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:

где , — координаты точки в мировой системе, пикс; , — в локальной системе, пикс; — угол поворота локальной системы координат относительно мировой, рад; , — координаты центра локальной системы в мировой, пикс; , — скорость перемещения центра робота, пикс/шаг.

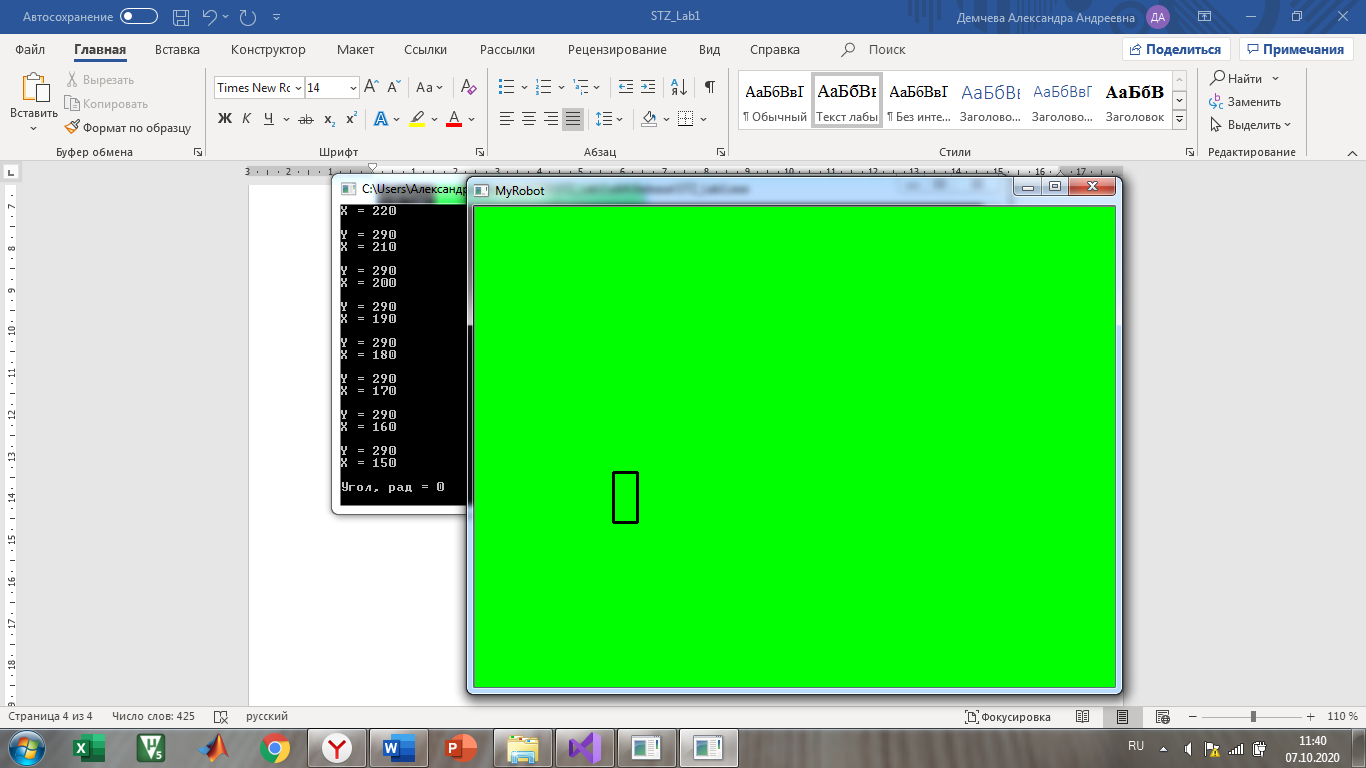
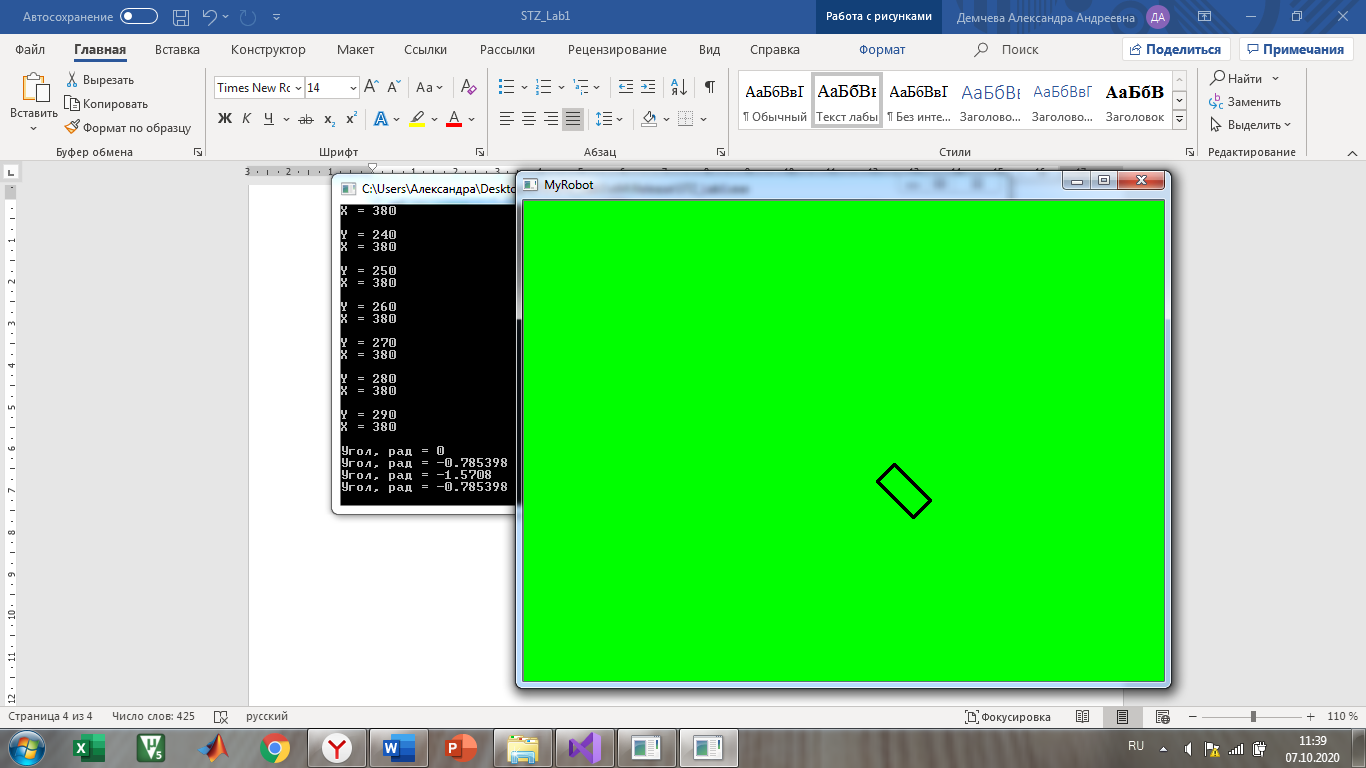
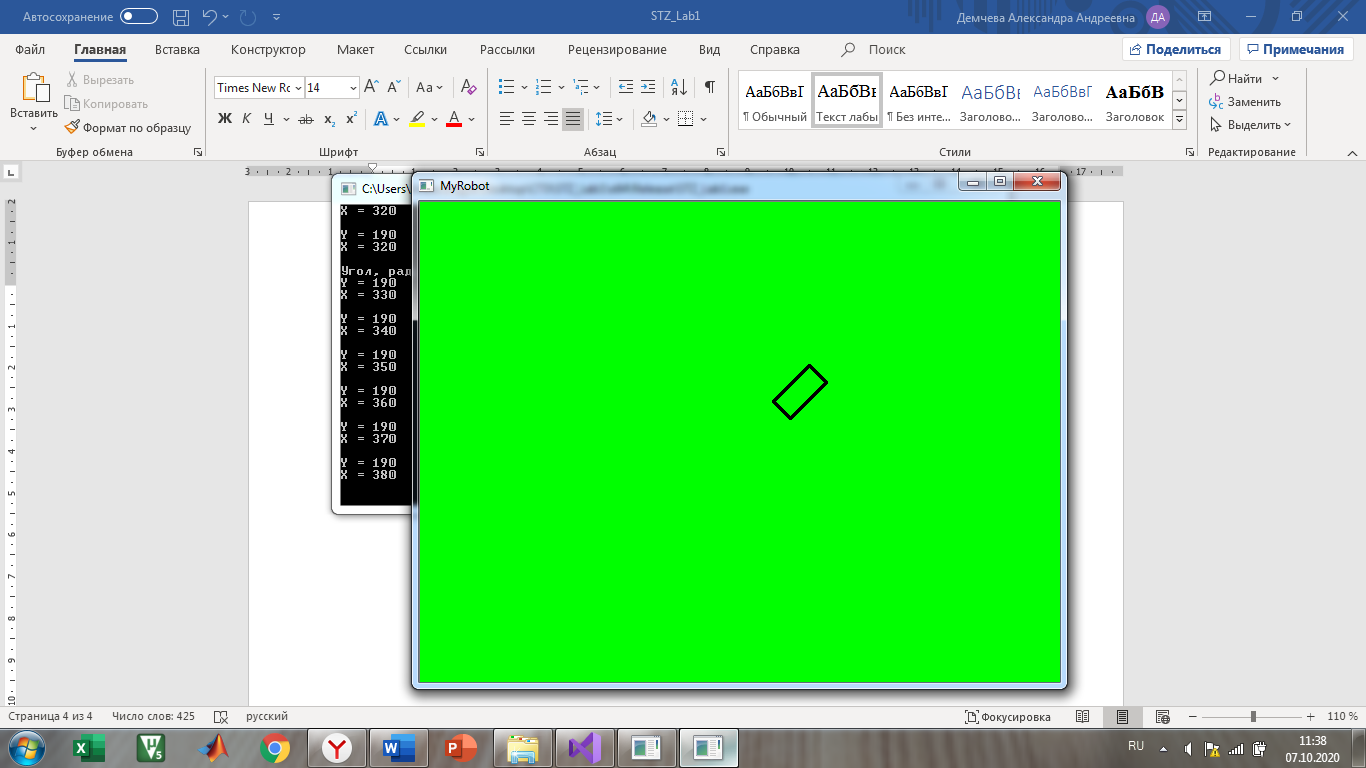
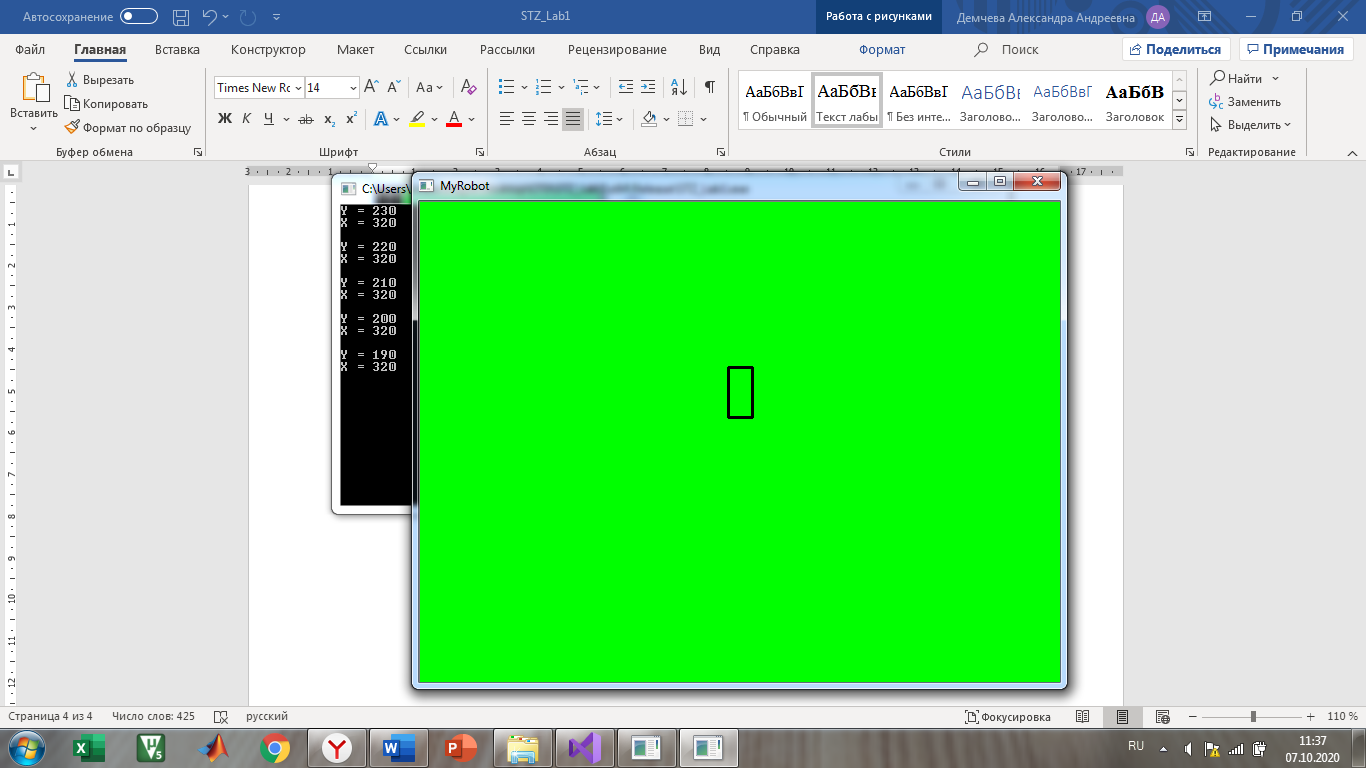
Мировой системой координат считается система координат изображения, заданная по умолчанию в OpenCV. Локальная система связана с центром робота, ось Oy направлена вверх, Ox — вправо.

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;

}