

Работа призера заключительного этапа
командной инженерной олимпиады школьников
Олимпиада Национальной технологической инициативы

Профиль «Водные робототехнические системы»

Кривега Владислав Алексеевич

Класс: 10

Город: г. Благовещенск

Школа: Лицей БГПУ

Регион: Амурская область

Уникальный номер участника: 258465

Команда на заключительном этапе: Underwater power

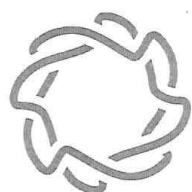
Параллель: 10-11

Результаты заключительного этапа:

Индивидуальная часть															Командная часть				Результат			
Физика				Информатика											Итого:	Макс. Балл				Всего		
1	2	3	4	1	2	3	4	5	6	7	8	9	10	11			1	2			3	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	200	12	10	13	35	21

Индивидуальная часть

Персональный лист участника с номером 258465:



Олимпиада НТИ

ФИО Кривеня Владислав Алексеевич

Город Благовещенск

Школа № лицей БГТУ

Физика

Командная инженерная олимпиада «Олимпиада НТИ»

Направление БРС

Предмет _____

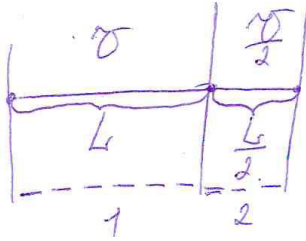
Номер участника 258465

2,1.

Σ 0,2

0,2

1.1. Ответ: $\frac{L}{2}$



Рассмотрим 2 участка.
Когда 2 объект заедет в зону на участок 2, то его скорость уменьшится в два раза $\frac{v}{2}$, а значит за одно и то же время они пройдут разные пути, отличающиеся на $\frac{L}{2}$, ну а дальше они будут ехать с одинаковой относительно друг друга скоростью на расстоянии $\frac{L}{2}$ друг от друга.

0,2

1.2. Второй быстрее.

0

2.1 Дано:

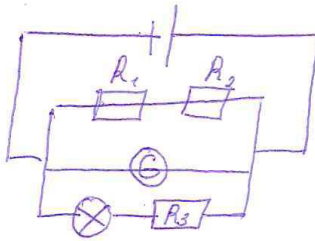
$$R_1 = 1 \text{ Ом}$$

$$R_2 = 1 \text{ Ом}$$

$$R_3 = 4 \text{ Ом}$$

$R_{\text{общ}}$

Решение



$$R_{\text{общ}1} = R_1 + R_2;$$

$$R_{\text{общ}1} = 1 + 1 = 2 \text{ (Ом)}$$

$$R_{\text{общ}2} = \frac{R_{\text{общ}1} \cdot R_3}{R_{\text{общ}1} + R_3}$$

$$R_{\text{общ}2} = \frac{2 \cdot 4}{2 + 4} = \frac{8}{6} = 1,33 \text{ (Ом)}$$

Ответ: 1,33 Ом.

0

Командная часть

Результаты были получены в рамках выступления команды: Underwater power

Личный состав команды:

- Русинович Андрей Сергеевич
- Кривега Владислав Алексеевич

```
#include <murAPI.hpp>
#include <thread>

int detectCircle(cv::Mat &&img) {

    int count = 0;

    if (img.empty()) {
        return 0;
    }

    static cv::Mat image;
    static std::vector<std::vector<cv::Point> > contours;
    static std::vector<cv::Point> hull;
    image = img.clone();
    cv::Mat toDraw = img.clone();
    cv::cvtColor(image, image, CV_BGR2GRAY);
    cv::GaussianBlur(image, image, cv::Size(5, 5), 2);

    double cannyParams = cv::threshold(image, image, 0, 255,
CV_THRESH_BINARY_INV + CV_THRESH_OTSU);
    cv::Canny(image, image, cannyParams, cannyParams / 2.0F);
    cv::findContours(image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);

    Object objectToRet;
    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 bEllipse = cv::fitEllipse(contours.at(i));
        cv::convexHull(contours.at(i), hull, true);
        cv::approxPolyDP(hull, hull, 15, true);
        if (!cv::isContourConvex(hull)) {
            continue;
        }

        double area = cv::contourArea(contours.at(i));
        cv::Rect r = cv::boundingRect(contours.at(i));
        double radius = r.width / 2.0;

        if (std::abs(1.0 - ((double) r.width / (double) r.height)) <= 0.2

    }
}
```

```

        std::abs(1.0 - (area / (CV_PI * std::pow(radius, 2.0)))) <=
0.2) {
            count++;
            cv::circle(toDraw, bEllipse.center, radius,
cv::Scalar(255, 255, i * 10), 3);
        }
        cv::waitKey(1);
    }
    return count - count / 2;
}

void keepYaw(double& yts) {
    double e = mur.getYaw() - yts;
    int kp = 2;
    mur.setPortA(15 - e * kp);
    mur.setPortB(15 + e * kp);
}

void keepDepth(double& dts) {
    double depth = mur.getInputAOne();
    mur.setPortC((dts - depth) * 10 * (-1));
}

void rotate_line(double& yts, bool& line) {
    if (!line) {
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::RECTANGLE) {
                if (obj.angle < 90) {
                    yts += obj.angle;
                } else {
                    yts -= std::fabs(90 - obj.angle);
                }
                mur.removeDetectorFromList(Object::RECTANGLE, 0);
                mur.add
                line = true;
            }
        }
    }
}

int main() {
    double yts = 0.0, dts = 120.0;
    bool line = false;
    mur.addDetectorToList(Object::RECTANGLE, 0);
    while (1) {
        keepYaw(yts);
        keepDepth(dts);
        rotate_line(yts, line);
        int cnt = 0;
        if (line) {
            for (auto obj : mur.getDetectedObjectsList(0)) {
                if (obj.type == Object::CIRCLE) {
                    cnt = detectCircle(mur.getCameraOneFrame());
                }
            }
        }
    }
}
return 0;

