МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Курсовая работа

По дисциплине

«Объектно-ориентированное программирование»

на тему:

«Техническое зрение»

Студент группы 33331506/00401 _____ И.А. Варламов

Преподаватель _____ М.С. Ананьевский

«__»_____2023

Санкт-Петербург

2023

Содержание

Введение

OpenCV (Open Source Computer Vision Library) - это библиотека с открытым исходным кодом, которая предоставляет набор инструментов для обработки изображений и видео, включая функции компьютерного зрения, машинного обучения и обработки сигналов. Она широко используется в различных областях, таких как робототехника, автоматизация, медицина, видеонаблюдение и многих других.

OpenCV предоставляет богатый набор функций для обработки изображений и видео, таких как фильтрация, сегментация, детектирование объектов, распознавание образов, слежение, распознавание лиц, а также многие другие. Библиотека также поддерживает работу с различными типами камер и видеоисточников, включая камеры USB, IP-камеры и видеофайлы.

OpenCV используется во многих проектах компьютерного зрения и робототехники, и позволяет создавать мощные и эффективные системы обработки изображений и видео. Благодаря своей открытой архитектуре и богатому функционалу, OpenCV становится все более популярным среди научных и инженерных сообществ, а также среди разработчиков программного обеспечения и стартапов в области компьютерного зрения и робототехники.

Основные принципы, на которых основана работа OpenCV, включают следующее:

- **Многообразие функций** — OpenCV содержит более 2500 функций, которые позволяют выполнять широкий спектр задач в области обработки изображений и видео, включая фильтрацию, сегментацию, детектирование объектов, распознавание образов, отслеживание движения и многое другое.

- **Кроссплатформенность** — OpenCV может быть использована на различных платформах, включая Windows, Linux, Mac OS, Android и iOS, что делает ее универсальным инструментом для разработки программ в области компьютерного зрения

- **Поддержка различных типов камер** — OpenCV поддерживает множество типов камер, что позволяет использовать ее в различных приложениях, включая робототехнику, медицину, видеонаблюдение и многие другие.

1 Основная часть

1.1 Принцип работы

Для начала необходимо определиться, что необходимо для получения желаемого результата, путем поиска в различных инструментах был построен псевдокод программы:

1. Распознавание объектов на видео
2. Обработка полученной информации
3. Обработка логики принятия решения о движении
4. Отправка на робота набора точек и режима движения

1.1.1 OpenCV

Библиотека opencv.h предоставляет набор функций для работы техническим зрением. Она позволяет захватывать изображение и совершать различные манипуляции с каждым отдельным кадром, .

opencv.h содержит функционал для захвата изображения и преобразования каждого отдельного фрейма. Например, в работе используется инструструмент для поиска ArUco меток для калибровки изображения.

Так же среди функций, которые мы будем использовать при работе с библиотекой, выделим следующие:

- **cv::VideoWriter()** — класс для записи видео в файл;
- **cv::VideoCapture()** — связывает сокет с определенным адресом;
- **cv::Mat()** — это класс в библиотеке OpenCV, который представляет матрицу пикселей;
- **cv::Vec2f()** — это класс в библиотеке OpenCV, который представляет вектор из двух элементов типа float. Он может использоваться для хранения и обработки двумерных векторов;

- **cv::Point()** —это класс в библиотеке OpenCV, который представляет точку на двумерной плоскости с целочисленными координатами x и y;

- **cv::cuda::GpuMat()** — это класс в библиотеке OpenCV, который представляет матрицу пикселей на графическом процессоре (GPU). Он является аналогом класса cv::Mat для работы с изображениями на GPU и обеспечивает быстрый доступ и обработку изображений на устройствах с поддержкой CUDA.;

2 Программный код

В этой главе нам нужно организовать два синтаксически подобных кода. Организовано многопоточное программирование при помощи **std::tread**. Код организован в соответствии с пунктом 1.1. Работа выполнена в **CLion**, также отдельно был создан **CMakeList.txt** с указанием библиотек

