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 *<подпись>* А.А. Ларионов

Преподаватель *<подпись>* В.В. Варлашин

«__»_____2020 г.

Санкт-Петербург
2020

ЗАДАНИЕ

Создать класс *C++*, позволяющий на изображении построить контур и промоделировать движения робота. Методы класса должны обеспечивать движение по прямой, в бок и поворот в пределах заданной рабочей плоскости. Для выполнения задания использовать библиотеку *OpenCV 4.0*.

РЕАЛИЗАЦИЯ КЛАССА

Ключевым элементом работы является класс *MyRobot* (листинг 1), состоящий из заголовочного файла *my_robot.h* и файла реализации *my_robot.cpp*.

Свойства класса определяют следующие параметры:

- характерные размеры робота (длина и ширина корпуса, ширина средства передвижения),
- линейную и угловую скорость робота,
- рабочую плоскость,
- положение робота на плоскости (координаты центра и угловых точек, угол поворота локальной системы координат робота относительно мировой).

Методы класса можно разделить на три группы: задающие параметры, возвращающие параметры и осуществляющие движение и отрисовку. При этом размеры робота указываются только однажды, т.е. в конструкторе.

К методам, задающим параметры, относятся *setSpeed, setAngularSpeed, setArea, setCenter, setAngle*. Значения скоростей также можно определить в конструкторе. Задание размеров рабочей плоскости и координат центра возможно двумя способами: через конкретные величины или посредством передачи изображения. При указании координат центра положение угловых точек в мировой системе высчитывается в соответствии с размерами робота. При задании угла происходит перерасчет координат угловых точек аналогично методу *rotate*, о котором речь пойдет далее.

К методам, возвращающим параметры, относятся *getSpeed, getAngularSpeed, getCenter, getAngle*. Информация, полученная с помощью них, может использоваться для вывода значений параметров в командную строку или на изображение рабочей плоскости. В работе применяется последний вариант (Приложение 1).

К методам, осуществляющим движение и отрисовку, относятся *move, rotate* и *draw*. Рассмотрим каждый в отдельности.

Листинг 1 – Класс *MyRobot*

```
class MyRobot
{
public:
    MyRobot();
    MyRobot(const float width, const float heignt, const float trackWidth, float speed,
float angularSpeed);
    ~MyRobot();
private:
    float m_width;
    float m_height;
    float m_trackWidth;

    float m_speed;
    float m_angularSpeed;

    Size2i m_area;

    Point2f m_center;
    Point2f cornerDots[8];
    float m_angle;
public:
    void setSpeed(float speed);
    void setAngularSpeed(float angularSpeed);
    void setArea(const Size2i area);
    int setArea(const Mat image);
    int setCenter(Mat image);
    int setCenter(float x, float y);
    void setAngle(float angle);

    float getSpeed();
    float getAngularSpeed();
    Point2f getCenter();
    float getAngle();

    int move(int straight, int sideways);
    int rotate(bool clockDirection);

    int draw(Mat& iImage, Mat& oImage);
};
```

1) Метод *move* позволяет осуществлять движение робота по прямой (вдоль локальной оси *y*) или в бок (вдоль локальной оси *x*) с одинаковой скоростью, равной заданной линейной, путем перерасчета координат центра и угловых точек. Математически это описывается следующим выражением

$$X_i = X_{i-1} + RU,$$

$$\begin{pmatrix} x_м \\ y_м \\ 1 \end{pmatrix}_i = \begin{pmatrix} x_м \\ y_м \\ 1 \end{pmatrix}_{i-1} + \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix},$$

где X_{i-1} и X_i – вектор однородных координат точки в мировой системе до и после перемещения соответственно, R – матрица поворота, U – вектор скорости в локальной системе координат робота, α – угол поворота локальной системы относительно мировой, рад.

Наличие и знак составляющих скорости по осям *x* и *y* локальной системы определяются значением параметров метода *sideways* и *straight* соответственно. Нулевое значение параметра означает отсутствие составляющей, положительное или отрицательное значение – положительный или отрицательный знак. Положительный знак соответствует направлению по оси.

Учтено ограничение перемещения робота заданной плоскостью – центр робота не может пересечь границу. Это реализовано пропорцией – для вычисленного перемещения высчитывается значение, которое робот действительно может пройти до пересечения с одной из границ.

