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

Студенты гр. 3331506/70401 Тестерева М.Н.
Каретный Я.М.
Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург
2020 г.

Задание

Необходимо создать программу, которая выводила бы анимацию движения роботелеги (нарисовать вид сверху схематично) по траектории, выбираемой пользователем: движение вперед, назад, влево, вправо, поворот по и против часовой стрелки. Движение должно быть ограничено размером выводимой области.

Дополнительно: реализовать произвольное движение роботелеги с отскоком от границ выводимого поля и изменением цвета корпуса робота в спектре HSV.

Выполнение задания

Отчёт выполнен по итогу дополнительного задания!

В файле проекта *My_Robot_2.cpp* содержится функция *main()*, где находятся:

- команды создания робота и рабочего поля;

```
8   int main()
9   {
10      const Scalar white(255, 255, 255);
11      const Scalar black(0, 0, 0);
12
13      //Задание размеров робота
14      float width = 100;
15      float lenght = 150;
16      float wheelWidth = 25;
17      float wheelDiameter = 40;
18
19      //Создание робота
20      MyRobot robot(width, lenght, wheelWidth, wheelDiameter, 0, 0, 0, 0, 0, 0, black);
21      robot.setSpeed(15);
22      robot.setAngularSpeed(10 * 3.14 / 180);
23      robot.setAngle(3.14*20);
24      robot.setFlag(0);
25      robot.setColorFlag(0);
26      robot.setAngleMotion(0);
27      robot.setColor(black);
28
29      //Создание поля и задание центра
30      Size size(600, 600);
31      Mat field(size, CV_8UC3, white);
32      robot.setArea(field);
33      robot.setCenter(field);
```

Scalar – класс, представляющий собой массив размером до 4-ёх элементов, который обычно используется для хранения значений цветов (в формате BGR).

Mat – класс, имеющий структуру одномерного массива, представленного матрицей, использующийся для хранения изображений.

- главная управляющая функция робота (пользователем) по ключевым клавишам:

```
37      //Основная функция управления
38      while (waitKey(25) != 27) //27 -- Esc
39      {
40          key = waitKey(25);
41
42          Mat imageOfRobot(size, CV_8UC3, white);
43
44          if ((key == 'w') || (key == 'W'))
45          {
46              robot.move(1);
47          }
48
49          if ((key == 'a') || (key == 'A'))
50          {
51              robot.move(2);
52          }
53
54          if ((key == 's') || (key == 'S'))
55          {
56              robot.move(3);
57          }
58
59          if ((key == 'd') || (key == 'D'))
60          {
61              robot.move(4);
62          }
63
64          //Поворот против часовой
65          if ((key == 'q') || (key == 'Q'))
66          {
67              robot.move(5);
68          }
69
70          //Поворот по часовой
71          if ((key == 'e') || (key == 'E'))
72          {
73              robot.move(6);
74          }
```

- функция позиционирования точек робота и его отрисовки, а далее – вывод изображения на экран:

```
76          //Отрисовка робота
77          robot.position_and_draw(field, imageOfRobot);
78          //Вывод изображения на экран
79          imshow("imageOfRobot", imageOfRobot);
80      }
```

В файле библиотеки *my_robot_2.h* задан класс *MyRobot*, его параметры и методы:

```
#pragma once

#include <iostream>
#include "opencv2/core.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <cmath>


using namespace cv;
using namespace std;

class MyRobot
{
public:
    MyRobot();
    MyRobot(float width, float heignt, float wheelWidth,
        float wheelDiameter, float speed, float angularSpeed, float angle,
        int flag, int color_flag, float angle_motion, Scalar color);
    ~MyRobot();

    //сеттеры
    void setAngle(float angle);
    void setFlag(int flag);
    void setColorFlag(int color_flag);
    void setColor(Scalar color);
    void setSpeed(const float speed);
    void setAngularSpeed(const float angularSpeed);
    void setAngleMotion(float angle_motion);
    void setArea(const Size2i area);
    int setArea(Mat image);
    int setCenter(Mat image);
    int setCenter(float x, float y);

    //геттеры
    float getSpeed(float speed);
    float getAngularSpeed(float angularSpeed);

    //движение робота
    int move(int8_t number);

    //позиционирование и отрисовка
    int position_and_draw(Mat& iImage, Mat& oImage);

    //проверка
    int check(float x, float y);

private:
    float m_speed;
    float m_angularSpeed;
    float m_angle;
    float m_width;
    float m_height;
    float m_wheelWidth;
    float m_wheelDiameter;
    int m_flag;

    Scalar m_color;
    int m_color_flag;
    float m_angle_motion;

    Size2i m_area;

    Point2f m_center;
};
```