```

```
}
```

```
#include <murAPI.hpp>  
#include <thread>
```

```
Object red (cv:: Mat img) {  
    cv::Scalar lower(0, 144, 42);  
    cv::Scalar upper(13, 255, 249);  
    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_TREE, CV_CHAIN_APPROX_NONE);  
    Object objectToRet;  
  
    for (std::size_t i = 0; i < contours.size(); i++) {  
        if (contours.at(i).size() < 100) {  
            continue;  
        }  
        if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {  
            continue;  
        }  
  
        cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));  
        objectToRet.x = (int) bEllipse.center.x;  
        objectToRet.y = (int) bEllipse.center.y;  
        objectToRet.angle = bEllipse.angle;  
        for (auto obj : mur.getDetectedObjectsList(0)) {  
            if (obj.type == Object::CIRCLE) objectToRet.type =  
Object::CIRCLE;  
        }  
        return objectToRet;  
    }  
    return objectToRet;  
}
```

```
Object yellow (cv:: Mat img) {  
    cv::Scalar lower(6, 135, 44);  
    cv::Scalar upper(38, 255, 249);  
    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_TREE, CV_CHAIN_APPROX_NONE);  
    Object objectToRet;  
  
    for (std::size_t i = 0; i < contours.size(); i++) {  
        if (contours.at(i).size() < 100) {  
            continue;  
        }  
        if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {  
            continue;  
        }  
  
        cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));  
        objectToRet.x = (int) bEllipse.center.x;  
        objectToRet.y = (int) bEllipse.center.y;
```

```

        objectToRet.angle = bEllipse.angle;
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::CIRCLE) objectToRet.type =
Object::CIRCLE;
        }
        return objectToRet;
    }
    return objectToRet;
}

Object green (cv:: Mat img) {
    cv::Scalar lower(50, 66, 5);
    cv::Scalar upper(110, 255, 210);
    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_TREE, CV_CHAIN_APPROX_NONE);
    Object objectToRet;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 100) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 300.0) {
            continue;
        }

        cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));
        objectToRet.x = (int) bEllipse.center.x;
        objectToRet.y = (int) bEllipse.center.y;
        objectToRet.angle = bEllipse.angle;
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::CIRCLE) objectToRet.type =
Object::CIRCLE;
        }
        return objectToRet;
    }
    return objectToRet;
}

int detectCircle(cv::Mat &&img) {

    int count = 0;

    if (img.empty()) {
        return 0;
    }

    static cv::Mat image;
    static std::vector<std::vector<cv::Point> > contours;
    static std::vector<cv::Point> hull;
    image = img.clone();
    cv::Mat toDraw = img.clone();
    cv::cvtColor(image, image, CV_BGR2GRAY);
    cv::GaussianBlur(image, image, cv::Size(5, 5), 2);

    double cannyParams = cv::threshold(image, image, 0, 255,
CV_THRESH_BINARY_INV + CV_THRESH_OTSU);
    cv::Canny(image, image, cannyParams, cannyParams / 2.0F);
    cv::findContours(image, contours, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);

