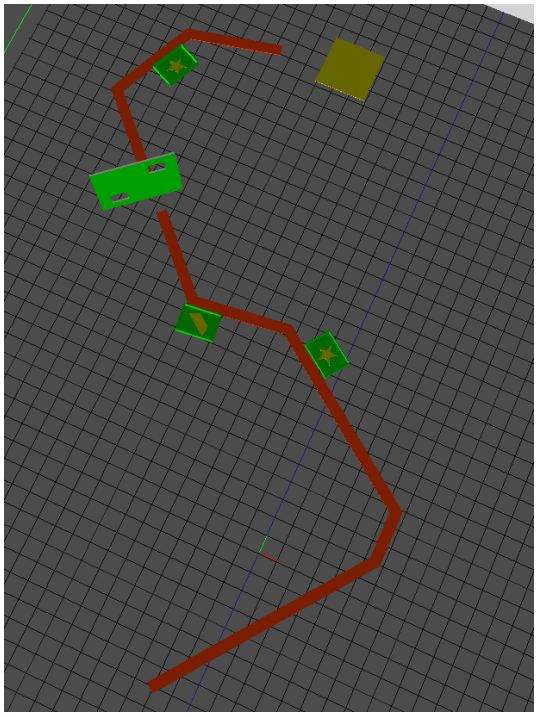


3D-модель и реализация устройства №1



3D-модель и реализация устройства №2

Задача 5.1.2. (30 баллов)



Робот стартует над началом оранжевой линии. Отдельно баллы за прохождение по линии не начисляются. Баллы начисляются за следующие задачи:

- Сбросить два маркера в корзины со звездочкой. Порядок маркеров в корзине может меняться. 5 баллов за маркер. Попадание в корзину с трапецией – 5 штрафных баллов.
- Выстрелить в нижнее отверстие с зеленой стороны мишени (7.5 баллов). Выстрелить в верхнее отверстие мишени с желтой стороны (7.5 баллов).
- Сесть аппаратом на желтый квадрат (5 баллов).

Попытка завершается если:

- Аппарат сел на желтый квадрат.

- Аппарат всплыл.
- Закончилось время.

Время на выполнение задачи: 2 минуты

Максимальное количество баллов: 30

Количество попыток сдачи решений: 3

Система оценки

Команда передают на флешке или отправляют на специальную почту решения в виде проекта MUR IDE. Организаторы проверяют решение на 3 сценах и в течение 30 минут отправляют команде результат тестирования: какие задачи в каждой сцене выполнил робот и сколько баллов команда получила за каждую сцену. Результатом является среднее арифметическое за три сцены. Команда может отправлять решения 3 раза.

В зачет идет лучшее решение команды.

Например, за первое решение команда заработала 20 баллов, за второе решение – 11 баллов, за третье – 13 баллов. В зачет пойдет 20 баллов.

Решение

Даная задача состоит из нескольких подзадач:

1. Пройти вдоль линии;
2. Сбросить маркеры в корзины со звездой;
3. Выстрелить в нижнее отверстие с зеленой стороны мишени;
4. Выстрелить в верхнее отверстие с желтой стороны мишени;
5. Сесть на желтую площадку.

Для решения данных задач участнику необходимо реализовать:

- регулятор управления курсом и глубиной (подойдет релейный или простой пропорциональный);
- алгоритмы поиска линии, определения фигуры в корзинах, поиск мишени.

Начнем с реализации регуляторов глубины и курса.

```
void yd_p_reg(double yaw, double depth, int power) {
  double u = mur.getYaw() - yaw;
  if (u < 0.0) {
    u += 360.0;
  }
  if (u > 180.0) {
    u -= 360.0;
  }
  u *= 1.3;
  double ddepth = mur.getInputAOne() - depth;
  ddepth *= 6;
  mur.setPorts(-power + u, -power - u, -ddepth, 0);
}
```

```
14
15 void yd_p_reg_timed(double yaw, double depth, int power, long long time) {
16   Timer t;
17   t.start();
18   while (t.elapsed() < time) {
19     yd_p_reg(yaw, depth, power);
20   }
21 }
22
23 double from_cv_angle(double angle) {
24   auto new_angle = angle;
25   if (new_angle > 90) {
26     new_angle = (-1.0) * (180.0 - new_angle);
27   }
28   return new_angle;
29 }
30
31 void yd_p_reg_line(double depth, int power) {
32   auto img = mur.getCameraOneFrame();
33   auto obj = detect_line(img);
34   double u = (0.0 - from_cv_angle(obj.angle)) * 1.3;
35   double ddepth = (mur.getInputAOne() - depth) * 6.0;
36   double u_lag = 160.0 - obj.x;
37   mur.setPorts(-power + u, -power - u, -ddepth, -u_lag);
38 }
```

