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 Сомов А. С.

Преподаватель Варлашин В. В.

«13» 10 2020 г.

Санкт-Петербург

2020

my_robot.h

```
#ifndef
TEST_MY_ROBOT_H
                #define TEST_MY_ROBOT_H

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

                using namespace std;
                using namespace cv;

                class Robot
                {

                public:
                    Robot() = default;

                    Robot(const Size2f m_areaSize, const Size2f robotBodySize, const Size2f
                &m_robotWheelSize, float motionShift,
                          float rotationShift);

                    void playRobot();

                    void drownRobot(Scalar color);

                private:
                    const Size m_areaSize;
                    const Size m_robotBodySize;
                    float m_robotRotationShift;
                    int m_robotSpeed;
                    int m_robotRotationSpeed;
                    float m_robotMotionShift;
                    float m_robotRotateAngle;
                    float m_currentAngle;
                    float m_currentTowerAngle;
                    float m_rotationMatrix[6];
                    int m_directionOfRotation;
                    float m_tmpAngle;
                    float m_tmpAngleTower;
                    Point2f m_tmpRobotBody[28];
                    Point2f m_robotBody[17];
                    Point2f m_robotTower[11];
                    Mat img;
```

```
    Scalar robotColor;
    Scalar color;
//Функция обеспечивающая поворот робота относительно его центра
    void robotRotation(Point2f *array, int length);
//Функция, обеспечивающая считывание нажатий клавишь и запуска необходимых
функций
    void robotMotion();
//Функция, проверяющая выход робота за пределы фона
    void borderCheck();
//Функция сохраняющая координаты робота до их изменения
    void saveArray();
//Функция, осуществляющая движение вперёд
    void forwardMove(Point2f *array, int length);
//Функция, осуществляющая движение назад
    void backMove(Point2f *array, int length);
//Функция, осуществляющая движение вправо
    void rightMove(Point2f *array, int length);
//Функция, осуществляющая движение влево
    void leftMove(Point2f *array, int length);

};

#endif //TEST_MY_ROBOT_H
```

main.cpp

```
#include
"my_robot/my_robot.h"

int main() {
    Size2f areaSize(1920, 1080);
    Size2f robotBodySize(80, 200);
    Size2f robotWheelSize(30, 30);
    Scalar robotColor = {0, 0, 0};
    Scalar color = {0, 200, 0};
    Robot robot(areaSize, robotBodySize, robotWheelSize, 10, 3);
    robot.playRobot();

}
```

