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 Демчева А.А.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург
2020

Задание

Написать программу, позволяющую вывести на фоновое изображение контур робота с учетом задаваемых в ходе выполнения программы перемещений и значения угла поворота робота. Перемещения осуществляются в плоскости изображения по вертикали и по горизонтали, поворот — на 360°. При перемещении контур робота не должен выходить за границы фонового изображения.

При выполнении использовать средства библиотеки *OpenCV 4.0*.

Выполнение задания

При выполнении задания были использованы возможности классов языка C++. Все функции были написаны в виде методов класса *MyRobot*, интерфейс которого был объявлен в заголовочном файле *my_robot.h*, а реализация описана в файле *my_robot.cpp*.

Параметры контура робота (ширина, длина корпуса робота, ширина и диаметр колес) задаются непосредственно при объявлении экземпляра класса. Параметры движения (фоновое изображение, величина границ, скорость и угловая скорость) — с помощью set-функций класса: *setArea, setBorders, setStartPoint, setSpeed, setAngularSpeed.*

Функции *setArea* и *setStartPoint* были перегружены, чтобы дать возможность пользователю задавать фоновое изображение объектом типа *Mat* или *Size2i*, а начальную точку положения робота — объектом типа *Mat* или непосредственно через координаты *x* и *y*.

В случае необходимости пользователь может получить значения скоростей, используя get-функции *getSpeed* и *getAngularSpeed*.

Движение робота осуществляется согласно следующему уравнению:

$$\begin{pmatrix} x_м \\ y_м \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & t_x \\ sin(\alpha) & cos(\alpha) & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_л \\ y_л \\ 1 \end{pmatrix} + \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix}$$

где $x_м$, $y_м$ — координаты точки в мировой системе, пикс; $x_л$, $y_л$ — в локальной системе, пикс; α — угол поворота локальной системы координат относительно мировой, рад; t_x, t_y — координаты центра локальной системы в мировой, пикс; v_x, v_y — скорость перемещения центра робота, пикс/шаг.

Мировой системой координат считается система координат изображения, заданная по умолчанию в OpenCV. Локальная система связана с центром робота, ось Oy направлена вверх, Ox — вправо.

Функция *setStartPoint* задает также положение угловых точек корпуса робота в локальной системе координат и начальное значение угла поворота в радианах.

При запуске программы в цикле while происходит считывание нажатой клавиши. В зависимости от того, какая именно клавиша была нажата, рассчитывается положение начала локальной системы координат или величина угла наклона этой системы координат относительно мировой (функция *move*). Учитывается тот факт, что в OpenCV начало системы координат фонового изображения расположено в левом верхнем углу. Затем рассчитываются координаты угловых точек корпуса робота. Если одна из них выходит за заданные в функции *setBorders* границы изображения, поднимается соответствующий флаг, движение робота в этом направлении останавливается до тех пор, пока не будет совершено перемещение в противоположную сторону.

Функция *draw* производит отрисовку контура робота на фоновом изображении по рассчитанным значениям координат угловых точек корпуса.

На рисунке 1 приведены результаты работы программы. Далее – листинги использованных функций.

