**종합설계 프로젝트 수행 보고서**

|  |  |
| --- | --- |
| **프로젝트명** | **객체 탐지를 응용한 스마트폰 블랙박스** |
| **팀 번호** | **S4-7** |
| **문서 제목** | **수행계획서( )**  **2차 발표 중간보고서( )**  **3차 발표 중간보고서( )**  **4차 수행 보고서( )**  **최종 수행 보고서( o )** |

**2021.11.19**

**팀원 : 김동현 (팀장)**

**심정수 (팀원)**

**지도교수 : 전광일 교수 (인)**

**문서 수정 내역**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **작성일** | **대표작성자** | **버전(Revision)** | **수정내용** |  |
| 2021.12.26 | 김동현 (팀장) | 1.0 | 수행계획서 | 최초작성 |
| 2021.03.02 | 김동현 (팀장) | 2.0 | 2차 발표자료 | 설계서 추가 |
| 2021.05.06 | 김동현 (팀장) | 3.0 | 3차 발표 자료 | 프로토타입 추가 |
| 2021.06.17 | 김동현 (팀장) | 4.0 | 4차 수행 보고서 | 시험 결과 수정 |
| 2021.08.17 | 심정수 (팀원) | 5.0 | 4차 수행 보고서 | 서식 수정 |
| 2021.11.19 | 심정수 (팀원) | 6.0 | 최종 수행 보고서 | 결론 추가 |

**문서 구성**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **진행단계** | **프로젝트**  **계획서 발표** | **중간발표1**  **(3월)** | **중간발표2**  **(5월)** | **학기말발표**  **(6월)** | **최종발표**  **(10월)** |
| **기본양식** | 계획서 양식 | 계획서 양식 | 계획서 양식 | 계획서 양식 | 계획서 양식 |
| **포함되는**  **내용** | Ⅰ. 서론  (1~6)  Ⅱ. 본론  (1~3)  참고자료 | Ⅰ. 서론  (1~6)  Ⅱ. 본론  (1~4)  참고자료 | Ⅰ. 서론  (1~6)  Ⅱ. 본론  (1~5)  참고자료 | Ⅰ. 서론  (1~6)  Ⅱ. 본론  (1~7)  참고자료 | I  II  III |

**이 문서는 한국산업기술대학교 컴퓨터공학부의 “종합설계”교과목에서 프로젝트**

**“객체탐지를 응용한 스마트폰 블랙박스”를 수행하는 (S4-7,김동현, 심정수)들이**

**작성한 것으로 사용하기 위해서는 팀원들의 허락이 필요합니다.**

**목 차**

**Ⅰ. 서론**

1. 작품선정 배경 및 필요성 2

2. 기존 연구/기술 동향 분석 5

3. 개발 목표 6

4. 팀 역할 분담 6

4.1. Github 주소 6

4.2. 역할 분담 6

5. 개발 일정 7

6. 개발 환경 8

**Ⅱ. 본론**

1. 개발 내용 9

2. 문제 및 해결방안 10

3. 시험시나리오 11

4. 상세 설계 13

5. Prototype 구현 17

6. 시험 / 테스트 결과 19

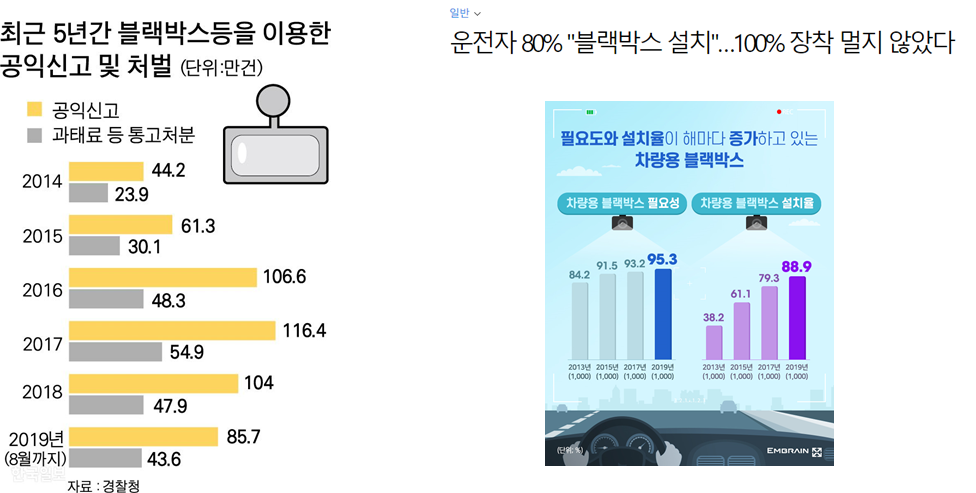
7. Coding / DEMO 21

**Ⅲ . 결론**

1. 연구 결과 60
2. 최종 결론 61
3. 참고 문헌 61

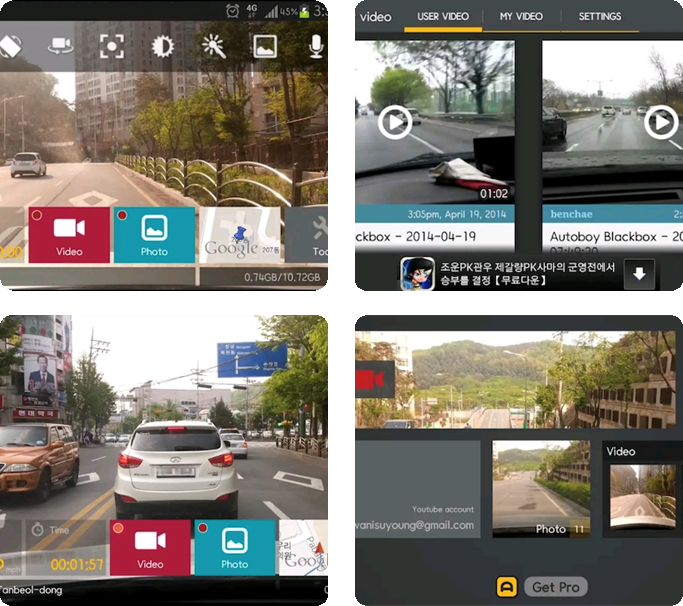
Ⅰ. 서론

|  |
| --- |
| **1. 작품선정 배경 및 필요성** |



<경찰청 2019.08 통계자료> <EMBRAIN 리서치 기업 2019.02.22 통계자료>

|  |  |
| --- | --- |
| **개발 배경** | **내 용** |
| 블랙박스 설치 비율 증가 | 시장조사전문기업 엠브레인 트렌드모니터가 차량용 블랙박스와 관련한 인식조사를 실시한 결과, 2019년 운전하는 차량에 블랙박스를 장착한 운전자는 88.9%였다. 2013년에는 38.2%만이 차량용 블랙박스를 장착했다고 응답 했으나, 불과 수년 새 50%의 증가율을 보였다. 이처럼 블랙박스의 필요성이 증가하고 있는 가운데 여전히 설치하지 않은 차량이 약 240만대 존재하며 원인으로 비용, 설치 과정의 부담 등이 있었다. 스마트폰을 블랙박스로 활용함으로써 해당 문제를 해결하고 블랙박스 사용성을 높이고자 한다. |
| 블랙박스를 영상을 통한 민원신고 증가 | 2019년 경찰청 통계자료에 따르면 교통법규 위반에 관한 공익신고가 한달에 약 10만건 접수된다고 한다. 또한 경찰은 공익신고 증가가 실제 교통안전 개선에도 영향을 주고 있다고 평가했다. 단순 비교는 어려울 수 있으나 공익 신고 증가와 교통사고 사망자 수 감소가 상당 부분 연관성이 있다고 밝혔다. 교통 법규 위반 신고를 위해서는 블랙박스 영상에서 필요한 부분을 추출해야하는 번거로움을 감수 해야 하는데 스마트폰 블랙박스에 자동 영상 추출기능을 탑재함으로써 보다 원활하게 공익신고가 이루어 질 수 있도록 하고자 한다. |



**<Google Play Store 1위 블랙박스 App – 오토보이 블랙박스>**

|  |  |
| --- | --- |
| **필요성** | **내 용** |
| 문제 | * 현재 시중에서 판매중인 블랙박스의 경우 약 30만원 상당의 비용을 지불해야하며 별도로 설치 비용을 부담해야 함. * 현재 출시 되어있는 블랙박스 App들의 경우 주행속도, 충격감지, 녹화 화질 선택등 기본적인 기능만 제공하고 있음. * 사고 영상, 공익 신고 영상을 위해 필요 영상을 찾아내어 추출하는 번거로운 과정을 감수해야 함 |
| 개선 | * 스마트폰을 블랙박스로 활용함으로써 블랙박스 사용을 위해 부담해야 하는 비용 문제를 해결함 * 블랙박스 앱의 기본기능에서 나아가 영상처리를 통해 부가적인 알림 기능들을 제공하여 운전자의 안전주행을 유도함 * 하이라이트 추출 기능을 탑재함으로써 영상 추출의 번거로운 과정을 해결함 |

|  |
| --- |
| **2. 기존 연구 / 기술 동향 분석** |

|  |  |
| --- | --- |
| **개발내용** | **내 용** |
| **국내 관련 연구** | **< 차량용 블랙박스 영상을 이용한 환경적응적 실시간 차선인식 연구 >**  - 소개 : 주행 중인 차선을 인식하여 이탈 경고 제공  - 장점 : 라이다 센서를 이용하여 보다 정확한 차선 인식  - 단점 : 블랙박스 녹화 영상만으로 구현할 수 없음, 별도의 센서 필요  **< 블랙박스 사각거리 이내의 근접차량 진행거리 산출방법에 관한 연구>**  - 소개 : 블랙박스 영상에서 보이는 상대방 차량 높이 변화값을 분석하여 차량 진행거리 산출  - 장점 : 카메라 촬영각도를 삼각함수로 산출한 후 사고차량의 지상고 변화와수평 이동거리 관계식 정리함으로써 높은 정확도의 결과 값 산출  - 단점 : 설치 각도에 따라 같은 상황에도 다른 결과 값 산출 |
| **국외**  **관련 연구** | **< Anticipating Accidents in Dashcam Videos>**  - 소개 : RNN을 응용하여 영상 처리를 통해 교통사고를 예측하는 알고리즘  - 장점 : 74.35%의 높은 정밀도를 달성  - 단점 : RNN방식으로 정확도는 높으나 속도가 느려 실시간에는 부적합  **< Classification of Crash and Near\_Crash Events from Dashcam Videos and Telematicc >**  - 소개 : 블랙박스 영상을 분석하여 충돌 및 충돌에 가까운 이벤트를 분류하는 연구  - 장점 : Convolutional Neural Network를 기반으로 하여 87%이상의 정확도를 보임  - 단점 : CNN방식으로 실시간에 부적합, 충돌 여부를 물리적으로 측정 불가. |
| **공통적인 문제점** | **<별도의 장치 필요>**   * 거리 및 차선 검출을 위해 고가의 라이다 센서 탑재 필요 * 블랙박스에서 실시간으로 영상처리를 수행할 수 없음   **< 실시간 처리에 부적합한 영상처리 알고리즘 >**   * CNN, RNN등 정확도가 높은대신 속도가 느린 알고리즘을 사용함으로써 실시간처리에 부적합함   **< 하나의 장치와 하나의 기능이 일대일로 대응 >**   * 스마트폰처럼 다양한 센서를 기반으로 복합적인 기능 제공 불가 |

|  |
| --- |
| **3. 개발 목표** |

|  |
| --- |
| **최종목표** |
| * 고가의 블랙박스를 스마트폰으로 대체하여 블랙박스를 설치하지 않은 차량의 운전자가 부담없이 블랙박스 기능을 이용할 수 있음 * 운전자에게 실시간으로 주행 차선 이탈 방지 알림 기능을 제공함 * 운전자에게 실시간으로 앞차와의 안전 거리 유도 알림 기능을 제공함 * 충돌 시 신고 팝업을 띄워 운전자가 보다 쉽게 신고할 수 있도록 충돌 감지 알림 기능을 제공함 * 충돌 이벤트 발생시 전후 하이라이트 영상을 추출­ * 진로 변경 위반 감지 기능을 통해 하이라이트 영상을 추출 |

|  |
| --- |
| **4. 팀 역할 분담** |

­

**4.1. Github 주소**

|  |
| --- |
| **Github 주소** |
| <https://github.com/kpuce2021/Easy> |

**4.2. 역할 분담**

|  |  |  |
| --- | --- | --- |
| **팀원** | **Github ID** | **역할 분담** |
| 김동현 | Dhkim0405 | * 팀회의 주관 및 일정 조율 * Android opencv환경 구축 * Opencv차선 검출 기반 이벤트 체크 및 알림 기능 구현 * Opencv 거리측정기반 충돌알림 기능 구현 * Android App 구현 * Android 센서기반 충돌감지 알림기능 구현 |
| 심정수 | Simking95 | * AWS\_EC2 가상 머신 환경 구축 * Ubuntu OpenCV 영상처리 환경 구축 * AWS\_RDS DB 서버 구축 * DB 상세설계 * Ubuntu node.js 웹서버 환경 구축 * Node.js 웹서버 MySQL DB 연동 * Android, EC2 영상 파일, 센서 정보 전송 기능 구현 * Android App 구현 * Android 센서기반 충돌감지 알림 기능 구현 * 하이라이트 추출 배치 프로그램 구현 |

|  |
| --- |
| **5. 개발 일정** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **추진내용** | **추진일정** | | | | | | | | | | | | | |
| **11월** | **12월** | **1월** | **2월** | **3월** | **4월** | **5월** | **6월** | **7월** | **8월** | **9월** | **10월** | **11월** |
| 아이디어 구체화 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 자료수집 및 개발환경 구축 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 시스템 설계 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 구현 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 데모 테스트 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 문서화 및 발표 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 최종 보고서 작성 |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |
| --- |
| **6. 개발 환경** |

|  |  |
| --- | --- |
| **적용 기술** | **내용** |
| 개발 환경 | * Ubuntu 20.04.2.0 LTS(Focal Fossa) * Mac OS 10.15.7 (Catalina) * Galaxy s10e (Android Q 10.0) |
| 개발 언어 | * C++ * Java * Java Script * MySQL |
| 개발 도구 | * Android Studio * MySQL Workbench * Xcode * Visual Studio Code |
| 서버 환경 | * Amazon EC2 Ubuntu 20.04.2.0 LTS |
| 주요 라이브러리 | * OpenCV * Android API |
| 데이터 베이스 | * AWS RDS MySQL |

Ⅱ. 본론

|  |
| --- |
| **1. 개발 내용** |

**1.1. 개발 내용**

|  |  |
| --- | --- |
| **분류** | **내용** |
| 전체  기능 | * 차량에 설치한 카메라를 이용하여 촬영하고 해당 영상을 이용하여 차량 주행 정보를 분석하고 운전자에게 주행 보조 알림기능을 제공하고 원활한 민원신고를 위한 하이라이트 영상을 추출 * 전방 차량의 주행 경로를 추적하여 좌표 데이터로 변환하고 해당 좌표 값과 실제 거리의 좌표 값을 비교하여 전방 주행차량과의 거리를 파악하여 안전거리 확보를 위한 알림기능 제공 * 주행중인 차선의 검출 유무를 통해 운전자가 주행중인 차량이 올바른 경로에서 벗어났을 경우 알림을 통해 안전한 주행을 유도 * 안드로이드 센서(GPS, 가속도, 자이로스코프)를 응용하여 주행중인 차량의 충돌을 감지하고 전방 차량 트래킹 결과를 반영하여 신고를 위한 팝업 기능 제공 * 주행이 끝난 영상을 서버로 전송하여 진로 변경 위반 감지 및 충돌 감지에 따른 하이라이트 영상을 추출 * 데이터베이스를 구축하여 로그인/회원가입 기능을 구현하고 사용자 정보를 저장 |
| 영상  분석 | * 허프 변환 메서드(Houghlines)를 통해 직선 검출 * 차량을 Detection 하여 Tracking 수행 * Tracking -> update를 통해 실시간으로 객체 좌표 파악 * 파악한 좌표값을 기반으로 이벤트 발생 |
| 서버 | * 서버는 node.js를 사용한다. 사용언어는 java Script를 이용 * 데이터베이스는 MySQL을 이용하며, 회원가입시 전달받은 회원정보를 저장함 * 영상처리를 위한 가상서버는 AWS\_EC2를 이용하고 Ubuntu 서버에 openCV개발 환경을 구축 * Node.js npm모듈을 이용하여 데이터베이스와 서버를 연결함 * 셸 스크립트 배치 프로그램을 구현, 자동 하이라이트 추출 |

|  |
| --- |
| **2. 문제 및 해결방안** |

|  |  |
| --- | --- |
| **문제** | **해결 방안** |
| 스마트폰 성능 부족으로 인한 영상처리 속도가 저하되는 문제 | * Region Of Interest(관심영역)을 지정하여 연산 최소화 차량의 좌 우측 하단에 영역을 지정하여 차선을 검출 |
| 차선 이외의 불필요한 직선 검출 되는 문제 | * Region Of Interest(관심 영역)을 좌 우측 하단에 지정하여 차선이 위치해 있는 공간에 직선 검출 수행 * HoughLines 메서드를 통해 검출된 직선의 기울기(세타)의 범위를 제한하여 차선과 유사한 기울기를 가진 직선 검출 |
| Detection 수행 시 옆 차선의 차량이 검출되는 문제 | * Region Of Interest(관심 영역)을 현재 차량이 주행중인 차선으로 제한 하여 Detection 횟수 최소화 * 영상의 width/2 값과 가장 가까운 거리의 x좌표를 가진 차량을 Tracker에게 전달 |
| Detection한 객체를 둘러싼 Rect 크기가 객체의 위치나 크기에 상관없이 변해 Rect의 크기를 이용한 거리탐지 정확도 저하 | * 주행 속도 60km/h 이상의 상황에서 차량 detection을 수행하고 수행 결과로 얻은 Rect의 크기를 계산하여 전방 약 30m에 해당하는 차량에 대한 알림 발생 |
| App 내에서 하이라이트 영상을 추출 했을 때 성능 저하 | * AWS에서 제공하는 클라우드 컴퓨팅 서비스인 EC2를 사용해 App과 서버를 연동하여 영상파일 전송만으로 하이라이트 추출이 성능과 무관하게 가능하도록 하고 Local 서버의 물리적 한계점을 극복 |
| 서버상의 opencv 개발환경 구축 시 EC2 사용용량 한계 및 성능 저하 | * Opencv 설치 과정중 cmake 빌드에서 멀티 쓰레드를 통한 빌드 과정의 의존성 에러가 발생 * AWS프리티어 서비스의 성능한계를 예상하고 python라이브러리를 제외한 c++라이브러리만 install 한 후 cmake 환경 변수 재설정 |
| 서버상에서 테스트를 위한 opencv 영상처리 과정중 gtk 라이브러리가 서버와 연결되지 않음 | * Local환경에서 ubuntu 서버에서 작업한 opencv 결과물을 확인하기 위해 텍스트 기반으로 만들어졌던 유닉스/리눅스 시스템에서 그래픽 인터페이스를 사용할 수 있는 xming사용 |
| AWS EC2를 재 가동시 AWS에서 제공하는 IP주소가 변경 | * EC2에서 제공하는 탄력적 IP를 사용하여 인스턴스 재 가동시 고정된 주소로 접근 할 수 있게 함 |
| 안드로이드 to 서버 접근 시 http프로토콜 오류 | * https 프로토콜 사용으로 네트워크 보안 요구사항 충족 |

|  |
| --- |
| **3. 시험시나리오** |

|  |  |
| --- | --- |
| **시나리오명** | **내 용** |
| 사용자 스마트폰 | * 차량에 스마트폰을 거치하여 차량 전방을 촬영하여 영상정보를 수집함 * 수집된 영상정보를 분석하여 실시간 알림 기능을 제공함 * 수집된 영상정보를 서버로 전송하여 결과를 저장함 * 스마트폰 센서 기반 충돌 감지 기능 제공함 |
| 서버 | * 스마트폰으로 부터 영상정보를 전달 받아 분석하여 결과를 전송 * 사용자 회원가입시 입력받은 정보를 데이터베이스에 저장 |
| 데이터베이스 | * 회원 정보 저장 |

|  |
| --- |
| **시나리오 (전체)** |
|  |

|  |
| --- |
| **스토리 보드** |
|  |

|  |
| --- |
| **4. 상세 설계** |

|  |
| --- |
| **시스템 상세 설계(개요)** |
|  |

|  |
| --- |
| **시스템 상세 설계(영상처리)** |
|  |

|  |
| --- |
| **시스템 상세 설계(영상처리)** |
|  |

|  |
| --- |
| **시스템 모듈** |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 |

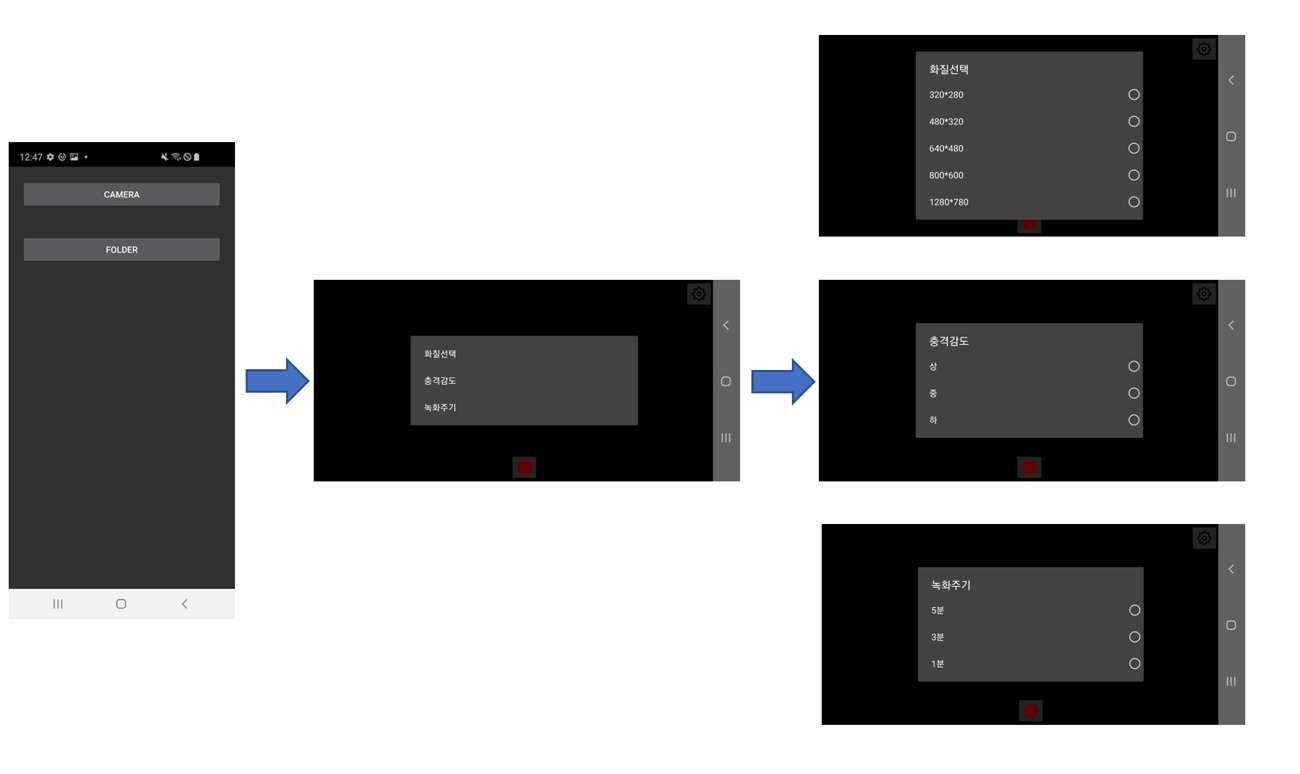
|  |  |
| --- | --- |
| **분류** | **내용** |
| 로그인/회원가입 모듈 | 모듈   * Body-parser   핵심 메서드   * App.post() : Post 방식으로 데이터 request, response * INSERT INTO USERS() VALUES() : Post방식으로 받아온 데이터를 DB테이블에 저장 * Select\*from Users where UserEmail : 이메일을 이용해 해당 아이디를 포함한 테이블이 있는지 확인하고, 비밀번호가 일치하는지 검사 |
| 블랙박스 모듈 | 모듈   * import org.opencv.android.OpenCVLoader * import android.media.MediaRecorder   핵심 메서드   * onCameraFrame() : 스마트폰 카메라로부터 수집되는 영상정보를 프레임단위로 전달 * MediaRecorder.setAudioSource() : 음성 데이터 저장 * MediaRecorder.setVideoSource() : 비디오 소스 설정 * MediaRecorder.setOutputFile() : 저장경로 설정 * MediaRecorder.start() : 영상 및 음성 데이터 녹화   결과   * 스마트폰내부 저장소(지정된 경로)에 녹화된 파일 저장 |
| 충돌감지 모듈 | 모듈   * import android.location.Location * import android.hardware.Sensor   핵심 메서드   * locationManager.requestUpdates() : 현재 gps정보 업데이트 * locationManager.getLastKnownLocation() : 현재 위치값 받아오기 * onSensorChanged() : 센서 값이 변경될 때 실행되는 이벤트 함수 * SensorEvent() : 센서값을 저장 * Detect\_crash() : gps를 통해 얻은 차량의 속도가 급격하게 줄어듦과 동시에 자이로스코프 센서와 가속도 센서의 값이 특정 값 이상으로 상승하면 이벤트 발생 |
| 이탈방지 모듈 | 모듈   * import cv2   핵심 메서드   * GaussianBlur() : 중앙값에 가중치를 더 주고 주변은 흐리게 함으로써 노이즈 제거 * Canny() : 가장자리 검출을 통해 이미지 이진화 * HoughLines() : 검출된 엣지들에 잇는 점을 적용하여 직선 검출   반환 값 : vector<2f> : 검출된 직선의 정보가 담겨있는 벡터 |
| 거리측정 모듈 | 모듈   * import cv2 * import cv2/tracking.hpp   핵심 메서드   * detectMultiScale() : 파라미터로 받은 .xml파일 기반으로 객체 검출 수행 * alarmBasedDetection() : detection수행 결과로 얻은 Rect의 size를 계산하여 전방 약 30m에 해당 하는 차량에 대한 알림 발생 |
| 진로 변경 위반 모듈 | 모듈   * import cv2 * import cv2/tracking.hpp   핵심 메서드   * detectMultiScale() : 파라미터로 받은 .xml파일 기반으로 객체 검출 수행 * tracker.update() : 객체의 위치 변화를 프레임 단위로 업데이트 * vehicle\_Interrupt() : 차량 tracking으로 얻은 객체의 좌표가 약 10m 이내의 거리에서 차량 전방에 지정된 영역 내 횡으로 움직일 경우 이벤트 발생 |
| DB연동 모듈 | 모듈   * var mysql=require(‘mysql’);   핵심 메서드   * connection() : node.js 서버에서 DB에 접근 하기 위해 사용 * connection.query() : DB에 연결하여 실제 데이터베이스의 값을 다룸 |
| 데이터 전송 모듈 | 모듈   * retrofit   핵심 메서드   * retrofit.create() : ServiceAPI 객체 생성 * Call<ResponseBody> upload() : Server에 mp4파일 업로드 |
| 하이라이트 추출 모듈 | 모듈   * Import opencv/cv.hpp   핵심 메서드   * VideoWrite.write() : 이벤트 발생 구간 전후 15초에 해당하는 영상 Frame을 저장 |

|  |
| --- |
| **5. prototype 구현** |

|  |
| --- |
| **녹화 영상** |



|  |
| --- |
| **녹화 설정** |

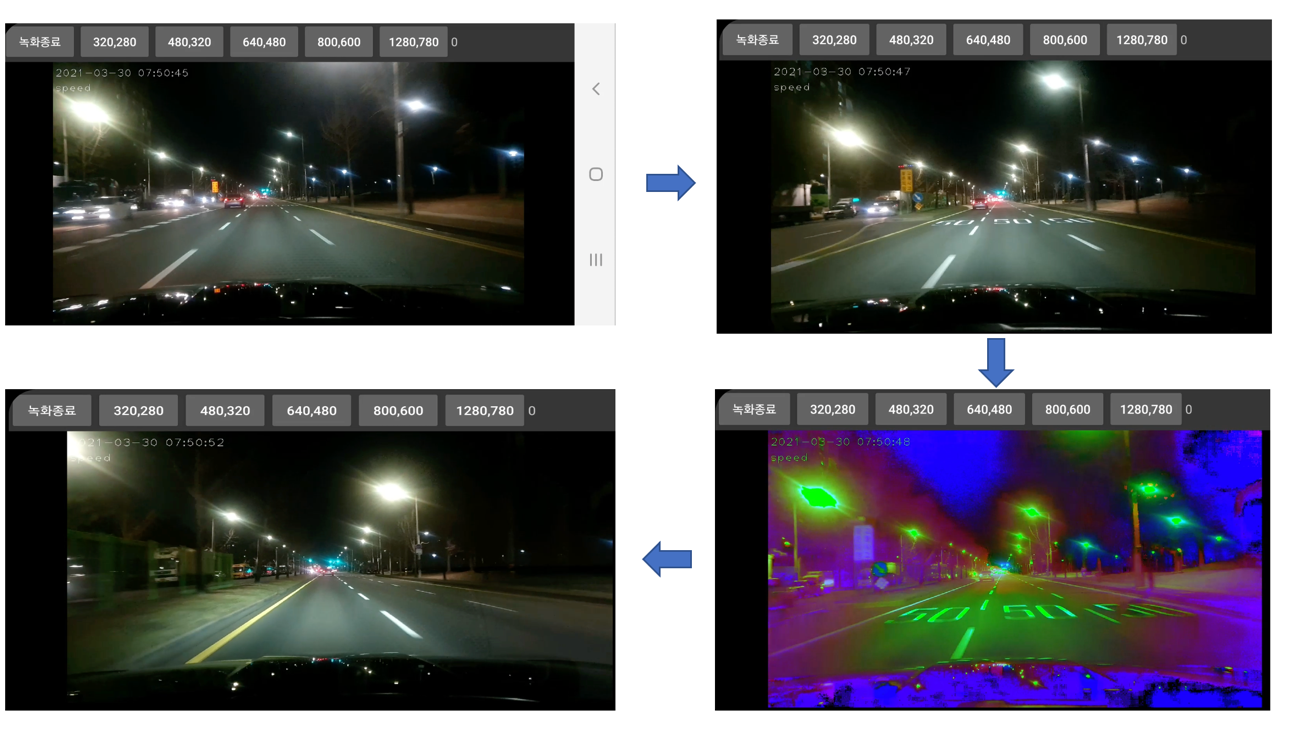


|  |
| --- |
| **거리감지 영상** |

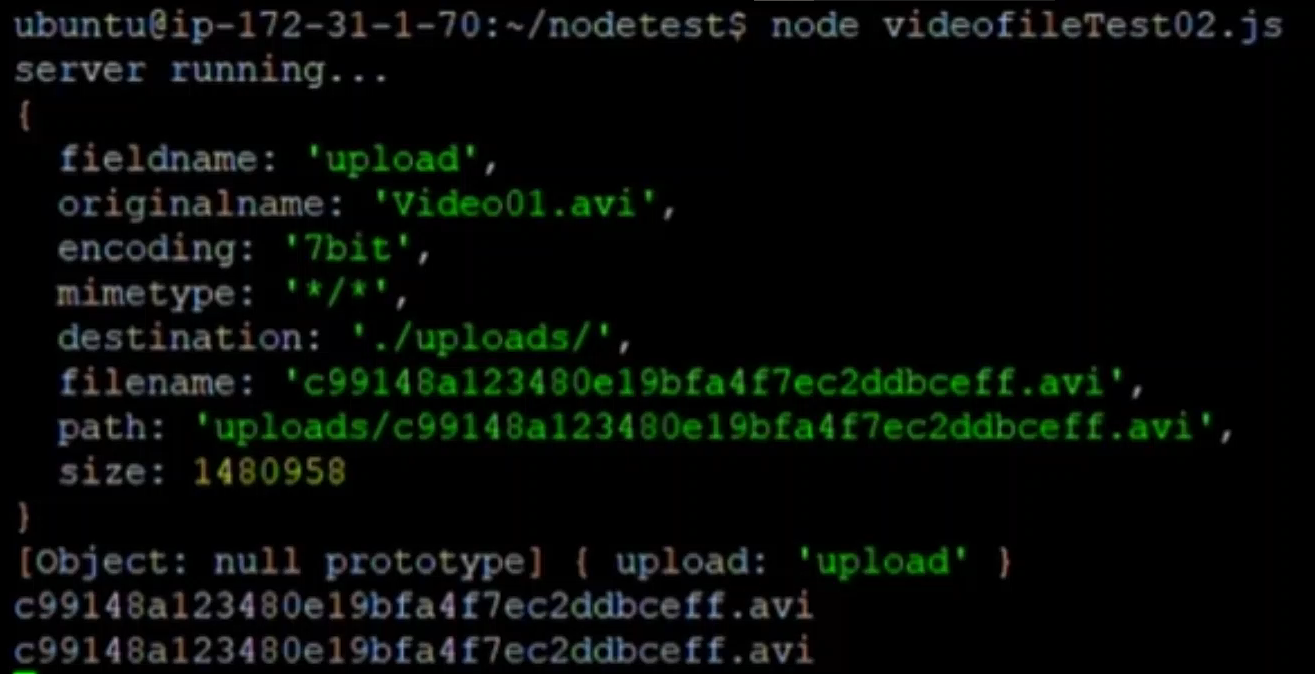
텍스트, 하늘, 도로, 길이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| **차선감지 영상** |



|  |
| --- |
| **데이터 송,수신 (Server)** |



|  |
| --- |
| **배치 프로그램** |



|  |
| --- |
| **6. 테스트 / 결과** |

|  |  |  |
| --- | --- | --- |
| **구분** | **시험 내용** | **결과** |
| **서버/데이터베이스** | **상호 연결이 잘 이루어지는가 (로그인/회원가입)** | **O** |
| **영상 수신이 원활히 이루어지는가 (영상타입/용량)** | **O** |
| **영상 송신이 원활히 이루어지는가 (영상타입/용량)** | **O** |
|  | **Shell 명령어를 통한 순서가 논리적 구조로 구성되었는가** | **O** |

|  |  |  |
| --- | --- | --- |
| **구분** | **시험 내용** | **결과** |
|  | **디바이스로 부터 영상 수집이 원활이 이루어지는가(프레임 단위)** | **O** |
| **영상 저장이 원활히 이루어 지는가(5분단위)** | **O** |
| **영상처리 수행 중 gps가 정상적으로 동작 하는가** | **O** |
| **네이티브 환경** | **충격 감도에 따른 충격 감지가 정상적으로 동작 하는가** | **O** |
|  | **이벤트에 따른 알림이 정상적으로 동작하는가** | **O** |
|  | **수집된 영상 정보를 바탕으로 디텍션 모듈이 잘 동작하는가** | **O** |
|  | **추출 정보가 사용자에게 적절히 제공되는가** | **O** |

|  |  |  |
| --- | --- | --- |
| **구분** | **시험 내용** | **결과** |
| **영상 처리**  **(서버)** | **다양한 상황에서 끼어들기 감지 기능이 높은 정확도를 유지하는가** | **O** |
| **저장된 데이터 기반 하이라이트 추출이 잘 동작하는가** | **O** |

|  |  |  |
| --- | --- | --- |
| **구분** | **시험 내용** | **결과** |
| **영상 처리**  **(네이티브)** | **차선 인식이 원활히 이루어 지는가(인식 정확도)** | **O** |
| **차선 인식 기반 알림 시점이 적합한가(실시간성)** | **O** |
|  | **거리 감지 기능이 원활히 이루어 지는가** | **O** |

|  |
| --- |
| **7. Code&DEMO** |

|  |
| --- |
| **Android Camera** |
| **public class MainActivity extends AppCompatActivity implements CameraBridgeViewBase.CvCameraViewListener2, SensorEventListener , LocationListener {**  **private CameraBridgeViewBase mOpenCvCameraView; // camera 로부터 얻은 화면을 View에 띄울 변수 선언**  **protected void onCreate(Bundle savedInstanceState) {**  **super.onCreate(savedInstanceState);**  **//camera로 부터 얻은 데이터를 띄울 전체 화면 생성**  **getWindow().setFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN,**  **WindowManager.LayoutParams.FLAG\_FULLSCREEN);**  **getWindow().setFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON,**  **WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON);**    **mOpenCvCameraView = findViewById(R.id.activity\_surface\_view);**  **mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);**  **mOpenCvCameraView.setCvCameraViewListener(this);**  **mOpenCvCameraView.setCameraIndex(0); // front-camera(1), back-camera(0)**  **}**  **// BaseLoaderCallBack 메소드 호출시 opencv 종속 카메라 라이브러리 로드**  **private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {**  **@Override**  **public void onManagerConnected(int status) {**  **switch (status) {**  **case LoaderCallbackInterface.SUCCESS:**  **{**  **mOpenCvCameraView.enableView();**  **} break;**  **default:**  **{**  **super.onManagerConnected(status);**  **} break;**  **}**  **}**  **};**  **// Frame 단위 영상 처리 수행 리턴된 Mat을 View에 전달**  **public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {**  **matInput = inputFrame.rgba();**  **if(record\_flag==true) { //record 시작 포인트 및 녹화중 상태**  **int w=matInput.cols();**  **int h=matInput.rows();**  **if(!onRecording){ //녹화 시작 포인트에만 해당**  **time=getTime();**  **baseDir\_Recorder = Environment.getExternalStorageDirectory().getPath(); // 기본 저장 경로**  **pathDir\_Recorder = baseDir\_Recorder + File.separator+"/녹화영상/"+getTime()+".avi";**  **//information of VideoFile**  **info\_path=baseDir\_Recorder + File.separator+"/녹화정보/"+getTime()+".txt";**  **//voideoWriter를 통해 avi영상을 저장하기 위해 writer set**  **// android opencv 의 경우 avi외에 포맷을 지원하지 않음 .**  **videoWriter=new VideoWriter(pathDir\_Recorder, VideoWriter.fourcc('M','J','P','G'),20,new Size(w,h), true);**  **videoWriter.open(pathDir\_Recorder, VideoWriter.fourcc('M','J','P','G'),20,new Size(w,h), true);**  **onRecording=true;**  **location\_flag=true; //delete**  **}**  **if(!videoWriter.isOpened()){**  **finish();**  **}**  **//현재 시간 화면에 출력**  **putText(matInput, getTime(), new Point(5,40), FONT\_HERSHEY\_PLAIN, 2, new Scalar(255,255,255));**  **//현재 속도 화면에 출력**  **putText(matInput, Integer.toString((int)speed\_location), new Point(5,80), FONT\_HERSHEY\_PLAIN, 2, new Scalar(255,255,255));**  **videoWriter.write(matInput); // video writer 수행**  **Mat canny = new Mat(); // 원본 Mat canny;**  **Mat Roi1, Roi2, Roi;**  **if (matResult == null) {**  **matResult = new Mat(matInput.rows(), matInput.cols(), matInput.type()); // 연산 최소화를 위한 영역 지정**  **}**  **count = ConvertImage(matInput.getNativeObjAddr(), matResult.getNativeObjAddr(), count); // 차선 검출 메서드 호출 native-lib 구현**  **if (count > 18) {**  **alarmImage(matInput.getNativeObjAddr(), matResult.getNativeObjAddr()); // 이벤트 발생 시점->화면전환**  **//tone.startTone(ToneGenerator.TONE\_CDMA\_PIP, durationOfAlarm); //차선 이탈 알림**  **}**  **detect(cascadeClassifier\_car, matInput.getNativeObjAddr(), matResult.getNativeObjAddr()); // cars.xml 기반 차량 검출 메서드 호출**  **}**  **return matInput;**  **}** |

|  |
| --- |
| **충격감지** |
| **public class MainActivity extends AppCompatActivity implements CameraBridgeViewBase.CvCameraViewListener2, SensorEventListener , LocationListener {**  **//variable 가속도 센서**  **private SensorManager sensorManager;**  **private Sensor senAccelerometer;**  **private long lastUpdate=0;**  **private float last\_x, last\_y, last\_z;**  **private int COLLISION\_THRESHOLD=2000; //충돌임계값 -> 값이 낮을 수록 작은 충돌에도 이벤트 발생**  **protected void onCreate(Bundle savedInstanceState) {**  **super.onCreate(savedInstanceState);**  **setContentView(R.layout.activity\_main);**  **//가속도 센서 설정**  **sensorManager =(SensorManager)this.getSystemService(SENSOR\_SERVICE);**  **senAccelerometer=sensorManager.getDefaultSensor(Sensor.TYPE\_LINEAR\_ACCELERATION);**  **sensorManager.registerListener(MainActivity.this, senAccelerometer,SensorManager.SENSOR\_DELAY\_NORMAL);**  **}// end of onCreate**  **@Override**  **public void onSensorChanged(SensorEvent sensorEvent) {**  **Sensor mySensor = sensorEvent.sensor;**  **if(mySensor.getType() == Sensor.TYPE\_LINEAR\_ACCELERATION){**  **float x = sensorEvent.values[0];**  **float y = sensorEvent.values[1];**  **float z = sensorEvent.values[2];**  **long curTime = System.currentTimeMillis(); // 현재시간**  **// 0.1초 간격으로 가속도값을 업데이트**  **if((curTime - lastUpdate) > 100) {**  **long diffTime = (curTime - lastUpdate);**  **lastUpdate = curTime;**    **double collision\_detect = Math.sqrt( Math.pow(z - last\_z,2)\*100 + Math.pow(x-last\_x,2)\*10+ Math.pow(y-last\_y,2)\*10)/ diffTime \* 10000;**  **if (collision\_detect > COLLISION\_THRESHOLD) {**  **//지정된 수치이상 흔들림이 있으면 실행**  **//tone.startTone(ToneGenerator.TONE\_CDMA\_PIP, durationOfAlarm); //check 주석삭제**  **info\_crash++;**  **} else {**  **}**  **//갱신**  **last\_x = x;**  **last\_y = y;**  **last\_z = z;**  **}**  **}**  **}** |

|  |
| --- |
| **차선 인식** |
| **bool find\_left(Mat\* frame, vector<Vec2f> lines){ // 수집된 영상정보의 왼쪽 부분에서 검출된 직선중 왼쪽 차선을 검출**  **bool flag=false; // 정상적으로 차선 검출시 true 리턴**  **int minIndex=0; // 가장 작은 세타 값을 가진 직선의 Index를 저장할 변수**  **if(lines.empty()!=1){**  **for(int i=0; i<lines.size(); i++){**  **if(lines[minIndex][1]>lines[i][1]&&lines[i][1]<1.1&&lines[i][1]>0.8) { // 세타 범위내 가장 값이 작은 직선의 Index 저장**  **minIndex = i;**  **}**  **}**  **if(lines[minIndex][1]<1.1&&lines[minIndex][1]>0.8){**  **drawLines(frame, lines[minIndex][0], lines[minIndex][1]);**  **flag=true; // 해당 범위내 직선의 존재한다면 true 리턴**  **}**  **}**  **return flag;**  **}**  **//========================================================================================**  **bool find\_right(Mat \*frame, vector<Vec2f> lines){ // 수집된 영상정보의 오른쪽 부분에서 검출된 직선중 오른쪽 차선을 검출**  **bool flag=false;**  **int maxIndex=0;**  **if(lines.empty()!=1){**  **for(int i=0; i<lines.size(); i++){**  **if(lines[maxIndex][1]<lines[i][1]&&lines[i][1]>2.1&&lines[i][1]<2.5){ // 세타 범위내 가장 값이 큰 직선의 Index 저장**  **maxIndex=i; //최대 세타 인덱스**  **}**  **}**  **if(lines[maxIndex][1]>2.1&&lines[maxIndex][1]<2.5){**  **drawLines(frame, lines[maxIndex][0], lines[maxIndex][1]);**  **flag=true; // 해당 범위내 직선의 존재한다면 true 리턴**  **}**  **}**  **return flag;**  **}** |

|  |
| --- |
| **차량 트래킹** |
| **Rect trackCars(Mat \*detectFrame, Mat \*baseFrame,bool \*init\_flag, bool \*track\_flag,Rect target, Ptr<Tracker> tracker, int \*process\_size, CascadeClassifier classifier, vector<int> \*vector\_x, bool \*indicatorFlag){ //vectorsave**    **vector<Rect> cars;**    **Mat frame=\*detectFrame;**  **Mat tempFrame=\*baseFrame;**  **Mat grayFrame;**    **int direction;**  **cvtColor(\*detectFrame, grayFrame, COLOR\_BGR2GRAY);**    **if(target.x/2<tempFrame.cols/2){**  **direction=0;**  **}else{**  **direction=1;**  **}**    **if(\*init\_flag==true){**    **startFrame=frameCount; // 영상 편집에 사용될 start Frmae -> event 객체 생성하여 저장**  **\*indicatorFlag=false;**    **vector\_x->clear(); //delete**  **//vector\_mat->clear(); //vectorsave**    **tracker->init(grayFrame, target); //parameter flag 가 false 이면 tracker 초기화 detection결과를 전달하고 이를 트래킹 하도록 초기화**  **vector\_x->push\_back(target.x/2); //delete**        **// tracker->init(\*baseFrame, target); //parameter flag 가 false 이면 tracker 초기화 detection결과를 전달하고 이를 트래킹 하도록 초기화**  **Rect drawCars=target;**  **drawCars.y=target.y+2\*tempFrame.rows/5;**  **rectangle(\*baseFrame, drawCars, Scalar( 0, 255, 0 ), 2, 1 );**    **Mat temp=frame(target);**  **//cvtColor(temp, temp, COLOR\_BGR2GRAY);**  **//classifier.detectMultiScale(temp, cars);**  **}else{**  **tracker->update(grayFrame, target); //flag param true 이면 트래킹 이어서 수행 -> 어떤 객체를 트래킹 해야할지 init되었으므로 트래킹 이어서 수행**    **//tracker->update(grayFrame, target); //flag param true 이면 트래킹 이어서 수행 -> 어떤 객체를 트래킹 해야할지 init되었으므로 트래킹 이어서 수행**  **vector\_x->push\_back(target.x/2); //delete**  **//vector\_mat->push\_back(frame); vectorsave**    **Rect drawCars=target;**  **drawCars.y=target.y+2\*tempFrame.rows/5;**    **rectangle(\*baseFrame, drawCars, Scalar( 255, 0, 0 ), 2, 1 );**    **putText(\*baseFrame, "Tracking car", Point(drawCars.x,drawCars.y+30), FONT\_HERSHEY\_DUPLEX, 1, Scalar(255,255,255));**    **//delete start**    **string str\_locationX = to\_string(drawCars.x/2);**  **putText(\*baseFrame, "location x : "+str\_locationX, Point(drawCars.x,drawCars.y+60), FONT\_HERSHEY\_DUPLEX, 1, Scalar(255,255,255));**    **Mat temp=tempFrame(drawCars);**    **if(\*indicatorFlag==false){**  **\*indicatorFlag=findIndicator(temp, direction); //findIndicator는 indicatorflag가false일때만 true를 검사 -> threshold 값 체크할것**  **}**      **//true -> false로 변경 해야함 !!! && 해당 조건식 충족한 상태로 반복되는 문제 해결해야함 딱 한번만 수행하도록 설정 해야함**  **if(vector\_x->back()>250&&vector\_x->front()<100&&\*indicatorFlag==true){ //내 차선으로 진입할 경우 그리고 indicator 가 false일 경우 Event 객체를 생성하여 videowrite method 에 전달**  **endFrame=frameCount;**  **cout<<"start frame : "<<startFrame<<endl;**  **cout<<"end Frame: "<<endFrame<<endl;**    **videoEvents.push\_back(VideoEvent("change\_test.mp4", "RightIndicator", startFrame-20, endFrame+20));**  **extractVideo(videoEvents);**      **\*track\_flag=false;**    **}**      **//delete end**  **//cvtColor(temp, temp, COLOR\_BGR2GRAY);**    **imshow("tracking cars", temp);**    **}**    **\*process\_size=target.width\*target.height;**  **int tempSize=\*process\_size;**    **//check point**    **if(\*process\_size<90000||vector\_x->back()>250){//if(cars.empty()||\*process\_size<100000){**  **\*process\_size=0;**  **\*track\_flag=false;**  **}else{**  **\*track\_flag=true;**  **}**    **return target;**  **}** |

|  |
| --- |
| **방향 지시등 인식** |
| **bool findIndicator(Mat target, int direction){**  **double percent;**  **double count=0;**  **Mat img\_hsv;**  **bool resultFlag=false;**  **double threshold=0.1;**  **//target=target(Rect(target.rows/2,target.cols/4,target.rows/2, 2\*target.cols/4));**  **if(direction==0){ // tracking중인 차량이 좌측-> 우측 방향인 경우**  **target=target(Rect(target.rows/2, target.cols/4, target.rows/2, 2\*target.cols/4));**  **}else{ //tracking 중인 차량이 우측->좌측인 경우**  **target=target(Rect(0, target.cols/4, target.rows/2, 2\*target.cols/4));**  **}**  **double width=target.cols;**  **double height=target.rows;**  **int bpp=target.channels();**  **double imageSize=width\*height\*bpp;**  **cvtColor(target, img\_hsv, COLOR\_BGR2HSV);**  **Mat yellow\_mask;**  **Mat yellow\_image;**  **Scalar lower\_yellow=Scalar(20,20,100);**  **Scalar upper\_yellow=Scalar(32,255,255);**  **inRange(img\_hsv, lower\_yellow, upper\_yellow, yellow\_mask);**  **bitwise\_and(target, target, yellow\_image, yellow\_mask);**  **for(int i=0; i<height; i++){**  **for(int j=0; j<width\*3; j++){**  **if(yellow\_image.at<uchar>(i, j)<200){**  **yellow\_image.at<uchar>(i, j)=0;**  **}else{**  **count++;**  **}**  **}**  **}**  **percent=(count/imageSize)\*100;**  **imshow("yellow\_image", yellow\_image);**  **imshow("img\_hsv", img\_hsv);**  **if(percent>threshold){**  **resultFlag=true;**  **}**  **//cout<<"result is : "<<resultFlag<<" ";**  **//cout<<percent<<"%"<<endl;**  **return resultFlag;**  **}** |

|  |
| --- |
| **거리감지** |
| **extern "C"**  **JNIEXPORT jboolean JNICALL**  **Java\_com\_mobileprogramming\_twelve\_MainActivity\_detectCar(JNIEnv \*env, jobject thiz,**  **jlong cascade\_classifier\_car,**  **jlong mat\_addr\_input,**  **jdouble gradient\_left,**  **jdouble intercept\_left,**  **jdouble gradient\_right,**  **jdouble intercept\_right) {**  **Mat &img\_input = \*(Mat \*) mat\_addr\_input; //input frame**  **Mat detectFrame=img\_input(Rect(3\*img\_input.cols/10, 0, 4\*img\_input.cols/10, img\_input.rows)); // detect를 위한 관심 영역지정**  **int size=0;**  **bool flag=false;**  **int tmpIndex;**  **int resIndex=0;**  **Mat temp=img\_input;**  **Mat detectTemp=detectFrame;**  **double threshold=2.0;**  **cvtColor(detectFrame, detectFrame, COLOR\_BGR2GRAY);**  **std::vector<Rect> cars; //검출된 차량의 정보를 저장할 벡터 자료형**  **int standard=temp.cols/2; // 기준 해당 위치에 가장 가까운 객체를 트래킹 하도록 설정**  **((CascadeClassifier \*) cascade\_classifier\_car)->detectMultiScale(detectFrame, cars); // 차량 검출 수행 -> 결과 cars벡터에 저장**  **if(cars.empty()){**  **cout<<"fail to find cars"<<endl;**  **//checkPoint2**  **}else{**  **for(int i=1; i<cars.size(); i++){**  **tmpIndex=i;**  **if(abs(standard-cars[resIndex].x+cars[resIndex].width/2)>abs(standard-cars[tmpIndex].x+cars[tmpIndex].width/2)){**  **resIndex=tmpIndex;**  **}**  **}**  **cars[resIndex].x=cars[resIndex].x+3\*temp.cols/10; //찾은 차량의 좌표x 좌표 화면 detectFrame-> 전체 화면 Frame상의 위치**  **rectangle(img\_input ,cars[resIndex], Scalar(255,0,255),2);**  **double frameSize=img\_input.cols\*img\_input.rows; //전체 화면 넓이**  **double carSize=cars[resIndex].width\*cars[resIndex].height; //찾은 차량의 rect넓이**  **double distanceRate=carSize/frameSize\*100; // 찾은 차량의 Rect가 전체 화면에서 차지하는 비율**  **//detection 된 차량의 정중앙 x,y 좌표**  **int location\_x=cars[resIndex].x+cars[resIndex].width/2;**  **int location\_y=cars[resIndex].y+cars[resIndex].height/2;**  **if(distanceRate>threshold&&(location\_y-intercept\_left)/gradient\_left<location\_x&&location\_x<(location\_y-intercept\_right)/gradient\_right){ //차량의 비율이 threshold(2.0=30m)보다 클 경우 true return**  **flag=true;**  **cvtColor(img\_input, img\_input, COLOR\_BGR2HLS);**  **}**  **}**  **return flag;**  **}** |

|  |
| --- |
| **로그인/회원가입(안드로이드)** |
| **로그인/회원가입 안드로이드측 코드**  **serviceApi ( interface 선언 )**  **/\*API 인터페이스**  **\* 필요한 통신이 새로운 사용자 정보를 삽입할 수 있는 join과 사용자의 이메일 + 비밀번호를 대조해 볼 수 있는 login.**  **\* 인터페이스를 하나 만들어 서버에 어떤식으로 요청을 보내고 응답을 받을 것이지 미리 정의.**  **\* 내 node.js 에서 작성한 회원가입,로그인 API에 맞춰 작성.**  **\* \*/**  **public interface ServiceApi {**  **/\***  **\* 두가지 모두 POST 방식이 필요하므로 @POST 어노테이션을 사용**  **\* userLogin 함수는 request 통신 시 email과 password를 필드로 가짐 / reponse 타입은 Call<UserData>**  **\* userJoin 함수는 request 통신시 UserData 타입을 body로 가짐 / reponse 타입은 Call<ResponseBody>**  **\* \*/**  **@POST("/user/login")**  **Call<LoginResponse> userLogin(@Body LoginData data);**  **//Call<서버의 응답 데이터 정의 클래스>**  **@POST("/user/join")**  **Call<JoinResponse> userJoin(@Body JoinData data);**  **}**  **RetrofitClient**  **/\*서비스 객체 생성**  **\* ServiceAPI 인터페이스의 구현체를 만들기위해 Retrofit 클래스를 사용**  **\* 서비스 객체를 초기화 및 생성**  **\***  **\* \*/**  **//레트로핏 클라이언트 생성**  **public class RetrofitClient {**  **//https 통신시 handshake 오류 발생 >> http통신으로 변경**  **private final static String BASE\_URL = "서버DNS주소:3000";**  **private static Retrofit retrofit = null;**  **private RetrofitClient(){**  **}**  **public static Retrofit getClient(){**  **if(retrofit == null){**  **// Retrofit 객체 초기화**  **retrofit = new Retrofit.Builder()**  **// 요청을 보낼 bae\_url설정**  **.baseUrl(BASE\_URL)**  **// Gson컨버터 사용**  **.addConverterFactory(GsonConverterFactory.create())**  **.build();**  **}**  **return retrofit;**  **}**  **}**  **JoinActivity**  **//회원가입 성공시 처리**  **private void startJoin(JoinData data) {**  **//서버에서 보낸 응답 데이터로 회원가입 처리완료**  **service.userJoin(data).enqueue(new Callback<JoinResponse>() {**  **@Override**  **public void onResponse(Call<JoinResponse> call, Response<JoinResponse> response) {**  **JoinResponse result = response.body();**  **Toast.makeText(JoinActivity.this, result.getMessage(), Toast.LENGTH\_SHORT).show();**  **showProgress(false);**  **if (result.getCode() == 200) {**  **finish();**  **}**  **}**  **//회원 가입 에러시 처리**  **@Override**  **public void onFailure(Call<JoinResponse> call, Throwable t) {**  **Toast.makeText(JoinActivity.this, "회원가입 에러 발생", Toast.LENGTH\_SHORT).show();**  **Log.e("회원가입 에러 발생", t.getMessage());**  **showProgress(false);**  **}**  **});**  **}**  **로그인 코드**  **LoginActivity**  **//로그인 성공시 처리**  **private void startLogin(LoginData data) {**  **//서버에서 보낸 응답 데이터로 로그인 처리완료**  **service.userLogin(data).enqueue(new Callback<LoginResponse>() {**  **@Override**  **public void onResponse(Call<LoginResponse> call, Response<LoginResponse> response) {**  **LoginResponse result = response.body();**  **Toast.makeText(MainActivity.this, result.getMessage(), Toast.LENGTH\_SHORT).show();**  **showProgress(false);**  **}**  **//로그인 에러시 처리**  **@Override**  **public void onFailure(Call<LoginResponse> call, Throwable t) {**  **Toast.makeText(MainActivity.this, "로그인 에러 발생", Toast.LENGTH\_SHORT).show();**  **Log.e("로그인 에러 발생", t.getMessage());**  **showProgress(false);**  **}**  **});**  **}** |

|  |
| --- |
| **파일전송(안드로이드)** |
| **apiService (intreface 생성)**  **// video파일 전송을 위해 body에 part 형식으로 데이터를 담아 서버측으로 전송**  **// 바이너리 파일 전송을 위해 Multipart 어노테이션 사용**  **public interface ApiService {**  **@Multipart**  **@POST("/upload")**  **Call<ResponseBody> postVideo(@Part MultipartBody.Part video, @Part("upload") RequestBody name );**    **}**  **//-----------------------------------------------------------------------**  **<?xml version="1.0" encoding="utf-8"?>**  **<network-security-config xmlns:android="http://schemas.android.com/apk/res/android">**  **<domain-config cleartextTrafficPermitted="true">**  **<domain includeSubdomains="true">ec2-13-124-56-124.ap-northeast-2.compute.amazonaws.com</domain>**  **</domain-config>**  **</network-security-config>**  **<!-- 안드로이드 http 프로토콜 접속 시 예외발생 조치-->**  **//-----------------------------------------------------------------------**  **//파일전송 부**  **private void multipartVideoUploads() {**  **//전송할 파일의 경로 저장 = mediaPath1**  **File file1 = new File(mediaPath1);**  **// 파일전송을 위한 파싱**  **RequestBody reqFile = RequestBody.create(MediaType.parse("\*/\*"), file1); // 파일 형식**  **// 파일을 part 형식으로 body에 담음**  **MultipartBody.Part body = MultipartBody.Part.createFormData("upload", file1.getName(), reqFile);**  **RequestBody name = RequestBody.create(MediaType.parse("text/plain"), "upload");**  **//전송부분 apiService 인터페이스에 선언해놓은 post 방식으로 이미지를 보냄**  **Call<ResponseBody> req = apiService.postVideo(body, name);**  **//이후 처리 부분**  **req.enqueue(new Callback<ResponseBody>() {**  **@Override**  **public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {**  **if (response.code() == 200) {**  **Toast.makeText(getApplicationContext(),"Uploaded Successfully!",Toast.LENGTH\_SHORT).show();**  **}**  **Toast.makeText(getApplicationContext(), response.code() + " ", Toast.LENGTH\_SHORT).show();**  **}**  **// 전송 실패시 처리**  **@Override**  **public void onFailure(Call<ResponseBody> call, Throwable t) {**  **Toast.makeText(getApplicationContext(), "Request failed", Toast.LENGTH\_SHORT).show();**  **t.printStackTrace();**  **}**  **});**  **}** |

|  |
| --- |
| **파일 수신** |
| **/파일 다운로드를 위해 Streaming 어노테이션 사용**  **public interface RetrofitInterface {**  **@Streaming**  **@GET**  **Call<ResponseBody> downloadFileByUrl(@Url String fileUrl);**    **}**  **//---------------------------------------------------------**  **//retrofit download**  **private void downloadVideoFile() {**  **//클라이언트와 통신할 서버측 주소 입력**  **RetrofitInterface downloadService = createService(RetrofitInterface.class, "서버DNS주소:3001");**  **Call<ResponseBody> call = downloadService.downloadFileByUrl("/uploads/Video01.avi"); // 다운받을파일의 서버측 경로**  **//파일 송신부 RetrofitInterface에 선언해 놓은Streaming 방식으로 파일 송신**  **call.enqueue(new Callback<ResponseBody>() {**  **@Override**  **public void onResponse(Call<ResponseBody> call, final Response<ResponseBody> response) {**  **if (response.isSuccessful()) {**  **Log.d(TAG, "Got the body for the file");**  **Toast.makeText(getApplicationContext(), "Downloading...", Toast.LENGTH\_SHORT).show();**  **downloadVideoFileTask = new DownloadVideoFileTask();**  **downloadVideoFileTask.execute(response.body());**  **} else {**  **Log.d(TAG, "Connection failed " + response.errorBody());**  **}**  **}**  **@Override**  **public void onFailure(Call<ResponseBody> call, Throwable t) {**  **t.printStackTrace();**  **Log.e(TAG, t.getMessage());**  **}**  **});**  **}**  **//---------------------------------------------------------**  **//다운 받을 파일에 대한 설정 메소드**  **private class DownloadVideoFileTask extends AsyncTask<ResponseBody, Pair<Integer, Long>, String> {**  **@Override**  **protected void onPreExecute() {**  **super.onPreExecute();**  **}**  **@Override**  **protected String doInBackground(ResponseBody... urls) {**  **saveToDisk(urls[0], "Video01.avi"); // 다운받을 파일명 설정**  **return null;**  **}**  **protected void onProgressUpdate(Pair<Integer, Long>... progress) {**  **Log.d("API123", progress[0].second + " ");**  **if (progress[0].first == 100)**  **Toast.makeText(getApplicationContext(), "File downloaded successfully", Toast.LENGTH\_SHORT).show();**  **if (progress[0].second > 0) {**  **int currentProgress = (int) ((double) progress[0].first / (double) progress[0].second \* 100);**  **progressBar.setProgress(currentProgress);**  **txtProgressPercent.setText("Progress " + currentProgress + "%");**  **}**  **if (progress[0].first == -1) {**  **Toast.makeText(getApplicationContext(), "Download failed", Toast.LENGTH\_SHORT).show();**  **}**  **}**  **public void doProgress(Pair<Integer, Long> progressDetails) {**  **publishProgress(progressDetails);**  **}**  **@Override**  **protected void onPostExecute(String result) {**  **}**  **}** |

|  |
| --- |
| **로그인/회원가입(서버)** |
| **var mysql = require('mysql'); // mysql데이터 베이스**  **var express = require('express'); //node.js 서버구축을 위한 웹 프레임워크**  **var bodyParser = require('body-parser'); //클라이언트 POST request data의 body로부터 파라미터를 추출하기 위한 모듈**  **var app = express();**  **app.use(bodyParser.json());**  **app.use(bodyParser.urlencoded({extended: true}));**  **//node.js 통신 포트 설정**  **app.listen(3000,function () {**  **console.log('server running...');**  **});**  **//mysql 데이터 베이스 정보 저장**  **var connection = mysql.createConnection({**  **host: "DB dns",**  **user: "user",**  **database: "loginSample",**  **password: "pwd",**  **port: 3306**  **});**  **//회원가입을 위해서 사용자 측에서 POST 방식으로 보낸 이메일, 닉네임, 비밀번호 정보를 얻어 users 테이블에 삽입**  **app.post('/user/join', function (req, res) {**  **console.log(req.body);**  **var userEmail = req.body.userEmail;**  **var userPwd = req.body.userPwd;**  **var userName = req.body.userName;**    **//삽입을 수행하는 sql문**  **var sql = 'INSERT INTO Users (UserEmail, UserPwd, UserName) VALUES (?, ?, ?)';**  **var params = [userEmail, userPwd, userName];**  **//만들어둔 데이터베이스 connection으로 쿼리문 사용**  **// sql 문의 ?는 두번째 매개변수로 넘겨진 params의 값으로 치환**  **connection.query(sql, params, function (err, result) {**  **var resultCode = 404;**  **var message = 'error has occured';**  **if (err) {**  **console.log(err);**  **} else {**  **resultCode = 200;**  **message = 'succeed register.';**  **}**  **res.json({**  **'code': resultCode,**  **'message': message**  **});**  **});**  **});**  **//로그인**  **//로그인을 위해서 이메일을 이용해 해당 아이디를 포함한 row가 있는지 확인하고, 있다면 비밀번호가 일치하는지 조회를 통해 검사**  **app.post('/user/login', function (req, res) {**  **var userEmail = req.body.userEmail;**  **var userPwd = req.body.userPwd;**  **var sql = 'select \* from Users where UserEmail = ?';**  **//result 값이 없을 경우엔 계정이 없음을, 패스워드 불일치 시엔 불일치 메시지를, 일치할 경우 환영하는 메세지를 json으로 돌려줌**  **connection.query(sql, userEmail, function (err, result) {**  **var resultCode = 404;**  **var message = 'error has occured';**  **if (err) {**  **console.log(err);**  **} else {**  **if (result.length === 0) {**  **resultCode = 204;**  **message = 'Non-exist account!';**  **} else if (userPwd !== result[0].UserPwd) {**  **resultCode = 204;**  **message = 'Wrong pwd';**  **} else {**  **resultCode = 200;**  **message = 'succesful' + result[0].UserName + 'welcome!';**  **}**  **}**  **res.json({**  **'code': resultCode,**  **'message': message**  **});**  **})**  **});** |

|  |
| --- |
| **파일 수신(서버)** |
| **/\*express 를 이용해 사용자가 받은 데이터를 저장해주는 모듈\*/**  **var multer, storage;**  **multer = require('multer')**  **var fs = require('fs'); //node 파일 모듈**  **storage = multer.diskStorage({**  **//저장경로 설정**  **destination: './uploads/',**  **//저장 파일 이름 설정**  **filename: function(req, file, cb) {**  **cb(null,file.originalname);**  **}**  **});**  **// Post file**  **//안드로이드 -> 서버 파일수신**  **app.post("/upload",multer({storage: storage}).single('upload'), function(req, res) {**  **console.log(req.file);**  **console.log(req.body);**  **console.log(req.file.filename);**  **//서버 응답 코드 전송**  **return res.status(200).end();**  **});** |

|  |
| --- |
| **파일 송신(서버)** |
| **/\*express 를 이용해 사용자가 받은 데이터를 저장해주는 모듈\*/**  **var multer, storage;**  **multer = require('multer')**  **var fs = require('fs'); //node 파일 모듈**  **storage = multer.diskStorage({**  **//저장경로 설정**  **destination: './uploads/',**  **//저장 파일 이름 설정**  **filename: function(req, file, cb) {**  **cb(null,file.originalname);**  **}**  **});**  **// 서버 -> 안드로이드 파일 송신**  **app.get('/uploads/:upload', function (req, res){**  **file = req.params.upload; //요청받은 파일을 클라이언트측을 업로드**  **console.log(req.params.upload);**  **var video = fs.readFileSync(\_\_dirname + "/uploads/" + file);**  **res.writeHead(200, {'Content-Type': 'video/\*' });**  **res.end(video, 'binary');**  **});** |

|  |
| --- |
| **배치 프로그램(main.sh)** |
| **#!/bin/bash**  **#**  **# mainScript - nodeScript / highlightScript 실행**  **#**  **#**  **##################################################**  **#**  **# 1. node check 스크립트 실행**  **#**  **##################################################**  **check=`ps | grep PGM\_Extract | wc | awk '{print$1}'` # PGM\_Extract 프로그램 실행시 1 출력**  **if (($check!=1))**  **##PGM\_Extract**  **then**  **echo " 3. PGM\_Extract is not running.."**  **# 하이라이트 추출 프로그램 실행**  **else**  **echo " 3. PGM\_Extract is already running.."**  **# 하이라이트 추출 프로그램 실행중**  **fi**  **while ((1))**  **do**  **echo -e ""**  **echo -e " +----------------+"**  **echo -e " | script Start |"**  **echo -e " +----------------+"**  **echo -e ""**  **echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**  **./nodeScript**  **# node check script 실행**  **echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**  **#################################################**  **#**  **# 1. highlight 스크립트 실행**  **#**  **#################################################**  **./highlightScript**  **# highlight extract Script 실행**  **#echo -e "\n"**  **echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**  **#echo sleep**  **sleep 10**  **#echo sleep**  **done**  **#echo done** |

|  |
| --- |
| **배치 프로그램(node.sh)** |
| **#!/bin/bash**  **#**  **# nodeScript - nodecheck / noderun**  **#**  **#########################################**  **#**  **# 1. node run check**  **#**  **#echo -e "\n"**  **#echo -e "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**  **echo -e " ------ nodeScript Start ------\n"**  **PGM\_NAME=node**  **#node run check**  **echo " 1. $PGM\_NAME run check"**  **#echo " result [1] == already running "**  **#echo -e " result [other] == not running\n"**  **check=`ps -ef | grep $PGM\_NAME | wc | awk '{print$1}'`**  **# check=`ps | grep $PGM\_NAME | wc | awk '{print$1}'`**  **echo -e " 2. result [" $check "]\n"**  **#Start node**  **if (($check!=1))**  **then**  **#echo " 3. $PGM\_NAME is not running.. let's run it now!"**  **nohup node server.js &**  **else**  **echo " 3. $PGM\_NAME is already running.."**  **fi**  **echo -e ""**  **echo -e " ------ nodeScript End ------\n"**  **#source code has already been compiled** |

|  |
| --- |
| **배치 프로그램(DBinsert.sh)** |
| **#!/bin/bash**  **#**  **# mysql insert Script**  **#**  **###################################################**  **#echo mysql -e select \* from users;**  **password="\*\*\*\*"**  **echo "insert into androidDB.videoTBL values('test1', 'test1');" | mysql -u"root" -p$password** |

|  |
| --- |
| **배치 프로그램(videoExtract.sh)** |
| **#!/bin/bash**  **#**  **# highlightScript - file comparison / file move / highlight extract / file delete**  **#**  **#############################################**  **#반복실행#**  **#영상 파일 하나씩 추출 과정 반복#**  **echo -e " ------ highlightScript Srart ------\n"**  **echo -e " 1. 각 dir 이름 추출\n"**  **videodirname=$(find ./ -name video\_andToser)**  **echo -e " video dirname 추출 =" $videodirname**  **txtdirname=$(find ./ -name txt\_andToser)**  **echo -e " txt dir name 추출=" $txtdirname "\n"**  **#######################################################################################**  **echo -e " 2. 각dir 첫 파일 추출\n"**  **videofullname=$(ls $videodirname -t | tail -1)**  **#echo $videofullname**  **videofirstname=$(basename "./video\_andToser/"$videofullname .avi)**  **echo -e " video dir first file name = " $videofirstname**  **txtfullname=$(ls $txtdirname -t | tail -1)**  **#echo $txtfullname**  **txtfirstname=$(basename "./txt\_andToser/"$txtfullname .txt)**  **echo -e " txt dir first file name =" $txtfirstname "\n"**  **#######################################################################################**  **echo -e " 3. txtdir videodir 파일명 체크\n"**  **if [ "$videofirstname" == "$txtfirstname" ]; then**  **echo -e " videofirstname == txtfirstname 이 같음 \n"**  **# mv 로 둘다 현재 작업 폴더로 이동**  **mv $videodirname/$videofullname ./**  **mv $txtdirname/$txtfullname ./**  **echo -e " 4. 하이라이트 추출\n"**  **./PGM\_Extract $videofirstname**  **#!!!!!!!! ./VideoWrite0905 $인자로 videofirstname [수정]**  **echo -e "\n"**  **ExtractedfirstVideo=$(ls ./ExtractedVideo -tr | tail -1)**  **echo -e " @@ DB insert\n"**  **password="123"**  **echo " insert into androidDB.videoTBL values('$ExtractedfirstVideo', '$videodirname');" | mysql -u"root" -p$password**    **echo -e " 5 . original file delete\n"**  **mv $videofullname ./AfterExtract**  **mv $txtfullname ./AfterExtract**  **elif [ "$videofirstname" != "$txtfirstname" ]; then**  **echo -e " videofirstname == txtfirstname 이 같지 않음\n"**  **# break**  **fi**  **echo -e " ------ highlightScript End ------\n"** |

ⅠⅠⅠ. 결론

|  |
| --- |
| **1. 연구 결과** |

|  |
| --- |
| **안드로이드 환경 동작** |

텍스트, 실외, 밤하늘이(가) 표시된 사진

자동 생성된 설명텍스트, 모니터, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| **서버 환경 동작** |

텍스트, 도로, 길이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| **설명** |

**위 사진들은 실제 구동 영상의 일부이다. 실제 구동 결과 개발 초기에 계획했던 대부분의 기능이 기 대 이상의 성능을 보여주었다. 직선 검출에 의한 차선 인식은 영상 프레임의 수준이 일정하게 유지 된 다는 가정하에 실제 양산 중인 차량에 탑재된 차선 이탈 보조 기능과 유사한 성능을 보여주었다. 충돌 센서의 경우 실제 판매중인 약 60만원 가량의 블랙박스와 유사한 성능을 보여주었다. 진로 변경 위반 모듈의 경우 비교 대상이 없어 객관적 성능 판단은 어려우나 Tracker의 성능 이 좋은 수준을 유지하면 고속도로 환경에서는 정상적으로 동작하는 것을 확인했다. 클라이언트와 서버간 통신에서 개발 초기에 겪어야 했던 비 동기식 통신과 서버내 파일 동기화에 대한 문제는 서버내 배치 프로그램을 동작시켜 해결 하였다. 다수의 트래픽을 수용할 만한 서버 가 준비되지 않아 동시 접속자에 대한 테스트를 수행 하지는 못 했으나 서버 증축이 이루어진다면 다수의 사용자에게 서비스를 제공할 수 있을 것으로 기대해 본다.**

|  |
| --- |
| **2. 최종 결론** |

**우리가 개발한 애플리케이션은 모바일 Native 환경에서 영상 처리 기능을 제공한다. 현재 블랙박스 애플리케이션은 다수 존재하나, 부가적인 서비스를 제공하는 앱은 많지 않으며, 영상처리와 같은 무거 운 작업을 수행하는 블랙박스는 더더욱 찾아보기 어렵다. 앞서 말한 바와 같이 영상처리 기능을 Native 환경에서 구현하는 것이 모바일 기기의 성능에 영향을 주는 것을 많이 경험해 볼 수 있었다. 그러나 모바일 디바이스의 성능은 큰 폭으로 발전하고 있으며, 현재 수준의 기기들로도 차선 검출과 같은 기능은 충분히 수행할 수 있는 정도이다. 이러한 기능은 최신 자동차를 소유하고 있지 않은 운전자들에게 주행 보조 시스템이라는 경험을 제공해 줄 수 있을 것으로 기대한다. 아쉽 게도 다수의 트래픽을 수용할 만한 서버가 준비 되어있지 않지만, 영상 처리에서의 성능 부하 문제를 잘 해결 한다면 API 서버 없이 앱 마켓 에도 출시 해 볼만 하다고 기대 해 볼 수 있다.**

|  |
| --- |
| **3. 참고 문헌** |

**[1] retrofit - https://square.github.io/retrofit/**

**[2] express - https://expressjs.com/ko/4x/api.html**

**[3] AWS\_EC2 - https://docs.aws.amazon.com/ec2/index.html?nc2=h\_ql\_doc\_ec2**

**[4] Amazon\_RDS - https://docs.aws.amazon.com/rds/index.html?nc2=h\_ql\_doc\_rds**

**[5] mysql - https://dev.mysql.com**

**[6] 황선규 지음, Opencv 4로 배우는 컴퓨터 비전과 머신러닝, 2019년**

**[7] Straight Line Detection Using PCA and Hough Transform, 오정수,한국정보통신학회논문지\_Vol.22.No.2:227-232, Feb 2018**

**[8] Flower Recognition System Using OpenCV on Android Platform, 조우 지음 , 전남대학교 도서관(여수 캠퍼스) , 2017**

**[9] Android developers documantation**

**[10] pyimagesearch - https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/**