В файле проекта my_robot_2.cpp с помощью сеттеров задаём используемые переменные:

```
19   //Задание флага поворота
20   void MyRobot::setFlag(int flag){ ... }
24
25   void MyRobot::setAngleMotion(float angle_motion){ ... }
29
30   //Задание флага для отслеживания изменения цвета робота
31   void MyRobot::setColorFlag(int color_flag){ ... }
35
36   void MyRobot::setColor(Scalar color){ ... }
40
41   //Задание линейной скорости робота
42   void MyRobot::setSpeed(const float speed){ ... }
46
47   //Задание угловой скорости робота
48   void MyRobot::setAngularSpeed(const float angularSeed){ ... }
52
53   //Задание угла поворота
54   void MyRobot::setAngle(float angle){ ... }
58
59   void MyRobot::setArea(const Size2i area){ ... }
63
64   //Определение размера поля
65   int MyRobot::setArea(Mat image){ ... }
77
78   //Определение центра
79   int MyRobot::setCenter(Mat image){ ... }
97
98   //Проверка расположения центра
99   int MyRobot::setCenter(float x, float y){ ... }
111
```

Также там определяем методы класса:

```
176  //Функция проверки, не вышел ли робот за границы поля
177  int MyRobot::check(float x, float y) {
178      if ((x > (int)m_speed) and (x < m_area.width - (int)m_speed)
179          and (y > (int)m_speed) and (y < m_area.height - (int)m_speed))
180      {
181          return 0;
182      }
183      else return -1;
184  }
```

Функция, перемещающая центр робота вдоль осей с заданной линейной скоростью, увеличивающая угол поворота робота и запоминающая направление движения.

```
125  //Функция движения робота - перемещение центра вдоль осей или увеличение угловой скорости
126 int MyRobot::move(int8_t number)
127 {
128     key_press = number;
129 
130     switch (number)
131     {
132         //вверх
133     case 1:
134         m_center.x = (m_center.x - m_speed * cos(3.14 / 2 - m_angle));
135         m_center.y = (m_center.y - m_speed * sin(3.14 / 2 - m_angle));
136 
137         break;
138 
139         //влево
140     case 2:
141         m_center.x = (m_center.x + m_speed * cos(3.14 + m_angle));
142         m_center.y = (m_center.y - m_speed * sin(3.14 + m_angle));
143 
144         break;
145 
146         //вниз
147     case 3:
148         m_center.x = (m_center.x + m_speed * cos(3.14 / 2 - m_angle));
149         m_center.y = (m_center.y + m_speed * sin(3.14 / 2 - m_angle));
150 
151         break;
152 
153         //вправо
154     case 4:
155         m_center.x = (m_center.x - m_speed * cos(3.14 + m_angle));
156         m_center.y = (m_center.y + m_speed * sin(3.14 + m_angle));
157 
158         break;
160         //для поворота против часовой
161     case 5:
162         m_angle = m_angle + m_angularSpeed;
163         m_flag = 2;
164         break;
165 
166         //для поворота по часовой
167     case 6:
168         m_angle = m_angle - m_angularSpeed;
169         m_flag = 1;
170         break;
171 
172     }
173     return 0;
174 };
```

Функция, совмещающая в себе позиционирование точек робота в соответствии с его расположением и его отрисовку.