Наши регуляторы готовы.

Перейдем к поиску линии.

Для решения данной задачи мы воспользуемся алгоритмом бинаризации и встроенными в OpenCV алгоритмами определения угла.

```
1  Object detect_line(cv::Mat &image) {
2    static const cv::Scalar lower(0, 100, 60);
3    static const cv::Scalar upper(20, 255, 255);
4    Object to_ret;
5    cv::Mat img = image.clone();
6
7    cv::cvtColor(img, img, CV_BGR2HSV);
8    cv::inRange(img, lower, upper, img);
9    std::vector<std::vector<cv::Point>> contours;
10   cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
11   auto max_area = 0;
12
13   for (std::size_t idx = 0; idx < contours.size(); ++idx) {
14
15     if (contours.at(idx).size() < 5) {
16       continue;
17     }
18
19     std::vector<cv::Point> hull;
20     cv::convexHull(contours.at(idx), hull, true);
21     cv::approxPolyDP(hull, hull, 24, true);
22
23     if (hull.size() < 3UL) {
24       continue;
25     }
26
27     auto ellipse = cv::fitEllipse(contours.at(idx));
```

```
    to_ret.x = ellipse.center.x;
    to_ret.y = ellipse.center.y;
    to_ret.angle = ellipse.angle;
  }
  return to_ret;
}
```

Алгоритм определения линий готов.

Перейдем к распознаванию звезды и трапеции.

Данный детектор нам необходим для определения фигуры в корзинах.

```
Object detect_start(cv::Mat &image) {
  static const cv::Scalar lower(23, 200, 94);
  static const cv::Scalar upper(50, 255, 122);
  Object to_ret;
  cv::Mat img = image.clone();

  cv::cvtColor(img, img, CV_BGR2HSV);
  cv::inRange(img, lower, upper, img);
  cv::imshow("W", img);
  cv::waitKey(1);
  std::vector<std::vector<cv::Point>> contours;
  cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

  auto max_area = 0;

  for (std::size_t idx = 0; idx < contours.size(); ++idx) {

    if (contours.at(idx).size() < 5) {
      continue;
    }

    std::vector<cv::Point> hull;
    cv::convexHull(contours.at(idx), hull, true);
    cv::approxPolyDP(hull, hull, 24, true);

    if (hull.size() > 3UL && hull.size() < 5UL) {
      to_ret.type = Object::CIRCLE;
    } else {
      to_ret.type = Object::TRIANGLE;
    }

    auto ellipse = cv::fitEllipse(contours.at(idx));
    to_ret.x = ellipse.center.x;
    to_ret.y = ellipse.center.y;
    to_ret.angle = ellipse.angle;
  }
  return to_ret;
}
```

Алгоритм распознавания зеленого готов. Мы будет условно считать звезду «кругом», а трапецию «квадратом».

Добавим алгоритм движения к центру корзины.

Для этого 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;
}
```

У нас готовы алгоритмы распознавания корзины и движения к ее центру.

Перейдем к поиску мишени и алгоритму движения к ее центру.

Искать мишень мы будем при помощи бинаризации по желтому и зеленому цвету, выбирая самый «высокий» или «низкий» квадрат в кадре, в зависимости от стороны.

