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 Патрушева А.И.

Преподаватель Варлашин В.В.

« »\_\_\_\_\_\_\_\_2020 г.

Санкт-Петербург

2020

**Задание**

Методами OpenCV реализовать движение и поворот робота.

**Алгоритм работы**

1. Сканирование нажатых кнопок;
2. Расчет положения центра робота в зависимости от нажатой кнопки;
3. Создание и поворот всех точек корпуса робота;
4. Соединение точек линиями и вывод изображения.

**Описание робота**

Для создания робота был создан класс *MyRobo,* его параметры представлены ниже:

cv::Size2i m\_area; //размер фона

float m\_speed; //скорость движения

float m\_angularSpeed; //скорость поворота

cv::Point2f m\_center; //к-ты центра прямоугольника

float m\_angle; //угол поворота робота

float m\_width; //ширина корпуса

float m\_height; //длина корпуса

float m\_wheelWidth; //ширина колес

float m\_wheelDiameter; //диаметр колес

cv::Size2i m\_restrictedArea; //границы

При создании класса вызывается конструктор, в котором заполняются все параметры робота.

**Сканирование кнопок**

Сканирование кнопок происходит в цикле *while* до момента пока пользователь не нажмет кнопку с кодом 27(Esc).

while (waitKey(25) != 27)

{

key = cv::waitKey(5);

Mat imageWithRobot(size, CV\_8UC3, white);

if ((key == 'w') || (key == 'W'))

{

robot.move(1);

}

Каждый раз заходя в цикл создается новый задний фон с помощью команды:

Mat imageWithRobot (size, CV\_8UC3, white);

Задний фон представляет из себя матрицу размером size, заполненную пикселями белого цвета.

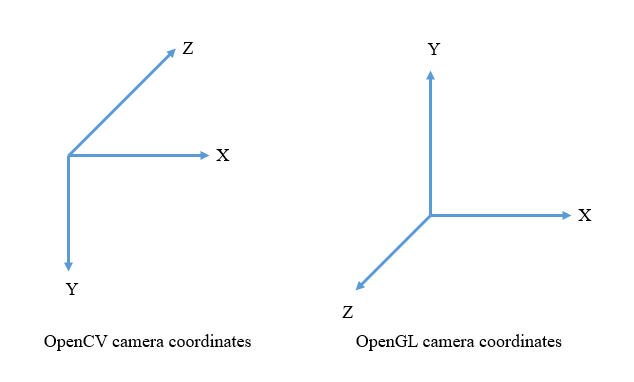
Scalar white(255, 255, 255);

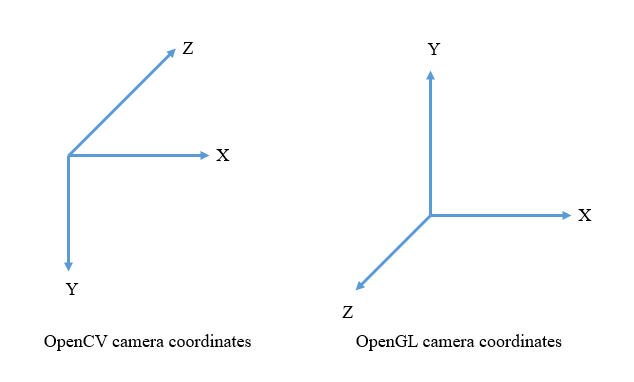
Size size(1600, 900);

Далее начинается сканирование кнопок и, в зависимости от нажатой кнопки вызывается или функция *move( )* или *rotate( )*, движение и поворот соответственно.

**Расчет положения центра робота**

Для перемещения центра робота используется функция *move( ),* которая рассчитывает центр координат в глобальной системе координат в зависимости от нажатой кнопки. Ниже изображены две системы координат: XY ­­— система координат изображений в OpenCV, X’Y’ ­­— система координат робота.





Y‘

**Y‘**

Рассмотрим часть кода с преобразованием координат на примере движения вверх:

void MyRobo::move(int8\_t number)

{

if (number == 1)

{

m\_center.x = m\_center.x + m\_speed \* sin(m\_angle);

m\_center.y = m\_center.y - m\_speed \* cos(m\_angle);

break;

}

В данном случае, двигаясь в системе координат робота вверх, в системе координат изображения мы двигаемся в сторону уменьшения Y, поэтому из

координаты центра будет вычитаться скорость движения.

Так как задание предполагает поворот робота, в выражения добавляется умножение скорости на *sin* и *cos.*

Функция *checkBorders( )* используется для запрещения роботу выходить за границы изображения:

int8\_t MyRobot::checkBorders()

{

if (m\_center.y > (m\_area.height - m\_barrier.height))

{

return 2; //движение вверх больше невозможно

}

if ((point.x > (m\_restrictedArea.width)) || (point.y > (m\_restrictedArea.height)) || (point.x < 100) || (point.y < 100))

{

return -1;

}

else

{

//std::cout << point.x << '\n';

//std::cout << point.y << '\n';

return 0;

}

Ограничивается некая зона на расстоянии от края изображения, за которую не может выходить центр робота. Если центр робота пересек эту границу, то функция возвращает передает ошибку, в функции *move( )* присваиваются предыдущие значения центра:

if (checkBorders(m\_center) == (-1))

{

std::cout << 1 << '/n';

m\_center = previousValue;

}

**Поворот робота**

Для поворота робота используется функция *rotate()*, чья логика аналогична функции *move():*

int32\_t MyRobo::rotate(int8\_t number)

{

switch (number)

{

case 1:

{

m\_angle = m\_angle – m\_angularSpeed;

break;

}

Увеличение и уменьшение угла задается с помощью константы *m\_angularSpeed* при нажатии определённых кнопок.