```

```

Object objectToRet;
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 bEllipse = cv::fitEllipse(contours.at(i));
    cv::convexHull(contours.at(i), hull, true);
    cv::approxPolyDP(hull, hull, 15, true);
    if (!cv::isContourConvex(hull)) {
        continue;
    }

    double area = cv::contourArea(contours.at(i));
    cv::Rect r = cv::boundingRect(contours.at(i));
    double radius = r.width / 2.0;

    if (std::abs(1.0 - ((double) r.width / (double) r.height)) <= 0.2
    &&
        std::abs(1.0 - (area / (CV_PI * std::pow(radius, 2.0)))) <=
0.2) {
        count++;
        cv::circle(toDraw, bEllipse.center, radius,
cv::Scalar(255, 255, i * 10), 3);
    }
    cv::waitKey(1);
}
return count - count / 2;
}

void keepYaw(double& yts, bool& circ) {
    if (!circ) {
        double e = mur.getYaw() - yts;
        int kp = 2;
        double powerA = 15 - e * kp, powerB = 15 + e * kp;
        mur.setPortA(powerA);
        mur.setPortB(powerB);
    }
}

void keepDepth(double& dts, bool& circ) {
    if (!circ) {
        double depth = mur.getInputAOne();
        mur.setPortC((dts - depth) * 10 * (-1));
    }
}

void rotate_line(double& yts, bool& line, bool& circl) {
    if (!line) {
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::RECTANGLE) {
                if (obj.angle < 90) {
                    yts += obj.angle;
                } else {
                    yts -= std::fabs(90 - obj.angle);
                }
            }
            mur.removeDetectorFromList(Object::RECTANGLE, 0);
            mur.addDetectorToList(Object::CIRCLE, 0);
            circl = true;
        }
    }
}

```



```

        line = true;
    }
}

}

int main() {
    double yts = 0.0, dts = 135.0;
    bool line = false, circ = false, cnt1 = true, cnt2 = true, cnt3 =
true, circl = false, tol80 = false;
    mur.addDetectorToList(Object::RECTANGLE, 0);
    while (1) {
        keepYaw(yts, circ);
        keepDepth(dts, circ);
        rotate_line(yts, line, circl);
        if (circl) {
            Object obj1 = green(mur.getCameraOneFrame());
            if (obj1.type == Object::CIRCLE) {
                if (cnt1) {
                    circ = true;
                    mur.setPortA(0);
                    mur.setPortB(0);
                    mur.setPortC(-70);
                    sleepFor(5000);
                    circ = false;
                    cnt1 = false;
                }
            }
            Object obj2 = yellow(mur.getCameraOneFrame());
            if (obj2.type == Object::CIRCLE) {
                if (cnt2) {
                    circ = true;
                    mur.setPortA(0);
                    mur.setPortB(0);
                    mur.setPortC(-70);
                    sleepFor(5000);
                    circ = false;
                    cnt2 = false;
                }
            }
            Object obj3 = red(mur.getCameraOneFrame());
            if (obj3.type == Object::CIRCLE) {
                if (cnt3) {
                    circ = true;
                    mur.setPortA(0);
                    mur.setPortB(0);
                    mur.setPortC(-70);
                    sleepFor(5000);
                    circ = false;
                    cnt3 = false;
                    tol80 = true;
                    circl = false;
                }
            }
        }
    }
    if (tol80) {
        circ = true;
        float yaw = mur.getYaw();
        float dpth = mur.getInputAOne();
    }
}

```

```

        float yts = , dts = 100;
        if (yaw > yts) {
            mur.setPortA(0);
            mur.setPortB(30);
        }
        if (yaw < yts) {
            mur.setPortB(30);
            mur.setPortB(0);
        }
        mur.setPortC((dts - dpth) * 10 * (-1));
    }

}
return 0;
}

#include <murAPI.hpp>
#include <thread>

Object yellow (cv:: Mat img) {
    cv::Scalar lower(23, 113, 166);
    cv::Scalar upper(33, 153, 252);
    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_TREE, CV_CHAIN_APPROX_NONE);
    Object objectToRet;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 20) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 200.0) {
            continue;
        }

        cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));
        objectToRet.x = (int) bEllipse.center.x;
        objectToRet.y = (int) bEllipse.center.y;
        objectToRet.angle = bEllipse.angle;
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::CIRCLE) objectToRet.type =
Object::CIRCLE;
        }
        return objectToRet;
    }
    return objectToRet;
}

Object red (cv:: Mat img) {
    cv::Scalar lower(0, 122, 80);
    cv::Scalar upper(185, 199, 166);
    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_TREE, CV_CHAIN_APPROX_NONE);
    Object objectToRet;

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