Из-за отличия системы координат в *OpenCV* от принятой мировой необходимо инвертировать координаты точек и движение по оси *y*. Системы координат, используемые в работе, показаны на рисунке 1.

Код метода приведен в листинге 2.

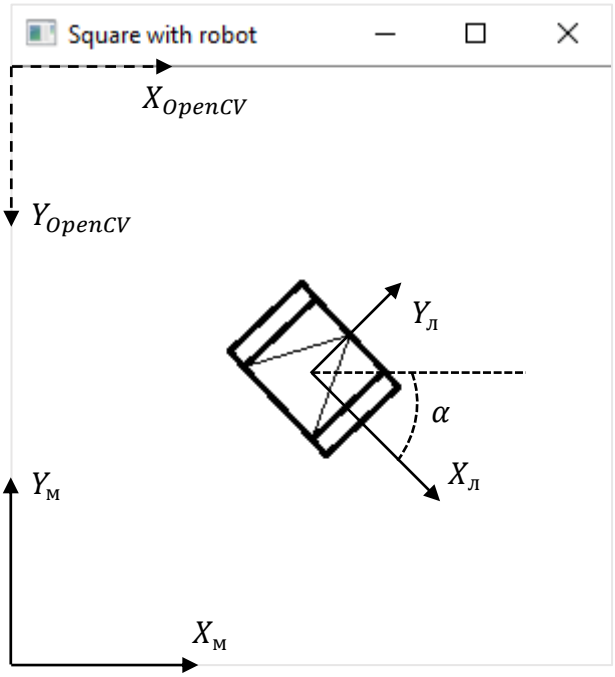

Рисунок 1 – Системы координат

Листинг 2 – Метод *move*

```
int MyRobot::move(int straight, int sideways)
{
    Mat rSpeed = (Mat_<float>(3, 3) << cos(m_angle), -sin(m_angle), 0,
                                       sin(m_angle), cos(m_angle), 0,
                                       0, 0, 1);

    float signS = (float)(sideways > 0) - (float)(sideways < 0);
    float signF = (float)(straight > 0) - (float)(straight < 0);

    Mat vecSpeed = (Mat_<float>(3, 3) << signS * m_speed * (float)(sideways != 0), 0, 0,
                                         signF * m_speed * (float)(straight != 0), 0, 0,
                                         1, 0, 0);

    Mat result;
    result = rSpeed * vecSpeed;

    float incrementX = result.at<float>(0, 0);
    float incrementY = (-1) * result.at<float>(1, 0);

    float previousX = incrementX;
    float previousY = incrementY;

    if ((m_center.x + incrementX) < 0)
    {
        incrementX -= (m_center.x + incrementX);
        incrementY = previousY * (incrementX / previousX);
    }
    if ((m_center.x + incrementX) > m_area.width)
    {
        incrementX -= (m_center.x + incrementX) - m_area.width;
        incrementY = previousY * (incrementX / previousX);
    }

    if ((m_center.y + incrementY) < 0)
    {
        incrementY -= (m_center.y + incrementY);
        incrementX = previousX * (incrementY / previousY);
    }
    if ((m_center.y + incrementY) > m_area.height)
    {
        incrementY -= (m_center.y + incrementY) - m_area.height;
        incrementX = previousX * (incrementY / previousY);
    }

    m_center.x += incrementX;
    m_center.y += incrementY;
    ...
}
```

2) Метод *rotate* позволяет роботу повернуться вокруг центра на произвольный угол с шагом, равным значению угловой скорости. Угловая скорость задана в радианах. Математически это описывается следующим выражением

$$X = Hx,$$

$$\begin{pmatrix} x_{\text{м}} \\ y_{\text{м}} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(-\alpha) & \sin(-\alpha) & r_x \\ \sin(-\alpha) & \cos(-\alpha) & r_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{\text{л}} \\ y_{\text{л}} \\ 1 \end{pmatrix},$$

где X – вектор однородных координат точки в мировой системе, x – вектор однородных координат точки в локальной системе, H – матрица перехода от локальной системы к мировой, r_x и r_y – координаты центра локальной системы в мировой.

