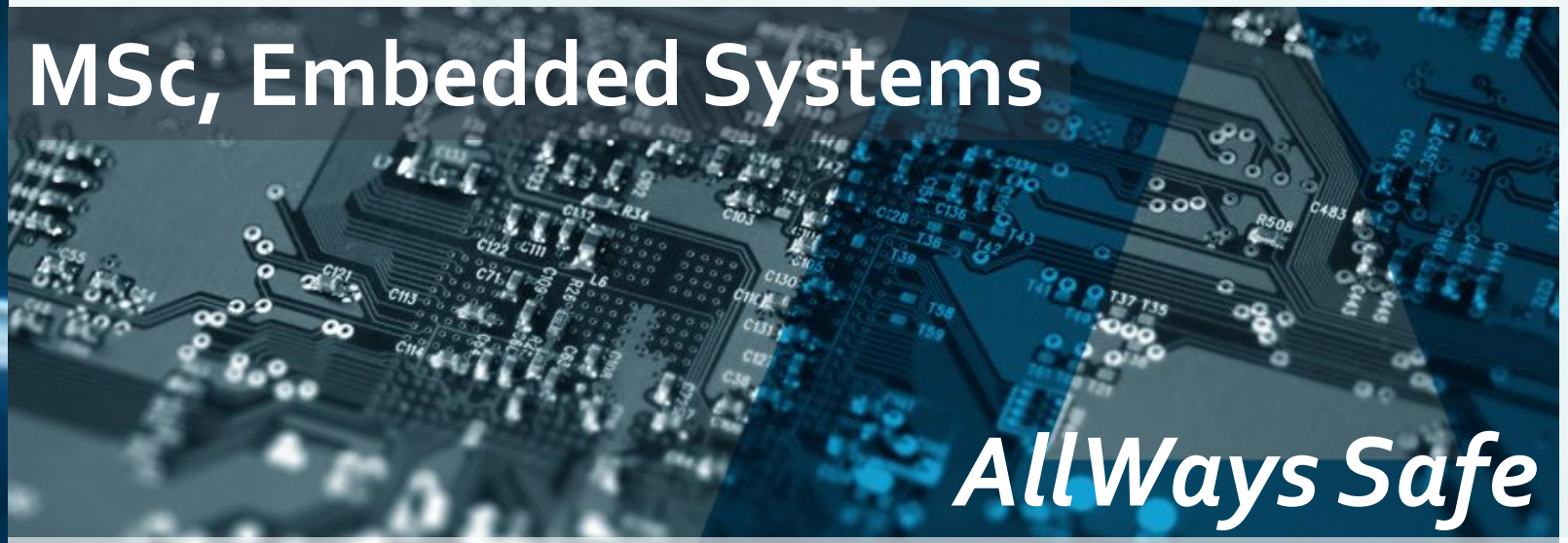
Final Presentation

MSc, Embedded Systems



AllWays Safe

André Martins, PG60192
Mariana Martins, PG60211

AllWays Safe

An Intelligent Traffic Management System

Enhancing Urban Mobility through V2I Communication and Adaptive Control

PROJECT BY GROUP 4 (ANDRÉ MARTINS & MARIANA MARTINS) | MASTER'S IN INDUSTRIAL ELECTRONICS AND COMPUTERS |
SUPERVISED BY ADRIANO TAVARES, PhD

Agenda

- Introduction
- The AllWays Safe Solution
- Hardware Architecture
- Technology Stack
- Package Diagram
- Database
- Monitoring System
- Linux Device Driver
- C++ Wrapper
- Emergency Vehicle Control Box
- Intersection Control Box

Introduction

Innovative and Emerging Technologies

- V2I (Vehicle-To-Infrastructure)
  - Enables vehicles to communicate with road infrastructure such as traffic lights, toll booths, and road sensors
  - Uses wireless technologies
  - Helps vehicles receive real-time traffic and road condition information
  - Improves traffic efficiency, safety, and fuel consumption
  - Plays a key role in Intelligent Transportation Systems (ITS)

Source: www.qualcomm.com

- V2X (Vehicle-To-Everything)
  - I2P (Infrastructure-To-Pedestrian)
    - Enables infrastructure to communicate with pedestrians and cyclists
    - Uses smartphones, wearables, or smart devices
    - Enhances safety for vulnerable road users
    - Often integrated with smart city technologies

Today's World

Static Systems in a Dynamic World

Source: www.post-gazette.com

Key Stat

54 Hours

Average time wasted per year by commuters in traffic in the USA.

- **Inefficiency:** Fixed-timer semaphores fail to adapt to real-time flow
- **Safety Risk:** Congestion increases driver fatigue and accident probability
- **Inaccessibility:** Standart intervals do not accommodate reduced mobility
- **Emergency Delays:** First responders blocked by static signaling

Problem Statement

A system based on Vehicle-To-Infrastructure communication, incorporating Infrastructure-To-Pedestrian interaction.

The system should enable **communication between traffic lights**, **emergency vehicles** and **pedestrians** in traffic intersections. Thus, it should allow for **alerts** and **proactive adjustments**. Additionally, it should keep track of some pedestrians movement, enhancing safety.

Requirements

- Functional:
  - **Command** Semaphores based on time and external events;
  - **Track** the path done by people with disabilities;
  - **Communicate** with nearby ambulances;
  - **Alert** pedestrians in case of an emergency situation.

- Non-Functional:
  - Provide a **friendly** User Interface;
  - **Durable**;
  - Low Market Price.

Constraints

- Technical:
  - Use the Raspberry Pi;
  - Use Buildroot's embedded Linux image;
  - Use C/C++ language (OOP).
  - Implement at least one Linux device driver;
  - Use UML for the analysis (OOD);
  - Use Pthreads.

- Non-Technical:
  - Project must be developed in groups with no more than 2 elements;
  - Project deadline at the end of the semester;
  - Limited budget.

The AllWays Safe Solution

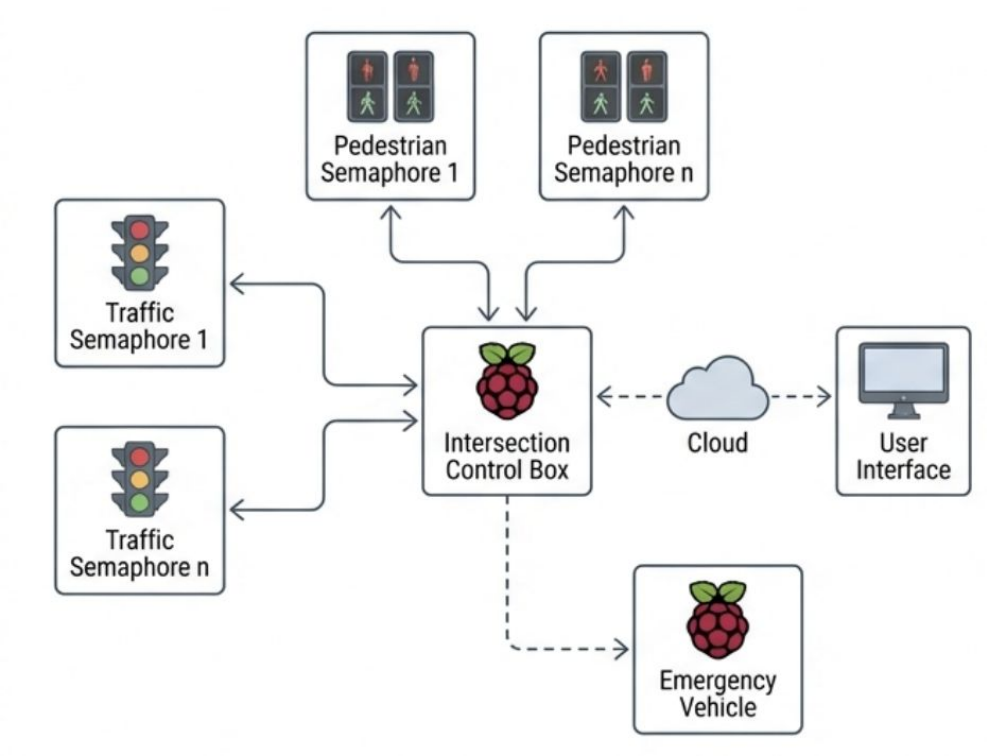
A modular V2I (Vehicle-To-Infrastructure) & I2P (Infrastructure-To-Pedestrian) ecosystem.

Affordability

Low-cost hardware vs expensive industrial material.

Inclusivity

Dedicated RFID and Audio cues for visually impaired and mobility-challenged users.



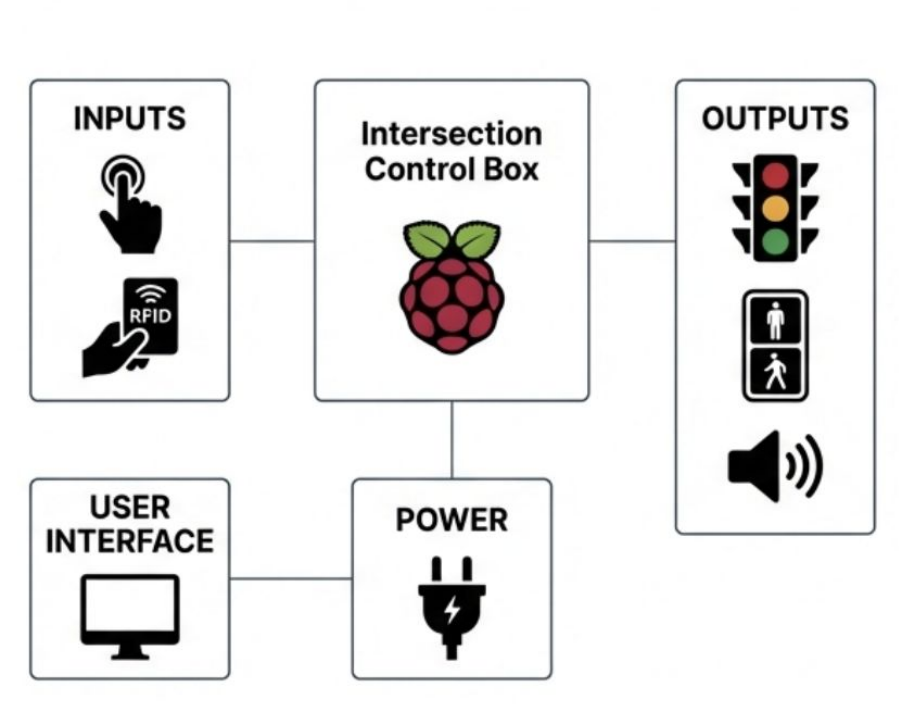
Independence

Local logic processing reduces latency of cloud-only reliable systems.

Monitoring

Current system state and pedestrian information is sent to cloud in real-time.

Intersection Control Box Hardware Architecture



Central Processing Unit

Raspberry Pi 4 Model B (Broadcom BCM2711, Quad-Core Cortex-A72)

Input Modules

Pedestrian: Push-buttons & RFC522 RFID Module

Output Modules

Visual: LEDs Traffic/Pedestrian Semaphores
Auditory: Active PWM Buzzer on Pedestrian Semaphore Green Light

Connectivity

Access-Point Mode (AP), via Data Distribution Service (DDS) Middleware

Power

Power Supply: LM50-23B05R2

Emergency Vehicle Hardware Architecture

Central Processing Unit

Raspberry Pi 4 Model B (Broadcom BCM2711, Quad-Core Cortex-A72)

Input Modules

Pedestrian: Analog-to-Digital Converter (ADC)

Output Modules

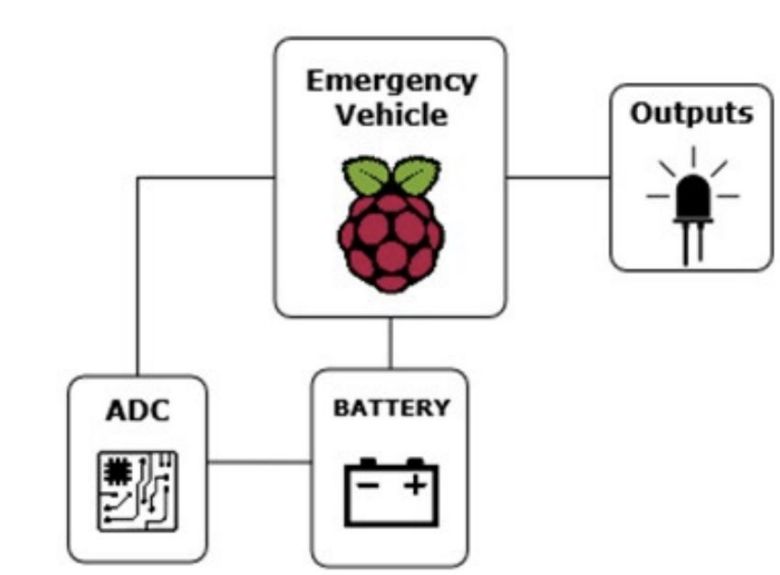
Visual: LED

Connectivity

Wi-Fi, using Data Distribution Service (DDS) Middleware

Power

Battery: 7.4V Li-ion Pack



Monitoring System

System

Any computer system that meets the following requirements.

Internet Connectivity

Wired Ethernet or reliable broadband internet access

Input Devices

Support for **keyboard and mouse** (USB or wireless)

Display

Monitor or display output compatible with the system

Operating System

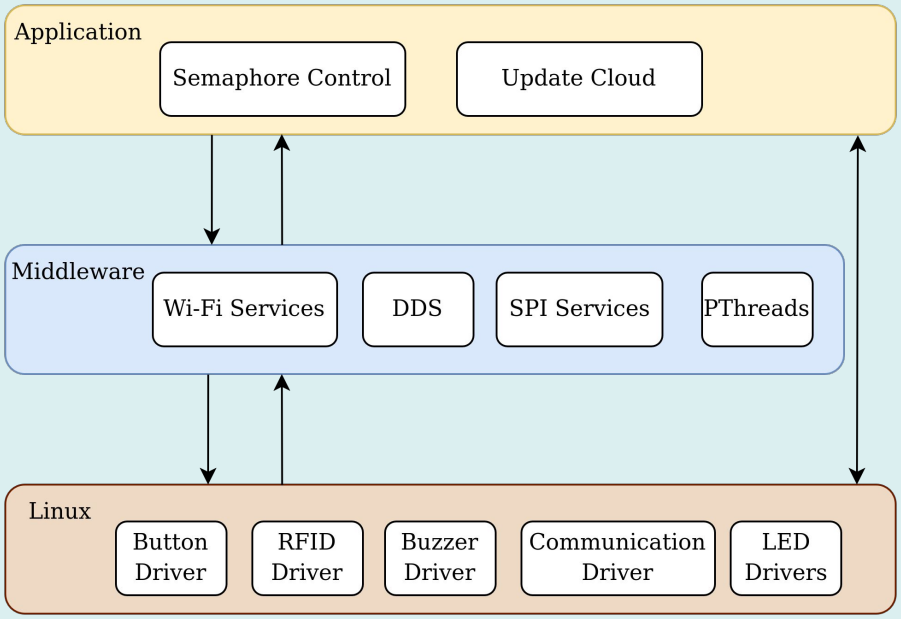
Capable of running a modern web browser or the required monitoring software

Network Access

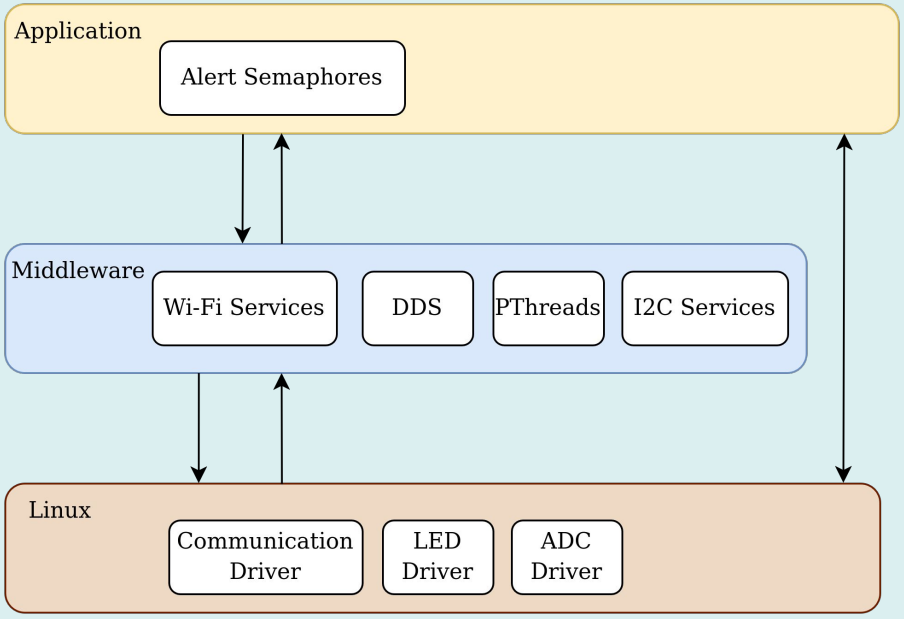
Ability to securely connect to a remote/cloud-hosted database connection.

Technology Stack

Intersection Control Box



Emergency Vehicle



Package Diagram

Model-View-Controller(MVC)

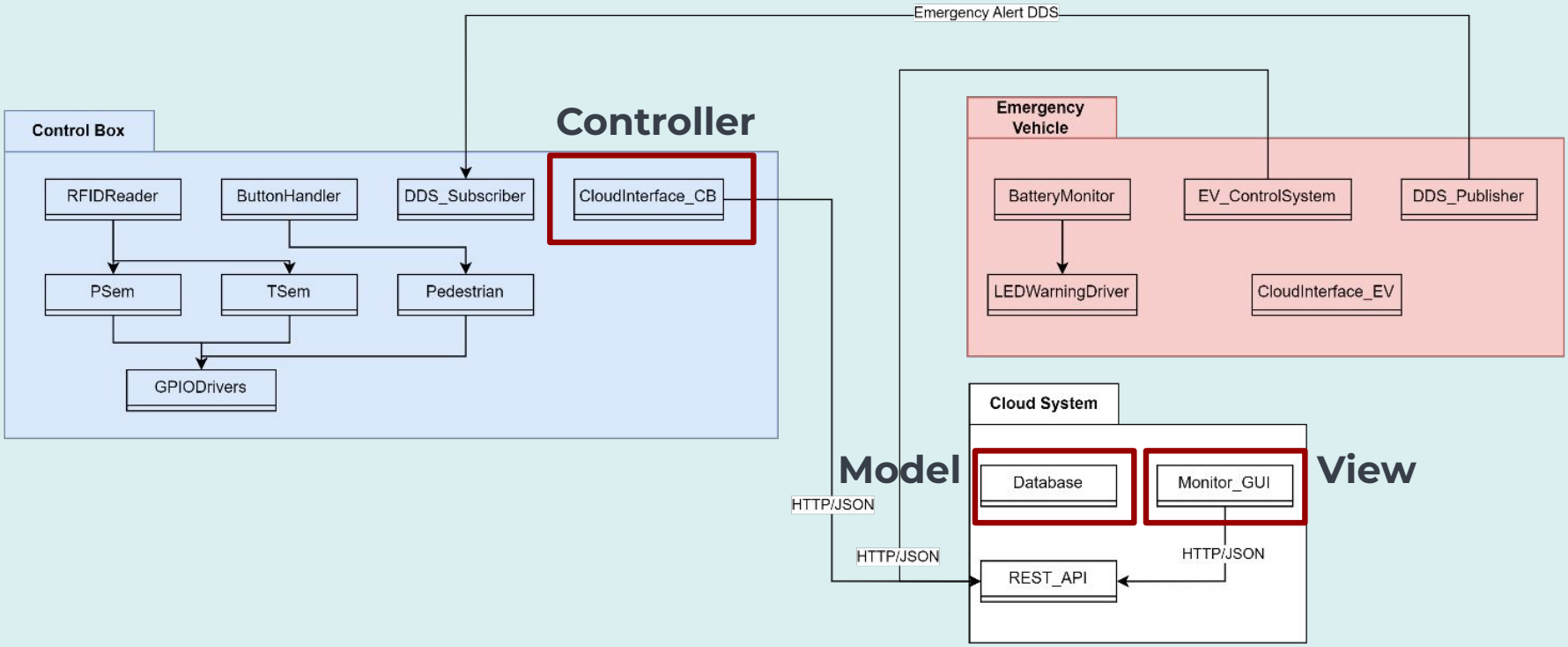


Image Configuration Buildroot

Intersection Control Box

- Access Point
- SPI
- NTP
- libgpio
- Curl
- JSON
- Fast DDS

Emergency Vehicle Control Box

- ADS1115
- WPA
- Curl
- JSON
- Fast DDS

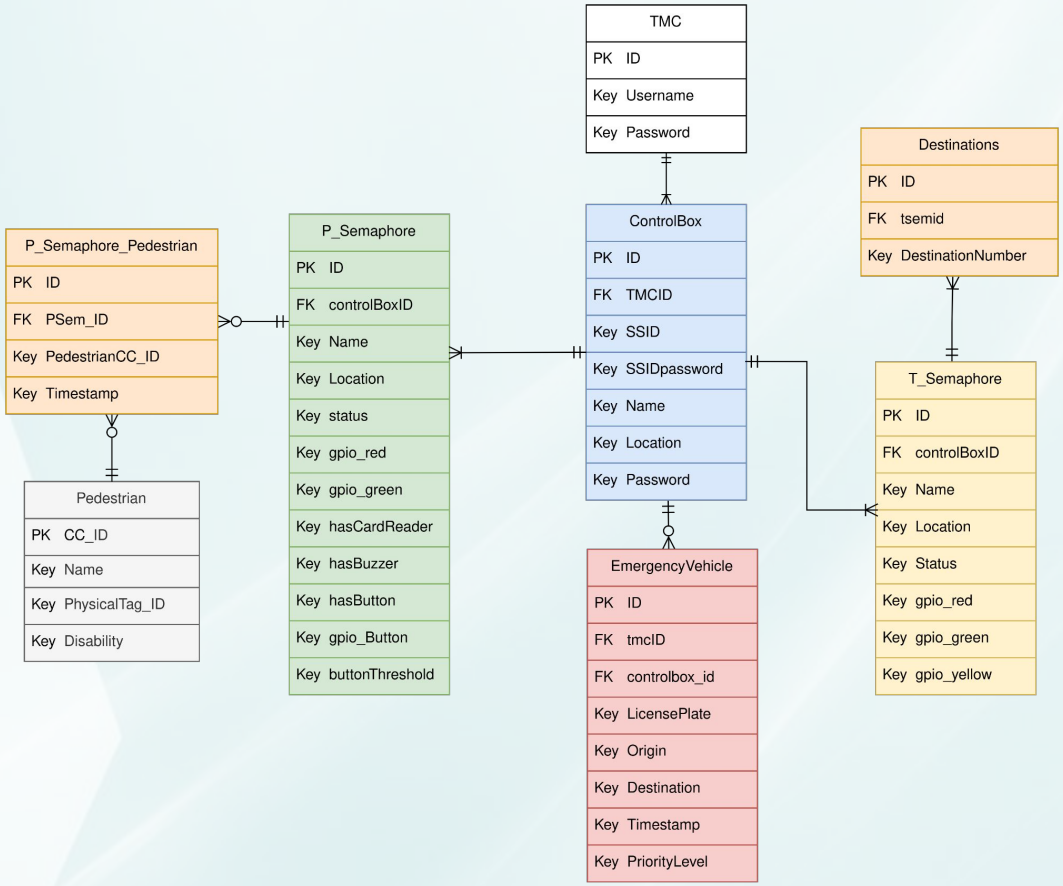
Supabase

CENTROALGORITMI

Fully managed **relational database built on top of PostgreSQL.**

Provides:

- Authentication;
- Storage;
- Real-time data synchronization.



Entity Relationship Diagram

Supabase

Example of an implemented table using SQL:

```
CREATE TABLE IF NOT EXISTS ControlBox (
    ID UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    TMCID UUID NOT NULL,
    Password TEXT NOT NULL,
    Name TEXT,
    Location TEXT,SSID TEXT NOT NULL,
    SSIDPassword TEXT NOT NULL,
    FOREIGN KEY (TMCID) REFERENCES TMC(ID) ON DELETE CASCADE
);
```

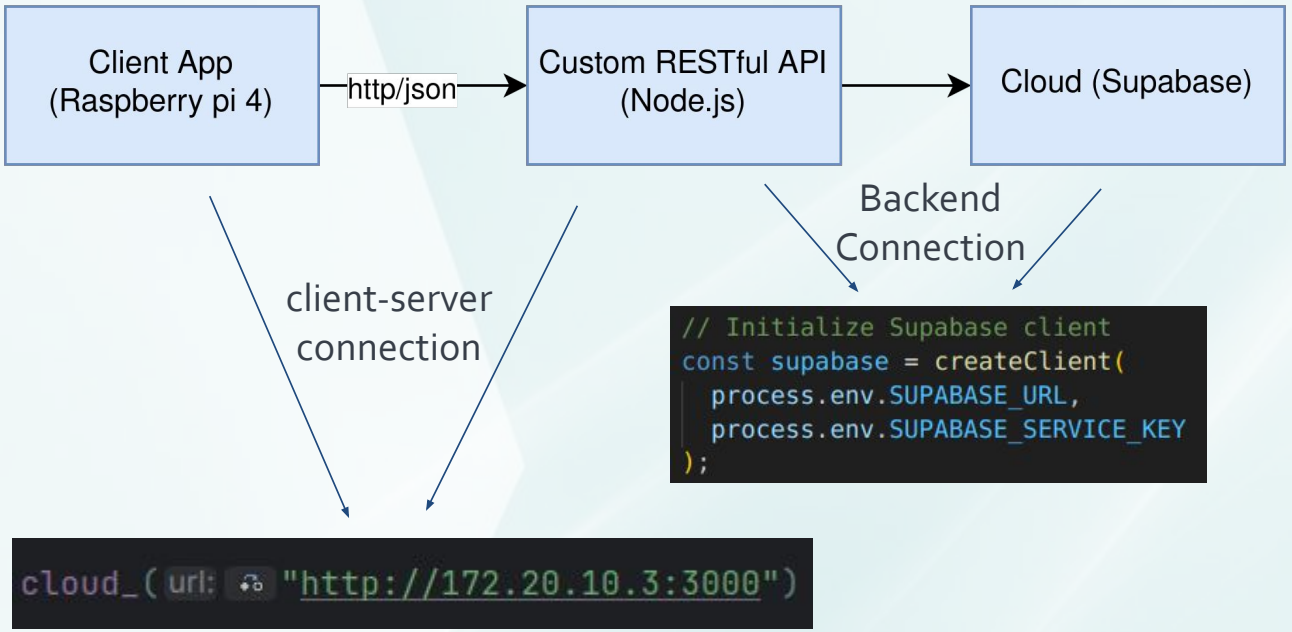
id uuid	password text	name text	location text	tmcid uuid	ssid text
cfc2d011-f8c2-43c	mari@1	raspMari.local	Avenida da Liberdade	8a7afae5-6814-4b1a-8ade-52f722...	wifimari

RESTful API

Full Control Over Functionality - Customized RESTful API

Endpoints designed exactly how our application needs them.

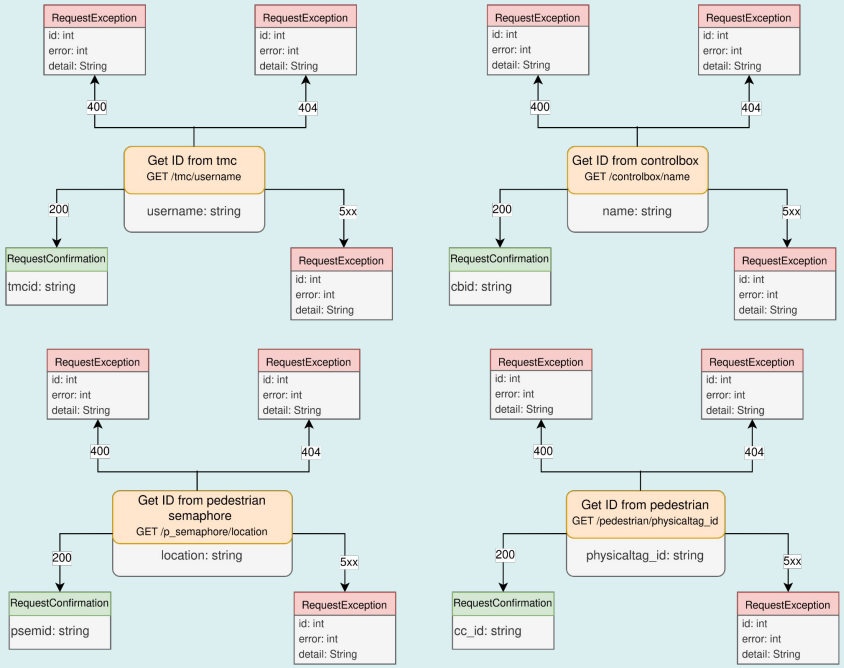
Only exposes the data and actions that make sense for our application.



RESTful API

Implementation of **GET**, **POST** and **PATCH** HTTP methods.

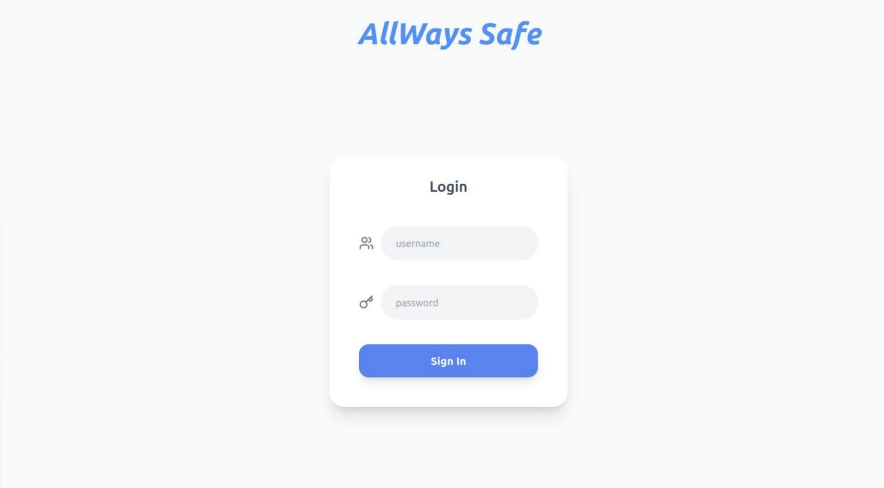
```
router.get("/id/:table/:identifier", async (req, res) => {
    try {
        const { table, identifier } = req.params;

        const TABLE_CONFIG = {
            p_semaphore: { identifierField: "location", idField: "id" },
            controlbox:  { identifierField: "name", idField: "id" },
            tmc:         { identifierField: "username", idField: "id" },
            pedestrian:  { identifierField: "physicaltag_id", idField: "cc_id" }
        };

        const { data, error } = await supabase
            .from(table)
            .select(idField)
            .eq(identifierField, identifier)
            .limit(1);

        if (error) {
            return res.status(500).json({
                error: 500,
                detail: error.message
            });
        }

        if (!data || data.length === 0) {
            return res.status(404).json({
                error: 404,
                detail: "Record not found"
            });
        }
        //returns ID
        return res.status(200).json({
            id: data[0][idField]
```



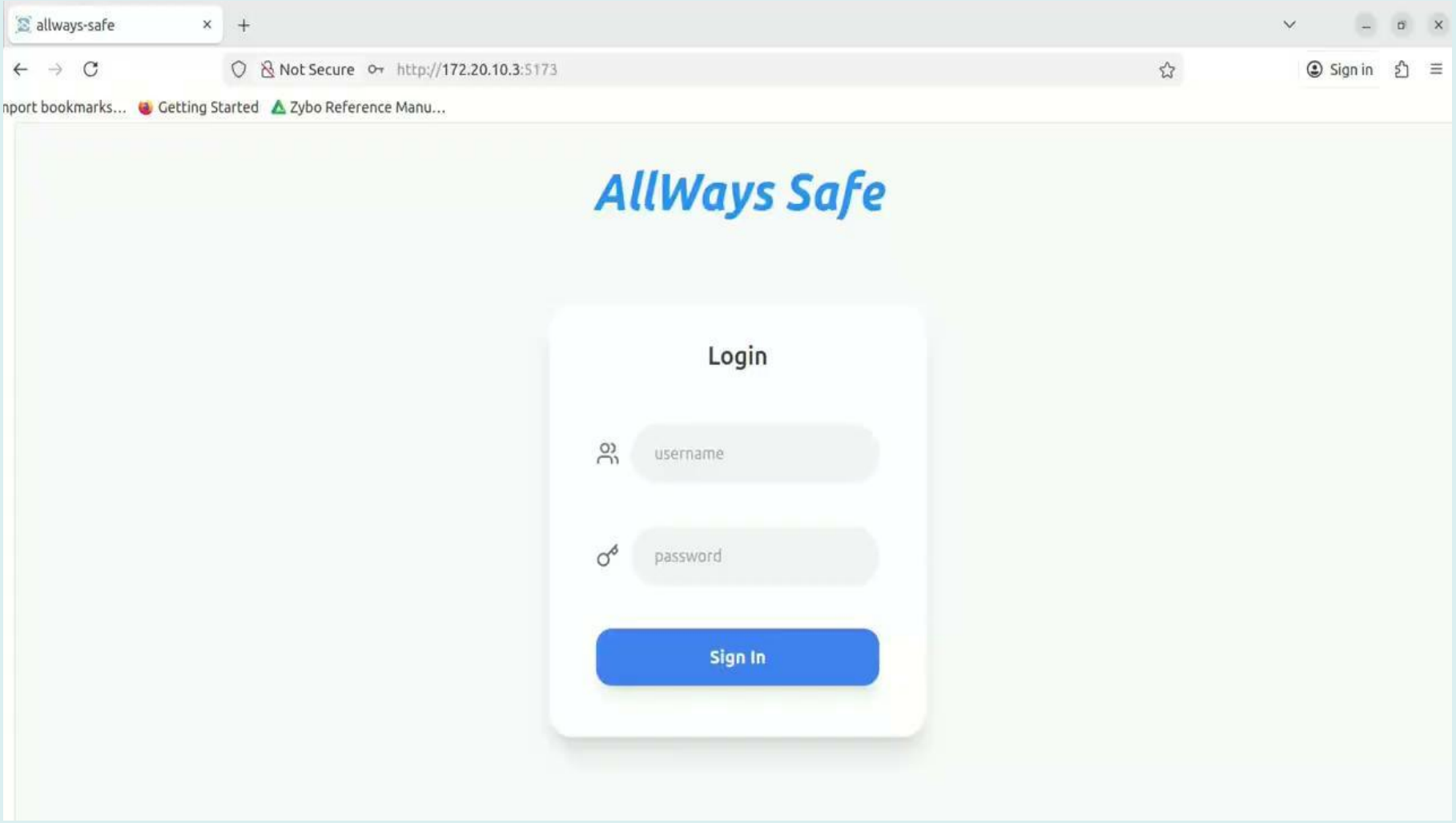
Graphical User Interface (GUI)

Monitoring System

- Enables authentication;
- Enables system configuration: Traffic/Pedestrian semaphores location, GPIO on board, and extras;
- Enables Pedestrian Registration;
- Enables visualization of Traffic/Pedestrian Semaphores state;
- Enables tracking of Pedestrians path on intersections;
- Enables tracking of Emergency vehicles path on intersections.



GUI


allways-safe
Not Secure
http://172.20.10.3:5173
Sign in
nport bookmarks...
Getting Started
Zybo Reference Manu...
AllWays Safe
Login
username
password
Sign In

Real Time Updates with GUI

ICB

```
self->patch_database(obj.table, "location", obj.location,
    "status", obj.status);
```

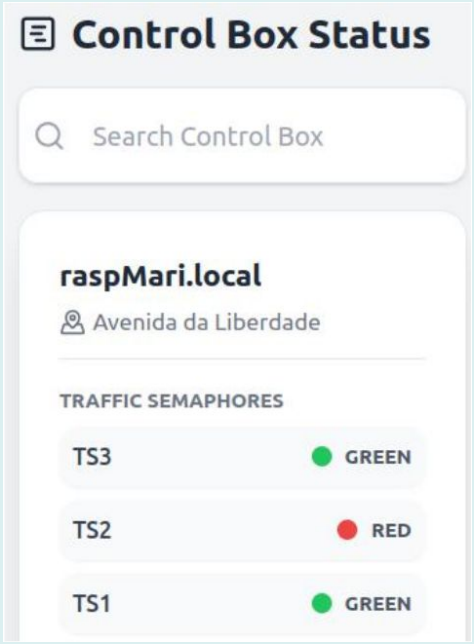
RESTful API

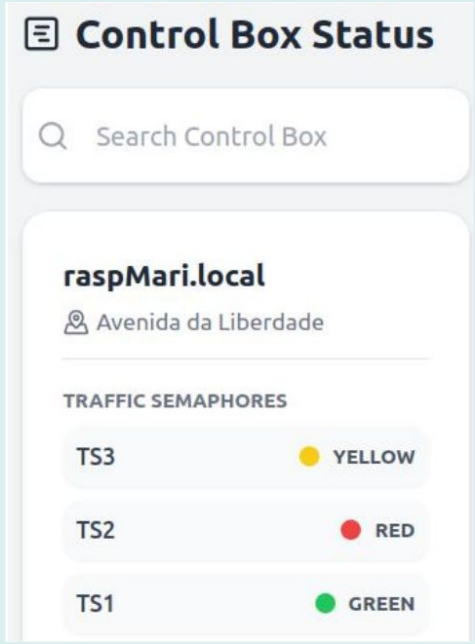
Supabase

name text	status TrafficColor
TS1	GREEN
TS2	RED
TS3	GREEN

name text	status TrafficColor
TS1	GREEN
TS2	RED
TS3	YELLOW

GUI





Monitoring System Test Cases

Test Case	Test Description	Expected Result	Real Result
Connection Test	Check connection to cloud	Can connect to cloud	GUI connected to the cloud
Data Insertion	Check write to cloud	Can insert items in cloud	Items inserted in cloud
Data Retrieval	Check read from cloud	Can read items from cloud	Items read from cloud and displayed in GUI
Data Alteration	Check altered items in cloud	Can alter items in cloud	Items altered in cloud through GUI
Data Consistency	Verify that updates made locally are reflected in the cloud database and vice versa	Data remains consistent between local and cloud databases	Updates are real-time and consistent
Unauthorized access attempt detected	Try access cloud with an invalid login	Access not granted	Invalid username and/or password - access not granted

Linux Device Driver

PWM Device Driver

The PWM Device Driver for RPi 4 Model B consists on the configuration of **various registers**, from **GPIO** addresses, **PWM** registers and **Clock** registers.

File Operations

The driver implements the standard Linux character device file operations:

Operation	Function	Description
open	`pwm_device_open`	Opens device, stores register pointer
release	`pwm_device_close`	Closes device, clears private data
read	`pwm_device_read`	Currently not implemented
write	`pwm_device_write`	Text-based interface: "freq duty"
unlocked_ioctl	`pwm_device_ioctl`	Binary interface for configuration

Frequency (1/Period) and *duty-cycle* configuration.

PWM RNG Registers - Range (Period)

RNG1 / RNG2 (32-bit)
Range Value (0x00000000 - 0xFFFFFFFF)

Purpose: Defines the PWM period in clock cycles.

Calculation:

$$\text{RNG} = \frac{f_{\text{clock}}}{f_{\text{desired}}}$$

PWM DAT Registers - Data (Duty Cycle)

DAT1 / DAT2 (32-bit)
Data Value (0x00000000 - RNG value)

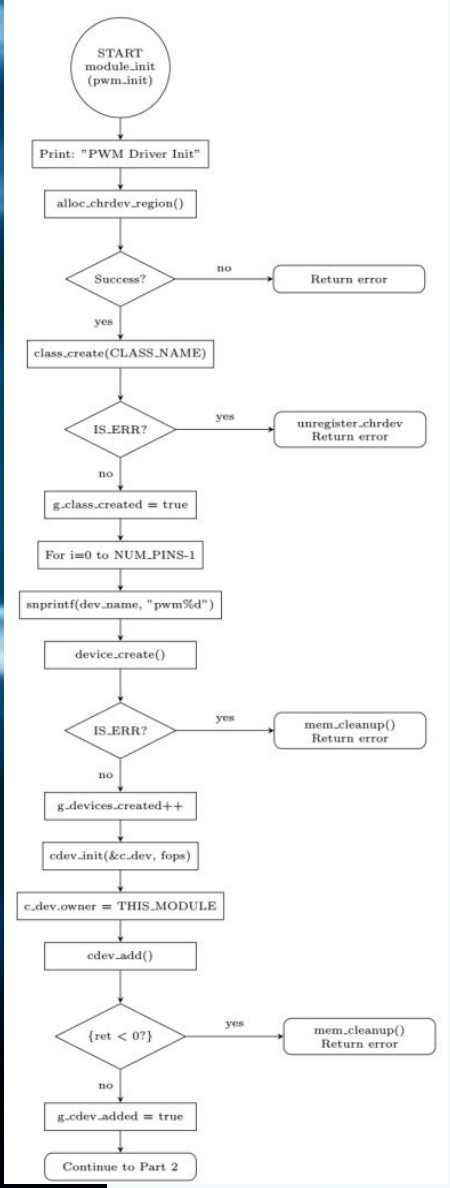
Purpose: Defines the number of clock cycles the output is HIGH.

Calculation:

$$\text{DAT} = \frac{\text{RNG} \times \text{duty\_cycle}}{100}$$

Linux Device Driver

PWM Device Driver



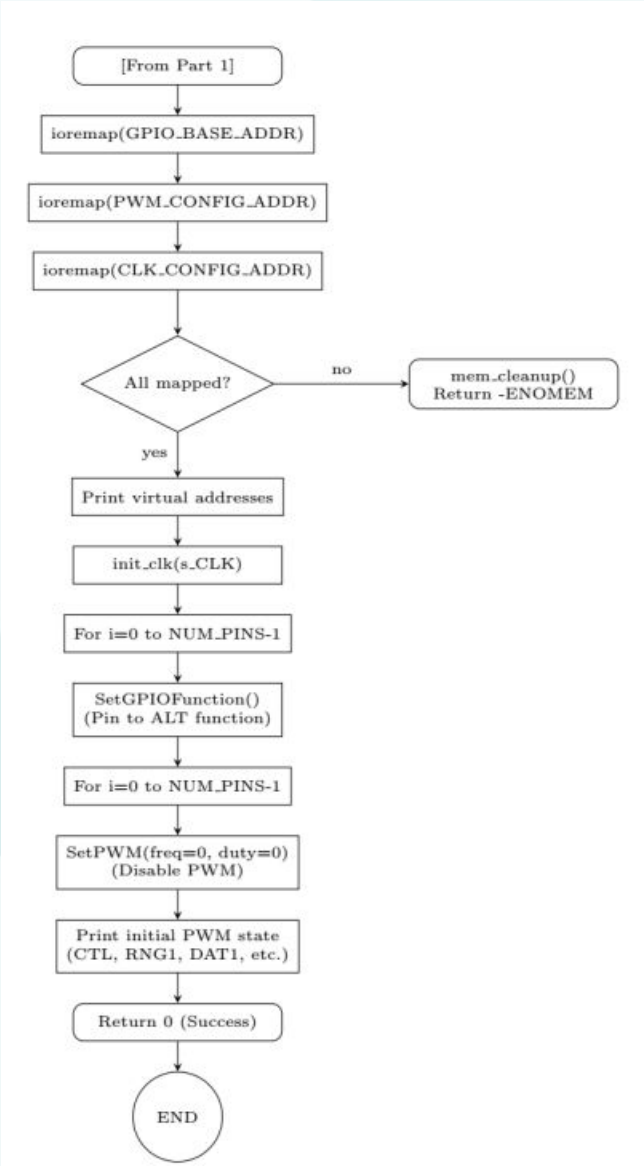
The PWM Device Driver for RPi 4 Model B includes initialization, exit, and configurations functions.

```
static int __init pwm_init(void)
{
  /* Allocate device numbers */
  alloc_chrdev_region(&DEV_major_minor, 0, NUM_PINS, DEVICE_NAME);
   /* Create class */
   class_create(CLASS_NAME);
   /* Create device nodes for each minor */
   for (int i = 0; i < NUM_PINS; i++) {
       char dev_name[16];
       snprintf(dev_name, sizeof(dev_name), "pwm%d", i);
       if (IS_ERR(dev_ret = device_create(pwm_class, NULL,
MKDEV(MAJOR(DEV_major_minor), i), NULL, dev_name))) {
           mem_cleanup();
           return PTR_ERR(dev_ret);
       }
       g_devices_created++;
   }

   /* Initialize character device - specify fops */
   cdev_init(&c_dev, &pwmDevice_fops);
   c_dev.owner = THIS_MODULE;

   /* Add character device to system */
   cdev_add(&c_dev, DEV_major_minor, NUM_PINS);
```

Device Driver Initialization (Simplified), *Pt.1*

Linux Device Driver

PWM Device Driver




```
   /** CONTINUATION **/
   /* Map hardware registers */
   s_GPIO  = (GPIORegister *)
ioremap(GPIO_BASE_ADDR,
sizeof(GPIORegister));
   s_PWM   = (PWMRegister *)
ioremap(PWM_CONFIG_ADDR,
sizeof(PWMRegister));
   s_CLK   = (CLKRegister *)
ioremap(CLK_CONFIG_ADDR,
sizeof(CLKRegister));

   /* Set up PWM clock first */
   init_clk(s_CLK);

   /* Configure GPIO pins for PWM
respective alternate function */
   for (int i = 0; i < NUM_PINS; i++)
       SetGPIOFunction(s_GPIO,
Pins[i], PWMconfig[Pins[i]].FSELx,
s_CLK);

   for (int i = 0; i < NUM_PINS; i++)
       SetPWM(s_PWM, Pins[i], 0, 0,
s_CLK);

   return 0;
}
```

Device Driver Initialization (Simplified), *Pt.2*

Linux Device Driver

PWM Device Driver

The defined ioctl operations :

```
#define PWM_SET_CONFIG   _IOW(PWM_IOC_MAGIC, 1, ioctl_pwm_config_t)
#define PWM_DISABLE      _IO(PWM_IOC_MAGIC, 3)
```

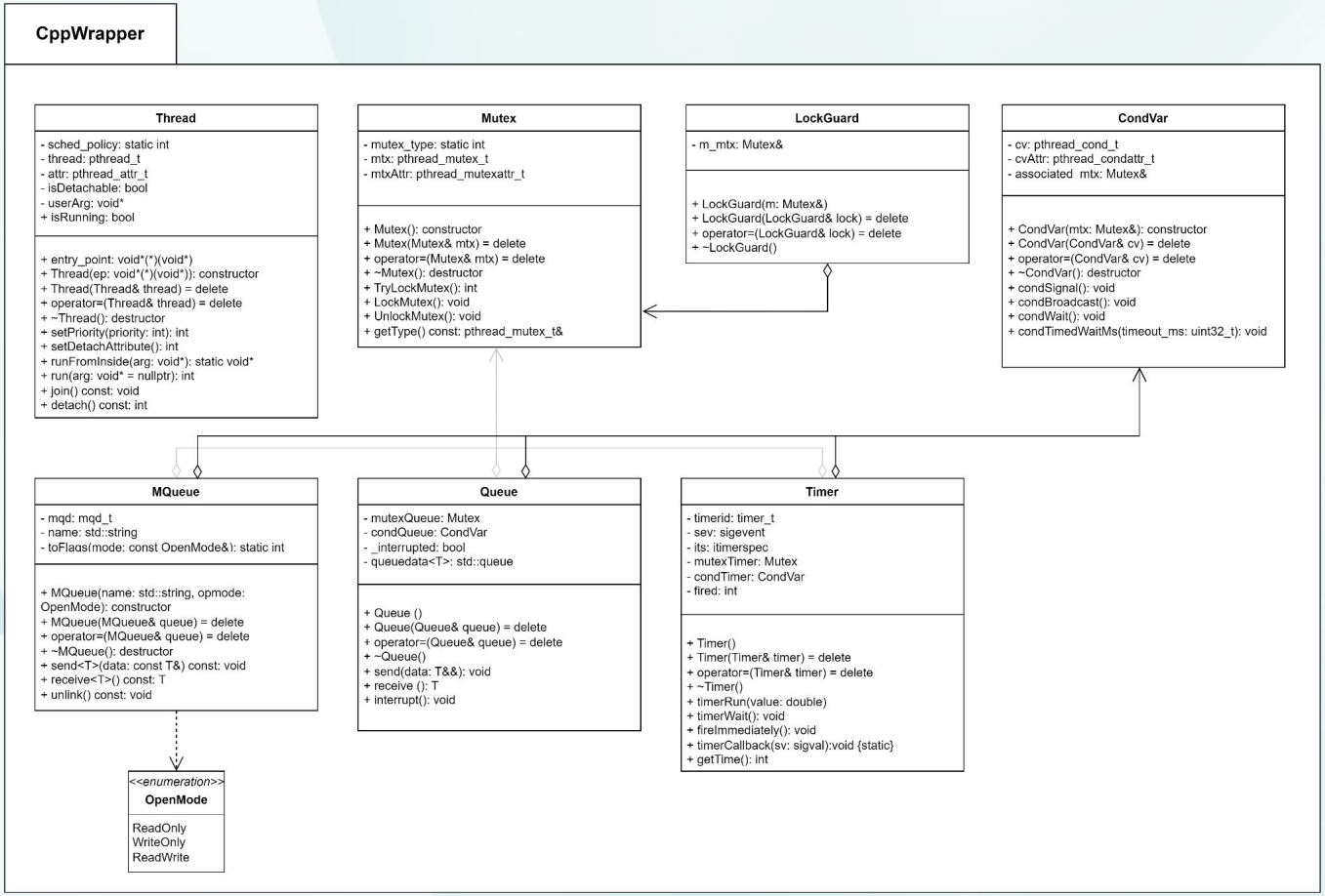
Can be called as per:

```
ioctl (file_descriptor, PWM_SET_CONFIG, &conf);  //update value
ioctl (file_descriptor, PWM_DISABLE, &conf);
```

These are handled under the **pwm_device_ioctl** function, indicated in the file operations

C++ Wrapper

The developed C++ Wrapper enables **RAII** mechanisms on POSIX threads, timers and IPC relevant mechanisms for the project. It also enables thread-safe mechanisms on the C++ std::queue.



C++ Wrapper

CppWrapper::Thread Implementation


```
namespace CppWrapper
{
    class Thread
    {
        static int sched_policy;
        pthread_t thread;
        pthread_attr_t attr;
        bool isDetachable;
        void* userArg;

    public:
        bool isRunning;

        void*(*entry_point)(void *);

        explicit Thread(void*(*ep)(
            void *));
        Thread(const Thread&)=delete;
        Thread& operator=(const
            Thread&)=delete;

        ~Thread();

        int setPriority(int priority)
            ;
        int setDetachAttribute();

        static void* runFromInside(
            void* arg);
        int run(void* arg = nullptr);
        void join() const;
        [[nodiscard]] int detach();
    };
}
```

Listing 4.9: POSIX Thread C++ Wrapper implementation

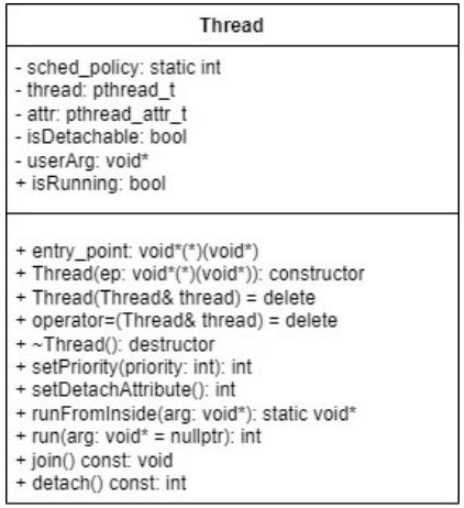


Figure 4.16: Class Diagram of POSIX Thread Wrapper

```
// Argument is of type Thread
void* Thread::runFromInside(void* arg)
{
    auto caller_thread = static_cast<Thread*>(arg);
    void* ret =
caller_thread->entry_point(caller_thread->userArg);
    // Code will reach this point upon Thread
function completion
    caller_thread->isRunning = false;

    return ret;
}

// creates the thread (it starts running) sets
isRunning.
// &Thread::runFromInside  is passed to ensure
variable isRunning is updated correctly
// arg is the executing function received argument
int Thread::run(void* arg)
{
    if (isRunning)
        return -EPERM;

    userArg = arg;

    int s = pthread_create(&thread, &attr,
&Thread::runFromInside, this); // this: is the
argument to the runFromInside function
    if (s != 0)
        throw std::runtime_error("Thread:
pthread_create");

    isRunning = true;

    return 0;
}
```

C++ Wrapper

CppWrapper::Queues Implementation


```
1  namespace CppWrapper
2  {
3   // Queue allows to have non-
       trivially copiable data, contrary
        to MQueue
4      template <typename T>
5      class Queue
6      {
7          Mutex mutexQueue;
8          CondVar condQueue;
9
10         bool _interrupted;
11
12         std::queue<T> queueData;
13     public:
14         Queue(): condQueue(mutexQueue
               ), _interrupted(false){};
15         Queue(const Queue& queue) =
               delete;
16         Queue& operator= (Queue&
               queue) = delete;
17         ~Queue();
18
19         void send (T&& data)
20         {/*...*/}
21
22         T receive ()
23         {/*...*/}
24
25         void interrupt ()
26         {/*...*/}
27     };
28 }
```

Listing 4.10: POSIX Thread C++ Wrapper implementation

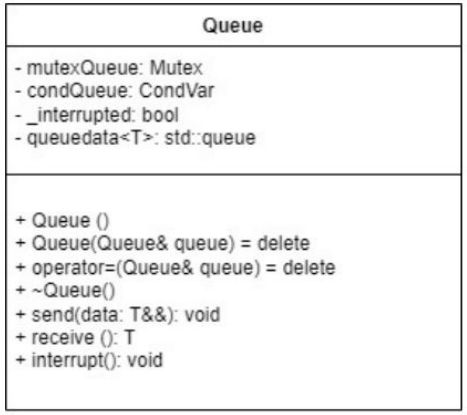


Figure 4.17: Class Diagram of C++ Queue Wrapper

The AllWays Safe

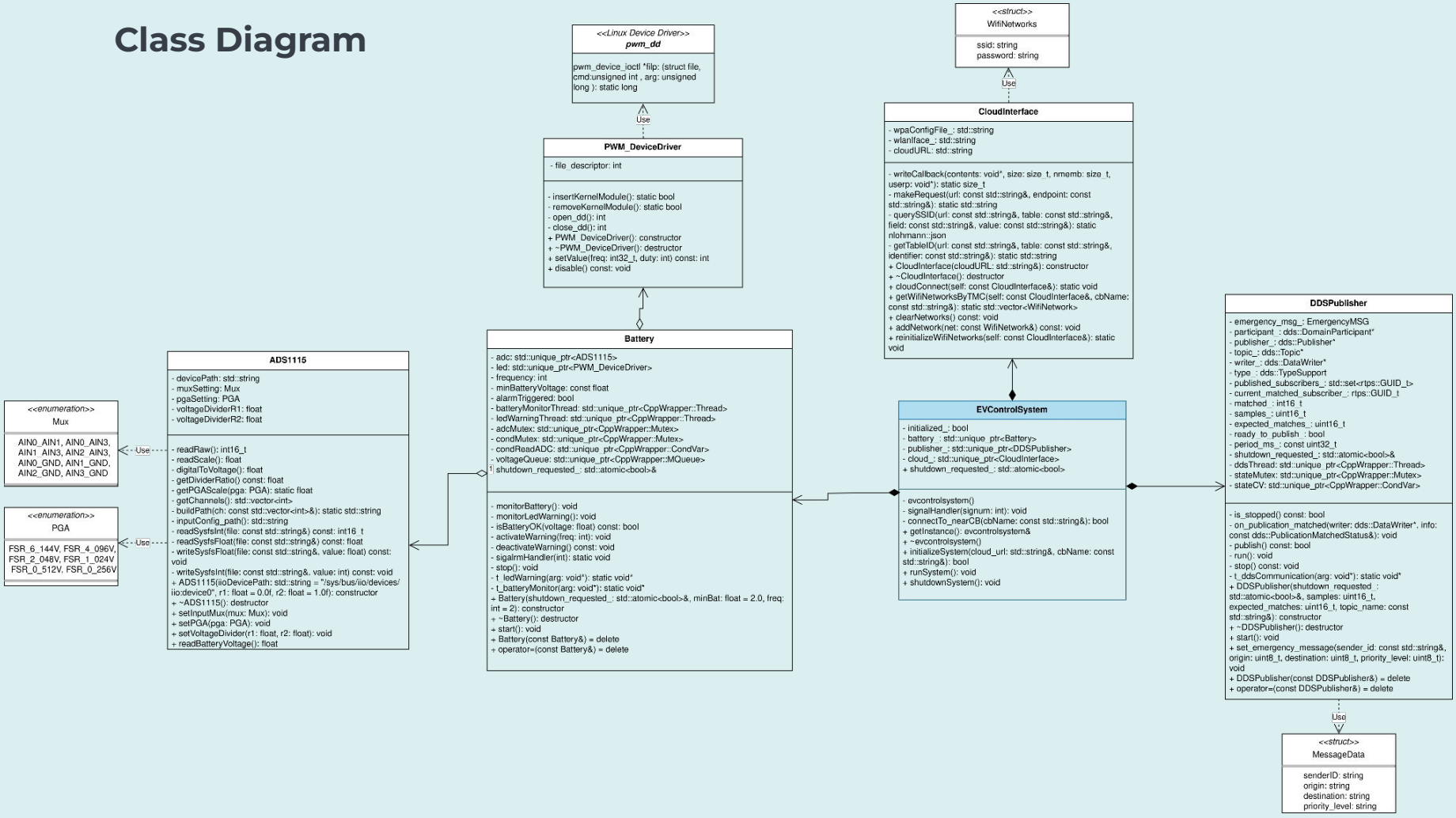
Emergency Vehicle Control Box Solution

- Cloud Connectivity for SSID Management;
- Data Distribution Service (DDS) for reliable, low-latency communication over Wi-Fi networks with ICB, for warning purposes;
- Emergency Vehicle Battery Monitoring;
- Low Battery Warning Mechanism.

Emergency Vehicle Control Box

CENTROALGORITMI

Class Diagram

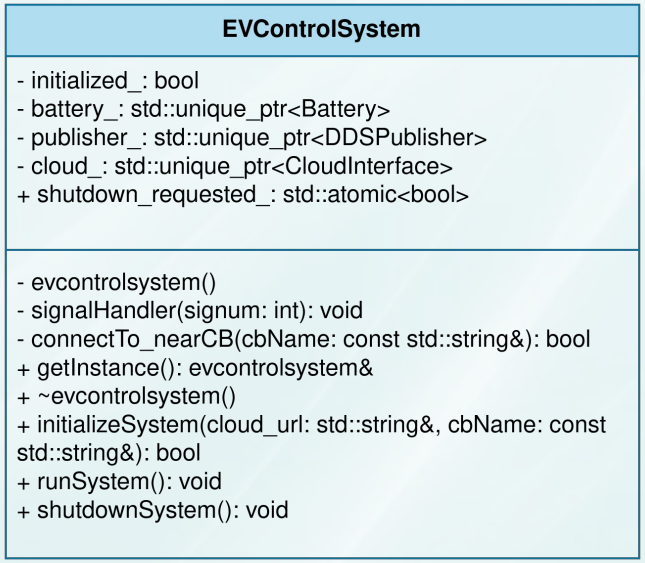


Emergency Vehicle Control Box

Design Patterns

Singleton

```
class evcontrolsystem {
private:
 // Private constructor
    evcontrolsystem();

public:
    static evcontrolsystem& getInstance
        ()
    {
        static evcontrolsystem instance;
        return instance;
    }

    ~evcontrolsystem();
};
```

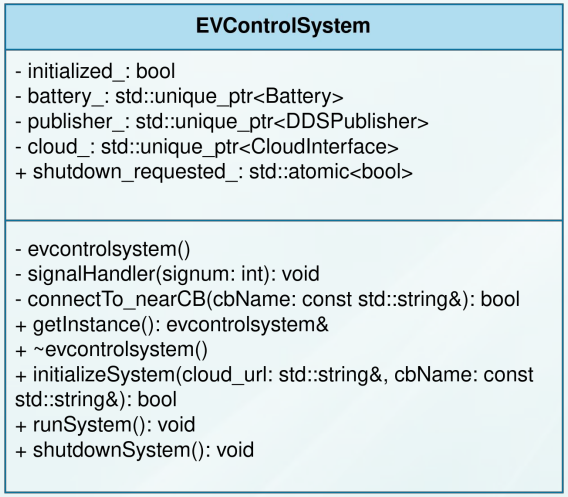


Emergency Vehicle Control Box

Design Patterns

Facade

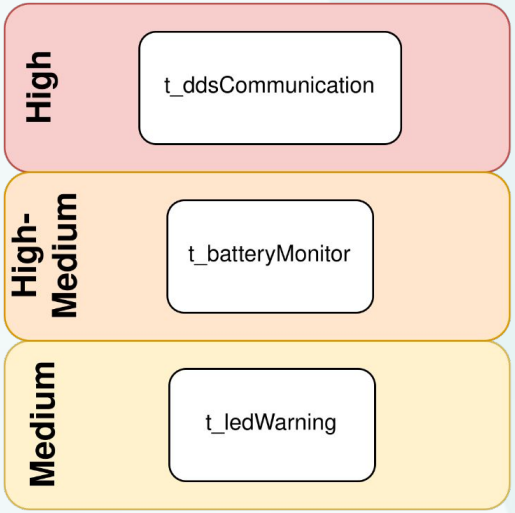
Exposes high-level methods hiding the intricate details of complex subsystems



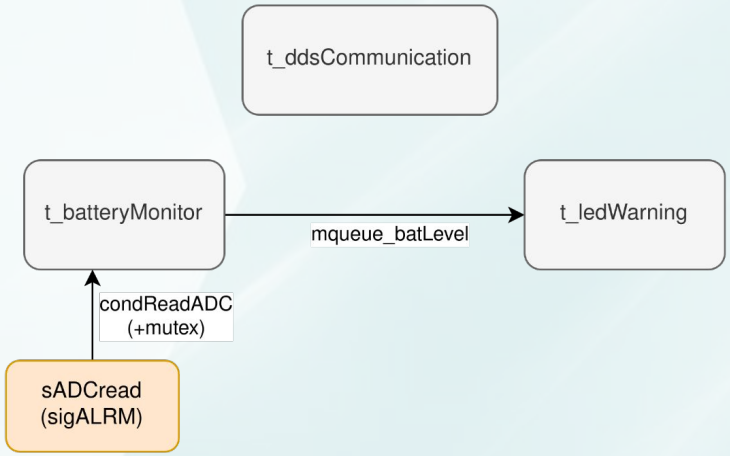

```
public:
    bool initializeSystem(std::string& cloud_url, const std::string
        & tmcName);
    void runSystem() const;
    void shutdownSystem();
};

void evcontrolsystem::runSystem() const
{
    if (!initialized_) return;
    battery_->start();
    publisher_->start();
}
```

Emergency Vehicle Control Box

Threads Overview




```
void Battery::start()
{
    signal( sig: SIGALRM, Battery::sigalrmHandler);

    batteryMonitorThread = std::make_unique<CppWrapper::Thread>(t_batteryMonitor);
    batteryMonitorThread->setPriority(60);
    batteryMonitorThread->run( arg: this);

    ledWarningThread = std::make_unique<CppWrapper::Thread>(t_ledWarning);
    ledWarningThread->setPriority(30);
    ledWarningThread->run( arg: this);

    itimerval timer{};
    timer.it_value.tv_sec = time_between_adcREAD;
    timer.it_interval.tv_sec = time_between_adcREAD;
    setitimer( which: ITIMER_REAL, &timer,  old: nullptr);
}
```



Emergency Vehicle Control Box

Start-up

```
evcontrolsystem *evSystem = &evcontrolsystem::getInstance();
std::string cloud_url = "http://172.20.10.3:3000";
std::string tmcName = "tmc1";
evSystem->initializeSystem([&]cloud_url, tmcName);
evSystem->runSystem();
```

Shutdown

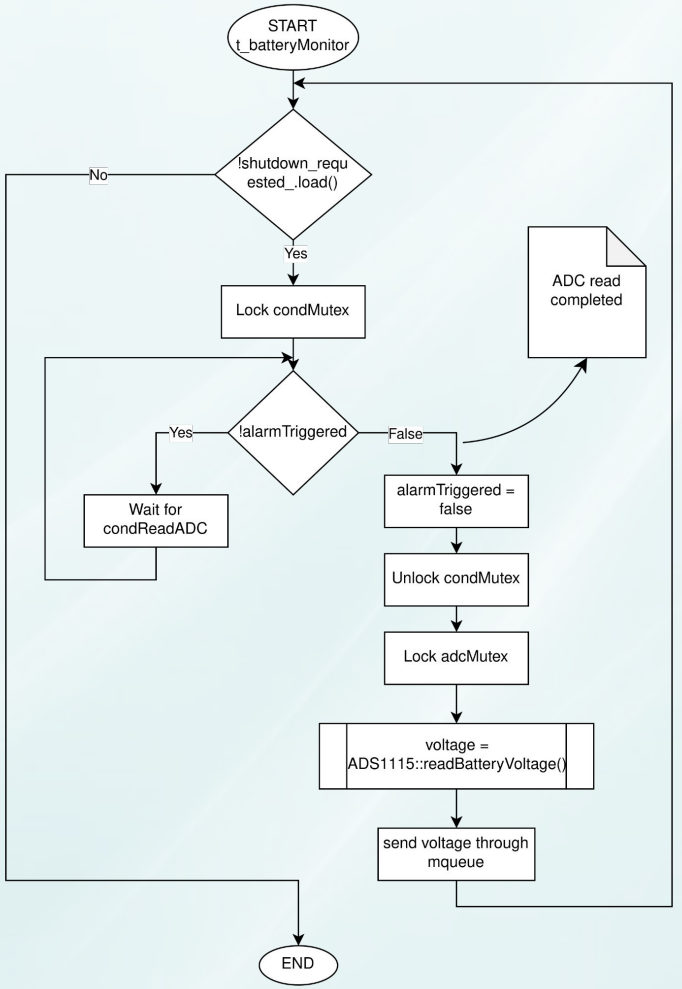
```
void evcontrolsystem::shutdownSystem()
{
    delete battery_;
    battery_ = nullptr;

    delete publisher_;
    publisher_ = nullptr;
}
```

```
void DDSPublisher::stop() const
{
    stateMutex->LockMutex();
    stateCV->condBroadcast();
    stateMutex->UnlockMutex();

    ddsThread->join();
}

DDSPublisher::~DDSPublisher()
{
    stop();

    if (participant_) {
        participant_->delete_contained_entities();
        dds::DomainParticipantFactory::get_instance()
            ->delete_participant(participant_);
    }
}
```

Emergency Vehicle Control Box

Battery Monitoring Thread

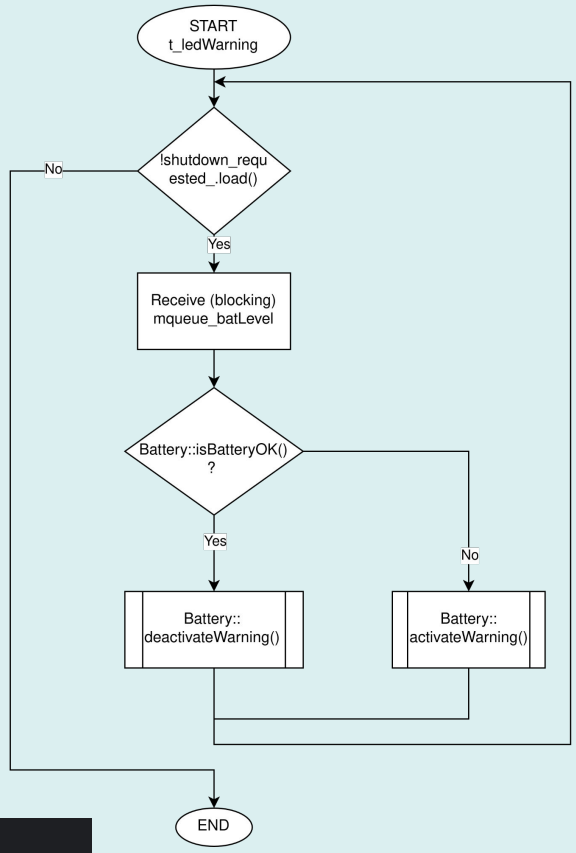
```
void Battery::monitorBattery()
{
    while (!shutdown_requested_.load
        ())
    {
        condMutex->LockMutex();
        while (!alarmTriggered)
            condReadADC->condWait();

        alarmTriggered = false;
        condMutex->UnlockMutex();

        float voltage;
        adcMutex->LockMutex();
        voltage = adc->
            readBatteryVoltage();
        adcMutex->UnlockMutex();

        std::cout << "[Battery]
            Current Battery Voltage: "
                << voltage << " V"
                    << std::endl;

        voltageQueue->send(voltage);
    }
}
```



Emergency Vehicle Control Box

Battery Warning Mechanism

```
void Battery::monitorLedWarning()
{
    while (!shutdown_requested_.load())
    {
        if (auto voltage = voltageQueue->receive<float>();
            isBatteryOK(voltage))
            deactivateWarning();
        else
            activateWarning(freq:2);
    }
}
```



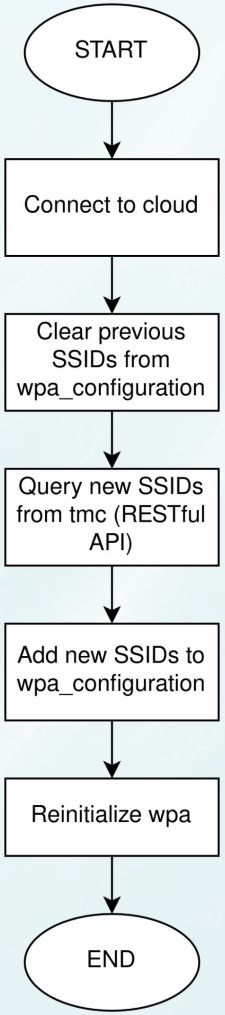
PWM device driver being used

```
void Battery::activateWarning( int freq)
{
    frequency = freq;
    int ret = led->setValue(freq:frequency, duty: STANDART_DUTY);
    if (ret < 0)
        throw std::runtime_error("Battery::activateWarning failed: invalid parameters");
}
```

Emergency Vehicle Control Box

Cloud Communication

```
bool evcontrolsystem::receive_SSIDs(const std::string& tmcName) const
{
    cloud_.clearNetworks();

    auto networks :vector<WifiNetwork> = cloud_.getWifiNetworksByTMC(tmcName);
    if (networks.empty()) return false;

    for (const auto& network : networks)
        cloud_.addNetwork(network);

    cloud_.reinitializeWifiNetworks();

    return true;
}
```



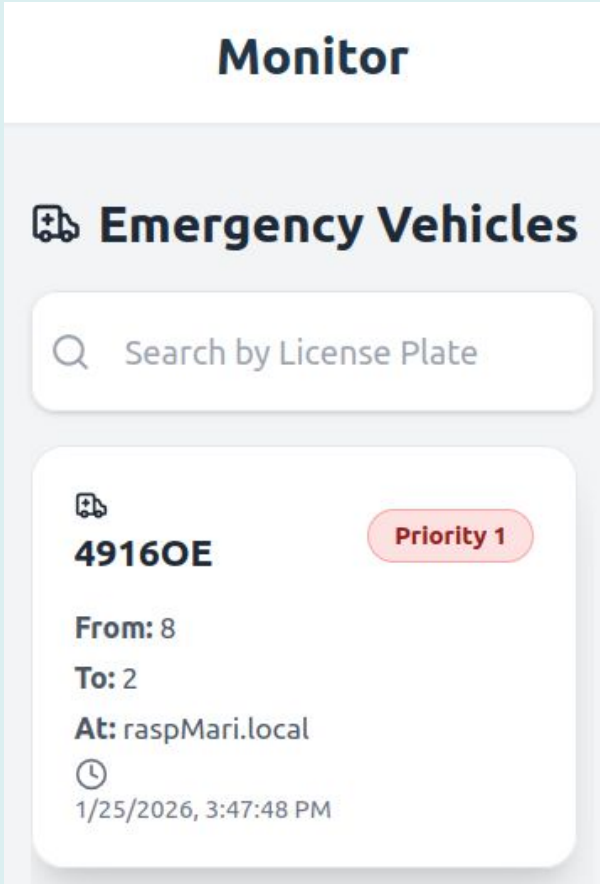
ESRG

Embedded Systems Research Group

Emergency Vehicle Control Box

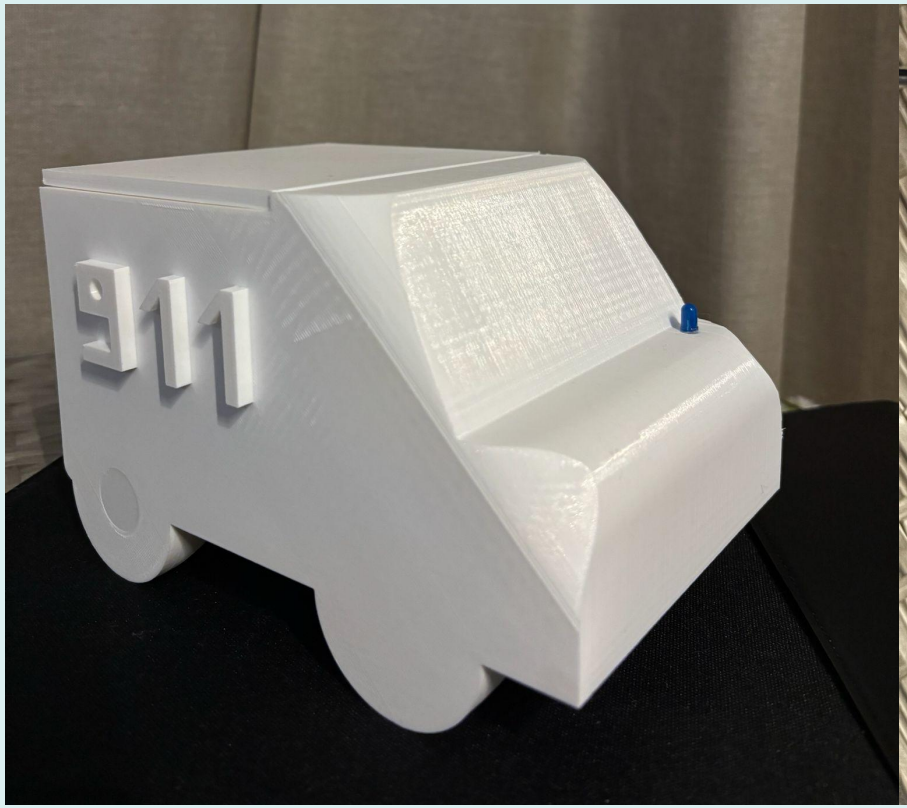
CENTROALGORITMI

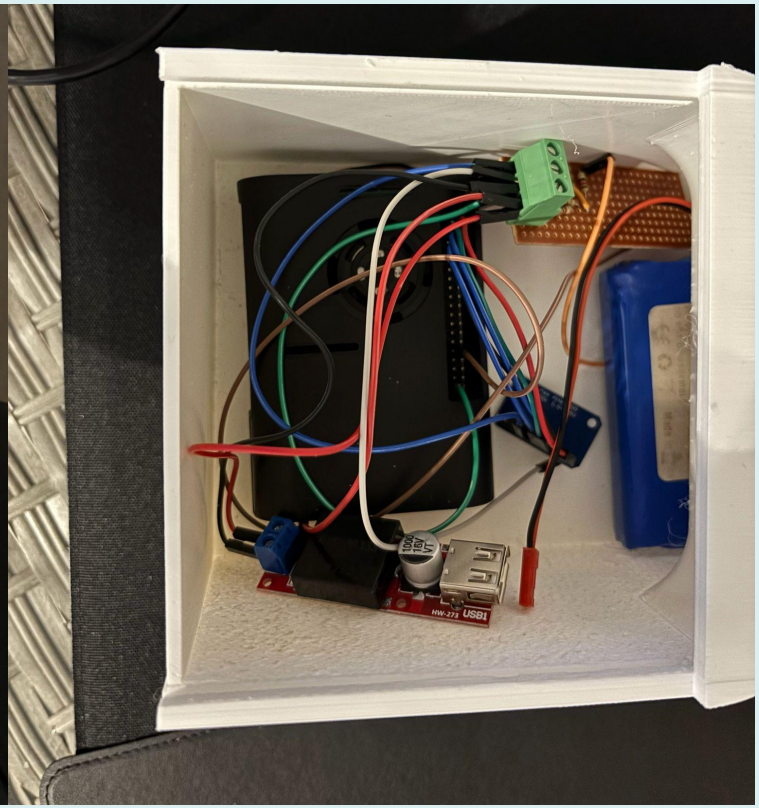
```
mariana@panda:~/Desktop/AllWaysSafe/AllWays-Safe$ ssh root@raspAndre.local
root@raspandre.local's password:
# cd /tmp/CLion/debug/
# ./evcontrolsystem
insmod: ERROR: could not insert module /root/pwm.ko: File exists
Module ALREADY loaded
Cloud connected successfully!
Selected interface 'wlan0'
OK
Successfully initialized wpa_supplicant
EVControlSystem initialized successfully.
Publisher initialized. Waiting for CB to warn...
Warning message sent:  : ID=4916OE, Origin=1, Destination=2, Priority level=1

>>> Subscriber disconnected <<<
[Battery] Current Battery Voltage: 1.9875 V
^C
Signal 2 received, stopping EVControlSystem...
[Battery] Current Battery Voltage: 1.98731 V
rmmod: ERROR: Module pwm is in use
Module ALREADY removed
Publisher closed.
# ps aux | grep -E "(evcontrolsystem)"
  752 root     grep -E (evcontrolsystem)
#
mariana@panda:~$ ssh root@raspAndre.local
root@raspandre.local's password:
# cd /tmp/CLion/debug/
# ./Subscriber
Subscriber running. Press Ctrl+C to stop.
Subscriber matched.
Message RECEIVED: ID=4916OE, Origin=1, Destination=2, Priority Level=1, Sent at:
00:13:14
^C
Signal 2 received, stopping Subscriber...
Subscriber finished.
# cd /etc
# cat wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
country=PT

network={
  ssid="wifimari"
  psk="wifimari"
  key_mgmt=WPA-PSK
}

network={
  ssid="IphoneMari"
  psk="mari2427"
  key_mgmt=WPA-PSK
}
```



Emergency Vehicle Control Box

Hardware Implementation





Emergency Vehicle Control Box

Test Cases

Test Case	Test Description	Expected Result	Real Result
Login in cloud	Verify authentication with valid credentials	System returns success, user authenticated and connected to Cloud	Success message sent from cloud and controlbox logged in
Logout off cloud	Disconnect user from cloud system	Session terminated successfully, Cloud disconnected	Control box logged out of cloud
ADS1115 Warning Activation	Battery voltage drops below threshold	Blinking LED turns on	Led blinked
ADS1115 Warning Deactivation	Battery voltage returns to normal levels	Blinking LED turns off	Led is off
ADS1115 Voltage Reading	Read battery voltage from ADS1115 sensor	Correct voltage reading	Battery voltage read correctly
Network Connection to Control Box AP check	Verify network connectivity status	Emergency Vehicle has IP address	Emergency vehicle connected to access point from intersection control box
DDS Message Publishing	Publish MessageData through DDSPublisher	Message published to topic successfully, timestamp and senderID correct	Emergency message sent to intersection control box
LED Blink Control	Control LED blinking through LedBlinkDriver	LED blinks at specified rate, ioctl commands executed successfully	Led blinked at specified rate

The AllWays Safe

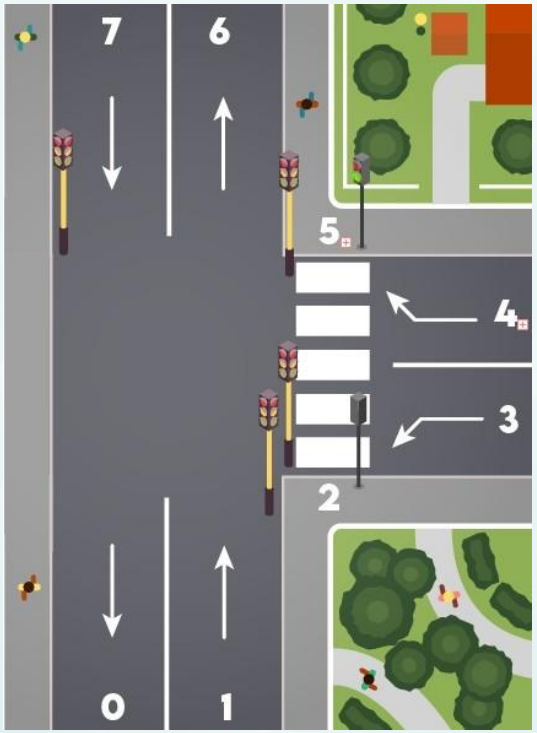
Intersection Control Box Solution

- Cloud-Bootstrapped Traffic System;
- Autonomously generated Traffic Orchestration;
- Traffic/Pedestrian Semaphores Control;
- Handle Pedestrian Input;
- Real-Time Cloud updates;
- Communication with Emergency Vehicles.

Intersection Control Box

CENTROALGORITMI

Cloud-Bootstrapped Traffic System



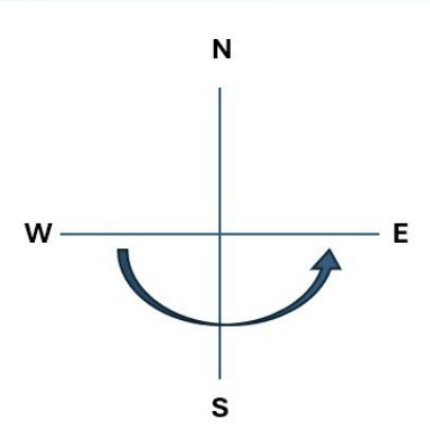
Intersection Control Box

Cloud-Bootstrapped Traffic System

Locations are integer numbers which identify a position of any existing route in an intersection.

Destinations can be defined for each Traffic Semaphore: these are the locations, in the intersection, to where vehicles can go from each traffic semaphore.

The criteria of starting from **West to East** in **counter-clockwise direction**.



Intersection Control Box

CENTROALGORITMI

Cloud-Bootstrapped Traffic System

Semaphore	Location (L)	Destination (D)
TSEM0	0	2, 5
TSEM3	3	9
TSEM6	6	7
TSEM8	8	2, 9
PSEM1	1	–
PSEM4	4	–

Number	Combinations
0	(TSEM0, TSEM6)
1	(TSEM6, TSEM8)
2	(TSEM6, PSEM1, PSEM4)
3	(TSEM6, TSEM3)

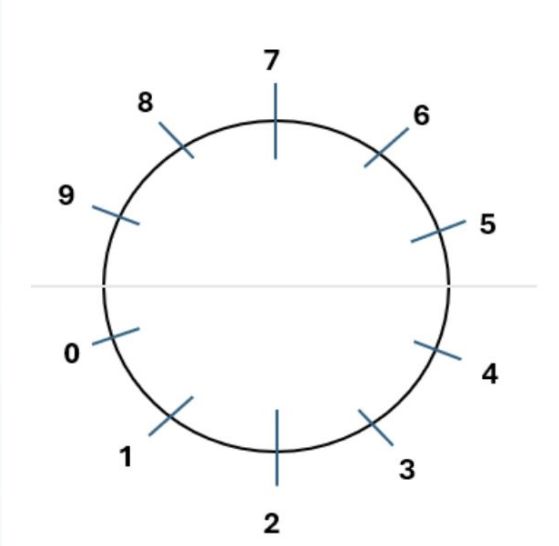
Intersection Control Box

Cloud-Bootstrapped Traffic System

The only remaining criteria to establish is the **algorithm that ensures if there is trajectory conflict or not**.

A **trajectory** is a possible path that can be done from a location to another location.
There is **Trajectory Conflict** when any trajectory from one semaphore **crosses** another trajectory from another semaphore

For this, a circular approach is used.



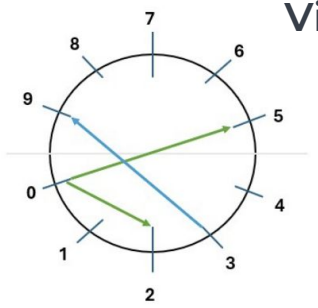
Intersection Control Box

CENTROALGORITMI

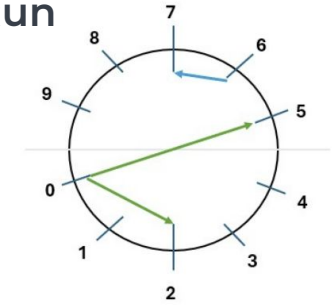
Cloud-Bootstrapped Traffic System

Number	Combinations
0	(TSEM0, TSEM6)
1	(TSEM6, TSEM8)
2	(TSEM6, PSEM1, PSEM4)
3	(TSEM6, TSEM3)

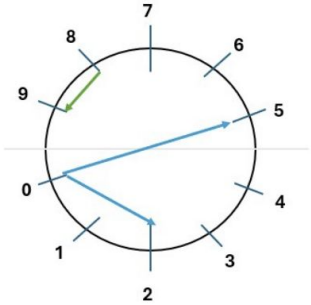
Visual Dry Run



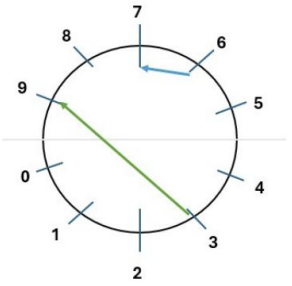
(a) Validation between Semaphore at Location 0 and Semaphore at Location 3: trajectories conflict



(b) Validation between Semaphore at Location 0 and Semaphore at Location 6: trajectories do not conflict



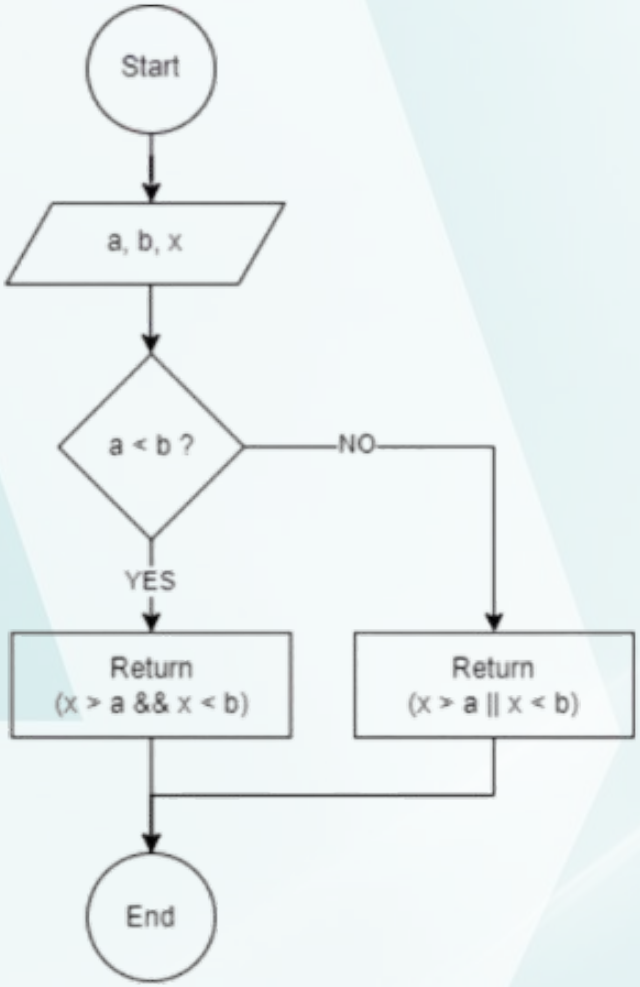
(c) Validation between Semaphore at Location 0 and Semaphore at Location 8: trajectories do not conflict



(d) Validation between Semaphore at Location 3 and Semaphore at Location 6: trajectories do not conflict

Intersection Control Box

Cloud-Bootstrapped Traffic System




```
bool TrafficControlSystem::isBetween
(const int a, const int b, const int x)
{
    if (a < b)
        return x > a && x < b;
    return x > a || x < b;
}
```

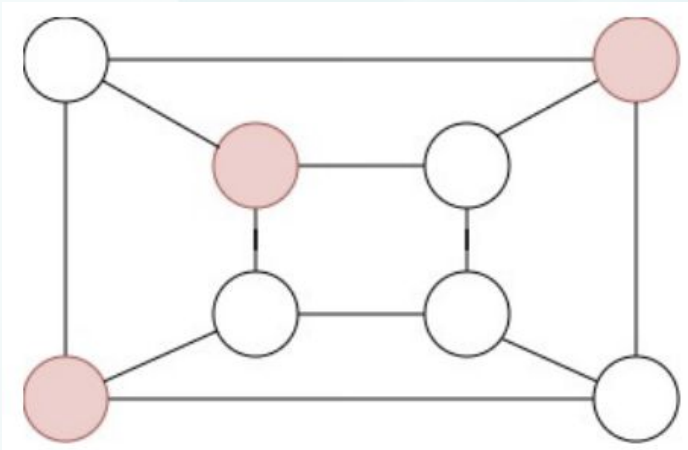
Intersection Control Box

Cloud-Bootstrapped Traffic System

With the criteria established, there was an emerging need of knowing how the configuration sets could be formed. But, there are relevant characteristics with which the **desired algorithm must comply**: it must **generate** the **minimum number of configurations** with the **maximal** number of semaphores possible.

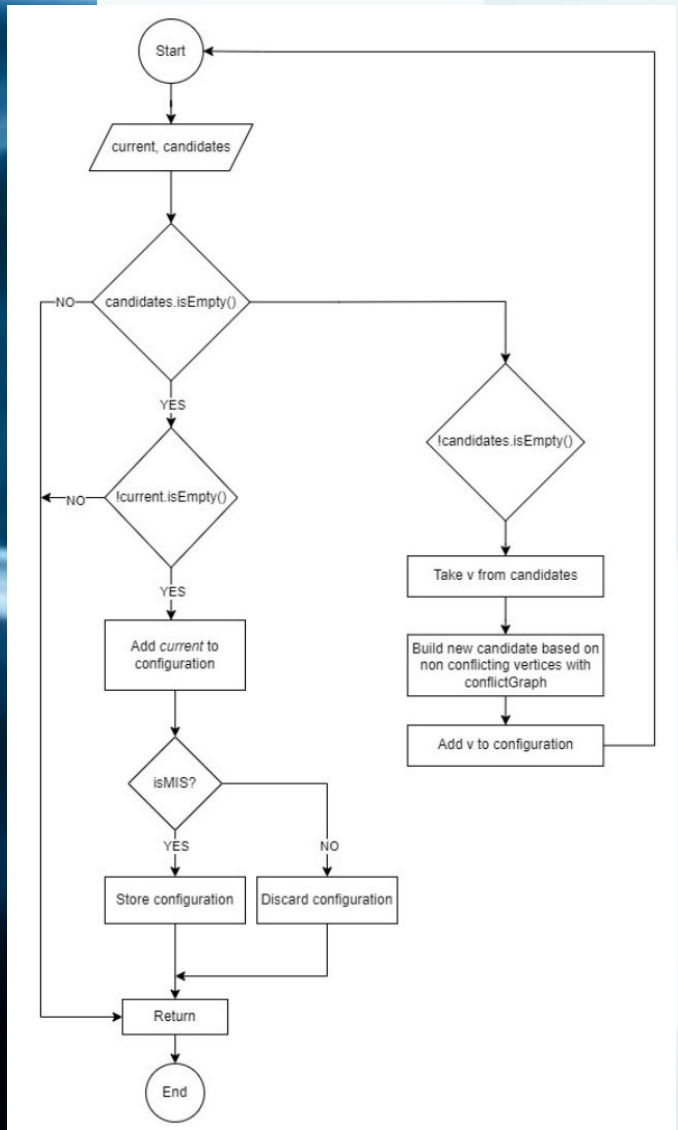
This problem is formulated under **graph theory** as the **Maximal Independent Set**, considering an **undirected graph**.

The conflicting vertices (red) are the semaphores whose trajectories conflict, whereas the others (white) don't have conflicting trajectories



Intersection Control Box

Cloud-Bootstrapped Traffic System



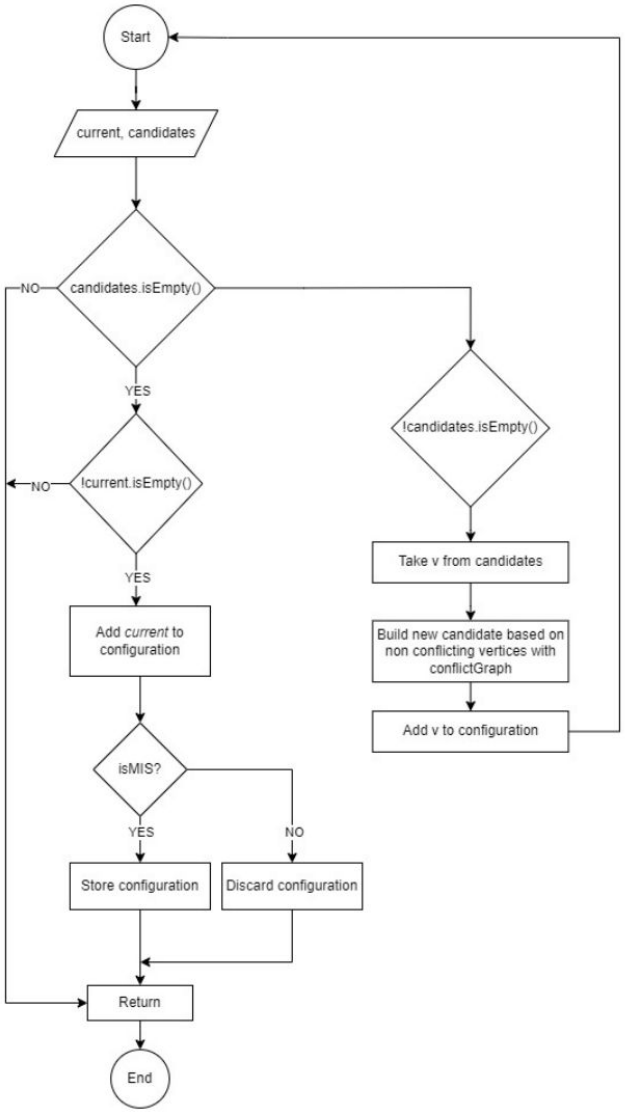
This reasoning was iterated through all semaphores and

- if an intersection is detected, then that semaphore is ignored for the current configuration and that branch is ignored.
- If there is no conflict, the algorithm should proceed to evaluate if there a viable configuration for that set of chosen semaphores (**recursive** iteration).

This approach is called ***Backtracking with prunning***, whose best case complexity is *O(N)* and worst case is *O(2^N)*. Therefore, is presents a better approach than brute-force backtracking.

Intersection Control Box

Cloud-Bootstrapped Traffic System

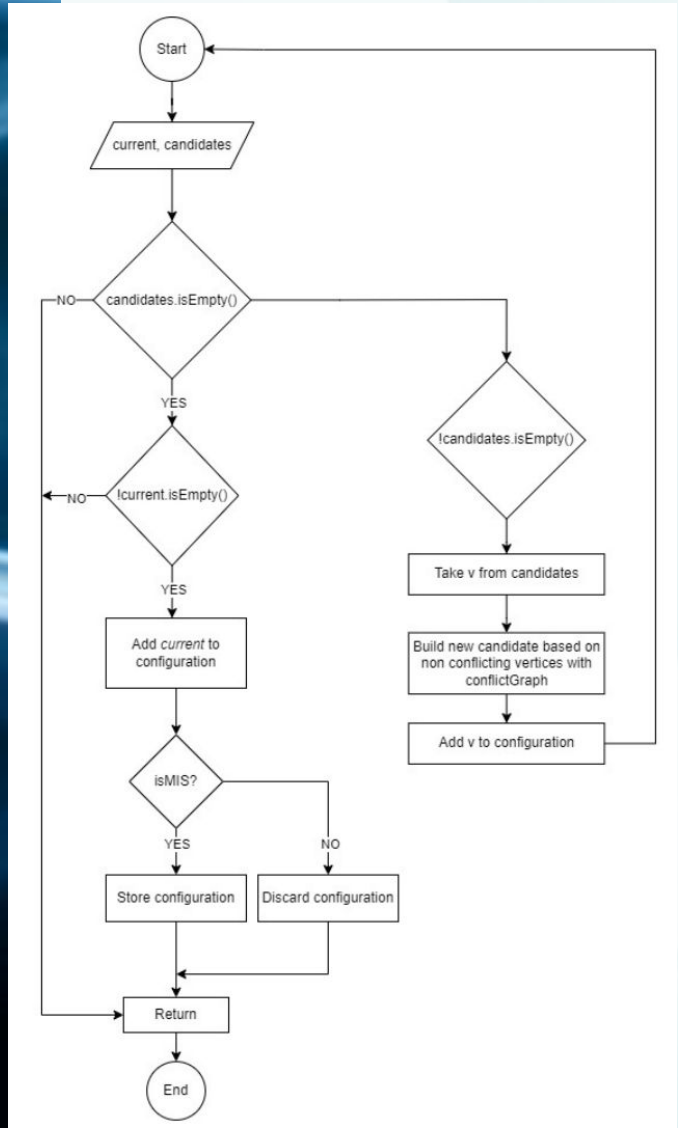



```
void TrafficControlSystem::backtrack(std::vector<int>& current, std::vector<int>&
candidates)
{
   if (candidates.empty()) {       // Last iteration of each configuration
       if (!current.empty()) {
           Configuration cfg;
           for (const int loc : current)
           {
              /* SIMPLIFIED: STORE ELEMENT IN CURRENT CONFIGURATION */
           }

           // Check if it is a maximal independent set - local validation:
           //      if any of the other locations (vertices) are compatible with
this config
           bool maximal = true;
           for (int v : vertices) {
               if (std::find(current.begin(), current.end(), v) == current.end()) {
                   bool ok = true;
                   for (int u : current)
                       if (conflictGraph[v][u]) {
                           ok = false;
                           break;
                       }
                   if (ok) {
                       maximal = false;
                       break;
                   }
               }
           }

           if (maximal)
               configurations.push_back(cfg);
       }
       return;
   }
```

Intersection Control Box

CENTROALGORITMI

Cloud-Bootstrapped Traffic System




```
  while (!candidates.empty()) {
        int v = candidates.back();
        candidates.pop_back();

        // New subset of compatible
candidates - next recursive iteration
        std::vector<int>
newCandidates;
        for (int u : candidates) {
            if (!conflictGraph[v][u])
// pruning

newCandidates.push_back(u);
        }

        current.push_back(v);
        backtrack(current,
newCandidates);
        current.pop_back();
    }
}
```

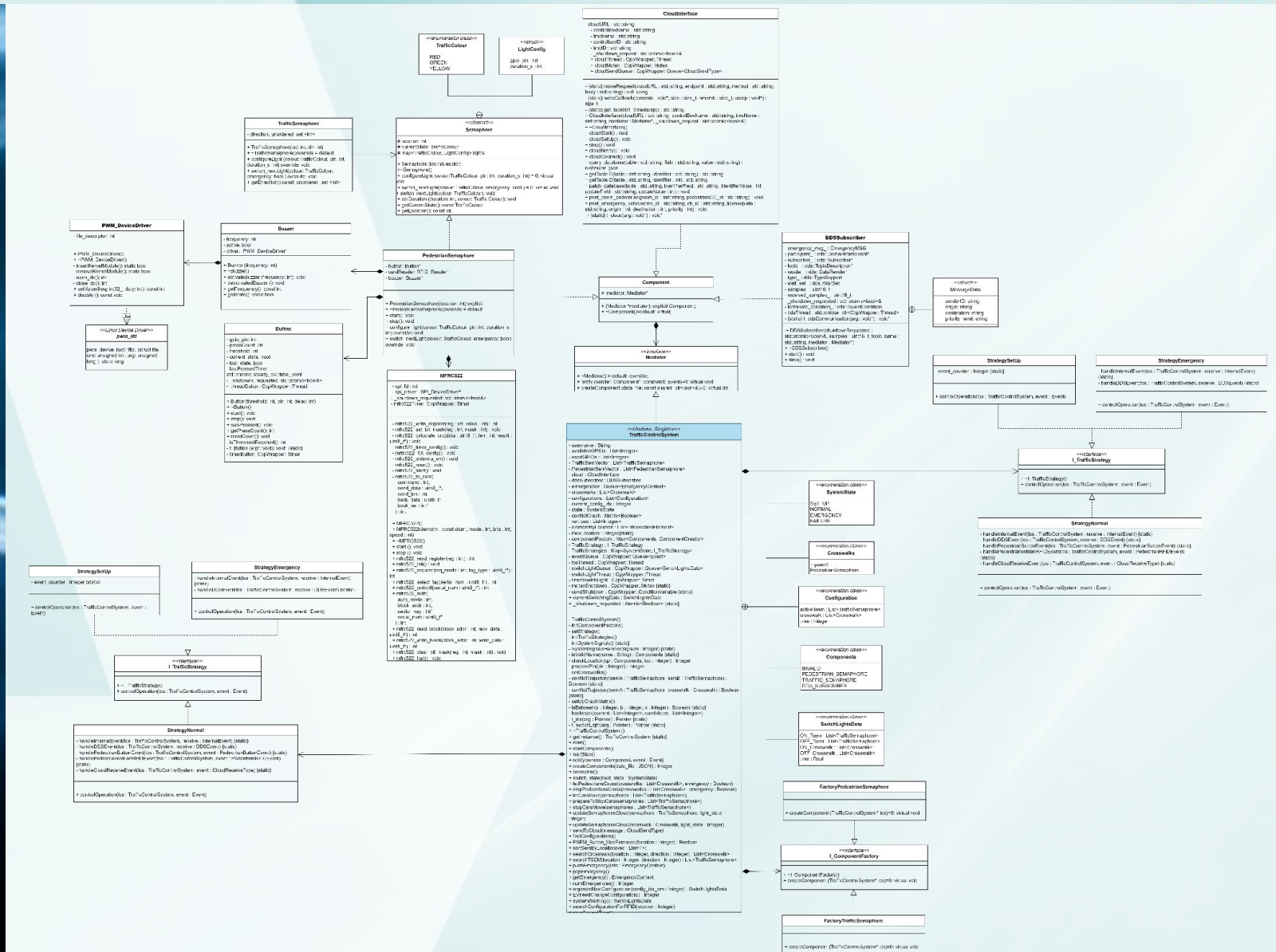
Simplified Backtracking Algorithm, *Pt.2* - Final