**Отрисовка робота**

Для отображения робота реализованы функции *calculateGlobalCoordinates( )* и *draw( ).*

Функция *calculateGlobalCoordinates( )* рассчитывает координаты точки в глобальной системе координат:

Point2i MyRobo::calculateGlobalCoordinates(Point2i point, float angle)

{

Mat rotateMat = (Mat\_<float>(3, 3) << cos(angle), -sin(angle), m\_center.x,

sin(angle), cos(angle), m\_center.y,

0, 0, 1);

Mat pointOfRobot = (Mat\_<float>(3, 3) << point.x, 0, 0,

point.y, 0, 0,

1, 0, 0);

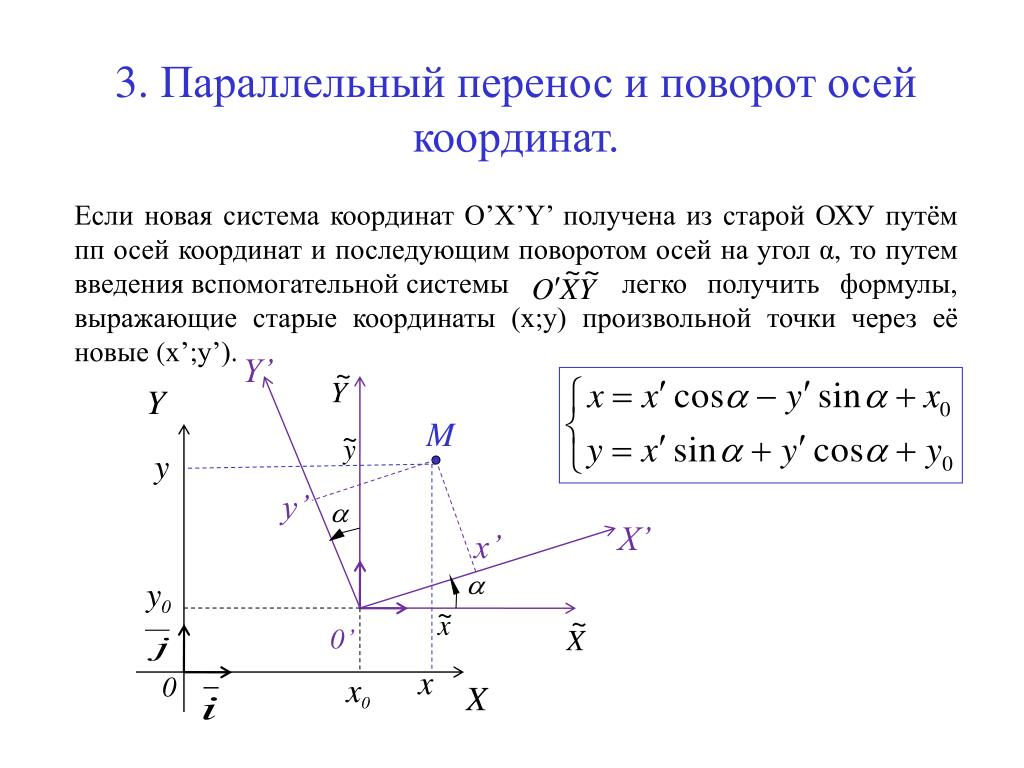
Mat result33 = rotateMat \* pointOfRobot;

Point2i result12(result33.at<float>(0, 0), result33.at<float>(1, 0));

return result12;

}

Рассмотрим поворот локальной системы координат, относительно глобальной:



Функция выполняет следующую операцию:

где – это угол поворота робота, *X* и *Y* – это координаты точки *М* после поворота в глобальной системе координат, *tx* и *ty* – это расстояние от центра глобальной системы координат до центра локальной системы, *x* и *y* – координаты точки *М* в локальной системе.

Метод класса draw( ) рассчитывает координаты точек корпуса и центры колес в локальной системе координат робота. А затем с помощью вызова *calculateGlobalCoordinates( )* передает определяет координаты всех точек в глобальной системе координат при повороте локальной системы. Далее вызываются функции *line( )*  и *circle( )*, которые рисуют отдельные части робота.

int32\_t MyRobo::draw(Mat& outpytImage)

{

//initializing the robot’s points relative to robot’s center

cv::Point2i pointTopLeft ((-m\_width / 2), (-m\_height / 2));

cv::Point2i pointTopRight ((m\_width / 2), (-m\_height / 2));

cv::Point2i pointBotLeft ((-m\_width / 2), ( m\_height / 2));

cv::Point2i pointBotRight ((m\_width / 2), ( m\_height / 2));

//applying rotation matrix for body's points

pointTopLeft = calculateGlobalCoordinates(pointTopLeft, m\_angle);

pointTopRight = calculateGlobalCoordinates(pointTopRight, m\_angle);

pointBotLeft = calculateGlobalCoordinates(pointBotLeft, m\_angle);

pointBotRight = calculateGlobalCoordinates(pointBotRight, m\_angle);

//drawing body

cv::line (outputImage, pointTopLeft, pointTopRight, black, thickness, lineType, shift); //Top line

cv::line (outputImage, pointTopLeft, pointBotLeft, black, thickness, lineType, shift); //Left line

cv::line (outputImage, pointTopRight, pointBotRight, black, thickness, lineType, shift); //Right line

cv::line (outputImage, pointBotLeft, pointBotRight, black, thickness, lineType, shift); //Bottom line