Задаем необходимые переменные, проверяем размеры поля:

```
186  //Функция позиционирования и отрисовки
187 int MyRobot::position_and_draw(Mat& iImage, Mat& oImage)
188  {
189      //Задание резервных копий положений и скоростей
190      Point2i m_center_res;
191      m_center_res.x = m_center.x;
192      m_center_res.y = m_center.y;
193      float m_angle_res;
194      m_angle_res = m_angle;
195
196      //Характеристики линий
197      int8_t thickness = 3;
198      int8_t lineType = 8;
199      int8_t shift = 0;
200
201      if (iImage.empty()) { ... }
205
206      if ((iImage.cols != m_area.width) || (iImage.rows != m_area.height)) { ... }
210
211      //Задание точек углов робота и его колес
212      //Углы корпуса робота:
213      Point2i pointTopLeft;
214      Point2i pointTopRight;
215      Point2i pointBotLeft;
216      Point2i pointBotRight;
```

Задаем начальное положение точек робота, например, верхнего левого колеса/левого верхнего угла корпуса:

```
234      //Позиционирование точек робота относительно центра - начальное положение
235      //для верхнего левого колеса:
236      pointWheelTL1.x = m_center.x - m_width / 2 - m_wheelWidth;
237      pointWheelTL1.y = m_center.y - m_height / 2;
238      pointWheelBL1.x = m_center.x - m_width / 2 - m_wheelWidth;
239      pointWheelBL1.y = m_center.y - m_height / 2 + m_wheelDiameter;
240      pointWheelBR1.x = m_center.x - m_width / 2;
241      pointWheelBR1.y = m_center.y - m_height / 2 + m_wheelDiameter;
```

```
264      //для углов корпуса робота:
265
266      pointTopLeft.x = m_center.x - m_width / 2;
267      pointTopLeft.y = m_center.y - m_height / 2;
```

Изменяем позиции точек робота в зависимости от угла поворота, например, по часовой стрелке для левого верхнего колеса и левого верхнего угла корпуса:

```
//Перепозиционирование для точек робота при повороте
//по часовой стрелке:
if (m_flag == 1)

    //Вращение прямоугольников происходит по окружностям:
    //для углов корпуса робота:
    int radius = sqrt((float)pow((m_height / 2), 2) + (float)pow((m_width / 2), 2));
    //для колес робота:
    int radius_middle = sqrt((float)pow(((m_height / 2) - m_wheelDiameter), 2) + (float)pow(((m_width) / 2), 2)); //Средний диаметр
    int radius_max = sqrt((float)pow(((m_height) / 2), 2) + (float)pow(((m_width / 2) + m_wheelWidth), 2)); //Наибольший диаметр
    int radius_min = sqrt((float)pow(((m_height / 2) - m_wheelDiameter), 2) + (float)pow(((m_width / 2) + m_wheelWidth), 2)); //Наименьший диаметр

    pointTopLeft.x = m_center.x + radius * cos(m_angle + 3.14 - atan(m_height / m_width));
    pointTopLeft.y = m_center.y - radius * sin(m_angle + 3.14 - atan(m_height / m_width));
```

```
    pointWheelTL1.x = m_center.x + radius_max * cos(m_angle + 3.14 - atan((0.5 * m_height) / (0.5 * m_width + m_wheelWidth)));
    pointWheelTL1.y = m_center.y - radius_max * sin(m_angle + 3.14 - atan((0.5 * m_height) / (0.5 * m_width + m_wheelWidth)));
    pointWheelBL1.x = m_center.x + radius_min * cos(m_angle + 3.14 - atan((0.5 * m_height - m_wheelDiameter) / (0.5 * m_width + m_wheelWidth)));
    pointWheelBL1.y = m_center.y - radius_min * sin(m_angle + 3.14 - atan((0.5 * m_height - m_wheelDiameter) / (0.5 * m_width + m_wheelWidth)));
    pointWheelBR1.x = m_center.x + radius_middle * cos(m_angle + 3.14 - atan((0.5 * m_height - m_wheelDiameter) / (0.5 * m_width)));
    pointWheelBR1.y = m_center.y - radius_middle * sin(m_angle + 3.14 - atan((0.5 * m_height - m_wheelDiameter) / (0.5 * m_width)));
```

При столкновении с границами поля перезадаём направление движения в необходимую сторону и дополнительным движением совершаем отскок:

```
    //Перезадание направления движения при столкновении с границами поля + отскок
    if ((check(pointWheelTL1.x, pointWheelTL1.y) != 0) and (key_press == 1))
    {
        key_press = 4;
        move(key_press);
    }
```