```
Object detect_green(cv::Mat &image) {
  static const cv::Scalar lower(50, 150, 160);
  static const cv::Scalar upper(80, 255, 255);
  Object to_ret;
  cv::Mat img = image.clone();

  cv::cvtColor(img, img, CV_BGR2HSV);
  cv::inRange(img, lower, upper, img);
  std::vector<std::vector<cv::Point>> contours;
  cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
  auto max_area = 0;
  std::vector<cv::RotatedRect> rects;
  for (std::size_t idx = 0; idx < contours.size(); ++idx) {

    if (contours.at(idx).size() < 5) {
      continue;
    }

    std::vector<cv::Point> hull;
    cv::convexHull(contours.at(idx), hull, true);
    cv::approxPolyDP(hull, hull, 24, true);

    if (hull.size() < 3UL) {
      continue;
    }

    auto ellipse = cv::fitEllipse(contours.at(idx));
    rects.push_back(ellipse);
  }

  auto min_index = 0;
  auto min_value = std::numeric_limits<int>::max();

```

```
34    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
35      if (rects.at(idx).center.y < min_value) {
36        min_index = idx;
37      }
38    }
39
40    auto &rect = rects.at(min_index);
41    to_ret.x = rect.center.x;
42    to_ret.y = rect.center.y;
43    to_ret.type = Object::RECTANGLE;
44    return to_ret;
45  }
46
47  Object detect_yellow(cv::Mat &image) {
48    static const cv::Scalar lower(0, 100, 60);
49    static const cv::Scalar lower(0, 150, 210);
50    static const cv::Scalar upper(85, 255, 255);
51    Object to_ret;
52    cv::Mat img = image.clone();
53
54    cv::cvtColor(img, img, CV_BGR2HSV);
55    cv::inRange(img, lower, upper, img);
56    std::vector<std::vector<cv::Point>> contours;
57    cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
58    auto max_area = 0;
59    std::vector<cv::RotatedRect> rects;
60    for (std::size_t idx = 0; idx < contours.size(); ++idx) {
61
62      if (contours.at(idx).size() < 5) {
63        continue;
64      }
65
66      std::vector<cv::Point> hull;
67      cv::convexHull(contours.at(idx), hull, true);
68      cv::approxPolyDP(hull, hull, 24, true);
69
70      if (hull.size() < 3UL) {
71        continue;
72      }
73
74      auto ellipse = cv::fitEllipse(contours.at(idx));
75      to_ret.x = ellipse.center.x;
76      to_ret.y = ellipse.center.y;
77      to_ret.angle = ellipse.angle;
78      to_ret.type = Object::RECTANGLE;
79    }
80
81    auto max_index = 0;
82    auto max_value = std::numeric_limits<int>::min();
83
84    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
85      if (rects.at(idx).center.y > max_value) {
86        max_index = idx;
87      }
88    }
89
90    auto &rect = rects.at(max_index);
91    to_ret.x = rect.center.x;
92    to_ret.y = rect.center.y;
93    return to_ret;
```

```
94
95    return to_ret;
96  }
```

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

```
1   Object detect_green(cv::Mat &image) {
2     static const cv::Scalar lower(50, 150, 160);
3     static const cv::Scalar upper(80, 255, 255);
4     Object to_ret;
5     cv::Mat img = image.clone();
6
7     cv::cvtColor(img, img, CV_BGR2HSV);
8     cv::inRange(img, lower, upper, img);
9     std::vector<std::vector<cv::Point>> contours;
10    cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
11    auto max_area = 0;
12    std::vector<cv::RotatedRect> rects;
13    for (std::size_t idx = 0; idx < contours.size(); ++idx) {
14
15      if (contours.at(idx).size() < 5) {
16        continue;
17      }
18
19      std::vector<cv::Point> hull;
20      cv::convexHull(contours.at(idx), hull, true);
21      cv::approxPolyDP(hull, hull, 24, true);
22
23      if (hull.size() < 3UL) {
24        continue;
25      }
26
27      auto ellipse = cv::fitEllipse(contours.at(idx));
28      rects.push_back(ellipse);
29    }
30
31    auto min_index = 0;
32    auto min_value = std::numeric_limits<int>::max();
33
34    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
35      if (rects.at(idx).center.y < min_value) {
36        min_index = idx;
37      }
38    }
39
40    auto &rect = rects.at(min_index);
41    to_ret.x = rect.center.x;
42    to_ret.y = rect.center.y;
43    to_ret.type = Object::RECTANGLE;
44    return to_ret;
45  }
46
47  Object detect_yellow(cv::Mat &image) {
48    static const cv::Scalar lower(0, 100, 60);
49    static const cv::Scalar lower(0, 150, 210);
50    static const cv::Scalar upper(85, 255, 255);
51    Object to_ret;
52    cv::Mat img = image.clone();
```

