지은 추가 내용

6.24

근무 환경 정리 및 업무 파악

6.25

스마트 주차장 Edge관련 코드 분석

VisionSensor/core/parkingspot.json에서 timelimit 0->10으로 수정하여 서버에서 실시간으로 주차 status 알맞게 변하도록 설정

6.26

션이사님 노트북 우분투 설치 및 환경설정, 스마트 주차장 Edge관련 코드 분석

MySQL DB에서 TimeLimit에 따라 주차상태 값이 Overtime으로 변할 수 있게 수정

6.27

션이사님 노트북 우분투 드라이버 문제 발생 -> 환경 재설정

Edge에서 기존의 2가지(파랑색, 빨강색)의 색으로 주차장 상태를 표현에서 3가지 상태로 수정

초록 : Available

빨강 : Occupied

파랑 : Overtime

6.28

MySQL Workbench 설치 및 간단한 내용 숙지 및 Edge와 admin 서버와 연동관련 학습

7.1

edge와 admin서버와 데이터 송수신 방법론 제시

Edge -> Admin으로 timestamp만 http 전송

DB와 리눅스 C기반 Edge연동을 통해

MySQL DB -> Edge를 통해 Policy, SPOT\_ID, TimeLimit 전송

7.2

(ParkingSpotManager.cpp)

MySQL DB와 C 기반 Edge와 연동을 통해 DB의 저장되어 있던 Policy, SPOT\_ID, TimeLimit 데이터 로드

기존에 있던 Edge 코드와 C 기반 Mysql 구현한 코드와 같이 make 할 때 Error 발생

-> Linux make, gcc 등 compile 하는 방법에 대해서 학습을 통해 해결

7.3

git에 대해 간단한 학습

MySQL DB Spot\_policy 테이블에 reservation 속성 추가를 통해 spot의 예약 유무를 판단.

카메라의 오작동으로 인해 DB와 서버에 생길 수 있는 문제에 대해 의논

(ParkingSpotManager.cpp)

C 기반 Edge에서 MySQL DB 테이블을 직접 수정할 수 있도록 자료조사

7.4

MySQL parking\_spot 속성값 변경

: reservation\_yn( 기존 [Y,N] -> 변경 후 [X,Y,N])` (현재 삭제함)

X : 일반주차

Y,N : 예약주차 유무

(ParkingSpotManager.cpp)

C 기반 Edge에서 MySQL DB 연동관련 함수들 모듈화

C 기반 Edge에서 MySQL DB 테이블을 직접 수정할 수 있도록 구현

connectDB() : DB 연결

updateDB() : DB 속성값 수정(Edge에서 주차정보들 DB로 전송)

pullDB() : DB에서 Policy, TimeLimit 값들을 Edge로 전송

7.5

Edge 부분(ParkingSpotManager.cpp)에서 MySQL 연동관련 코드 추가 및 모듈화

isExist(int id)

: 파라미터 id 로 MySQL 쿼리를 하고 해당 id로 spot이 없는 경우 0 을 반환하여 createDB() 함수를 호출할수 있도록 함/ spot이 있는 경우 1을 반환하여 updateDB() 함수를 호출할 수 있도록 구현

createDB(int id)

: table에 해당 id에 관한 spot이 없는 경우 table에 insert하도록 구현

MySQL 수정사항 (spot insert 때문에 default null 필요)

ALTER TABLE `seevider`.`parking\_spot`

CHANGE COLUMN `gps\_latitude` `gps\_latitude` DOUBLE NULL DEFAULT NULL ,

CHANGE COLUMN `gps\_longitude` `gps\_longitude` DOUBLE NULL DEFAULT NULL ,

CHANGE COLUMN `last\_hst\_seq` `last\_hst\_seq` BIGINT(20) NULL DEFAULT NULL ;

7.8

기존에 구현되어 있던 Edge 코드내에 함수를 활용하기 위해 코드분석

: enter(), exit(), timelimit, spot 등

7.9

MySQL DB의 end\_user\_reservation에서 들어온 차량번호 정보를 가지고 sequence를 쿼리를 하여 Edge에서 해당 예약의 start\_time, date, end\_time, date 데이터 값을 얻어오도록 구현

7.10

edge에서 enter, exit 시간이 현재 20190631T041205 이런 string 형태로 받아오고 DB에서는 201906040312 string값으로 받아옴

: 형태를 맞춰주기 위해 각각 year, month, day, hour, minutes로 나눠 int형으로 변환해주고 exit값과 예약 끝나는 시간을 비교해서 exit한 시간 > reservation 끝나는 시간이면 overtime이 되도록 구현함

7.11

MySQL 수정

end\_user\_reservation에 속성 추가

: used\_yn - 해당 시퀀스 예약이 호출되었는지 Y/N으로 표현하여 중복 호출을 피하기 위함

overtime관련 코딩 구상

: 예약 끝나는 시간 - enter 시간

: ex) 16 - 15 = 1분을 timelimit에 넣어줌

-> parking update에서 timelimit을 수정하는 것으로 파악

end\_use\_resrervation

today’s issue

1. 예약시간이 13:00 ~ 15:00 이지만 사용자가 이전시간 10:00에 ent 하는 경우 문제 발생 가능
2. 현재 시간이 3분가량 늦음 -> 이유는 아직 모르겠음

월요일에 해야할껄

예약 끝나는 시간 - ent 시간

16-15 = 1분을 timelimit에 넣어줌

parking update 에서 timelimit을 수정하는것으로 파악

7.15

차량이 들어오면 시간을 db에서 해당차량의 번호판을 가지고 쿼리를함

현재 번호판 인식이 안되고 있으므로 디폴트 값으로 “QISENS6”으로 설정하고 진행중

DB에서 해당하는 예약을 쿼리를 했으면 해당 튜플의 used\_yn 을 N에서 Y로 업데이트

시간 계산은 예약 끝나는 시간과 차량이 진입한 시간의 day, hour, minutes 각각 초로 바꾸어 계산한 뒤 해당값을 timelimit으로 설정해줌, 만약 들어간 시간이 예약 끝나는시간보다 애초에 over되어있으면 디폴트값으로 10초로 설정했음

* 이부분을 [plate\_number 쿼리 결과가 없으면]으로 계산하면 좋을 듯

!!cmake 관련!!

소스코드를 다른 컴퓨터에서 make하면 생기는 오류!

-> cmakelists.txt 파일내에서 경로를 재설정해줘야함

그리고 cmakefile과 cacahe를 삭제해주고 터미널창 열어서

$ cmake CMakeLists.txt

해당 명령을 치고 make를 다시해줘야함!!

MySQL관련

: 해당 SQL라이브러리를 쓰기위해 CMakeLists.txt.에서 -lmysqlclient 를 추가했었음!

!!!

7.16

edge\_front에서 코딩했던 프로젝트를 back(reservation edge)로 돌려서 적용하려했지만 make 문제발생

: opencv 및 관련 라이브러리를 다시 설치함, darknet 폴더에서도 다시 make함으로

문제해결

7.17

1. TK1 - qisens side s1
   1. 일반 고객들을 위한 디바이스
2. Res - qisens back s1
   1. 예약 고객들을 위한 디바이스
3. Nano\_Res – qisens back s2
   1. 예약 고객들을 위한 소형 디바이스

TK1 코드 복구 및 Res 디바이스 코드 백업

Nano\_Res os 재설치 및 환경설정

7.18, 7.19

Res 디바이스 추가한 코드 부분 정리

core/parkingspots.json

: 해당 파일은 json 형식으로 되어있으며 spot의 정보들을 담고 있음

spot의 정보들을 추가할때마다 연동되어 있는 mysql DB에 동기화

**ParkingSpot.h**

1. void pullDB()
2. void connect\_DB()
3. int entVehicle(const std::string PN, const std::string entryTime)
4. void extVehicle(const std::string PN, const std::string entryTime)
5. void startTimer(int res\_timeLimit) -> 파라미터를 추가함

**ParkingSpot.cpp**

전역변수들 추가

: DB로부터 예약 시작 시간과 끝나는 시간을 받아와 각각 year, month, hour, minutes로 세분화 시킴

**long long e\_y, e\_mo, e\_d, e\_h, e\_m**

: 차량 들어온 시간을 day, hour, minutes로 세분화 시킴

**void connect\_DB()**

: MySQL과 연동하기 위해 connect 함수

**int entVehicle(const std::string PN, const std::string entryTime)**

: 차량이 들어옴을 감지할때 호출되는데 차량의 PN(번호판) 과 entryTime(들어온시간)을 인자로 받음

: connect\_DB를 통해 MySQL과 연동하고 차량의 번호판으로 예약되어있고 예약이 사용되지 않은 건의 한에 쿼리를함

-> 문제점 : 한 사용자가 여러개의 예약건수를 신청함.

-> 해결방안 : 쿼리문에 ORDER BY 날짜순으로 한번 필터링, + 지금 시간대와 예약시간안에 포함되어있는 예약건을 불러드리는 쿼리를 함!! (구현 미완성)

: 예약 건을 처리하면 바로 used\_yn = Y로 업데이트 하여 중복 쿼리를 방지

* where platenumber + orderby startdate, starttime
* 지금 시간 이후로 불러오게 하면 used\_yn 필요 없나??

day, hour, minutes를 각각 초를 바꾸어 예약 끝나는 시간과 차

량 들어온 시간을 빼줌을 통해 timer를 설정하고 timer를 통해 occupied, overtime 상태를 구별하도록 구현함

e\_d = (e\_date % 10000) % 100 \* 86400; // 이거 바꿨는데 그건 코드상에서 확인하기

e\_h = e\_time / 100 \* 3600;

e\_m = e\_time % 100 \* 60;

현재는 이런식으로 초를 바꾸어 계산하는데 minutes

db\_sum = e\_d + e\_h + e\_m 여기에 year, month관련 초를 바꾸어 계산해주면 예약, 차량들어온 시간 관련 문제는 해결할수있음!!

result\_timer = db\_sum - ent\_sum; 를 통해 db에서 불러들인 예약 끝나는 시간과 차량 들어온 시간을 계산한뒤 빼주어 현재부터 예약시간 까지 남은 시간을 timer를 계산하여 timer가 끝나면 overtime으로 간주

**enter()**

: 주차공간에 차량이 들어오게되면 PN을 받고 startTimer()에 계산된 res\_timer를 인자로 넣어둠으로써 해당 spot에 timer를 걸어줌

**startTimer(int res\_timeLimit)**

: 위에서 언급한 seconds를 minutes로 바꾸어 시간계산 범위를 작게 할수 있는 부분임

mParkingTimer.expires\_from\_now(boost::posix\_time::seconds(res\_timeLimit))

여기서 seconds를 minutes로 바꿀수 있는지 확인 안해봄

**MainInterface.cpp**

(Log 카메라 오류발생) 카메라가 잡히지 않는다 싶으면

mVideoReader->open(1); 해당 1 값을 0~4까지 바꿔보는 해결방안이 있음

: usb 꽂는 포트 달라서

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*가장 중요하다고 생각\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void updateSpots(const Mat &frame, const pt::ptime& now)에서

parkingSpot->enter(frame.clone(), parkingSpot->ROI, now,"QISENS6");

parkingSpot->exit(frame.clone(), now, "QISENS6");

이 부분이 있는데 여기서 “QISENS6” 임의의 들어온 차량과 나간 차량의 번호판을 의미함

따라서 번호판 인식 기능이 구현이 된다면 해당 값을 들어온 차량의 번호판의 변수를 넣을수 있도록 추가 구현을 해야함!!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*가장 중요하다고 생각\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

현재 에러처리를 안해준 부분

해당 차량의 번호판으로된 예약이 DB에 되어있지 않으면 프로그램 종료되지 않게 예외처리를 해주어야함

또한 차량이 들어와서 ENTER했음에도 DB에 예약이 없다면 프로그램이 종료되어 이를 방지하도록 예외처리(except, catch)를 잡아서 처리를 해줘야함

**ParkingSpotManger.h**

void connectDB()

void updateDB(int id, std::string spotName, int x, int y, int h, int w)

bool isExist(int id)

void createDB(int id, std::string spotName, int x, int y, int h, int w)

void pullDB()

**ParkingSpotManger.cpp** // Edge에서 등록한 spot들을 DB에 등록할수있는 부분

// /core/parkingspot.json에서 spot에 대한 정보들을 추가하면 알아서 해줌

**void connectDB()**

: MySQL DB와 연동 하기 위함

: 함수내에서 exit(1) 코드 삭제 추천

**bool isExist(int id)** **(spot id 관련)**

: spot id값을 받아와 db와 연결후 해당 id가 DB에 있는지 쿼리한 후 있는지 없는지 True, False 값으로 반환해줌

없으면

createDB가 호출되고 DB에 해당 parking\_spot\_id, parking\_spot\_name, roi\_coordx, 등등 정보들을 토대로 DB 테이블에 INSERT함

있으면

updateDB가 호출되어 /core/parkingspot.json/ 내에 있는 정보들로 이용해 DB 업데이트함

**CMakeLists.txt**

.cpp 파일이나 .h파일을 새롭게 구현해서 추가한다면 해당 파일들을 여기에 추가해야한다.

또한 MySQL을 컴파일 하기위해서는 옵션 -lmysqlclient를 추가해야하는데 이것 또한 set(BOOST\_LIB\_LIST ~~~~ -lmysqlclient) 이런식으로 추가해주어

$ cmake CMakeLists.txt

$ make

해주어 컴파일을 성공적으로 가능하다.

**parkingspot**

**새로운 센서 추가할때 변경해야하는 부분**

/core/settings.ini : SensorID=이름을 수정해줘야함

/core/parkingspots.json : spot에 대한 정보들을 모두 새롭게 추가해줘야함

7.22

분리되어 있던 일반 주차석과 예약석 코드를 합치는 과정 (1)

MainInterface.cpp

현재는 :: 예약area든 일반area든 end\_user\_reservation table에서 쿼리 하고, 쿼리 결과가 없으면 일반area라고 가정, 쿼리 결과가 있으면 예약area라고 가정

run(), updateSpot()내에 있는 “QISENS6” 는 임의로 정의한 번호판 (Plate Number)임

그래서 번호판인식이 성공하다면 여기에 PN값을 넣어주어 인자를 전달해줘함

7.23

분리되어 있던 일반 주차석과 예약석 코드를 합치는 과정 (2)

ParkingSpot.cpp

bool check\_res(std::string PN)

: Plate Number를 받아서 예약되어있는 차량인지 확인하고 맞으면 예약석 관련 코드를 수행하고 아니면 일반석 코드를 수행할 수 있도록 boolean 값을 반환

void enter()에서 check\_res()가 true이면 예약시간에 관련한 starttimer(res\_timer, PN)가 호출되고 false이면 일반 예약시간 즉 policy의 timelimit으로 시간을 책정하는 starttimer()가 호출되어진다

Jetson Nano 환경 세팅

**DB 관련 필드 추가한 내용들**

**end\_user\_reservation**

**used\_yn**

**만든 이유: DB에서 해당 하는 예약 시퀀스(튜플)이 사용되어 없애기 위함, 중복 호출을 피하기 위함, 해당 예약이 불렸으면 Y, 아직 예약 N**

**설치해야하는 라이브러리**

sudo apt-get install libmysqlclient-dev

[**https://bitsoul.tistory.com/154**](https://bitsoul.tistory.com/154)

**parking spot 그리는 파일 : MainInterface.cpp**

**ip 주소 변환**

**core/serverdata.xml**

**main.cpp**

**최종폴더 : VisionSensor(merge)ㄴ**

**reservation 과 normal 코드 합침**

**end\_user\_reservation table에서 used\_yn을 통해 예약 중복관련을 피하기 위해 쿼리를 했는데 이부분이 보완이 필요함**

* **기존에는 order by start\_date, start\_time으로 정렬하여 쿼리를 했지만 이 방법보다는 차량이 들어온 시간(entrytime)과 end\_user\_reservation table의 start\_date, start\_time, end\_date, end\_time과 비교하여 적당한 로직으로 쿼리를 함**