my_robot.cpp

```
#include
"my_robot.h"


Robot::Robot(const Size2f m_areaSize, const Size2f robotBodySize, const Size2f
&m_robotWheelSize, float motionShift,
            float rotationShift) :
    m_robotMotionShift(motionShift),
    m_robotRotateAngle(rotationShift * M_PI / 180),
    m_directionOfRotation(1),
    m_currentAngle(0),
    m_areaSize(m_areaSize),
    m_robotRotationShift(rotationShift),
    m_robotBodySize(robotBodySize),
    m_currentTowerAngle(0),
    m_robotSpeed(m_robotSpeed),
    m_robotRotationSpeed(m_robotRotationSpeed)
{
    //Задаём координаты робота относительно центра фона
    m_robotBody[0].x = (m_areaSize.width / 2) - (m_robotBodySize.width / 2);
    m_robotBody[3].x = (m_areaSize.width / 2) - (m_robotBodySize.width / 2);
    m_robotBody[1].x = (m_areaSize.width / 2) + (m_robotBodySize.width / 2);
    m_robotBody[2].x = (m_areaSize.width / 2) + (m_robotBodySize.width / 2);

    m_robotBody[0].y = (m_areaSize.height / 2) + (m_robotBodySize.height / 2);
    m_robotBody[1].y = (m_areaSize.height / 2) + (m_robotBodySize.height / 2);
    m_robotBody[2].y = (m_areaSize.height / 2) - (m_robotBodySize.height / 2);
    m_robotBody[3].y = (m_areaSize.height / 2) - (m_robotBodySize.height / 2);

    m_robotBody[4].x = m_robotBody[0].x - m_robotWheelSize.width;
    m_robotBody[5].x = m_robotBody[0].x - m_robotWheelSize.width;
    m_robotBody[6].x = m_robotBody[0].x;
    m_robotBody[13].x = m_robotBody[0].x - m_robotWheelSize.width;
    m_robotBody[14].x = m_robotBody[0].x - m_robotWheelSize.width;
    m_robotBody[15].x = m_robotBody[0].x;

    m_robotBody[7].x = m_robotBody[1].x + m_robotWheelSize.width;
    m_robotBody[8].x = m_robotBody[1].x + m_robotWheelSize.width;
    m_robotBody[9].x = m_robotBody[1].x;
    m_robotBody[10].x = m_robotBody[1].x + m_robotWheelSize.width;
    m_robotBody[11].x = m_robotBody[1].x + m_robotWheelSize.width;
    m_robotBody[12].x = m_robotBody[1].x;

    m_robotBody[4].y = m_robotBody[0].y;
    m_robotBody[5].y = m_robotBody[0].y - m_robotWheelSize.height;
    m_robotBody[6].y = m_robotBody[0].y - m_robotWheelSize.height;
```

```
        m_robotBody[7].y = m_robotBody[0].y;
        m_robotBody[8].y = m_robotBody[0].y - m_robotWheelSize.height;
        m_robotBody[9].y = m_robotBody[0].y - m_robotWheelSize.height;

        m_robotBody[10].y = m_robotBody[2].y;
        m_robotBody[11].y = m_robotBody[2].y + m_robotWheelSize.height;
        m_robotBody[12].y = m_robotBody[2].y + m_robotWheelSize.height;
        m_robotBody[13].y = m_robotBody[2].y;
        m_robotBody[14].y = m_robotBody[2].y + m_robotWheelSize.height;
        m_robotBody[15].y = m_robotBody[2].y + m_robotWheelSize.height;
        m_robotBody[16].x = m_robotBody[0].x + (m_robotBodySize.width / 2);
        m_robotBody[16].y = m_robotBody[0].y - (m_robotBodySize.height / 2);
        /*****************************************************************/
        m_robotTower[0].x = m_robotBody[16].x - (m_robotBodySize.width / 4);
        m_robotTower[1].x = m_robotBody[16].x + (m_robotBodySize.width / 4);
        m_robotTower[2].x = m_robotBody[16].x + (m_robotBodySize.width / 3);
        m_robotTower[3].x = m_robotBody[16].x + (m_robotBodySize.width / 4);
        m_robotTower[4].x = m_robotBody[16].x - (m_robotBodySize.width / 4);
        m_robotTower[5].x = m_robotBody[16].x - (m_robotBodySize.width / 3);

        m_robotTower[0].y = m_robotBody[16].y - (m_robotBodySize.height / 6);
        m_robotTower[1].y = m_robotBody[16].y - (m_robotBodySize.height / 6);
        m_robotTower[3].y = m_robotBody[16].y + (m_robotBodySize.height / 6);
        m_robotTower[4].y = m_robotBody[16].y + (m_robotBodySize.height / 6);
        m_robotTower[2].y = m_robotBody[16].y;
        m_robotTower[5].y = m_robotBody[16].y;

        m_robotTower[6].x = m_robotBody[16].x - (m_robotBodySize.width / 16);
        m_robotTower[7].x = m_robotBody[16].x - (m_robotBodySize.width / 16);
        m_robotTower[8].x = m_robotBody[16].x + (m_robotBodySize.width / 16);
        m_robotTower[9].x = m_robotBody[16].x + (m_robotBodySize.width / 16);
        m_robotTower[10].x = m_robotBody[16].x;

        m_robotTower[6].y = m_robotTower[0].y;
        m_robotTower[7].y = m_robotTower[0].y - (m_robotBodySize.width / 3);
        m_robotTower[8].y = m_robotTower[0].y - (m_robotBodySize.width / 3);
        m_robotTower[9].y = m_robotTower[0].y;
        m_robotTower[10].y = m_robotBody[16].y;

        //Задаём цвета, матрицу поворота, а также настройки фона
        robotColor = {0, 0, 0};
        color = {0, 100, 0};
        img = {m_areaSize, CV_8UC3, color};
        m_rotationMatrix[0] = cos(m_robotRotateAngle);