```
53
54    cv::cvtColor(img, img, CV_BGR2HSV);
55    cv::inRange(img, lower, upper, img);
56    std::vector<std::vector<cv::Point>> contours;
57    cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
58    auto max_area = 0;
59    std::vector<cv::RotatedRect> rects;
60    for (std::size_t idx = 0; idx < contours.size(); ++idx) {
61
62      if (contours.at(idx).size() < 5) {
63        continue;
64      }
65
66      std::vector<cv::Point> hull;
67      cv::convexHull(contours.at(idx), hull, true);
68      cv::approxPolyDP(hull, hull, 24, true);
69
70      if (hull.size() < 3UL) {
71        continue;
72      }
73
74      auto ellipse = cv::fitEllipse(contours.at(idx));
75      to_ret.x = ellipse.center.x;
76      to_ret.y = ellipse.center.y;
77      to_ret.angle = ellipse.angle;
78      to_ret.type = Object::RECTANGLE;
79    }
80
81    auto max_index = 0;
82    auto max_value = std::numeric_limits<int>::min();
83
84    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
85      if (rects.at(idx).center.y > max_value) {
86        max_index = idx;
87      }
88    }
89
90    auto &rect = rects.at(max_index);
91    to_ret.x = rect.center.x;
92    to_ret.y = rect.center.y;
93    return to_ret;
94
95    return to_ret;
96  }
```

Для распознавания зеленой площадки мы будем использовать этот же алгоритм.

Собираем все вместе и получаем следующий код:

```
1   #include "murAPI.hpp"
2
3   Object detect_green(cv::Mat &image) {
4     static const cv::Scalar lower(50, 150, 160);
5     static const cv::Scalar upper(80, 255, 255);
6     Object to_ret;
7     cv::Mat img = image.clone();
8
9     cv::cvtColor(img, img, CV_BGR2HSV);
10    cv::inRange(img, lower, upper, img);
11    std::vector<std::vector<cv::Point>> contours;
```

```
12    cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
13    auto max_area = 0;
14    std::vector<cv::RotatedRect> rects;
15    for (std::size_t idx = 0; idx < contours.size(); ++idx) {
16
17      if (contours.at(idx).size() < 5) {
18        continue;
19      }
20
21      std::vector<cv::Point> hull;
22      cv::convexHull(contours.at(idx), hull, true);
23      cv::approxPolyDP(hull, hull, 24, true);
24
25      if (hull.size() < 3UL) {
26        continue;
27      }
28
29      auto ellipse = cv::fitEllipse(contours.at(idx));
30      rects.push_back(ellipse);
31    }
32
33    auto min_index = 0;
34    auto min_value = std::numeric_limits<int>::max();
35
36    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
37      if (rects.at(idx).center.y < min_value) {
38        min_index = idx;
39      }
40    }
41
42    auto &rect = rects.at(min_index);
43    to_ret.x = rect.center.x;
44    to_ret.y = rect.center.y;
45    to_ret.type = Object::RECTANGLE;
46    return to_ret;
47  }
48
49  Object detect_yellow(cv::Mat &image) {
50    static const cv::Scalar lower(0, 100, 60);
51    static const cv::Scalar lower(0, 150, 210);
52    static const cv::Scalar upper(85, 255, 255);
53    Object to_ret;
54    cv::Mat img = image.clone();
55
56    cv::cvtColor(img, img, CV_BGR2HSV);
57    cv::inRange(img, lower, upper, img);
58    std::vector<std::vector<cv::Point>> contours;
59    cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
60    auto max_area = 0;
61    std::vector<cv::RotatedRect> rects;
62    for (std::size_t idx = 0; idx < contours.size(); ++idx) {
63
64      if (contours.at(idx).size() < 5) {
65        continue;
66      }
67
68      std::vector<cv::Point> hull;
69      cv::convexHull(contours.at(idx), hull, true);
70      cv::approxPolyDP(hull, hull, 24, true);
71
```