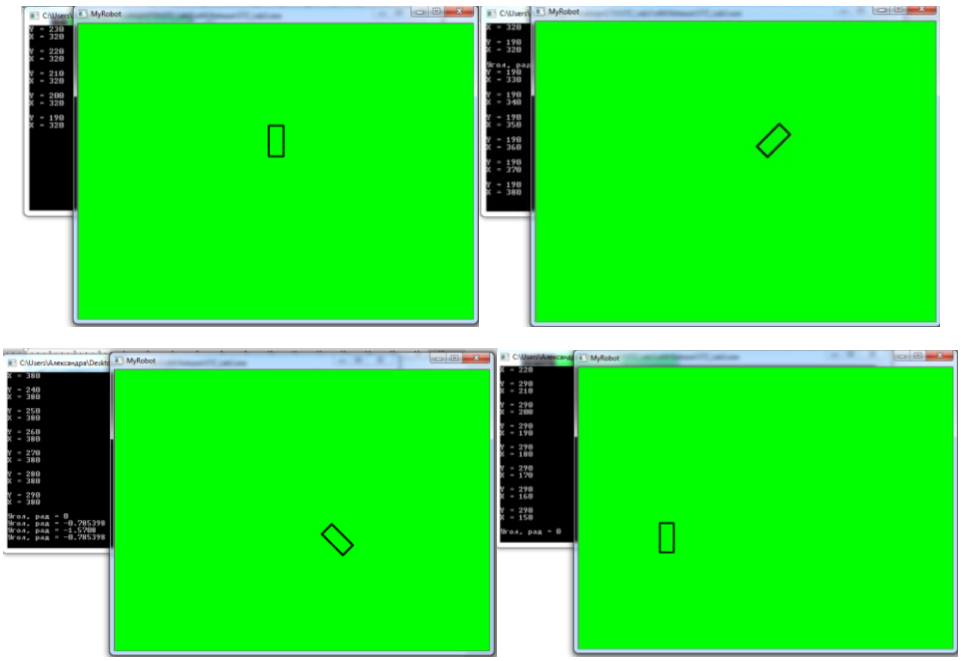

Рисунок 1 — Пример работы программы

Листинг 1 — Файл TVS_Lab1_DemchevaAA.cpp

```
#pragma once
#include <iostream>
#include "opencv2\core.hpp"
#include "opencv2\highgui.hpp"
#include "my_robot\my_robot.h"
using namespace cv;
using namespace std;

int main()
{
	setlocale(LC_ALL, "Russian");

	Size size(640, 480);
	Scalar color(0, 255, 0);
	Mat field(size, CV_8UC3, color);

	MyRobot robot(25, 50, 12, 12); //MyRobot(width, height, wheelWidth, wheelDiameter)

	robot.setArea(field);
	robot.setBorders(100,100);

	robot.setStartPoint(field);
	robot.setSpeed(10);
	robot.setAngularSpeed(CV_PI / 4);

	while(1)
	{
		char key = waitKey(10);
		robot.move(key);

		Mat newField = field.clone();
		robot.draw(newField);
		imshow("MyRobot", newField);
	}
	return 0;
}
```

Листинг 2 — Файл my_robot.h

```
#pragma once
#include <iostream>
#include "opencv2\core.hpp"
#include "opencv2\highgui.hpp"
#include "opencv2\imgproc.hpp"
using namespace cv;
using namespace std;

class MyRobot
{
public:
	MyRobot();
	MyRobot(int8_t width, int8_t height,
		float wheelWidth, float wheelDiameter);
	~MyRobot();

//Сеттеры
	void setSpeed(const float speed);
	void setAngularSpeed(const float angularSpeed);

	void setArea(const Size2i area);
	int setArea(Mat image);

	void setBorders(int16_t xBorder, int16_t yBorder);

	int setStartPoint(Mat image);
	int setStartPoint(float x, float y);

//Геттеры
	float getSpeed();
	float getAngularSpeed();

//Управление движением
	void move(char key);

//Отрисовка
	int draw(Mat& ioImage);

private:

	float m_angle; //угол поворота локальной системы координат робота

	Point2i m_center; //координаты центра робота

	Point2i m_pt1; //крайние точки корпуса робота в мировых координатах
	Point2i m_pt2;
	Point2i m_pt3;
	Point2i m_pt4;

	Point2i m_pt1_loc; //крайние точки корпуса робота
	Point2i m_pt2_loc;
	Point2i m_pt3_loc;
	Point2i m_pt4_loc;

	int8_t m_width; //ширина корпуса
	int8_t m_height; //длина корпуса

	int8_t m_halfWidth = m_width / 2;
	int8_t m_halfHeight = m_height / 2;

	int8_t m_wheelWidth;//ширина колес
	int8_t m_wheelDiameter; //диаметр колес
```

```
        float m_speed;
        float m_angularSpeed;

        Size2i m_area;

        int16_t  m_topBorder;
        int16_t  m_bottomBorder;
        int16_t  m_rightBorder;
        int16_t  m_leftBorder;

        bool m_isTBcrossed = false;
        bool m_isBBcrossed = false;
        bool m_isRBcrossed = false;
        bool m_isLBcrossed = false;
};
```