Минус перед углом поворота необходим также из-за отличия системы координат в *OpenCV*.

Направление вращения задается параметром метода *isClockDirection*. При логической единице вращение происходит по часовой стрелке, при логическом нуле – против.

Код метода приведен в листинге 3.

Листинг 3 – Метод *rotate*

```
int MyRobot::rotate(bool isClockDirection)
{
    if (isClockDirection == true)
    {
        m_angle -= m_angularSpeed;
    }
    else
    {
        m_angle += m_angularSpeed;
    }

    if (m_angle > (float)CV_2PI)
    {
        m_angle -= (float)CV_2PI;
    }
    if (m_angle < (float)(-CV_2PI))
    {
        m_angle += (float)CV_2PI;
    }

    Mat rCoordinates = (Mat_<float>(3, 3) << cos(-m_angle), -sin(-m_angle), m_center.x,
                                             sin(-m_angle), cos(-m_angle), m_center.y,
                                             0, 0, 1);

    Point2f defaultCornerDots[8] =
{ Point2f((m_center.x - (m_width / 2)), (m_center.y - (m_height / 2))),
  Point2f((m_center.x + (m_width / 2)), (m_center.y - (m_height / 2))),
  Point2f((m_center.x + (m_width / 2)), (m_center.y + (m_height / 2))),
  Point2f((m_center.x - (m_width / 2)), (m_center.y + (m_height / 2))),
  Point2f(((m_center.x - (m_width / 2)) - m_trackWidth), (m_center.y - (m_height / 2))),
  Point2f(((m_center.x + (m_width / 2)) + m_trackWidth), (m_center.y - (m_height / 2))),
```

```
  Point2f(((m_center.x + (m_width / 2)) + m_trackWidth), (m_center.y + (m_height / 2))),
  Point2f(((m_center.x - (m_width / 2)) - m_trackWidth), (m_center.y + (m_height / 2)))
};

    for (int i = 0; i < 8; i++)
    {
        Mat localDefaultCoordinates = (Mat_<float>(3, 3) <<
                                       (defaultCornerDots[i].x - m_center.x), 0, 0,
                                       (defaultCornerDots[i].y - m_center.y), 0, 0,
                                       1, 0, 0);

        Mat result;
        result = rCoordinates * localDefaultCoordinates;

        cornerDots[i].x = result.at<float>(0, 0);
        cornerDots[i].y = result.at<float>(1, 0);
    }

    return 0;
}
```

3) Метод *draw* строит контур робота линиями между его угловыми точками на рабочей плоскости. Здесь же происходит вывод значений параметров робота (листинг 4). Для удобства определения направления движения к контуру добавлена стрелка.

Листинг 4 – Вывод параметров

```
...
putText(oImage, "X center:", Point2i(20, 20), FONT_HERSHEY_COMPLEX, 0.5, Scalar(125));
putText(oImage, to_string(getCenter().x), Point2i(220, 20), FONT_HERSHEY_COMPLEX, 0.5,
Scalar(125));
...
```

УПРАВЛЕНИЕ

Создание робота, т.е. объекта класса *MyRobot*, управление и вывод на экран описано в основном файле *main.cpp*. Управление может осуществляться клавишами клавиатуры. Пример из работы приведен в таблице 1.

Таблица 1 – Управление движением робота

Клавиша	Соответствующий метод класса	Значение параметра / диапазон значений параметра	Назначение
W	*move*	*straight* = 1, *sideways* = 0	Движение по оси *y*
S	*move*	*straight* = -1, *sideways* = 0	Движение против оси *y*
D	*move*	*straight* = 0, *sideways* = 1	Движение по оси *x*
A	*move*	*straight* = 0, *sideways* = -1	Движение против оси *x*
E	*rotate*	*true*	Вращение по часовой стрелке
Q	*rotate*	*false*	Вращение против часовой стрелки
R	*setSpeed*	0,1…50	Увеличение линейной скорости
F	*setSpeed*	0,1…50	Уменьшение линейной скорости
V	*setAngularSpeed*	0,0.1…1	Увеличение угловой скорости
C	*setAngularSpeed*	0,0.1…1	Уменьшение угловой скорости

Приложение 1

ПРИМЕР РАБОТЫ ПРОГРАММЫ