```
72      if (hull.size() < 3UL) {
73        continue;
74      }
75
76      auto ellipse = cv::fitEllipse(contours.at(idx));
77      to_ret.x = ellipse.center.x;
78      to_ret.y = ellipse.center.y;
79      to_ret.angle = ellipse.angle;
80      to_ret.type = Object::RECTANGLE;
81    }
82
83    auto max_index = 0;
84    auto max_value = std::numeric_limits<int>::min();
85
86    for (std::size_t idx = 0; idx < rects.size(); ++idx) {
87      if (rects.at(idx).center.y > max_value) {
88        max_index = idx;
89      }
90    }
91
92    auto &rect = rects.at(max_index);
93    to_ret.x = rect.center.x;
94    to_ret.y = rect.center.y;
95    return to_ret;
96
97    return to_ret;
98  }
99
100 /*
101 Функция определения линии.
102 */
103 Object detect_line(cv::Mat &image) {
104   static const cv::Scalar lower(0, 100, 60);
105   static const cv::Scalar upper(20, 255, 255);
106   Object to_ret;
107   cv::Mat img = image.clone();
108
109   cv::cvtColor(img, img, CV_BGR2HSV);
110   cv::inRange(img, lower, upper, img);
111   std::vector<std::vector<cv::Point>> contours;
112   cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
113   auto max_area = 0;
114
115   for (std::size_t idx = 0; idx < contours.size(); ++idx) {
116
117     if (contours.at(idx).size() < 5) {
118       continue;
119     }
120
121     std::vector<cv::Point> hull;
122     cv::convexHull(contours.at(idx), hull, true);
123     cv::approxPolyDP(hull, hull, 24, true);
124
125     if (hull.size() < 3UL) {
126       continue;
127     }
128
129     auto ellipse = cv::fitEllipse(contours.at(idx));
130     to_ret.x = ellipse.center.x;
131     to_ret.y = ellipse.center.y;
```

```
132     to_ret.angle = ellipse.angle;
133   }
134   return to_ret;
135 }
136 
137 /*
138 Функция определения звезды.
139 */
140 Object detect_start(cv::Mat &image) {
141   static const cv::Scalar lower(23, 200, 94);
142   static const cv::Scalar upper(50, 255, 122);
143   Object to_ret;
144   cv::Mat img = image.clone();
145 
146   cv::cvtColor(img, img, CV_BGR2HSV);
147   cv::inRange(img, lower, upper, img);
148   cv::imshow("W", img);
149   cv::waitKey(1);
150   std::vector<std::vector<cv::Point>> contours;
151   cv::findContours(img, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
152 
153   auto max_area = 0;
154 
155   for (std::size_t idx = 0; idx < contours.size(); ++idx) {
156 
157     if (contours.at(idx).size() < 5) {
158       continue;
159     }
160 
161     std::vector<cv::Point> hull;
162     cv::convexHull(contours.at(idx), hull, true);
163     cv::approxPolyDP(hull, hull, 24, true);
164 
165     if (hull.size() > 3UL && hull.size() < 5UL) {
166       to_ret.type = Object::CIRCLE;
167     } else {
168       to_ret.type = Object::TRIANGLE;
169     }
170 
171     auto ellipse = cv::fitEllipse(contours.at(idx));
172     to_ret.x = ellipse.center.x;
173     to_ret.y = ellipse.center.y;
174     to_ret.angle = ellipse.angle;
175   }
176   return to_ret;
177 }
178 
179 void yd_p_reg(double yaw, double depth, int power) {
180   double u = mur.getYaw() - yaw;
181   if (u < 0.0) {
182     u += 360.0;
183   }
184   if (u > 180.0) {
185     u -= 360.0;
186   }
187   u *= 1.3;
188   double ddepth = mur.getInputAOne() - depth;
189   ddepth *= 6;
190   mur.setPorts(-power + u, -power - u, -ddepth, 0);
191 }
```