Листинг 3 — Файл my_robot.cpp

```
#pragma once
#include "my_robot.h"

MyRobot::MyRobot(int8_t width, int8_t height,
        float wheelWidth, float wheelDiameter):
        m_width(width),
        m_height(height),
        m_wheelWidth(),
        m_wheelDiameter()
{
        ;
};

MyRobot::~MyRobot()
{
        ;
};

//Сеттеры
void MyRobot::setSpeed(const float speed)
{
        m_speed = speed;
}

void MyRobot::setAngularSpeed(const float angularSpeed)
{
        m_angularSpeed = angularSpeed;
}

void MyRobot::setArea(const Size2i area)
{
        m_area = area;
}

int MyRobot::setArea(Mat image)
{
        if (image.empty())
        {
                return -1;
        }

        m_area.width = image.cols;
        m_area.height = image.rows;
```

```
	return 0;
}

void MyRobot::setBorders(int16_t xBorder, int16_t yBorder)
{
	m_topBorder = yBorder;
	m_bottomBorder = m_area.height - yBorder;

	m_leftBorder = xBorder;
	m_rightBorder = m_area.width - xBorder;

	return;
}

int MyRobot::setStartPoint(Mat image)
{
	if (image.empty())
	{
		return -1;
	}

	if ((image.cols / 2 > m_area.width) || (image.rows / 2 > m_area.height))
	{
		return -2;
	}

	m_center.x = (image.cols)/2;
	m_center.y = (image.rows)/2;

	m_pt1_loc.x = - m_halfWidth;
	m_pt1_loc.y = m_halfHeight;

	m_pt2_loc.x = m_halfWidth;
	m_pt2_loc.y = m_halfHeight;

	m_pt3_loc.x = m_halfWidth;
	m_pt3_loc.y = - m_halfHeight;

	m_pt4_loc.x = -m_halfWidth;
	m_pt4_loc.y = -m_halfHeight;

	m_angle = 0;

	return 0;
}

int MyRobot::setStartPoint(float x, float y)
{
	if ((x > m_area.width) || (y > m_area.height))
	{
		return -2;
	}

	m_center.x = x;
	m_center.y = y;

	m_pt1_loc.x = -m_halfWidth;
	m_pt1_loc.y = m_halfHeight;

	m_pt2_loc.x = m_halfWidth;
	m_pt2_loc.y = m_halfHeight;

	m_pt3_loc.x = m_halfWidth;
```

```
	m_pt3_loc.y = -m_halfHeight;

	m_pt4_loc.x = -m_halfWidth;
	m_pt4_loc.y = -m_halfHeight;

	m_angle = 0;

	return 0;
}

//Геттеры

float MyRobot::getSpeed()
{
	return m_speed;
}

float MyRobot::getAngularSpeed()
{
	return m_angularSpeed;
}

void MyRobot::move(char key)
{

	if (key == 56) //перемещение вверх 8
	{
		if (m_isTBcrossed == true)
		{
			cout << "Пересечение верхней границы" << endl;
		}
		else
		{
			m_center.y = m_center.y - m_speed;
			cout << "Y = " << m_center.y << endl;
			cout << "X = " << m_center.x << endl << endl;
			m_isBBcrossed = false;
		}
	}

	if (key == 50) //перемещение вниз 2
	{
		if (m_isBBcrossed == true)
		{
			cout << "Пересечение нижней границы" << endl;
		}
		else
		{
			m_center.y = m_center.y + m_speed;
			cout << "Y = " << m_center.y << endl;
			cout << "X = " << m_center.x << endl << endl;
			m_isTBcrossed = false;
		}
	}

	if (key == 52) //перемещение влево 4
	{
		if (m_isLBcrossed == true)
		{
			cout << "Пересечение левой границы" << endl;
		}
		else
		{
			m_center.x -= m_speed;
```

```
                cout << "Y = " << m_center.y << endl;