```

```

    if (contours.at(i).size() < 10) {
        continue;
    }
    if (std::fabs(cv::contourArea(contours.at(i))) < 100.0) {
        continue;
    }

    cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));
    objectToRet.x = (int) bEllipse.center.x;
    objectToRet.y = (int) bEllipse.center.y;
    objectToRet.angle = bEllipse.angle;
    for (auto obj : mur.getDetectedObjectsList(0)) {
        if (obj.type == Object::CIRCLE) objectToRet.type =
Object::CIRCLE;
    }
    return objectToRet;
}
return objectToRet;
}

Object green (cv:: Mat img) {
    cv::Scalar lower(34, 174, 114);
    cv::Scalar upper(70, 255, 189);
    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_TREE, CV_CHAIN_APPROX_NONE);
    Object objectToRet;

    for (std::size_t i = 0; i < contours.size(); i++) {
        if (contours.at(i).size() < 10) {
            continue;
        }
        if (std::fabs(cv::contourArea(contours.at(i))) < 100.0) {
            continue;
        }

        cv::RotatedRect bEllipse = cv::fitEllipse(contours.at(i));
        objectToRet.x = (int) bEllipse.center.x;
        objectToRet.y = (int) bEllipse.center.y;
        objectToRet.angle = bEllipse.angle;
        for (auto obj : mur.getDetectedObjectsList(0)) {
            if (obj.type == Object::CIRCLE) objectToRet.type =
Object::CIRCLE;
        }
        return objectToRet;
    }
    return objectToRet;
}

int main() {

    mur.initCamera(0);
    mur.addDetectorToList(Object :: CIRCLE, 0);

    int i = 0, cur = 0;
    int first = mur.getInputAOne() / 10, second = (int)mur.getInputAOne()
% 10, third = mur.getInputATwo() / 10;
    std::vector <int> cor = {1, 2, 3};

```

```

    bool circ = true, circle = true, circ_find_green = true, circ_find_red
= true, circ_find_yellow = true;
    const int toSet = 50;
    const int kP = 7;
    int powerB = -40, powerA = -55;

while (1) {
    int depth = mur.getInputBOne();
    int power = (depth - toSet) * kP * (-1);
    float yaw = mur.getYaw();
    float yts = yaw - 180;
    if (circ) {
        if (yts > 0) {
            mur.setPortA(0);
            mur.setPortB(powerB);
        }
        if (yts < 0) {
            mur.setPortA(powerA);
            mur.setPortB(0);
        }
        mur.setPortC(power);
    }

    if (circle) {
        Object obj1 = green(mur.getCameraOneFrame());
        Object obj2 = yellow(mur.getCameraOneFrame());
        Object obj3 = red(mur.getCameraOneFrame());

        if(circ_find_green) {
            if (obj1.type == Object :: CIRCLE) {
                cur++;
                circ_find_red = true;
                circ_find_yellow = true;
                circ_find_green = false;
            }
        }

        if (circ_find_yellow) {
            if (obj2.type == Object::CIRCLE) {
                cur++;
                circ_find_red = true;
                circ_find_green = true;
                circ_find_yellow = false;
            }
        }

        if (circ_find_red) {
            if (obj3.type == Object::CIRCLE) {
                cur++;
                circ_find_green = true;
                circ_find_yellow = true;
                circ_find_red = false;
            }
        }

        if (cur == cor[i]) {
            circ = false;
            mur.setPortA(0);
            mur.setPortB(0);
            mur.setPortC(70);
        }
    }
}

```

```

        sleepFor(12000);
        i++;
        circ = true;
        if (cur > cor[i]) {
            powerA *= -1;
            powerB *= -1;
        }
        if (cur < cor[i]) {
            powerB = -40;
            powerA = -55;
        }
        if (cur == 3) cur = 1;
    }
}
}
}
}

```

```

#include <murAPI.hpp>
#include <thread>

```

```

int main() {

    int power = -40;

    mur.setPortC(70);
    sleepFor(3000);

    mur.setPortC(-70);
    sleepFor(3000);

    mur.setPortC(0);

    Timer time;
    time.start();

    while (1) {

        float yaw = mur.getYaw() - 180;

        if (time.elapsed() < 24000) {

            if (yaw < 0.0) {
                mur.setPortA(power);
                mur.setPortB(0);
            }

            if (yaw > 0.0) {
                mur.setPortA(0);
                mur.setPortB(power);
            }
        }

        if (time.elapsed() > 24000 && time.elapsed() < 26000) {
            mur.setPortA(50);
            mur.setPortB(52);
        }

        if (time.elapsed() > 26000) {

```

```
        mur.setPortA(-50);  
        mur.setPortB(50);  
        sleepFor(5000);  
  
        mur.setPortA(50);  
        mur.setPortB(-50);  
        sleepFor(5200);  
  
        mur.setPortA(0);  
        mur.setPortB(0);  
        sleepFor(100000);  
    }  
}  
return 0;  
}
```

