//külső borító

Debreceni Egyetem
Informatikai Kar

Robotautó világbajnokság

Témavezető: Dr. Bátfai Norbert
Beosztása: Adjunktus

Készítette: Fábián Kristóf – Szabolcs
Szak megnevezése: Programtervező Informatikus BSc

Tartalomjegyzék

<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 Világbajnokság 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 Világbajnokság “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. Azzal viszont egyet kell értenünk, hogy egy ilyen rendszer általában valós idejű, ritkán használják utó-feldolgozásra, így kiegészítettem az adatforrások tárházát webkamerák (közvetetett módon csatlakoztatott) illetve távolról elérhető IP kamerákra.

A dolgozatom első részében bemutatom a fejlesztés során használt eszközöket, ezt követően az előző, általam fejlesztett szoftverről ejtek meg 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 szofvterként is, LGPL/GLP licenc alatt [QtLicense] Emellett a Qt megvásárolható kereskedelmi licenc alatt is, mely tartalmaz olyan további funkcionalitásokat, mint például virtuális billentyűzet, adatvizualizáció. Egy közösség alapú adatgyűjtés szerint jelenleg 207 könyvtár [inqlude] 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 leegyszerüsíti 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 generáni 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 kövező 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 következőek:

- 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 használom. 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 használom. 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 létrehozn, majd ellenőrizni, hogy sikeresen meg tudjuk- e nyitnii:

```
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:

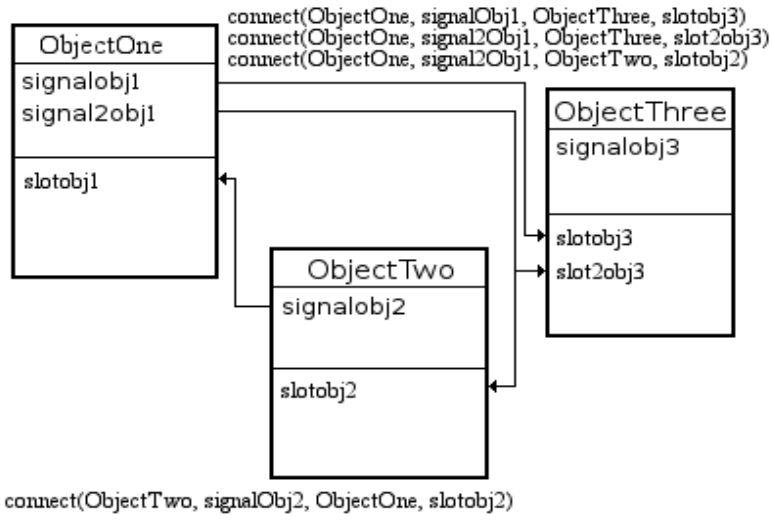
```
QsqlQuery query ("Select name from employees");

while (query.next())
{
      qDebug() << quey().value(0).toString();
}
```

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

Utolsónak említem 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 keretrendszertől. A legegyszerűbben egy felhasználói felület működésével lehet a slot-signal mechanizmus lényegességét ábrázolni. Például, ha egy egy elemen változtatunk a felhasználói felületünkön, az általában hatással lesz egy másik elemre is. Ez azt jelenti, hogy valamilyen módon kommunikációt kell létrehozzunk 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 `Q_OBJECT` 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 slothoz is hozzákapcsolhatjuk, ezáltal akár több példányt is értesíthetünk egy bizonyos példányban kialakult változásokról. Az alábbi ábra szemlélteti ezt:



1. ábra: Slot-signal mechanizmus absztakt 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 kövekezőképpen 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 metódusát meghívjuk, ha a kapcsolatot meg szeretnénk szűntetni:

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

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

```
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 megegyszerüsíti 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 dolgozatom 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 szofver 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 utó-feldolgozással 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..?), 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 forrásfájljaikat.

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 alkalmazásom, 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* bgSubstractor = new FrameDifference;
    cv::Mat frame;
    while (1)
    {
      cap.read(frame);
      cv::Mat background;
      cv::Mat foreground;
      if (!frame.empty())
       {
            bgSubstractor->process(frame, foreground, background);
            cv::imshow("Orignal", frame);
            cv::waitKey(30);
    }
    delete backgroundSubstractor;
    cap.release();
```

```
    return 0;
}
```

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

2. ábra: Eredeti képkocka illetve a kiemelt előtér

2.2 Forgalom számlálási megoldások összehasonlítása

Ebben a fejezetben rövid áttekintést vetek a jelenleg is használt módszerek egy részére, végezetül pedig az általam választott módszer mellett érvelek. Mivel a dolgozatom lényege az forgalmi adat szolgáltatása a Robocar City Emulator számára, így szükséges volt áttekintenem a lehetséges forgalom számlálási megoldásokat, hogy egy megfelelőt sikeresen kiválasszak. Az OOCWC platform felépítése A lehetséges megoldások közül választanom kellett egyet, amely számomra elérhető. Az elérhetőség definiálja azt, hogy sikeresen megvalósítható, a rendelkezésre álló idő keretén belül, a számomra elérhető eszközök segítségével, illetve a szükséges szakirodalmi források adottak hozzá. Ezek a módszerek, eszközök a következőek: humán erőforrású forgalomszámlálás, pneumatikus érzékelő, piezoelektromos érzékelő, infravörös érzékelő, videódetektor.

2.2.1 Humán erőforrású forgalomszámlálás

Egy nagyon egyszerű módszer, amivel adatra tehetünk szert az, ha felkérünk embereket, hogy számolják a város bizonyos pontján a forgalmat. Ahogy egy jármű elhalad, ezt az egyén feljegyzi, lehet ez egy jegyző lap, vagy egy elektronikus számláló, akár egy mobil alkalmazás. Ehhez a módszerhez tartozik a közösség alapú adatgyűjtés is [crowd-sourcing]. Ezzel a módszerrel elég pontosan meg tudjuk határozni egy bizonyos szakaszon a forgalom nagyságát, viszont ahhoz, hogy egy egész városnak megfelelő pontossággal mérni tudjuk a forgalmát, ahhoz kár több ezer emberre lenne szükségünk. Általában nem egész napos tevékenység ez, legtöbb esetben 5, 10, 15 perces intervallumokra vannak beosztva az egyének [manual-count]. Az, hogy folyamatos emberi jelenlétet igényel ez a módszer, kizáró tényezőnek minősült számomra.

2.2.2 Pneumatikus érzékelő

Egy másik, egyszerű elven működő, mégis igencsak pontos érzékelő ez. Az egész rendszer lényege az, hogy egy rugalmas anyaból készült tömlőt, levegővel a belsejében az úton keresztbe helyezünk el. Ekkor, abban az esetben, hogy egy jármű áthalad a tömlőn, a tömlő belsejében megváltozik a légnyomás, amelyet a tömlő valamelyik végében beszerelt membrán elektromos impulzussá fogja átalakítani. [pneumatic] A következő fázisban az impulzusokat számlálják, vagy akár továbbíthatják egy feldolgozó szoftver felé, így megkapva az áthaladt járművek számát. Ezt a módszert számomra nem igazán megfelelőnek tartottam, mivel szükséges lenne mélyebb elektronikai tudással rendelkeznem, hogy megfelelően implementálni tudjak.

2.2.3 Piezoelektronikus érzékelő

A piezoelektronikus érzékelő piezo kristályok segítségével működik, melyek külső nyomás hatására elektromos feszültséget generálnak, amikor pedig a nyomás megszűnik, elveszti a feszültséget [piezo]. Ezáltal ennek a szenzornak a segítségével legfőbbképpen olyan helyen tudunk pontosan mérni, ahol a forgalom nagyon ritkán, vagy egyáltalán nem áll. Ennek a feszültségnek a mértéke arányosan változik a kristályokra nehezedő súllyal, így akár a járművek súlyát is egyszerűen meg tudjuk közelíteni. Hasonlóan a pneumatikus

érzékelőkkel, a pezoelektronikus érzékelőkkel is tudunk egyszerűen sebességet mérni, ha egynél több érzékelőt helyezünk el egymás mellé párhuzamosan. Ezeket a szenzorokat az úttest alá szerelik, a járművek haladási irányával merőlegesen, tehát ugyanúgy keresztbe. Mivel az úttest alatt helyezkednek el, így az időjárási viszonyok nem befolyásolják a pontosságát [piezo-properties]. Innen adódik a fő hátránya is: az úttestet keresztbe fel kell vágni ahhoz, hogy elhelyezzük az érzékelőket. Ez viszont számomra lehetetlen feladat, így ez a megoldás sem bizonyult elfogadhatónak.

2.2.4 Infravörös érzékelő

Az infravörös érzékelők alapja a hősugárzás. Mivel minden anyag, amely hőmérséklete az abszolút nulla fok fölött az infravörös sugárzást bocsájt ki, így minden jármű is. A rendszer két fő részből áll: az egyik folyamatosan infravörös jeleket bocsát ki, a másik pedig egy érzékelő amely a jelenleg elhaladó járművek által visszavért infravörös sugarakat észleli. [infrared] Ezeket az érzékelőket általalában az utak fölé, állványokra, vagy mellé szerelik. Forgalom számlálás mellett sebességet mérhetünk a segítségükkel, illetve akár kategorizálhatjuk is a járműveket. Egyik hátránya, hogy folyamatos karbantartást igényel, a jeladók védőlencséjüket tisztítani kell, amely útzárat igényel. Erős havazás, illetve nagy köd esetén pontatlanok lehetnek. Ez a típusú érzékelő megfelelő lenne számomra, viszont mégis nemet kellett mondanom, mivel eme érzékelők ára jócskán egy hétköznapi kamera ára felett van.

2.2.5 Videó érzékelő

Ez az remdszer egy kamera által streamelt képet dolgot fel különböző képfeldolgozási algoritmusok segítségével, így lehetséges a járműveket szegmentálni, megszámlálni egy felvételen. Ehhez a módszerhez szükségünk van egy kamerára illetve egy feldolgozó egységre. A kamera képekekkel szolgál a feldolgozó egység felé. A feldolgozó egység feladata a kép feldolgozása, eredmény szolgáltatása, illetve az eredmény tárolása. A tárolás lehet akár egy lokális merevlemez, vagy egy távoli adatbázis. Mindkét esetben az eredménynek tartalmaznia kell a megfigyelés pontos idejét és helyét. Mivel általában a kamerák statikusak, tehát a helyzetüket nem változtatják, így nem szükséges GPS modullal

felszerelni, mivel elég, ha felszereléskor a feldolgozó egységben beállítjuk a kamera pozícióját. Legtöbb esetben a feldolgozó egységen futó szoftver rendelkezik grafikus felhasználói felülettel is, amely segítségével különböző, a képfeldolgozást befolyásoló paramétereket be lehet állítani. A többi módszernek egy nagy hátránya az, hogy limitált az érzékelők helyze. Viszont a megfelelő képfeldolgozási algoritmusok használatával bármilyen képről sikeresen kaphatunk megfelelő forgalmi adatot. Mivel képfeldolgozó könyvtárak bő tárházát érhetjük el mostmár, így viszonylag gyorsan fejleszteni lehet egy szoftvert amely ezt a feladatot el tudja végezni, többé-kevésbé pontosan. Ahhoz, hogy elfogadható eredményt érjünk el, nem feltétlen szükséges egy nagy teljesítményű érzékelővel rendelkeznünk, mivel jelen esetben csak a forgalom nagyságát mérjük.

2.3 Előzmények

2.3.1 Megelőző forgalomszámláló szoftver

A jelenlegi szoftver, ahogy említettem egy továbbfejlesztett verziója egy korábban általam kifejlesztett forgalomszámláló szofvtervnek. Az eredeti feladat leírása a képzés “Magas szintű programozási nyelvek 2” tárgy keretein belül lett megfogalmazva, ami a következő:

“ • Helén feladat. Írj olyan C++ programot, amely egy adott út forgalmát mutató videófelvételről kiszámolja az adott irányú forgalmat egy olyan függvény formájában, mely megadja, hogy az addig eltelt percekben mennyi jármű haladt át. Nem hegylakó feladat, 320 pont. Két kamerát tudok kölcsönözni a feladathoz.

Címkék: City, Map, videó és képfeldolgozás.” [UDPROG]

Ekkor már rendelkeztem csöppnyi képfeldolgozási tapasztalattal, mivel előző félévben az évkönyv “Labda követése” [BallTracking] című feladatát megoldottam. Ennek a feladatnak a segítségével áttekintést kapthattam a háttérlevállasztásról OpenCV segítségével. Rövid demóját lásd: https://www.youtube.com/watch?v=ffmZHzfMDMc A következő félév végén bemutatott forgalom számláló program már felhasználói felülettel is rendelkezett. Lehetőségünk volt lokális fájlokat betölteni, elindítani a feldolgozást, beletekerni, pillanatképet menteni, kézzel finom-hangolni a képfeldolgozást, a “felismert” objektumok

köré kontúrt megjeleníttetni. A háttér szegmentálást hasonló módon oldottam meg, mint a mostani alkalmazásomban, viszont az előzőben minden esetben két képkockát olvastam be, ahelyett, hogy az első beolvasás után letároljam azt, s a következő lépésben felhasználjam. A felvételre rajzolt vonalak úgynevezett kontrol egyenesek voltak. A vízszíntes vonal, illetve a három függőleges vonal által meghatározott metszéspontok határozták meg a kontrolpontokat. Ha egy jármű mind a három kontrolponton áthaladt, azt jelentette, elhaladt, szóval számoljuk. Ezt az egyszerű elvet követve egész pontos (körülbelül 85%- os) eredményt tudtunk elérni. Ugyanúgy Qt- t használtam a felhasználói felülethez, illetve eseménykezeléshez. Ezt az alkalmazást működés közben a következő linken lehet megtekinteni: https://www.youtube.com/watch?v=ffmZHzfMDMc

2.4 A Robotautó Világbajnokság

2.4.1 A platform koncepciója

2.4.2 <>

2.5 A szoftver elkészítésének folyamata

2.5.1 Tervezés

2.5.2 Fejlesztés

2.5.2.1 Videó források típusok

2.5.2.2 Adatbázis kapcsolatok kialakítása, kezelése

2.5.2.3 Szálkezelés

2.5.2.4 Videó feldolgozása

2.5.2.5 Járművek detektálása

2.5.3 Problémák a szoftver fejlesztése során

2.5.4 A szoftver tesztelése

2.5.5 Eredmények

2.6 A szoftver használata

2.6.1 Fordítása

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/

inqlude: Cornelius Schumacher, All libraries, 2017, https://inqlude.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

crowd-sourcing: , Crowd-sourced traffic data: driven by innovation , 2013, http://www.roadtraffic-technology.com/features/featurecrowd-sourced-traffic-data-android-smartphone/

manual-count: Ministry of Works and Transport, Traffic Data Collection and Analysis, , http://www.vegvesen.no/_attachment/336339/binary/585485

pneumatic: Rodford Edmiston, More Than You Ever Wanted to Know About Counting Cars, 2002, http://www.dcr.net/~stickmak/JOHT/joht02countingcars.htm

piezo: , Piezoelectricity, , https://en.wikipedia.org/wiki/Piezoelectricity

piezo-properties: U.S. Department of Transportation, Traffic Monitoring Guide, 2013, https://www.fhwa.dot.gov/policyinformation/tmguide/tmg_fhwa_pl_13_015.pdf

UDPROG: , The Yearbook of the Programmers of University of Debrecen , 2015