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 Ляпцев И.А.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

С помощью методов *OpenCV* реализовать движение робота по заданному полю, его поворот на месте, а также ограничение на выезд за пределы.

Задачи

1) Реализовать вывод изображения робота на экран;

2) Реализовать продольное и поперечное движения робота;

3) Реализовать поворот робота относительно центра его корпуса;

4) Реализовать ограничения на выезд за пределы заданной области.

Ход работы

Общий алгоритм выполнения программы:

1) Ожидание нажатия кнопок управления;

2) Вычисление координат точек робота, если нажата какая-то кнопка продольного или поперечного движения;

3) Вычисление координат точек робота, если нажата какая-то из кнопок поворота;

4) Проверка на пересечение роботом границ заданного поля;

5) Вывод нового изображения робота на экран.

Весь процесс находится в цикле *while()*, пока не будет нажата клавиша «*esc*».

Для того, чтобы на экране не отображалось предыдущее изображение робота, создан отдельный объект класса Mat – общий фон. Главное изображение становится клоном общего фона, таким образом стирается предыдущее изображение робота, а затем уже на пустом экране рисуется следующее положение робота.

Класс *my_robot*

Для создания робота был создан класс *my_robot.* Данный класс обладает параметрами, показанными на рисунке 1.

```
private:
    Size2i m_area; //область движения
    float m_speed; //скорость движения в пикселях
    float m_angle; //текущий угол поворота
    Point2f m_center; //координаты центра
    float m_width; //ширина робота
    float m_height; //высота робота
    float m_wheelWidth; //ширина колес
    float m_wheelDiameter; //диаметр (= высота) колес
    float m_angularSpeed; //скорость поворота в градусах
};
```

Рисунок 1 – Параметры класса

Координаты центра устанавливаются с помощью функции *setCenter(const Mat& image)*, которая в начальный момент времени устанавливает центр робота в центр заданного поля.

Область движения устанавливается с помощью функции *setArea(const Mat& image)*, которая областью движения робота выбирает созданный объект класса *Mat*.

Все остальные параметры устанавливаются при вызове конструктора.

Основные функции

1) Функция *move()*

Сканирование нажатия клавиш продольного и поперечного движений и одновременно с этим вычисление новых координат центра робота осуществлено в функции *Move()*.

После нажатия какой-либо из клавиш происходит проверка на то, разрешено ли двигаться в этом направлении или нет, и, если нет, новые координаты центра сбрасываются.

Реализация функции представлена на рисунке 2.

```
int my_robot::move(int movecase)
{
    const Point2f prev_center = m_center;

    switch (movecase)
    {
    case 1: //w
        setPoints(difference_x:0, difference_y:m_speed);
        break;
    case 2: //s
        setPoints(difference_x:0, difference_y:-m_speed);
        break;
    case 3: //a
        setPoints(difference_x:m_speed, difference_y:0);
        break;
    case 4: //d
        setPoints(difference_x:-m_speed, difference_y:0);
        break;
    default:;
    }

    const Point2f new_borderDot[8] = { //граничные точки при угле наклона 0 и 45 градусов
        Point2f((-m_width / 2 - m_wheelWidth) * cos(_left:0) + m_center.x - (-m_height / 2) * sin(_left:0), //14
            _y:(-m_width / 2 - m_wheelWidth) * sin(_left:0) + m_center.y + (-m_height / 2) * cos(_left:0)),
        Point2f((m_width / 2 + m_wheelWidth) * cos(_left:0) + m_center.x - (-m_height / 2) * sin(_left:0), //15
            _y:(m_width / 2 + m_wheelWidth) * sin(_left:0) + m_center.y + (-m_height / 2) * cos(_left:0)),
        Point2f((-m_width / 2 - m_wheelWidth) * cos(_left:0) + m_center.x - (m_height / 2) * sin(_left:0), //4
            _y:(-m_width / 2 - m_wheelWidth) * sin(_left:0) + m_center.y + (m_height / 2) * cos(_left:0)),
        Point2f((m_width / 2 + m_wheelWidth) * cos(_left:0) + m_center.x - (m_height / 2) * sin(_left:0), //5
            _y:(m_width / 2 + m_wheelWidth) * sin(_left:0) + m_center.y + (m_height / 2) * cos(_left:0)),
        Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:45 / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:45 / 180 * PI), //14.1
            _y:(-m_width / 2 - m_wheelWidth) * sin(_x:45 / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:45 / 180 * PI)),
        Point2f((m_width / 2 + m_wheelWidth) * cos(_x:45 / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:45 / 180 * PI), //15.1
            _y:(m_width / 2 + m_wheelWidth) * sin(_x:45 / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:45 / 180 * PI)),
        Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:45 / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:45 / 180 * PI), //4.1
            _y:(-m_width / 2 - m_wheelWidth) * sin(_x:45 / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:45 / 180 * PI)),
        Point2f((m_width / 2 + m_wheelWidth) * cos(_x:45 / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:45 / 180 * PI), //5.1
            _y:(m_width / 2 + m_wheelWidth) * sin(_x:45 / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:45 / 180 * PI))
    };

    for (int i = 0; i < 8; i++)
    {
        if (new_borderDot[i].y < 0 || new_borderDot[i].x < 0 || new_borderDot[i].y > static_cast<float>(m_area.height) || new_borderDot[i].x > static_cast<float>(m_area.width))
        {
            m_center = prev_center;
        }
    }

    return 0;
}
```