2.0.1 Основной файл main

Основном коде программы мы сначала подключаем все библиотеки и модули, потом инициализируем функцию объявления параметров, и уже в бесконечном цикле захватываем изображение с камеры и обрабатываем его

```
#include <pylon/PylonIncludes.h>
#include <QDebug>
#include <QSize>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/aruco.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/cudaimgproc.hpp>
#include <opencv2/cudawarping.hpp>
#include <settings.h>
#include <performance.h>
#include <videowriter.h>
#include <sortcvpoints.h>
#include <framegrabber.h>
#include <trectdetector.h>
#include <puckdetector.h>
#include <puckpredictor.h>
#include <gamealgorithm.h>
#include <cvgui.h>
void initBaslerParameters(std::string filename)
{
    Pylon::PylonInitialize();
    try
    {
        Pylon::CInstantCamera camera(Pylon::CTlFactory::GetInstance
           ().CreateFirstDevice());
        qDebug() << "Using device " << camera.GetDeviceInfo().
           GetModelName();
        camera.Open();
        Pylon::CFeaturePersistence::Load(filename.c_str(), &camera.
           GetNodeMap(), true);
        camera.Close();
    }
    catch (const Pylon::GenericException &e)
    {
        qDebug() << "An exception occurred." << e.GetDescription();
    }
}
int main()
{
    Settings programSettings;
    programSettings.Load();
```

```
    if (!programSettings.baslerPFSFilePath.isEmpty())
    {
        initBaslerParameters(programSettings.baslerPFSFilePath.
           toStdString());
    }
    cvGUI gui(programSettings);
    Performance FPSCounter;
    FrameGrabber frameGrabber(programSettings);
    TRectDetector tableBorderDetector;
    PuckDetector puckDetector;
    PuckPredictor puckPredictor;
    GameAlgorithm game;
    while (true)
    {
        FPSCounter.resetCounter();
        frameGrabber.grab();
        frameGrabber.warp(programSettings);
        frameGrabber.crop(programSettings);
        puckDetector.detect(frameGrabber, programSettings);
        if (puckDetector.isPuckDetected())
        {
            puckPredictor.predict(frameGrabber, puckDetector,
               programSettings);
            if (puckPredictor.predictedPointRSP.x != -1)
            {
                game.process(puckDetector, puckPredictor,
                   programSettings, frameGrabber.frameHeight);
            }
        }
        FPSCounter.stopCounter();
        gui.displayWindows(programSettings,
                           FPSCounter,
                           frameGrabber,
                           tableBorderDetector,
                           puckDetector,
                           puckPredictor);
        if (gui.processKeyboard(programSettings,
                                frameGrabber,
                                tableBorderDetector))
        {
            return 0;
        }
    }
}
```

2.0.2 Алгоритм поведения робота

Этот подраздел опишет основные принципы принятия решения и поведения робота. Код для последующего анализа представлен ниже:

```
#ifndef GAMEALGORITHM_H
#define GAMEALGORITHM_H
#include <QDebug>
#include <puckdetector.h>
#include <puckpredictor.h>
#include <udpsender.h>
#include <settings.h>
#define ROBOT_STRIKER_POSITION programSettings.robotStrikerPosition
#ifndef MAX_ROBOT_REACH
#define MAX_ROBOT_REACH (long long)programSettings.
   robotStrikerPosition - programSettings.robotMotionRange
```

```
#endif
#define TRIGGER_LINE (long long)(MAX_ROBOT_REACH) - programSettings
   .Kp*abs(puckDetector.puckAverageSpeed[2])/100
#define PUCK_SPEED_SLOW_F programSettings.puckSpeedSlowF
#define PUCK_SPEED_FAST_F programSettings.puckSpeedFastF
#define PUCK_SPEED_SLOW_B programSettings.puckSpeedSlowB
#define PUCK_SPEED_FAST_B programSettings.puckSpeedFastB
class GameAlgorithm
{
private:
    UDPSender udpSender;
    std::map<std::string, bool> stagesMap;
    void initStagesMap();
    void setState(std::string state);
    bool checkState(std::string state);
public:
    GameAlgorithm();
    void process(PuckDetector puckDetector, PuckPredictor
       puckPredictor, Settings programSettings, int frameHeight);
    void processBackward(PuckDetector puckDetector, PuckPredictor
       puckPredictor, Settings programSettings, int frameHeight);
    void processForward(PuckDetector puckDetector, PuckPredictor
       puckPredictor, Settings programSettings, int frameHeight);
};
#endif // GAMEALGORITHM_H
```