                cout << "X = " << m_center.x << endl << endl;
                m_isRBcrossed = false;
        }

    }

    if (key == 54) //перемещение вправо 6
{
        if (m_isRBcrossed == true)
        {
                cout << "Пересечение правой границы" << endl;
        }
        else
        {
                m_center.x += m_speed;
                cout << "Y = " << m_center.y << endl;
                cout << "X = " << m_center.x << endl << endl;
                m_isLBcrossed = false;
        }
}

    if (key == 51) // вращение против чс 3
{
     m_angle += m_angularSpeed;
            if (m_angle >= 6.28)
            {
                m_angle = 0;
            }
            cout << "Угол, рад = " << m_angle << endl;
}

    if (key == 49) //вращение по чс  1
{
     m_angle -= m_angularSpeed;
            if (m_angle <= -6.28)
            {
                m_angle = 0;
            }
            cout << "Угол, рад = " << m_angle << endl;
}

    m_pt1.x = m_pt1_loc.x * cos(m_angle) - m_pt1_loc.y * sin(m_angle) + m_center.x;
    m_pt1.y = m_pt1_loc.x * sin(m_angle) + m_pt1_loc.y * cos(m_angle) + m_center.y;

    m_pt2.x = m_pt2_loc.x * cos(m_angle) - m_pt2_loc.y * sin(m_angle) + m_center.x;
    m_pt2.y = m_pt2_loc.x * sin(m_angle) + m_pt2_loc.y * cos(m_angle) + m_center.y;

    m_pt3.x = m_pt3_loc.x * cos(m_angle) - m_pt3_loc.y * sin(m_angle) + m_center.x;
    m_pt3.y = m_pt3_loc.x * sin(m_angle) + m_pt3_loc.y * cos(m_angle) + m_center.y;

    m_pt4.x = m_pt4_loc.x * cos(m_angle) - m_pt4_loc.y * sin(m_angle) + m_center.x;
    m_pt4.y = m_pt4_loc.x * sin(m_angle) + m_pt4_loc.y * cos(m_angle) + m_center.y;


    //Проверка пересечения верхней границы
    if ((m_pt1.y < m_topBorder) || (m_pt2.y < m_topBorder) ||
            (m_pt3.y < m_topBorder) || (m_pt4.y < m_topBorder))
    {
            m_isTBcrossed = true;
    }

    //Проверка пересечения нижней границы
    if ((m_pt1.y > m_bottomBorder) || (m_pt2.y > m_bottomBorder) ||
```

```
		(m_pt3.y > m_bottomBorder) || (m_pt4.y > m_bottomBorder))
	{
		m_isBBcrossed = true;
	}

	//Проверка пересечения правой границы
	if ((m_pt1.x > m_rightBorder) || (m_pt2.x > m_rightBorder) ||
		(m_pt3.x > m_rightBorder) || (m_pt4.x > m_rightBorder))
	{
		m_isRBcrossed = true;
	}

	//Проверка пересечения левой границы
	if ((m_pt1.x < m_leftBorder) || (m_pt2.x < m_leftBorder) ||
		(m_pt3.x < m_leftBorder) || (m_pt4.x < m_leftBorder))
	{
		m_isLBcrossed = true;
	}

	return;
}

int MyRobot::draw(Mat& ioImage)
{

	Scalar lineColor(0, 0, 0);

	line(ioImage, m_pt1, m_pt2, lineColor, 2, 8, 0);
	line(ioImage, m_pt2, m_pt3, lineColor, 2, 8, 0);
	line(ioImage, m_pt3, m_pt4, lineColor, 2, 8, 0);
	line(ioImage, m_pt4, m_pt1, lineColor, 2, 8, 0);

	line(ioImage, m_pt1, m_pt5, lineColor, 2, 8, 0);
	line(ioImage, m_pt5, m_pt6, lineColor, 2, 8, 0);
	line(ioImage, m_pt6, m_pt7, lineColor, 2, 8, 0);

	return 0;
}
```