```
192
193 void yd_p_reg_timed(double yaw, double depth, int power, long long time) {
194   Timer t;
195   t.start();
196   while (t.elapsed() < time) {
197     yd_p_reg(yaw, depth, power);
198   }
199 }
200
201 double from_cv_angle(double angle) {
202   auto new_angle = angle;
203   if (new_angle > 90) {
204     new_angle = (-1.0) * (180.0 - new_angle);
205   }
206   return new_angle;
207 }
208
209 void yd_p_reg_line(double depth, int power) {
210   auto img = mur.getCameraOneFrame();
211   auto obj = detect_line(img);
212   double u = (0.0 - from_cv_angle(obj.angle)) * 1.3;
213   double ddepth = (mur.getInputAOne() - depth) * 6.0;
214   double u_lag = 160.0 - obj.x;
215   mur.setPorts(-power + u, -power - u, -ddepth, -u_lag);
216 }
217
218 bool move_to_v_center(int x_val) {
219   constexpr auto center_v = 320 / 2;
220   auto center_diff = x_val - center_v;
221   if (std::abs(center_diff) < 25) {
222     mur.setPortD(0);
223     return true;
224   }
225   if (center_diff < 0) {
226     mur.setPortD(-15);
227   }
228   if (center_diff > 0) {
229     mur.setPortD(15);
230   }
231   return false;
232 }
233
234 bool move_to_hv_center_front(int x, int y) {
235
236   constexpr auto center_v = 320 / 2;
237   constexpr auto center_h = 240 / 2;
238   mur.setPortA(0);
239   mur.setPortB(0);
240   auto center_x_diff = x - center_v;
241   auto center_y_diff = y - center_h;
242   if (std::abs(center_x_diff) < 10 && std::abs(center_y_diff) < 10) {
243     mur.setPortD(0);
244     return true;
245   }
246   if (center_x_diff < 0) {
247     mur.setPortD(-15);
248   }
249   if (center_x_diff > 0) {
250     mur.setPortD(15);
251   }
```

```
252   if (center_y_diff < 0) {
253     mur.setPortC(-15);
254   }
255   if (center_y_diff > 0) {
256     mur.setPortC(15);
257   }
258   return false;
259 }
260
261 int main() {
262   yd_p_reg_timed(0, 70, 30, 3000);
263
264   auto rotate_to_180 = []() {
265     auto yaw = mur.getYaw();
266     yd_p_reg_timed(yaw + 180.0, 70, 30, 3000);
267   };
268
269   auto rotate_to_yellow = []() {
270     auto yaw = mur.getYaw();
271     yd_p_reg_timed(yaw, 30, 30, 15000);
272     yd_p_reg_timed(yaw + 180.0, 70, 30, 3000);
273   };
274
275   auto line_and_bin = []() {
276     for (;;) {
277       auto img = mur.getCameraOneFrame();
278       yd_p_reg_line(100, 15);
279       auto star_obj = detect_start(img);
280
281       if (star_obj.type == Object::NONE) {
282         continue;
283       }
284
285       if (star_obj.type == Object::RECTANGLE) {
286         break;
287       }
288
289       bool in_center = false;
290       mur.setPorts(0, 0, 0, 0);
291       while (!in_center) {
292         img = mur.getCameraOneFrame();
293         star_obj = detect_start(img);
294         in_center = move_to_v_center(star_obj.x);
295         mur.setPortC((mur.getInputAOne() - 100.0) * -6);
296       }
297       mur.drop();
298       break;
299     }
300   };
301
302   auto shoot_green_side = []() {
303     for (;;) {
304       auto img = mur.getCameraTwoFrame();
305       yd_p_reg_line(100, 15);
306       auto green_side = detect_green(img);
307
308       if (green_side.type == Object::NONE) {
309         continue;
310       }
311
```

```
312       bool in_center = false;
313       mur.setPorts(0, 0, 0, 0);
314       while (!in_center) {
315         img = mur.getCameraTwoFrame();
316         green_side = detect_green(img);
317         in_center = move_to_hv_center_front(green_side.x, green_side.y);
318         mur.setPortC((mur.getInputAOne() - 100.0) * -6);
319       }
320       mur.shoot();
321       break;
322     }
323   };
324
325   auto shoot_yellow_side = []() {
326     for (;;) {
327       auto img = mur.getCameraTwoFrame();
328       yd_p_reg_line(100, 15);
329       auto yellow_side = detect_yellow(img);
330
331       if (yellow_side.type == Object::NONE) {
332         continue;
333       }
334
335       bool in_center = false;
336       mur.setPorts(0, 0, 0, 0);
337       while (!in_center) {
338         img = mur.getCameraTwoFrame();
339         yellow_side = detect_yellow(img);
340         in_center = move_to_hv_center_front(yellow_side.x, yellow_side.y);
341         mur.setPortC((mur.getInputAOne() - 100.0) * -6);
342       }
343       mur.shoot();
344       break;
345     }
346   };
347
348   auto stop_at_yellow = []() {
349     for (;;) {
350       auto img = mur.getCameraOneFrame();
351       yd_p_reg_line(100, 15);
352       auto yellow_side = detect_yellow(img);
353
354       if (yellow_side.type == Object::NONE) {
355         continue;
356       }
357
358       bool in_center = false;
359       mur.setPorts(0, 0, 0, 0);
360       while (!in_center) {
361         img = mur.getCameraOneFrame();
362         yellow_side = detect_yellow(img);
363         in_center = move_to_hv_center_front(yellow_side.x, yellow_side.y);
364         mur.setPortC((mur.getInputAOne() - 100.0) * -6);
365       }
366       yd_p_reg_timed(mur.getYaw(), 200, 30, 30000);
367       break;
368     }
369   };
370
371   line_and_bin();
```

```
  line_and_bin();
  shoot_green_side();
  rotate_to_yellow();
  shoot_yellow_side();
  rotate_to_180();
  line_and_bin();
  stop_at_yellow();
}
```

Задача 5.1.3. (40 баллов)

Задача выполняется в бассейне. Характеристики бассейна зависят от места проведения финала.

Необходимо запрограммировать подводного робот, который должен выполнить под водой две подзадачи.

В расстоянии до 1 метра от старта находится полоска, размеры 14×75 см, которая направлена на щит. Аппарат должен найти полоску и повернуться по направлению к щиту. Щит расположен на расстоянии от 2 до 5 метров от полоски.

Размеры щита - 60 × 60 см, в центре щита имеется отверстие диаметром 25 см. Робот должен выстрелить торпедой с расстояния не менее 10 см от щита так, чтобы торпеда самостоятельно прошла сквозь отверстие.

За поиск полоски, правильный поворот и ориентацию робота дается 10 баллов.

За торпедирование мишени дается 30 баллов.

Система оценки

Во время финальных заездов каждой команде будет дано 5 минут на выполнение задачи. В течение отведенного времени робот может погружаться и всплывать неограниченное количество раз. Допускается перепрограммирование. В зачет идет лучшая попытка финального заезда. Время в течение выполнения миссии не останавливается.

Расстояние от старта до полоски, расстояние от полоски до щита, угол поворота полоски, место старта могут меняться перед финальным заездом.

Решение

Для решения данной задачи нам достаточно заглубиться, пройти вперед и подать тягу на подключенное к порту D устройство для стрельбы.

Пример программы-решения

Ниже представлено решение на языке C++

```
#include <murAPI.hpp>

void yd_p_reg(double yaw, double depth, int power) {
  double u = mur.getYaw() - yaw;
```

```
  if (u < 0.0) {
    u += 360.0;
  }
  if (u > 180.0) {
    u -= 360.0;
  }
  u *= 1.3;
  double ddepth = mur.getInputAOne() - depth;
  ddepth *= 6;
  mur.setPorts(-power + u, -power - u, -ddepth, 0);
}

void yd_p_reg_timed(double yaw, double depth, int power, long long time) {
  Timer t;
  t.start();
  while (t.elapsed() < time) {
    yd_p_reg(yaw, depth, power);
  }
}

int main() {
  //Берем текущий курс
  auto yaw = mur.getYaw();
  // Заглубляемся на 1м и идем прямо 15 секунда
  yd_p_reg_timed(yaw, 100, 30, 15000);
  //Стреляем
  mur.setPortD(100);
  //Всплываем.
  yd_p_reg_timed(yaw, 0, 0, 5000);
}
```