        m_rotationMatrix[1] = -sin(m_robotRotateAngle);
        m_rotationMatrix[2] =
                m_robotBody[16].x * (1 - cos(m_robotRotateAngle)) +
    m_robotBody[16].y * sin(m_robotRotateAngle);
```

```
    m_rotationMatrix[3] = sin(m_robotRotateAngle);
    m_rotationMatrix[4] = cos(m_robotRotateAngle);
    m_rotationMatrix[5] =
            m_robotBody[16].y * (1 - cos(m_robotRotateAngle) -
m_robotBody[16].x * sin(m_robotRotateAngle));

};

void Robot::drownRobot(Scalar color)
{
    line(img, m_robotBody[0], m_robotBody[1], color, 2);
    line(img, m_robotBody[1], m_robotBody[2], color, 2);
    line(img, m_robotBody[2], m_robotBody[3], color, 2);
    line(img, m_robotBody[3], m_robotBody[0], color, 2);

    line(img, m_robotBody[2], m_robotBody[11], color, 2);
    line(img, m_robotBody[3], m_robotBody[14], color, 2);

    line(img, m_robotBody[0], m_robotBody[4], color, 2);
    line(img, m_robotBody[4], m_robotBody[5], color, 2);
    line(img, m_robotBody[5], m_robotBody[6], color, 2);

   line(img, m_robotBody[1], m_robotBody[7], color, 2);
   line(img, m_robotBody[7], m_robotBody[8], color, 2);
   line(img, m_robotBody[8], m_robotBody[9], color, 2);

  line(img, m_robotBody[2], m_robotBody[10], color, 2);
  line(img, m_robotBody[10], m_robotBody[11], color, 2);
  line(img, m_robotBody[11], m_robotBody[12], color, 2);

  line(img, m_robotBody[3], m_robotBody[13], color, 2);
  line(img, m_robotBody[13], m_robotBody[14], color, 2);
  line(img, m_robotBody[14], m_robotBody[15], color, 2);

 line(img, m_robotTower[0], m_robotTower[1], color, 2);
 line(img, m_robotTower[1], m_robotTower[2], color, 2);
 line(img, m_robotTower[2], m_robotTower[3], color, 2);
 line(img, m_robotTower[3], m_robotTower[4], color, 2);
 line(img, m_robotTower[4], m_robotTower[5], color, 2);
 line(img, m_robotTower[5], m_robotTower[0], color, 2);
 line(img, m_robotTower[6], m_robotTower[7], color, 2);
 line(img, m_robotTower[7], m_robotTower[8], color, 2);
 line(img, m_robotTower[8], m_robotTower[9], color, 2);
 line(img, m_robotTower[9], m_robotTower[6], color, 2);

 if (color == robotColor)
 {
     imshow("Empty window", img);
```

```
    }
}

void Robot::robotMotion()
{
    while (true)
    {
        //ожидаем нажатия
        switch (waitKey(60))
        {
            case 'w':
                //закрашиваем робота цветом фона
                drownRobot(color);
                //сохраняем текущие кординаты точек робота
                saveArray();
                //вызываем функции движения в нужную сторону
                //тела робота
                forwardMove(m_robotBody, 17);
                //башни робота
                forwardMove(m_robotTower, 11);
                //проверяем пересечение границ крайними угловыми точками робота
                borderCheck();
                //отрисовываем робота на фоне
                drownRobot(robotColor);
                break;
            case 's':
                drownRobot(color);
                saveArray();
                backMove(m_robotBody, 17);
                backMove(m_robotTower, 11);
                borderCheck();
                drownRobot(robotColor);
                break;
            case 'd':
                drownRobot(color);
                saveArray();
                rightMove(m_robotBody, 17);
                rightMove(m_robotTower, 11);
                borderCheck();
                drownRobot(robotColor);
                break;
            case 'a':
                drownRobot(color);
                saveArray();
                leftMove(m_robotBody, 17);
                leftMove(m_robotTower, 11);
                borderCheck();
                drownRobot(robotColor);
```

```
                break;
            case 27:
                exit(0);
        }

        switch (waitKey(60))
        {
            case 'e':
                //выбор коэфффициента поворота по часосвой или против часовой
стрелки
                m_directionOfRotation = 1;
                //сохраняем координаты робота
                saveArray();
                //отслеживание текущего угла поворота
                m_currentAngle -= m_robotRotationShift;
                //закраска робота на фоне
                drownRobot(color);
                //пересчёт координат при путём умножения на матрицу поворота
                robotRotation(m_robotBody, 17);
                break;
            case 'q':
                m_directionOfRotation = -1;
                saveArray();
                m_currentAngle += m_robotRotationShift;
                drownRobot(color);
                robotRotation(m_robotBody, 17);
                break;
            case 'j':