2.0.3 Пользовательский интерфейс

Этот подраздел содержит код, в котором написал GUI:

```
#include "cvgui.h"
void cvGUI::setupTrackbarsWindow(Settings &programSettings)
{
    cv::namedWindow(TRACKBARS_NAME, cv::WINDOW_KEEPRATIO);
    cv::resizeWindow(TRACKBARS_NAME, 2000, 2000);
    cv::createTrackbar("houghParam1", TRACKBARS_NAME, &
       programSettings.houghParam1, 1024);
    cv::createTrackbar("houghParam2", TRACKBARS_NAME, &
       programSettings.houghParam2, 1024);
    cv::createTrackbar("puckMinRadius", TRACKBARS_NAME, &
       programSettings.puckMinRadius, 500);
    cv::createTrackbar("puckMaxRadius", TRACKBARS_NAME, &
       programSettings.puckMaxRadius, 500);
    cv::createTrackbar("robotStrikerPosition", TRACKBARS_NAME,
                       &programSettings.robotStrikerPosition,
                       programSettings.cameraResolution.width());
    cv::createTrackbar("robotMotionRange", TRACKBARS_NAME,
                       &programSettings.robotMotionRange,
                       programSettings.cameraResolution.width());
    cv::createTrackbar("puckPositionYLimit", TRACKBARS_NAME,
                       &programSettings.puckPositionYLimit,
                       programSettings.cameraResolution.width());
    cv::createTrackbar("robotGateYLinit", TRACKBARS_NAME,
                       &programSettings.robotGateYLimit,
                       programSettings.cameraResolution.width());
    cv::createTrackbar("playerZoneMargin", TRACKBARS_NAME,
                       &programSettings.playerZoneMargin,
                       programSettings.cameraResolution.width());
    cv::createTrackbar("puckSpeedSlowF", TRACKBARS_NAME,
                       &programSettings.puckSpeedSlowF, 100);
```

```
    cv::createTrackbar("puckSpeedFastF", TRACKBARS_NAME,
                       &programSettings.puckSpeedFastF, 100);
    cv::createTrackbar("puckSpeedSlowB", TRACKBARS_NAME,
                       &programSettings.puckSpeedSlowB, 100);
    cv::createTrackbar("puckSpeedFastB", TRACKBARS_NAME,
                       &programSettings.puckSpeedFastB, 100);
    cv::createTrackbar("Kp", TRACKBARS_NAME,
                       &programSettings.Kp, 20000);
}
void cvGUI::showInfo(Settings programSettings)
{
    cv::Mat info = cv::Mat::ones(cv::Size(300, 300), CV_8U);
    cv::putText(info, "Welcome to KRAH", cv::Point(5, 20), cv::
       FONT_HERSHEY_SIMPLEX, 0.7, cv::Scalar(255, 0, 0));
    cv::imshow(TRACKBARS_NAME, info);
    cv::waitKey(1);
}
cvGUI::cvGUI(Settings &programSettings)
{
    setupTrackbarsWindow(programSettings);
    videoWriter = new VideoWriter(programSettings.videoFolderPath);
    FPSColor = CV_COLOR_GREEN;
    isRecording = false;
}
bool cvGUI::processKeyboard(Settings &programSettings, FrameGrabber
    frameGrabber, TRectDetector tableBorderDetector)
{
    char pressedKey = cv::waitKey(1);
    switch (pressedKey)
    {
    case 'c':
        tableBorderDetector.detectTableBorders(frameGrabber,
           programSettings);
        break;
    case 'h':
        showInfo(programSettings);
        break;
    case 'q':
        return true;
    case 's':
        programSettings.Save();
        qDebug() << "Settings were successfully saved";
        break;
    case 'r':
        if (isRecording)
        {
            isRecording = false;
            FPSColor = CV_COLOR_GREEN;
            videoWriter->release();
            qDebug() << "Recording was stopped";
        }
        else
        {
            isRecording = true;
            FPSColor = CV_COLOR_RED;
            qDebug() << "Recording was started";
        }
        break;
    case 'l':
        programSettings.Load();
        cv::destroyWindow(TRACKBARS_NAME);
        setupTrackbarsWindow(programSettings);
        qDebug() << "Settings were successfully loaded";
        break;
    }
```

```
    return false;
}
void cvGUI::displayWindows(Settings &programSettings,
                           Performance FPSCounter,
                           FrameGrabber frameGrabber,
                           TRectDetector tableBorderDetector,
                           PuckDetector puckDetector,
                           PuckPredictor posPredictor)
{
    if (GPU_ACCELERATION)
    {
        cv::cuda::cvtColor(frameGrabber.gpuFrame, frameGrabber.
           gpuFrame, cv::COLOR_GRAY2RGB);
        frameGrabber.gpuFrame.download(frameGrabber.frame);
    }
    else
    {
        cv::cvtColor(frameGrabber.frame, frameGrabber.frame, cv::
           COLOR_GRAY2RGB);
    }
    if (isRecording)
    {
        videoWriter->write(frameGrabber.recordFrame);
    }
    if (puckDetector.currentPoint.x != -1)
    {
        cv::circle(frameGrabber.frame,
                   cv::Point(puckDetector.currentPoint.x,
                             puckDetector.currentPoint.y),
                   puckDetector.currentPoint.z, CV_COLOR_GREEN, 2);
    }
    for (uint i = 0; i < puckDetector.puckTrajectoryPointsVector.
       size(); i++)
    {
        cv::drawMarker(frameGrabber.frame, puckDetector.
           puckTrajectoryPointsVector[i], CV_COLOR_BLUE);
    }
    cv::line(frameGrabber.frame,
             cv::Point(ROBOT_STRIKER_POSITION, 0),
             cv::Point(ROBOT_STRIKER_POSITION, frameGrabber.
                frameHeight),
             CV_COLOR_BLUE, 2);
    cv::line(frameGrabber.frame,
             cv::Point(MAX_ROBOT_REACH - programSettings.Kp * abs(
                puckDetector.puckAverageSpeed[2]) / 100, 0),
             cv::Point(MAX_ROBOT_REACH - programSettings.Kp * abs(
                puckDetector.puckAverageSpeed[2]) / 100,
                frameGrabber.frameHeight),
             CV_COLOR_BLUE, 2);
    cv::line(frameGrabber.frame,
             cv::Point(MAX_ROBOT_REACH, 0),
             cv::Point(MAX_ROBOT_REACH, frameGrabber.frameHeight),
             CV_COLOR_RED, 2);
    cv::line(frameGrabber.frame,
             cv::Point(PLAYER_ZONE, 0),
             cv::Point(PLAYER_ZONE, frameGrabber.frameHeight),
             CV_COLOR_RED, 2);
    //
    cv::line(frameGrabber.frame,
             cv::Point(0, frameGrabber.frameHeight / 2 -
                programSettings.puckPositionYLimit),
             cv::Point(frameGrabber.frameWidth, frameGrabber.
                frameHeight / 2 - programSettings.
                puckPositionYLimit),
             CV_COLOR_RED, 2);
```

```
    cv::line(frameGrabber.frame,
             cv::Point(0, frameGrabber.frameHeight / 2 +
                programSettings.puckPositionYLimit),
             cv::Point(frameGrabber.frameWidth, frameGrabber.
                frameHeight / 2 + programSettings.
                puckPositionYLimit),
             CV_COLOR_RED, 2);
    //
    cv::line(frameGrabber.frame,
             cv::Point(0, frameGrabber.frameHeight / 2 -
                programSettings.robotGateYLimit),
             cv::Point(frameGrabber.frameWidth, frameGrabber.
                frameHeight / 2 - programSettings.robotGateYLimit),
             CV_COLOR_GREEN, 2);
    cv::line(frameGrabber.frame,
             cv::Point(0, frameGrabber.frameHeight / 2 +
                programSettings.robotGateYLimit),
             cv::Point(frameGrabber.frameWidth, frameGrabber.
                frameHeight / 2 + programSettings.robotGateYLimit),
             CV_COLOR_GREEN, 2);
    //
    if (posPredictor.predictedPointRSP.x != -1)
    {
        cv::circle(frameGrabber.frame, posPredictor.
           predictedPointRSP, 20, CV_COLOR_WHITE);
    }
    if (posPredictor.predictedPointMRR.x != -1)
    {
        cv::circle(frameGrabber.frame, posPredictor.
           predictedPointMRR, 20, CV_COLOR_WHITE);
    }
    //
    cv::putText(frameGrabber.frame, std::to_string(FPSCounter.
       getAverageFPS()), cv::Point(80, 80), cv::
       FONT_HERSHEY_SIMPLEX, 1, FPSColor);
    cv::imshow(VIDEO_WINDOW_NAME, frameGrabber.frame);
}
#endif // GAMEALGORITHM_H
```

Заключение

Работа с компьютернем зрением на C++ — это отличный способ познакомиться с основами OpenCV. В процессе работы были изучены основные функции библиотеки.

Был создана программа, в которой была проработана логика принятия решения, техническое зрение, был написан пользовательский интерфейс и налажена связь с роботом.

В целом, работа на C++ представляет собой интересный и познавательный опыт для разработчика, который может быть полезен при работе с техническим зрением.

Список литературы

[1] Адриан Кейлер и Гари Брадски *Learning OpenCV*. Kenneth L. Calvert, Massachusetts, 1nd Edition, 2000.

[2] *Beej's Guide to Network Programming.*