Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт

по лабораторной работе №1

Дисциплина: Техническое зрение

Тема: Моделирование движения робота с использованием библиотеки OpenCV

Студент гр. 3331506/70401 Соколов Д.А.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

Разработать программный модуль, который реализует анимацию движения робота (схематично) по траектории и с помощью управления по скорости. Выход за границы рабочей области должен быть ограничен.

Задачи

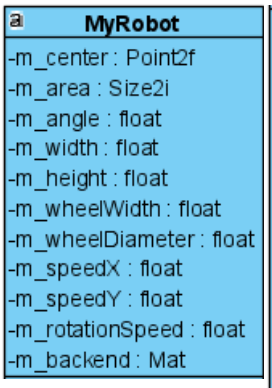
Проанализировав задание, были сформулированы следующие задачи:

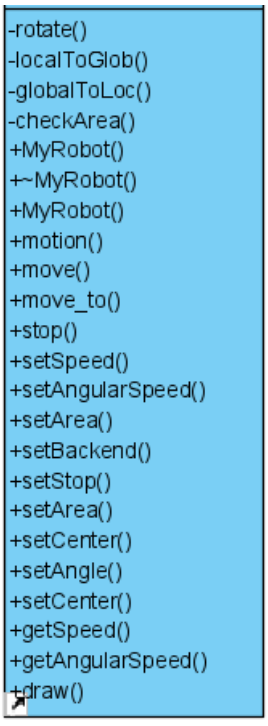
- Разработать интерфейс класса robot
- Разработать метод вращения и передвижения по скорости
- Ограничить рабочую зону для робота
- Реализовать метод движения по координатам

Ход работы

Интерфейс класса

UML диаграмма представлена справа на двух картинках. Подробнее о реализации отдельных методов будет изложено ниже, в следующих главах.





Вращение и передвижение

Для изменения положения и ориентации робота было решено применить управление по скорости. Метод *move* принимает 3 аргумента: линейную скорость по двум осям и угловую скорость (в радианах). Его исходный код показан на рисунке 1.

```
int MyRobot::move(float speedX, float speedY, float angularSpeed)
{
    setAngularSpeed(this->m_rotationSpeed + angularSpeed);
    setSpeed(this->m_speedX + speedX, this->m_speedY + speedY);
    return 0;
}
```

Рисунок 1 — Реализация метода *move*

Затем отрисовка изменений производится методом *draw*. Изображен на рисунке 2. Сначала обновляется поле *m_angle*, которое содержит угол ориентации робота в локальной системе координат. Затем рассчитываются координаты углов робота, и далее по ним рисуются линии. В конце — вывод на экран. Зачистка фона также предусмотрена. Создано поле класса, в котором хранится фоновое изображение.

```
int MyRobot::draw()
{
    this->setAngle(this->m_angle + this->m_rotationSpeed);

    cv::Point2f vertices[4];
    vertices[0] = localToGlob(this->m_height / 2, -this->m_width / 2);
    vertices[1] = localToGlob(this->m_height / 2, this->m_width / 2);
    vertices[3] = localToGlob(-this->m_height / 2, -this->m_width / 2);
    vertices[2] = localToGlob(-this->m_height / 2, this->m_width / 2);
    setCenter(((vertices[0].x + vertices[2].x)/2.0f), ((vertices[0].y + vertices[2].y)/2.0f));
    cv::Mat background = this->m_backend.clone();
    for (int i = 0; i < 4; i++)
    {
        cv::line(background, vertices[i], vertices[(i+1)%4], cv::Scalar(0,255,0), 2);
        // Задний бампер
        if (i == 0)
            cv::line(background, vertices[i], vertices[(i+1)%4], cv::Scalar(255,0,0), 2);
        // Передний бампер
        if (i == 2)
            cv::line(background, vertices[i], vertices[(i+1)%4], cv::Scalar(0,0,255), 2);
    }
    cv::imshow("robot", background);
    return 0;
}
```

Рисунок 2 — Реализация метода *draw*

Для отрисовки робота необходимо локальные координаты перевести в глобальные. Для этого написан метод *localToGlob*. Перевод осуществляется при помощи применения матрицы поворота и матрицы переноса. Итоговые формулы:

$$X = x` * \cos(\varphi) + tx - y` * \sin(\varphi)$$

$$Y = x` * \sin(\varphi) + ty + y` * \cos(\varphi)$$

Код показан на рисунке 3.

```
cv::Point2f MyRobot::localToGlob (float yLoc, float xLoc)
{
    float yLoc_ = yLoc + this->m_speedY;
    float xLoc_ = xLoc + this->m_speedX;

    float xGlob = xLoc_ * std::cos(this->m_angle) + this->m_center.x - yLoc_ * std::sin(this->m_angle);
    float yGlob = xLoc_ * std::sin(this->m_angle) + this->m_center.y + yLoc_ * std::cos(this->m_angle);

    //проверка выхода за границы

    checkArea(xGlob, yGlob);
    cv::Point2f out(xGlob, yGlob);
    return out;
}
```

Рисунок 3 — Реализация метода *localToGlob*

Ограничение рабочей зоны

Для решения данной задачи был разработан метод `void checkArea(float & x, float & y);` Принимает два аргумента – координаты угла робота. Если x и y выходят за границы разрешенной области – робот останавливается.

Пример реализации показан ниже, на рисунке 4.

```
void MyRobot::checkArea(float & x, float & y)
{
    if ((x >= this->m_area.width) || (x <= 0) || (y >= this->m_area.height) || (y <= 0))
    {
        this->stop();
        return;
    }
}
```

Рисунок 4 — Реализация метода *checkArea*

Движение по координатам

Для решения задачи движения в точку было решено добавить отрицательную обратную связи по положению. Высчитывается ошибка err. Затем в цикле регулируется скорость робота и его угол поворота, таким образом, чтобы ошибка была минимальна и стремилась к 0. Код представлен на рисунке 5.

```
int MyRobot::move_to(float xGlob, float yGlob, float angularSpeed)
{
    cv::Point3f dest(xGlob, yGlob, angularSpeed);
    cv::Point3f pose(m_center.x, m_center.y, m_angle);
    cv::Point3f err = dest - pose;

    while (int(err.x) != 0 || int(err.y) != 0 || std::abs(err.z) >= ACCURACY)
    {
        pose = cv::Point3f(m_center.x, m_center.y, m_angle);
        err = dest - pose;
        float xSpeed = regulator(int(err.x));
        float ySpeed = regulator(int(err.y));
        float aSpeed = regulator(err.z);
        setSpeed(xSpeed, ySpeed);
        setAngularSpeed(aSpeed);
        draw();
        cv::waitKey(1);
    }
    stop();
    return 0;

}
```

Рисунок 5 — Реализация метода *move_to*

Примечание: Исходный код можно найти по [ССЫЛКЕ]