Рисунок 2 – Реализация функции *move()*

Одной из особенностью функции *move()* является то, что начало системы координат изображения находится в верхнем левом углу, поэтому при движении вперед(клавиша «*w*») необходимо уменьшать *y*-координату центра робота.

2) Функция *rotate()*

Сканирование нажатия клавиш поворота и одновременно с этим вычисление нового угла поворота робота осуществлено в функции *rotate()*.

Для избежания возможного переполнения переменной значение угла сбрасывается каждый полный оборот.

Реализация показана на рисунке 3.

```
int my_robot::rotate(int movecase)
{
    m_angle += m_angularSpeed * movecase;

    if (m_angle >= 360)
    {
        m_angle -= 360;
    }

    if (m_angle <= 0)
    {
        m_angle += 360;
    }

    return 0;
}
```

Рисунок 3 – Обработка поворота

3) Функция *draw()*

Проверка на пересечение роботом границ заданного поля, а также вывод нового изображения робота на экран производится функцией *draw()*.

Модель робота с колесами представляет из себя 16 линий. Изначально в функции *draw()* вычисляются координаты точек каждой линии в глобальной системе координат, эти значения записываются в массив, из которого потом берутся необходимые две точки, по которым строится линия с помощью функции *line()*.

$$\left.\begin{aligned} x &= X\cos\alpha - Y\sin\alpha, \\ y &= X\sin\alpha + Y\cos\alpha. \end{aligned}\right\} \quad (1)$$

Массив полученных координат точек представлен на рисунке 4.

```
Point2f robodot[16] =
{
    Point2f(-m_width / 2 * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:m_angle / 180 * PI), //0
        _y:(-m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:m_angle / 180 * PI), //1
        _y:(m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((-m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:m_angle / 180 * PI),//2
        _y:(-m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:m_angle / 180 * PI), //3
        _y:(m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:m_angle / 180 * PI)),

    Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:m_angle / 180 * PI), //4
        _y:(-m_width / 2 - m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2 + m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2) * sin(_x:m_angle / 180 * PI), //5
        _y:(m_width / 2 + m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2 - m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //6
        _y:(-m_width / 2 - m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2 - m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((-m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2 - m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //7
        _y:(-m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2 - m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2 - m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //8
        _y:(m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2 - m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2 + m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (m_height / 2 - m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //9
        _y:(m_width / 2 + m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (m_height / 2 - m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),

    Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2 + m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //10
        _y:(-m_width / 2 - m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2 + m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((-m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2 + m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //11
        _y:(-m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2 + m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2 + m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //12
        _y:(m_width / 2) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2 + m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2 + m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2 + m_wheelDiameter) * sin(_x:m_angle / 180 * PI), //13
        _y:(m_width / 2 + m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2 + m_wheelDiameter) * cos(_x:m_angle / 180 * PI)),
    Point2f((-m_width / 2 - m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:m_angle / 180 * PI), //14
        _y:(-m_width / 2 - m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:m_angle / 180 * PI)),
    Point2f((m_width / 2 + m_wheelWidth) * cos(_x:m_angle / 180 * PI) + m_center.x - (-m_height / 2) * sin(_x:m_angle / 180 * PI), //15
        _y:(m_width / 2 + m_wheelWidth) * sin(_x:m_angle / 180 * PI) + m_center.y + (-m_height / 2) * cos(_x:m_angle / 180 * PI))
};
```

Рисунок 4 – Массив координат точек

Вывод

В ходе роботы проведено успешное моделирование движения робота по заданной области. Успешно изучены и использованы необходимые алгоритмы *OpenCV*.