Цикл, поддерживающий движение в заданном направлении до тех пор, пока робот внутри поля, иначе возвращающий его на шаг назад с помощью резервных копий переменных:

```
    //Общая проверка, не вышел ли робот за границы поля...
    if ((check((m_center.x - m_width / 2 - m_wheelWidth), (m_center.y - m_height / 2)) == 0) and
        (check((m_center.x + m_width / 2 + m_wheelWidth), (m_center.y - m_height / 2)) == 0) and
        (check((m_center.x + m_width / 2 + m_wheelWidth), (m_center.y + m_height / 2)) == 0) and
        (check((m_center.x - m_width / 2 - m_wheelWidth), (m_center.y + m_height / 2)) == 0) and
        (key_press != 5) and (key_press != 6))
    //...если нет, продолжаем движение в заданном направлении
    {
        move(key_press);
    }
    //...если да, то возвращаемся на шаг назад
    else {
        m_center.x = m_center_res.x;
        m_center.y = m_center_res.y;
        m_angle = m_angle_res;
    }
```

При столкновении робота с границей поля и сменой направления движения, цикл реализует смену цвета корпуса робота в спектре HSV:

```
443     //Смена цвета робота при столкновении с границей
444     if (key_press_res != key_press)
445     {
446         m_color_flag = m_color_flag + 10;
447         if (m_color_flag >= 360) { m_color_flag = 0; }
448         int Hue_color = m_color_flag;
449         int Hue_delenie = floor(Hue_color / 60);
450         int Saturation_color = 100;
451         int Value_color = 100;
452 
453         int V_min = ((100 - Saturation_color) * Value_color) / 100;
454         float a = (Value_color - V_min) * (Hue_color % 60) / 60;
455         int V_inc = V_min + (int)a;
456         int V_dec = Value_color - (int)a;
457 
458         Value_color = Value_color * 255 / 100;
459         V_min = V_min * 255 / 100;
460         V_inc = V_inc * 255 / 100;
461         V_dec = V_dec * 255 / 100;
462 
463         if ((Hue_delenie) == 0)
464         {
465             m_color[0] = V_min; m_color[1] = V_inc; m_color[2] = Value_color;
466             std::cout << "0) Red = " << Value_color << ", Green = " << V_inc << ", Blue = " << V_min << endl;
467         }
468         if ((Hue_delenie) == 1) { ... }
473         if ((Hue_delenie) == 2) { ... }
478         if ((Hue_delenie) == 3) { ... }
483         if ((Hue_delenie) == 4) { ... }
488         if ((Hue_delenie) == 5) { ... }
493     }
```

Отрисовка линий робота:

```
495     //Отрисовка
496     cv::line(oImage, pointTopLeft, pointTopRight, m_color, thickness, lineType, shift);
497     cv::line(oImage, pointTopLeft, pointBotLeft, m_color, thickness, lineType, shift);
498     cv::line(oImage, pointTopRight, pointBotRight, m_color, thickness, lineType, shift);
499     cv::line(oImage, pointBotLeft, pointBotRight, m_color, thickness, lineType, shift);
500 
501     cv::line(oImage, pointTopLeft, pointWheelTL1, m_color, thickness, lineType, shift);
502     cv::line(oImage, pointWheelTL1, pointWheelBL1, m_color, thickness, lineType, shift);
503     cv::line(oImage, pointWheelBL1, pointWheelBR1, m_color, thickness, lineType, shift);
504 
505     cv::line(oImage, pointTopRight, pointWheelTR2, m_color, thickness, lineType, shift);
506     cv::line(oImage, pointWheelTR2, pointWheelBR2, m_color, thickness, lineType, shift);
507     cv::line(oImage, pointWheelBR2, pointWheelBL2, m_color, thickness, lineType, shift);
508 
509     cv::line(oImage, pointBotRight, pointWheelBR3, m_color, thickness, lineType, shift);
510     cv::line(oImage, pointWheelBR3, pointWheelTR3, m_color, thickness, lineType, shift);
511     cv::line(oImage, pointWheelTR3, pointWheelTL3, m_color, thickness, lineType, shift);
512 
513     cv::line(oImage, pointBotLeft, pointWheelBL4, m_color, thickness, lineType, shift);
514     cv::line(oImage, pointWheelBL4, pointWheelTL4, m_color, thickness, lineType, shift);
515     cv::line(oImage, pointWheelTL4, pointWheelTR4, m_color, thickness, lineType, shift);
516 
517     return 0;
518 }
```

Реализация программы: