

Debreceni Egyetem
Informatikai Kar

Robotautó világbajnokság

Témavezető: Dr. Bátfai Norbert

Beosztása: Adjunktus

Készítette: Fábián Kristóf – Szabolcs

Szak megnevezése: Programtervező
Informatikus BSc

Tartalomjegyzék

1. Bevezetés.....	5
2. Tárgyalás.....	7
2.1 Használt szoftver eszközök.....	7
2.1.1 Qt.....	7
2.1.1.1 qmake.....	7
2.1.1.2 Modulok.....	9
2.1.1.3 Slot-signal mechanizmus.....	12
2.1.1.4 Qt Creator.....	15
2.1.2 Képfeldolgozási megoldások.....	15
2.1.3 BGSLib.....	16
2.1.3.1 Telepítése, használata.....	16
2.2 Előzmények.....	18
2.2.1 <legeleső képfeldolgozással kapcsolatos.....	18
2.2.2 Megelőző forgalomszámláló szoftver.....	19
2.3 A Robotautó Világbajnokság.....	19
2.3.1 A platform koncepciója.....	19
2.3.2 <>.....	19
2.4 A szoftver elkészítésének folyamata.....	19
2.4.1 Tervezés.....	19
2.4.2 Fejlesztés.....	19
2.4.2.1 Videó források típusok.....	19
2.4.2.2 Adatbázis kapcsolatok kialakítása, kezelése.....	19
2.4.2.3 Szálkezelés.....	19
2.4.2.4 Videó feldolgozása.....	19
2.4.2.5 Járművek detektálása.....	19

2.4.3	Problémák a szoftver fejlesztése során.....	19
2.4.4	A szoftver tesztelése.....	19
2.4.5	Eredmények.....	19
2.5	A szoftver használata.....	20
2.5.1	Fordítása.....	20
2.5.2	Indítása.....	20
2.5.3	A grafikus felhasználói felület használata.....	20
2.5.3.1	Új forrás kiválasztása.....	20
2.5.3.2	Adatbázisok kezelése.....	20
2.5.3.3	Fájl videó forrás információi.....	20
2.5.4	A parancssoros felhasználói felület használata.....	20
2.5.4.1	Használható parancsok.....	20
3.	Összefoglalás.....	20
4.	Köszönetnyilvánítás.....	20
5.	Irodalomjegyzék.....	21

<ábrajegyzék soon>

1. Bevezetés

Robotautó: amikor valaki mostanában ezt hallja, nagy eséllyel valamilyen science fiction filmbeli saját gondolkodással rendelkező járműre gondol. Nem is tudják, mennyire közel állunk ahhoz, hogy utakon a legtöbb autó tudatosan, önműködően közlekedjen, illetve milyen régóta folynak kutatások ezen a területen. Az első komolyabb próbálkozások az 1980-as évek vége fele kezdődtek. A PROMETHEUS [PROMETHEUS] projekt keretén belül Ernst Dickmanns csapatának sikerült először továbbfejleszteni egy hétköznapi járműt, hogy az külső (emberi) beavatkozás nélkül (vagy minimális) sikeresen közlekedjen közönséges utakon, akár hosszabb távokon keresztül. 1995- ben az Ernst Dickmanns csapata által módosított VaMP- nak [VaMP] becézett Mercedes 500 SEL típusú jármű egy 1000 mérföldnél is hosszabb utat tett meg minimális emberi beavatkozással München és Koppenhága között.

Mindig is vonzott az autók világa, gimnáziumos koromban lenyűgözött a Google által fejlesztett autonóm jármű. Tavaly ősszel átlépték a két millió mérföldes [recode] megtett távot. Hatalmas teljesítmény ez, így a Google az első, akinek sikerült ekkora távot teljesen önirányító járművel megtenni. Egy olyan világban, ahol az összes autó autonóm módon közlekedik a balesetek megszűnnek létezni. A World Health Organization szerint [WHO] 2013- ban 1.25 millió ember vesztette életét közúti balesetben. A McKinsey&Company konzultációs vállalat egy beszámolója szerint [McKinsey] ha 2025- re az utakon közlekedő járművek 10-20 %- a önirányításúak lennének, évi 30000-140000 emberi életet tudnánk megmenteni.

A Robotautó Világajnokság platformja egy számomra elérhető teret biztosított arra, hogy egy olyan szoftvert fejlesszek, ami valós forgalmi adatokkal szolgál a versenyzők számára. Ennek a szoftvernek a kimenetének a hatására a versenyzők által fejlesztett útválaszó algoritmusok új értelmet nyernek, mivel figyelembe kell venniük az aktuális forgalmat a városban (MAP). A Robotautó Világajnokság "Tetris-terv"- én az általam fejlesztett, nevezhetjük akár modulnak is, szoftver a ASA [OOCWC] (Automated Sensor Annotation). Korábbi fejlesztésem ezen a területen egy hasonló szoftver volt, Magasszintű programozási nyelvek 2 tárgy keretén belül. Habár viszonylag jó hatásfokon működött, nem lett volna elegendő csupán az, ha kibővítem adatbázis eléréssel, vagy újraszervezem a kódot.

Be kellett lássam, hogy két évvel ezelőtt sokkal alacsonyabb minőségű kódot írtam. Ahogy most visszanézek rá, csodálkozom néha, miért gondoltam akkor jónak néhány dolgot, ahogy írtam. Az is hiányzott a régi verzióból, hogy semmi féle tervezési mintát nem követett, viszont nekem ahhoz, hogy felhasználói felület nélkül is futtatható, használható alkalmazást építsek, szükséges az MVC (Model – View – Controller) tervezési mintát [MVC] követnem. Az is nagy hiány volt az előző verzióból, hogy csupán lokális fájlt tudott feldolgozni, viszont egy ilyen rendszer általában valós idejű, ritkán használják utó-feldolgozásra, így kiegészítettem az adatforrások tárházát webkamerák illetve távolról elérhető IP kamerákra.

A dolgozatom első részében bemutatom a fejlesztés során használt eszközöket, ezt követően az előző, általam fejlesztett szoftverről ejtek néhány szót. Az ezt követő fejezetben a Robotautó Világbajnokság platformjáról számolok be, amit a szoftver fejlesztéséről szóló fejezet követ, utólag pedig a szoftvert áttekintve a használatát mutatom be.

2. Tárgyalás

2.1 Használt szoftver eszközök

2.1.1 Qt

A Qt [Qt] (kiejtése az angol “cute” szóval megegyező) egy számos platformra fordítható keretrendszer, aminek segítségével PC, mobil, illetve beágyazott rendszerekre is fejleszthetünk alkalmazásokat C++ programozási nyelven. A többi keretrendszerhez eltérően, a Qt rendelkezik egy MOC (Meta – Object Compiler) [MOC] nevű kód-generátorral. Ez az generátor feldolgozza a Qt- ban írt forrásfájlokat. Ha valamely osztály deklarációban a MOC megtalálja a Q_OBJECT makrót akkor egy másik C++ forrásfájlt generál belőlük, ezt meta-objektum kódnak nevezzük. A Qt egyik sajátos funkciója, a slot-signal (magyarul?) a MOC-nak köszönhető. A slot-signal mechanizmusról bővebben a 2.1.1.3- as, “Slot-signal mechanizmus” nevű fejezetben írok. A MOC mellett a Qt még használ egy uic nevű kód-generátort. Az uic [uic] egy felhasználói felületet leíró (.ui) XML fájlt dolgoz fel, és C++ kódot generál belőle, ami a kézzel leprogramozott felhasználói felületnek felel meg.

Mivel számomra feltétel volt, hogy Open Source szoftvereket használjak, így a Qt nagyon jó választásnak tűnt, mivel elérhető szabad szoftverként is, LGPL/GLP licenc alatt [QtLicense] Emellett a Qt megvásárolható kereskedelmi licenc alatt is, mely tartalmaz olyan további funkciókat, mint például virtuális billentyűzet, adatvizualizáció. Egy közösség alapú adatgyűjtés szerint jelenleg 207 könyvtár [include] használja a Qt- t, különböző, viszont legtöbbjük valamely szabad szoftver licenc alatt érhető el.

2.1.1.1 qmake

A qmake [qmake] egy olyan eszköz, amely leegyszerűsíti számunkra a fordítás menetét eltérő platformokra. Egy projekt (.pro) fájlból generál számunkra egy úgynevezett Makefile- et, vagy akár projekt fájlt is tudunk a segítségével generálni, ha egyelőre csak a forrásfájlokkal rendelkezünk. Egy Makefile a GNU make által használt fájl, amely leírja,

hogy a make hogyan fordítsa illetve kapcsolja össze az objektum fájlokat egy futtatható állománnyá.

Vegyük példának az alábbi forrásokat:

```
krajsz@krajsz-Lenovo-G500s:~/ $ ls -lh
total 16K
-rw-rw-r-- 1 krajsz krajsz 172 apr 21 14:52 main.cpp
-rw-rw-r-- 1 krajsz krajsz 219 apr 21 14:52 mainwindow.cpp
-rw-rw-r-- 1 krajsz krajsz 291 apr 21 14:52 mainwindow.h
-rw-rw-r-- 1 krajsz krajsz 630 apr 21 14:52 mainwindow.ui
```

Ezután a qmake- nek átadva a -project kapcsolót tudunk generálni egy projekt fájlt. Átadhatjuk a forrásfájlokat argumentumként a qmake- nek, viszont, ha nem adunk meg egy fájlt sem, akkor az aktuális könyvtárban található forrásfájlok alapján generálja a projekt fájlt:

```
krajsz@krajsz-Lenovo-G500s:~/ $ qmake -project
krajsz@krajsz-Lenovo-G500s:~/ $ ls -l
total 20
-rw-rw-r-- 1 krajsz krajsz 172 apr 21 14:52 main.cpp
-rw-rw-r-- 1 krajsz krajsz 219 apr 21 14:52 mainwindow.cpp
-rw-rw-r-- 1 krajsz krajsz 291 apr 21 14:52 mainwindow.h
-rw-rw-r-- 1 krajsz krajsz 630 apr 21 14:52 mainwindow.ui
-rw-rw-r-- 1 krajsz krajsz 354 apr 21 15:21 testProgram.pro
```

Láthatjuk, hogy generálva lett egy testProgram.pro nevű állomány, aminek a tartalma a következő:

```
krajsz@krajsz-Lenovo-G500s:~/ $ cat testProgram.pro

TEMPLATE = app
TARGET = testProgram
INCLUDEPATH += .
# Input
HEADERS += mainwindow.h
FORMS += mainwindow.ui
SOURCES += main.cpp mainwindow.cpp
```

Mivel a qmake nem csak Qt- s projektekhez használható, így, mint jelen esetben is, kiegészítésre szorul a generált projekt fájl. Jelen esetben ahhoz, hogy a megfelelő fájlokat kapcsolja a make, a következő sorral kell kiegészítsük a projekt fájlunkat:

```
QT+= widgets
```


Ezután futtatjuk a qmake- et, átadva neki a generált projekt fájlunkat:

```
krajsz@krajsz-Lenovo-G500s:~/ $ qmake testProgram.pro
```

Ha nem kapunk semmi féle hibaüzenetet, akkor azt jelenti, sikeresen generálva lett egy Makefile. Láthatjuk, hogy mekkora segítség ez számunkra, mivel egy hét soros projekt fájlból egy 537 soros Makefile- et generált a qmake:

```
krajsz@krajsz-Lenovo-G500s:~/ $ wc -l Makefile
537 Makefile
```

Ezután a make- et futtatva láthatjuk a fordítás menetét:

```
krajsz@krajsz-Lenovo-G500s:~/ $ make
/usr/lib/x86_64-linux-gnu/qt5/bin/uic          mainwindow.ui          -o
ui_mainwindow.h
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG
-DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem
/usr/include/x86_64-linux-gnu/qt5 -isystem /usr/include/x86_64-
linux-gnu/qt5/QtWidgets -isystem /usr/include/x86_64-linux-
gnu/qt5/QtGui -isystem /usr/include/x86_64-linux-gnu/qt5/QtCore -I.
-I. -I/usr/lib/x86_64-linux-gnu/qt5/mkspecs/linux-g++-64 -o main.o
main.cpp
...
```

Ha hiba nélkül lefut a make, azt jelenti, hogy elkészült a futtatható állományunk.

2.1.1.2 Modulok

A Qt több, különálló modulból áll össze, ezek a következők:

- Qt Core
- Qt GUI
- Qt Multimedia
- Qt Multimedia Widgets
- Qt Network
- Qt QML
- Qt Quick
- Qt Quick Controls
- Qt Quick Dialogs
- Qt Quick Layouts

- Qt SQL
- Qt Test
- Qt Widgets

Ezek közül a dolgozatomban a Qt Core, Qt Multimedia, Qt SQL, illetve Qt Widgets modulokat használok. A Qt Core (“mag”) modulra alapszik az összes többi, ez a modul tartalmazza például a `QObject` osztályt, amely az összes többi osztály bázis osztálya a Qt-ban. Továbbá, a Core modul tartalmazza a konténer osztályokat is, mint például az általam is használt `QVector<T>`, illetve `QList<T>` osztályokat.

A következő módon tudunk egy `intVector` nevű, száz elemet tartalmazó vektort deklarálni:

```
QVector<int> intVector(100);
```

Ekkor, ha megadunk egy értéket második paraméterként a konstruktorban, amely típusa megegyezik a vektorban tárolt elemek típusával, akkor a vektor összes eleme azzal az értékkel fog rendelkezni. Ha nem adtunk meg második paraméterként alapértelmezett értéket a konstruktorban a vektor elemeinek, akkor a típushoz megfelelő alapértelmezett konstruktor meghívásával lesznek példányosítva.

A Qt Multimedia modul ahogy nevéből is következtethető, multimédia tartalom kezeléséhez nyújt eszközöket, mint például videók fájlok kezelése, audio fájlok kezelése, rendszer kamerákhoz hozzáférés. Az általam fejlesztett szoftverben ebből a modulból a `QCameraInfo` osztályt használok. A `QCameraInfo` osztály segítségével az alábbi módon lekérhetjük a rendszerünkben található összes kamerához tartozó információt:

```
QList<QCameraInfo> cameras = QCameraInfo::availableCameras();
```

A `QCameraInfo::availableCameras(QCamera::Position position = QCamera::UnspecifiedPosition)` metódus egy statikus metódusa a `QCameraInfo` osztálynak. A paraméter segítségével lekérhetünk csak bizonyos kamerákról információt, a

következő módon meghívva csak az elülső kamerák (például mobil készülékek elülső kamerái) információit fogja visszaadni:

```
... = QCameraInfo::availableCameras(QCamera::FrontFace);
```

A Qt SQL modul támogatja SQL adatbázisokkal kapcsolatos műveleteket. Az adatbázishoz való kapcsolódásról az `QSqlDatabase` felelős. Az alábbi módon tudunk egy adatbáziskapcsolatot létrehozni, majd ellenőrizni, hogy sikeresen meg tudjuk-e nyitni:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLite");
if (db.open())
{
    //sikeresen csatlakoztunk az adatbázishoz
}
else
{
    //sikertelen csatlakozás
}
```

Sikertelen kapcsolódás esetén a `QSqlError` osztály információval tud szolgálni a kapcsolat létrejöttét meggátló hibáról. Az `QSqlDatabase` osztály `QSqlError lastError()` `const` metódusa visszaadja az adatbázis kapcsolatban felmerülő utolsó hibával kapcsolatos információkat, amit a következő módon `QString`-ként (ahogy a neve is sugallja, egy karakterláncot ábrázoló osztály) meg tudjuk kapni:

```
const QString errorString = db.lastError().text();
```

Lekérdezések végrehajtása a `QSqlQuery` osztály segítségével lehetségesek. Lehetőségünk van DML, DDL, illetve adatbázis-specifikus parancsok végrehajtására is.

Egy egyszerű példa a működésére, a lekérdezés feldolgozására:

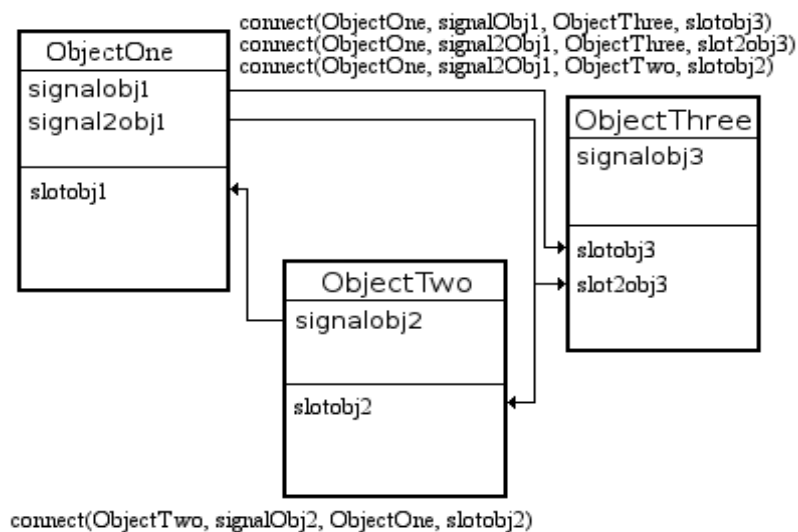
```
QSqlQuery query ("Select name from employees");
while (query.next())
{
    qDebug() << query().value(0).toString();
}
```

A `value(0)` függvény visszaadja lekérdezés által visszaadott rekord 0. indexű mezőjét.

Utolsónak említtem a Qt Widgets modult, amely a felhasználói felület felépítésére szolgál. Minden egyes grafikus felhasználói felületet alkotó osztály bázisosztálya a `QWidget` osztály. A `QWidget` osztály szolgáltat egy képernyőn megjeleníthető felületet, illetve gondoskodik a felhasználói bemenetek feldolgozásáról. Minden osztály ebben a modulban rendelkezik egy alapértelmezett esemény feldolgozó implementációval, viszont mivel ezek virtuális függvények, ezért fölül tudjuk írni őket, és saját implementációnkat használhatjuk.

2.1.1.3 Slot-signal mechanizmus

A slot-signal mechanizmus az egyik legfőbb olyan funkciója a Qt- nak amiben eltér a több keretrendszerétől. A legegyszerűbben egy felhasználói felület működésével lehet a slot-signal mechanizmus lényegességét ábrázolni. Például, ha egy elemet változtatunk a felhasználói felületünkön, az általában hatással lesz egy másik elemre is. Ez azt jelenti, hogy valamilyen módon kommunikációt kell létrehozunk a két elem között. A slot-signal mechanizmus épp ezt teszi, kommunikációt tudunk felépíteni bármely két osztály példánya között, amely bázisosztálya a `QObject` osztály. Ahhoz, hogy használni tudjunk ezt a mechanizmust, még egy feltételnek eleget kell tennünk. Minden osztály deklarációjának a legelején szerepelnie kell a `QObject` makrónak. Egy slot az lehet egy metódus vagy függvény (rendelkezhet visszatérítési értékkel), egy signal pedig egy “jel”, amit a példányok kibocsátani tudnak, ezek típusa csakis `void` lehet. Egy signalt akár több slot-hoz is hozzákapcsolhatjuk, ezáltal akár több példányt is értesíthetünk egy bizonyos példányban kialakult változásokról. Az alábbi ábra szemlélteti ezt:



1. ábra: Slot-signal mechanizmus absztrakt módon

Láthatjuk, hogy jelen esetben az `Object1` példány kommunikál az `Object2` illetve az `Object4` példánnyal is.

Egy rövid példa a használatukra:

```
class Counter
{
    Q_OBJECT
    ..
private:
    void methotThatChangesTheValue()
    {
        ..
        emit valueChanged(value);
    }

    int m_value;

Q_SIGNALS:
    void valueChanged(int value);
public Q_SLOTS:
    void setValue(const int value);
};
```

Tegyük fel, hogy eme `Counter` osztálynak két példányával dolgozunk, és szeretnénk, hogy az egyik értesüljön, ha a másik osztály kibocsátja a `valueChanged(int)` jelet, ezt a következőképpen tudjuk megvalósítani:

```
Counter counter;
Counter counterTwo;

connect(&counter,          &Counter::valueChanged,          &counterTwo,
        &Counter::setValue);
```

A fenti esetben feltételezem, hogy a fenti sor egy `QObject` leszármazott osztályában található, ellenkező esetben a `QObject::connect(...)` statikus függvényt kell alkalmaznunk. A fenti szintaxist a Qt 5-ös verziójával kezdődően használhatjuk, az előző verziókban a következőképpen nézne ki:

```
connect(&counter,          SIGNAL(valueChanged(int)),          &counterTwo,
        SLOT(setValue(int)));
```

Természesen lehetőségünk van a kapcsolat felbontására is, Qt 5-öt megelőzően az alábbi módon tudtuk ezt megtenni:

```
disconnect(&counter, SIGNAL(valueChanged(int)), &counterTwo,
SLOT(setValue(int)));
```

Qt 5- től kezdődően ezt egyszerűbben is megtehetjük, mivel a `QObject::connect(...)` egy `QMetaObject::Connection` példányt ad vissza, így írhatjuk a következőt:

```
QMetaObject::Connection connection = connect(&counter,
&Counter::valueChanged, &counterTwo, &Counter::setValue);
```

Ezután már csak annyi van hátra, hogy a `QObject::disconnect(...)` statikus módszerét meghívjuk, ha a kapcsolatot meg szeretnénk szüntetni:

```
QObject::disconnect(connection);
```

Viszont, ha ismét létre szeretnénk hozni a kapcsolatot, akkor hála a `QMetaObject::Connection` osztálynak, elegendő a következőt írunk:

```
QObject::connect(connection);
```

Lehetőségünk van Qt5 óta C++11 lambda kifejezésekhez is signalokat csatlakoztatni, például:

```
connect(this, &Counter::valueChanged, [=](const int newValue)
{
    this->doSomethingWithNewValue(newValue);
});
```

2.1.1.4 Qt Creator

A Qt Creator a The Qt Company által fejlesztett integrált fejlesztői környezet (IDE), amely meglehetősen megegyezősíti Qt szoftverek fejlesztését. Lehetőséget biztosít felhasználói felület kialakítására, drag-and-drop stílusban, ezentúl minden olyan képességgel bír, mint a legtöbb IDE, mint például kódkiegészítés, szintaktikai helyesség ellenőrzése. Mindezek felett az elterjedtebb verziókezelő rendszerek is támogatva vannak, mint például a Git, CVS, Perforce, Subversion. Mivel nagyon kényes lenne kézzel leprogramozni az egész felhasználói felület elrendezését, elemeinek példányosítását, beállítását, így számomra nagyon hasznosnak bizonyult.

2.1.2 Képfeldolgozási megoldások

Mivel a dolgozatomban legfontosabb funkcionalitását képfeldolgozás segítségével érem el, így szükséges volt egy stabil, széles funkcionalitással rendelkező, jól dokumentált képfeldolgozási könyvtárra. Az első gondolat természetesen az OpenCV-t [OpenCV] illette, mivel már előzőleg használtam (lásd 2.2-es fejezetben). Olyan vezető cégek is alkalmazzák az OpenCV-t, mint a Google, Yahoo, Intel, Honda, Toyota [ocvabout]. Viszont az is megfordult a fejemben, hogy nem feltétlen kellene egy ekkora könyvtárt függőségként használnom, reméltem, találok egy sokkal kisebbet, aminek egyszerűbb a telepítése is például. Az OpenCV telepítése időigényes, elég nagy tárterület szükséges a telepítésére még abban az esetben is, ha csak az OpenCV-ben elérhető modulok egy szűkebb részét vesszük figyelembe, amelyek szükségesek az általam fejlesztett szoftver fordításához (ezek mérete körülbelül 1.1-1.3 GiB). Egy csábító könyvtárra találtam keresés közben, amely 2.0-béta verziója C++14-ben van írva jelenleg. Ez a könyvtár a libCVD [libCVD], BSD licenc alatt érhető el, viszont elég szűk az utófeldolgozással kapcsolatos funkcióinak tárháza. Leginkább kép, illetve videó megjelenítéssel, mentéssel kapcsolatosan van jobban kifejlesztve egyelőre. Például bináris képen egy egybefüggő régió kontúrjának felismerése, letárolása számomra szükséges, viszont ilyet sajnos nem tud még ez a könyvtár. Ezen kívül az is az OpenCV mellett érvelt, hogy rátaláltam a BGSLib-re (bővebben a 2.3.?), mivel ez a könyvtár is az OpenCV-t használja. Az OpenCV jogi szempontból is megfelelő, mivel BSD licenccel van ellátva, ezáltal szabadon felhasználhatjuk, akár módosíthatjuk a forrásfájlokat.

2.1.3 BGSLib

A BGSLib[bgslibrary] egy C++-ban írt, OpenCV-re alapozott háttér leválasztó algoritmusokat tartalmazó könyvtár. Az általam kiválasztott járműfelismerési módszerben az egyik leglényegesebb lépés egy statikus képsorozaton a háttér leválasztása, kiemelése. Ezzel a módszerrel megkapjuk az előtérben helyezkedő területeket. Ezek, az előtérben lévő területek jelentik számunkra az elmozdulást. Ez a könyvtár pont erre szolgál, jelenleg 43 különböző háttér leválasztó algoritmust tartalmaz, egyszerűbbtől, bonyolultig. Mivel egy optimális környezetre épülő rendszerre építettem az alkalmazásomat, ezért számomra a minél egyszerűbb, minél gyorsabb algoritmus viszonyult a kiválasztottnak. Optimális környezet alatt értem azt,

hogy a statikus kamerák a környezetből adódó minimális rezgéseken kívül nem végeznek semmi féle plusz mozgást mozgást (kamera fej mozgatása, nem megfelelő rögzítésből adódó nagyobb amplitúdójú berezgések), a kamerák helyesen vannak bepozicionálva, vagyis az úttest felett, minél merőlegesebben a szemből érkező kocsikra, vagy ha úgy adódik az épp távolodókra.

2.1.3.1 Telepítése, használata

Ahhoz, hogy egyszerűen beépítsük az alkalmazásunkba a BGSLibrary- t, először le kell klónoznunk a tárolójukat:

```
krajsz@krajsz-Lenovo-G500s:~/$ git clone
https://github.com/andrewssobral/bgslibrary.git
Cloning into 'bgslibrary'...
...
```

Miután ezzel a lépéssel megvagyunk, bemásoljuk a `package_bgs` illetve `package_analysis` könyvtárakat a projektünk könyvtárába és egy üres `config` mappát hozunk létre, mivel ebbe tárolja le a BGSLibrary némely osztálynak beállítását XML fájlokban. Ezek után csak a megfelelő fejléceket kell `include-` olnunk. Egy minimális példa a használatára:

```
#include <cv.h>
#include <highgui.h>
#include "package_bgs/FrameDifferenceBGS.h"

int main(int argc, char** args)
{
    cv::VideoCapture cap;
    cap.open("video.avi");
    IBGS* bgSubtractor = new FrameDifference;
    cv::Mat frame;
    while (1)
    {
        cap.read(frame);
        cv::Mat background;
        cv::Mat foreground;
        if (!frame.empty())
        {
            bgSubtractor->process(frame, foreground, background);
            cv::imshow("Original", frame);
            cv::waitKey(30);
        }
        delete backgroundSubtractor;
        cap.release();
    }
}
```



```
    return 0;  
}
```

A fenti példát fordítás után futtatva a következő kimenetet kapjuk, a jobb oldali ablak automatikusan meg lesz nyitva:



2. ábra: Eredeti képkocka illetve a kiemelt előtér

2.2 Előzmények

2.2.1 <legeső képfeldolgozással kapcsolatos

2.2.2 Megelőző forgalomszámláló szoftver

2.3 A Robotautó Világbajnokság

2.3.1 A platform koncepciója

2.3.2 <

2.4 A szoftver elkészítésének folyamata

2.4.1 Tervezés

2.4.2 Fejlesztés

2.4.2.1 Videó források típusok

2.4.2.2 Adatbázis kapcsolatok kialakítása, kezelése

2.4.2.3 Szálkezelés

2.4.2.4 Videó feldolgozása

2.4.2.5 Járművek detektálása

2.4.3 Problémák a szoftver fejlesztése során

2.4.4 A szoftver tesztelése

2.4.5 Eredmények

2.5 A szoftver használata

2.5.1 Fordítása

2.5.2 Indítása

2.5.3 A grafikus felhasználói felület használata

2.5.3.1 Új forrás kiválasztása

2.5.3.2 Adatbázisok kezelése

2.5.3.3 Fájl videó forrás információi

2.5.4 A parancssoros felhasználói felület használata

2.5.4.1 Használható parancsok

3. Összefoglalás

4. Köszönetnyilvánítás

5. Irodalomjegyzék

Irodalomjegyzék

PROMETHEUS: , Programme for a european traffic system with highest efficiency and unprecedented safety, 1987-1995, <http://www.eurekanetwork.org/project/id/45>

VaMP: , Versuchsfahrzeug für autonome Mobilität und Rechnersehen, 1987-1995, <https://en.wikipedia.org/wiki/VaMP>

recode: , After two million miles, Google's robot car now drives better than a 16-year-old, 2016, <https://www.recode.net/2016/10/5/13167364/google-self-driving-cars-2-million-miles>

WHO: , Road traffic deaths, , http://www.who.int/gho/road_safety/mortality/en/

McKinsey: , Disruptive technologies: Advances that will transform life, business, and the global economy, 2013, <http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/disruptive-technologies>

OOCWC: Bátfai Norber, Besenczi Renátó, Mamenyák András, Ispány Márton, OOCWC: The Robocar World Championship Initiative, submitted manuscript, 2015

MVC: , Model/View Programming, , <http://doc.qt.io/qt-5/model-view-programming.html>

Qt: The Qt Company, Cross-platform software development for embedded & desktop, 2017, <https://www.qt.io/>

MOC: The Qt Company Ltd., Using the Meta-Object Compiler (moc), 2017, <http://doc.qt.io/qt-5/moc.html>

uic: The Qt Company Ltd., User Interface Compiler (uic), 2017, <http://doc.qt.io/qt-5/uic.html>

QtLicense: The Qt Company, Qt - Legal | Licensing, 2017, <https://www.qt.io/licensing/>

include: Cornelius Schumacher, All libraries, 2017, <https://include.org/all.html>

qmake: The Qt Company, qmake manual, 2017, <http://doc.qt.io/qt-5/qmake-manual.html>

bgslibrary: Sobral Andrews, BGSLibrary: An OpenCV C++ Background Subtraction Library, 2013, <https://github.com/andrewssobral/bgslibrary>