                m_directionOfRotation = -1;
                m_currentTowerAngle += m_robotRotationShift;
                drownRobot(color);
                saveArray();
                robotRotation(m_robotTower, 11);
                break;
            case 'l':
                m_directionOfRotation = 1;
                m_currentTowerAngle += m_robotRotationShift;
                drownRobot(color);
                saveArray();
                robotRotation(m_robotTower, 11);
                break;
        }

    }
}

void Robot::robotRotation(Point2f *array, int length)
{
```

```
    float tmp_x = 0, tmp_y = 0;
    for (int i = 0; i < length - 1; i++)
    {
        array[i].x -= array[length - 1].x;
        array[i].y -= array[length - 1].y;
        tmp_x = array[i].x;
        tmp_y = array[i].y;
        array[i].x = (tmp_x * cos(m_robotRotateAngle) + tmp_y *
m_directionOfRotation * (-sin(m_robotRotateAngle)));
        array[i].y = (tmp_x * m_directionOfRotation * sin(m_robotRotateAngle) +
tmp_y * (cos(m_robotRotateAngle)));
        array[i].x += array[length - 1].x;
        array[i].y += array[length - 1].y;
    }
    borderCheck();
    drownRobot(robotColor);
}


void Robot::borderCheck()
{
    for (int i = 4; i < 14; i += 3)
    {
        if ((m_robotBody[i].x >= m_areaSize.width) || (m_robotBody[i].x <= 0)
||
            (m_robotBody[i].y >= m_areaSize.height) || m_robotBody[i].y <= 0)
        {
            for (int j = 0; j < 17; j++)
            {
                m_robotBody[j] = m_tmpRobotBody[j];
            }
            for (int j = 17, g = 0; j < 28; j++, g++)
            {
                m_robotTower[g] = m_tmpRobotBody[j];
            }
            m_currentAngle = m_tmpAngle;
            m_currentTowerAngle = m_tmpAngleTower;
            return;
        }
    }
}

void Robot::saveArray()
{
    for (int i = 0; i < 17; i++)
    {
        m_tmpRobotBody[i] = m_robotBody[i];
    }
```

```
    for (int i = 17, j = 0; i < 28; i++, j++)
    {
        m_tmpRobotBody[i] = m_robotTower[j];
    }
    m_tmpAngle = m_currentAngle;
    m_tmpAngleTower = m_currentTowerAngle;
}

void Robot::playRobot()
{
    drownRobot(robotColor);
    robotMotion();
}

void Robot::forwardMove(Point2f *array, int length)
{
    for (int i = 0; i < length; i++)
    {
        if (sin((m_currentAngle) * M_PI / 180) == 0)
        {
            array[i].y -= m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == 1)
        {
            array[i].x -= m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == (-1))
        {
            array[i].x += m_robotMotionShift;
        } else
        {
            array[i].y -= m_robotMotionShift * cos((m_currentAngle) * M_PI /
180);
            array[i].x -= m_robotMotionShift * sin((m_currentAngle) * M_PI /
180);
        }
    }
}

void Robot::backMove(Point2f *array, int length)
{
    for (int i = 0; i < length; i++)
    {
        if (sin((m_currentAngle) * M_PI / 180) == 0)
        {
            array[i].y += m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == 1)
        {
            array[i].x += m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == (-1))
```

```
        {
            array[i].x -= m_robotMotionShift;
        } else
        {
            array[i].y += m_robotMotionShift * cos((m_currentAngle) * M_PI /
180);
            array[i].x += m_robotMotionShift * sin((m_currentAngle) * M_PI /
180);
        }
    }
}

void Robot::rightMove(Point2f *array, int length)
{
    for (int i = 0; i < length; i++)
    {
        if (sin((m_currentAngle) * M_PI / 180) == 0)
        {
            array[i].x += m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == 1)
        {
            array[i].y -= m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == (-1))
        {
            array[i].y += m_robotMotionShift;
        } else
        {
            array[i].y -= m_robotMotionShift * sin((m_currentAngle) * M_PI /
180);
            array[i].x += m_robotMotionShift * cos((m_currentAngle) * M_PI /
180);
        }
    }
}

void Robot::leftMove(Point2f *array, int length)
{
    for (int i = 0; i < length; i++)
    {
        if (sin((m_currentAngle) * M_PI / 180) == 0)
        {
            array[i].x -= m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == 1)
        {
            array[i].y += m_robotMotionShift;
        } else if (sin((m_currentAngle) * M_PI / 180) == (-1))
        {
            array[i].y -= m_robotMotionShift;
```

```
        } else
        {
            array[i].y += m_robotMotionShift * sin((m_currentAngle) * M_PI /
180);
            array[i].x -= m_robotMotionShift * cos((m_currentAngle) * M_PI /
180);
        }
    }
}
```