ESRG

Embedded Systems Research Group

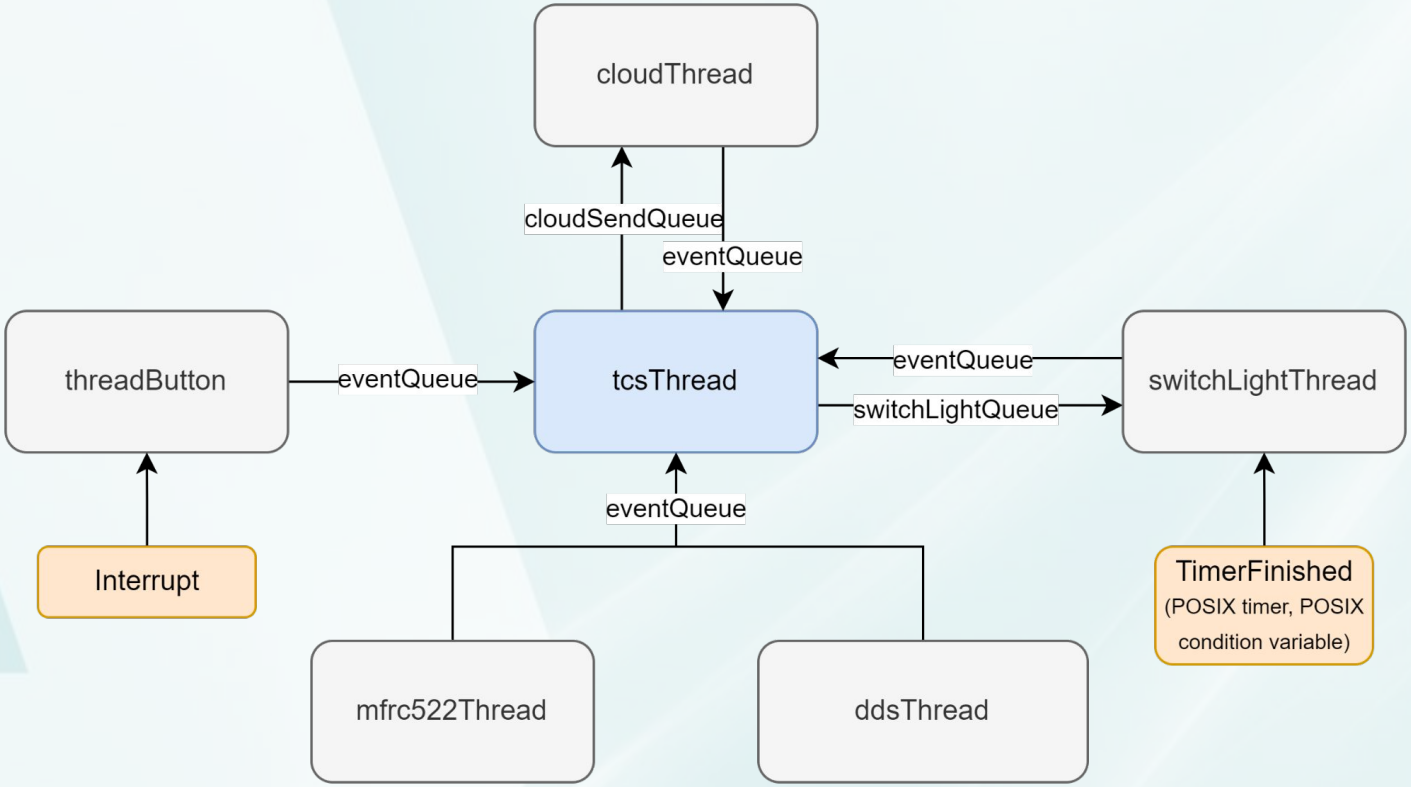
Intersection Control Box

CENTROALGORITMI



Intersection Control Box

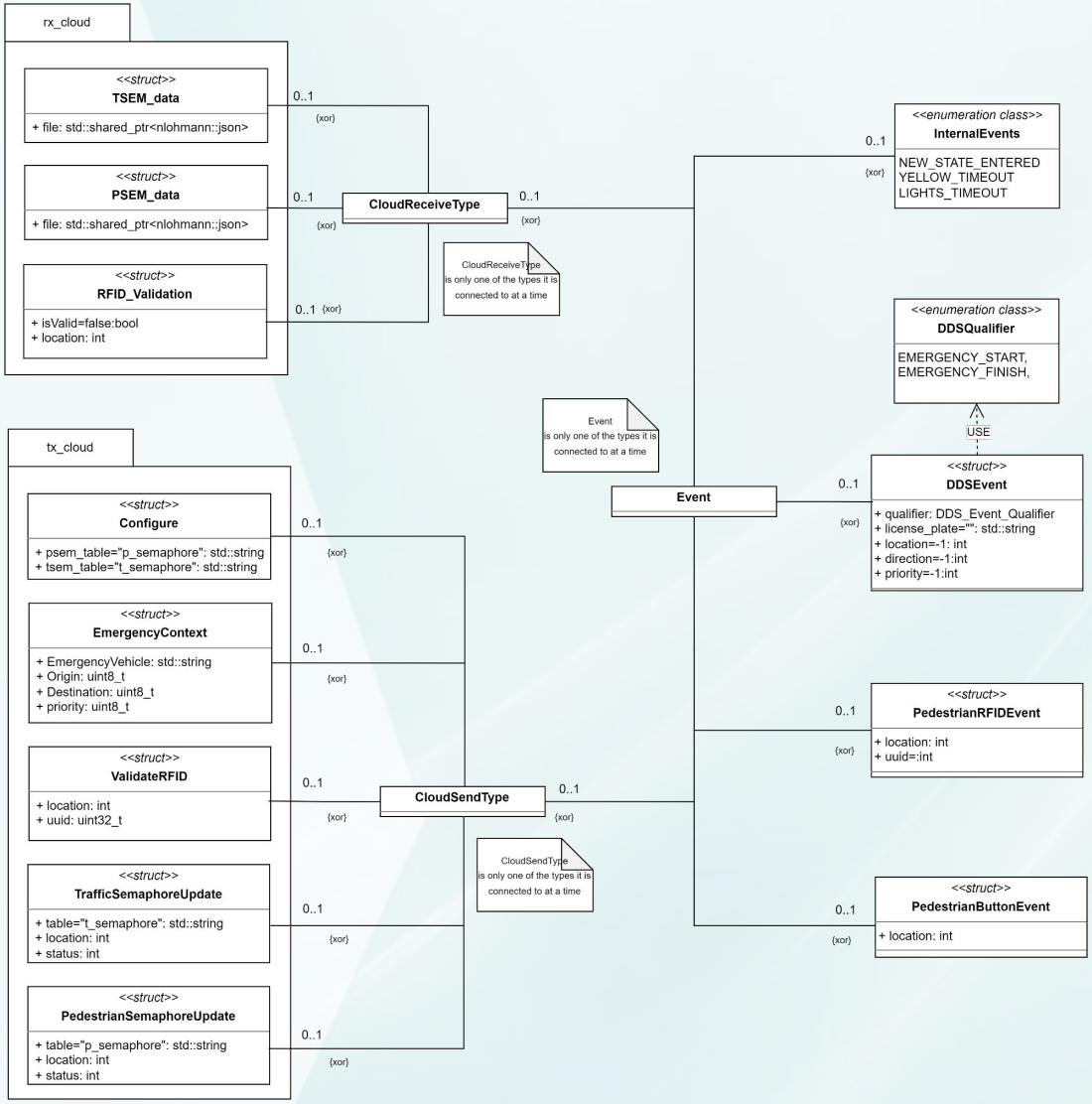
Task Overview



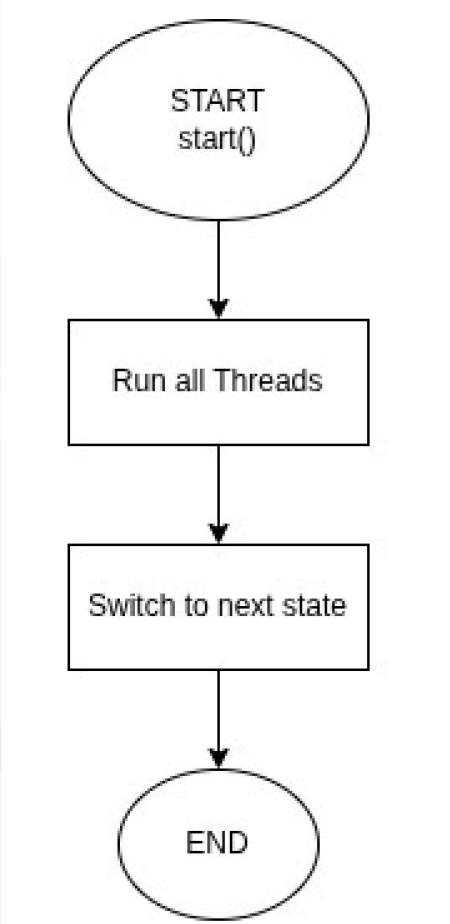
Intersection Control Box

CENTROALGORITMI

System Events



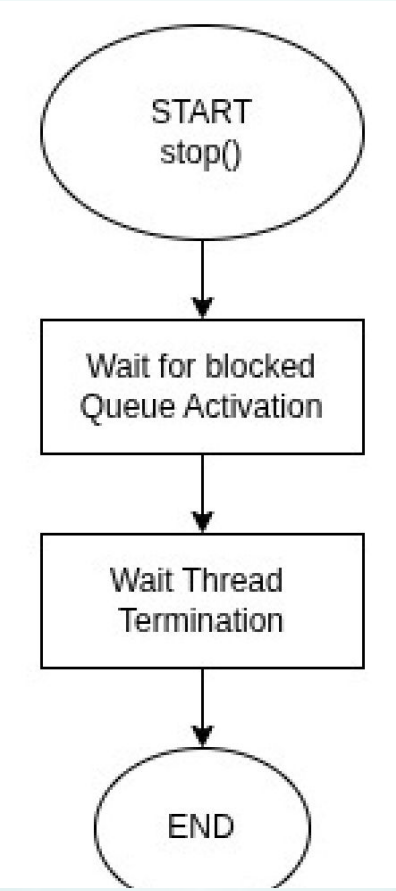
Intersection Control Box




```
void TrafficControlSystem::start()
{
    // Start threads
    tcsThread.run(this);
    switchLightThread.run(this);

    cloud.cloudStart();
    ddsSubscriber.start();

    switch_state(state);
}
```

Intersection Control Box


```
void TrafficControlSystem::waitStop()
{
    switchLightQueue.interrupt();
    eventQueue.interrupt();

    ddsSubscriber.stop();
    cloud.stop();

    for (auto& psem: PedestrianSemVector)
        psem->stop();

    switchLightThread.join();
    tcsThread.join();
}
```



Intersection Control Box

Control Box (TrafficControlSystem) Initialization - Constructor

```
TrafficControlSystem::TrafficControlSystem():
    username("raspMari.local"),
    cloud("http://192.168.1.185:3000", username, "tmc1", this,
_shutdown_requested),
    tcsThread(t_tcs),
    switchLightThread(t_switchLight),
    ddsSubscriber(_shutdown_requested, 0, "EmergencyAlert", this)
{
    state = SystemState::SET_UP;
    current_config_idx = 0;
    availableGPIOs = {1, 2, 3, 4, 5, 6, 7, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27};//22 total

    initComponentFactory();
    initTrafficStrategies();
    initSystemSignals();
}
```

Intersection Control Box

Design Patterns

- Singleton (*Meyers* Singleton)
- Mediator
- Strategy
- Factory
- Facade

Software Architectural Patterns

- Event-Driven Architecture (EDA) → leads to an Event-Driven Mediator/ Asynchronous Mediator

Concurrency Patterns

- Producer-Consumer

Intersection Control Box

Design Patterns

Singleton → Ensure only one instance of that class is ever created: one one TrafficControlSystem;

Facade → When interfacing with a system, simplicity is a pivotal. Therefore, an interface that hides all the program details is preferred. Thorough details are hidden by a simple interface;

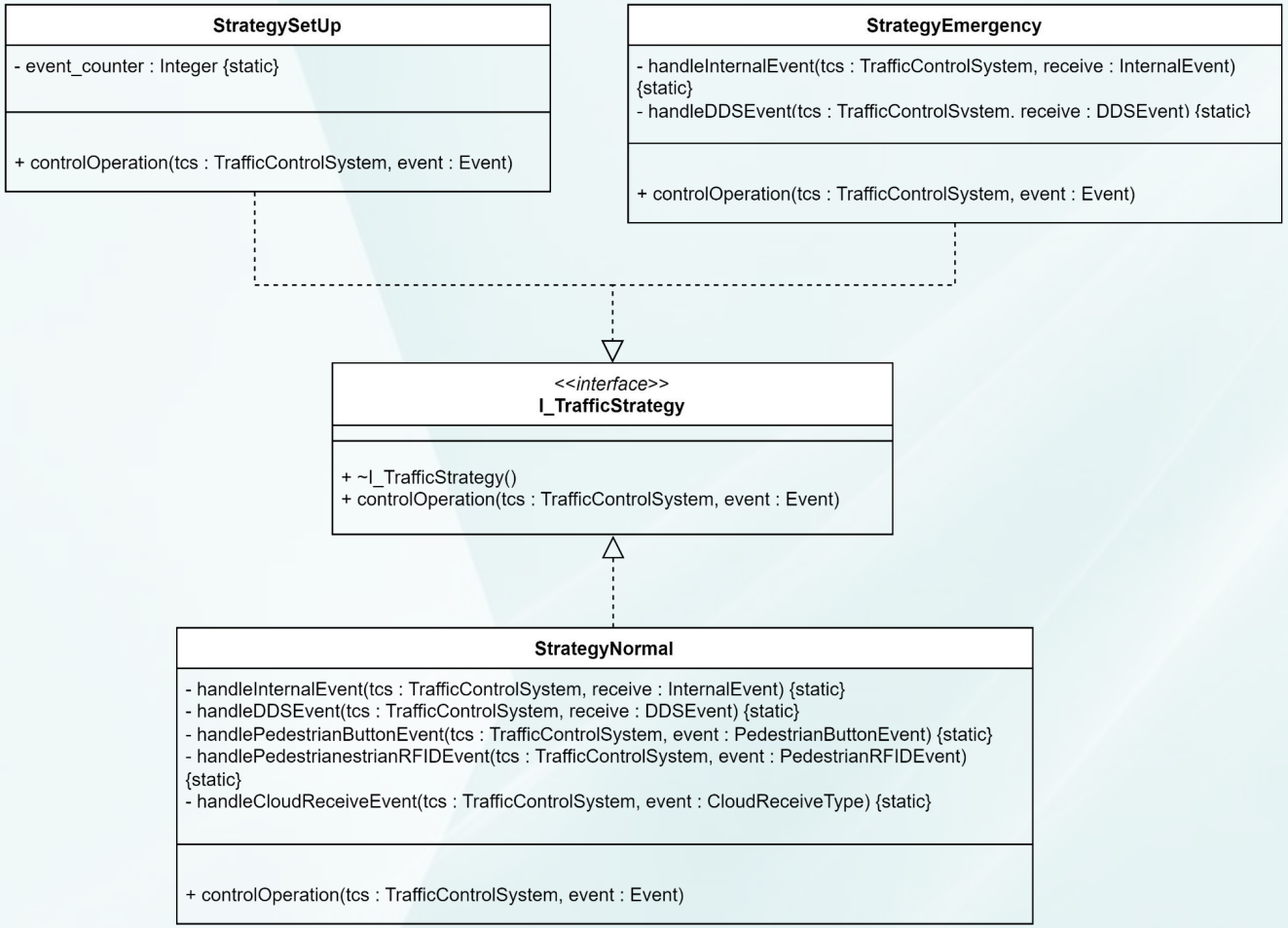
Factory → Considering that the system should be able of automatically creating its components and their attributes it requires an efficient way of creating these components *on-the-fly*. The Factory design pattern, provided by the TrafficControlSystem, enables such structure by providing a **dedicated interface for components creation**, which acts differently based on the component.

```
void TrafficControlSystem::initComponentFactory()
{

   componentFactory[Components::PEDESTRIAN_SEMAPHORE] =
    [this](const json& data) -> int
       {
            /* Implementation */
       };

   componentFactory[Components::TRAFFIC_SEMAPHORE]=
       [this](const json& data)
       {
            /* Implementation */
       };
```

Intersection Control Box

Design Patterns

Strategy → confer the system different strategies to deal with the system events, based on the system's current state



Intersection Control Box

Design Patterns

Strategy → confer the system different strategies to deal with the system events, based on the system's current state

```
void TrafficControlSystem::setStrategy()
{
    TrafficStrategy = TrafficStrategies[state].get();
}
```

```
void StrategyEmergency::controlOperation(TrafficControlSystem* tcs,
    Event& event)
{

    auto visitor = [tcs](auto&& receive)
    {
        using T = std::decay_t<decltype(receive)>;

        if constexpr (std::is_same_v<T, InternalEvent>)
            handleInternalEvent(tcs, receive);
        else if constexpr (std::is_same_v<T, DDSEvent>)
            handleDDSEvent(tcs, receive);
    };

    std::visit(visitor, event);
}
```

Intersection Control Box

Mediator → provide system orchestrator, via Mediator and Component Interface

```
class Mediator;

class Component
{
protected:
   Mediator *mediator;
public:
   explicit Component(Mediator
*mediator);
   virtual ~Component()=default;
};

class Mediator
{
public:
   virtual ~Mediator() = default;
   virtual void notify (Component*
sender, Event command)=0;
   virtual int createComponents
(const std::shared_ptr<json>&
data_file)=0;
};
```

```
1 class PedestrianSemaphore :
    public Semaphore, public
    Component {
2    /*...*/
3 };
```

Listing 4.24: PedestrianSemaphore component

```
1 class CloudInterface: public
    Component {
2    /*...*/
3 };
```

Listing 4.25: CloudInterface component

```
1 class DDSSubscriber: public dds
    ::DataReaderListener, public
    Component {
2    /*...*/
3 };
```

Listing 4.26: DDSSubscriber component

```
1 class TrafficControlSystem:
    public Mediator {
2    /*...*/
3 };
```

Listing 4.27: TrafficControlSystem Mediator

The *notify* method is used by all components when they want to notification the system about an event — mediator->notify(this, event) → Event Driven Architecture

```
void TrafficControlSystem::notify (Component* sender, Event event)
{
    std::cout << "TrafficSystem this: " << this << std::endl;
    eventQueue.send(std::move(event));
}
```

Intersection Control Box

Mediator → provide system orchestrator, via Mediator and Component Interface

Producer-Consumer

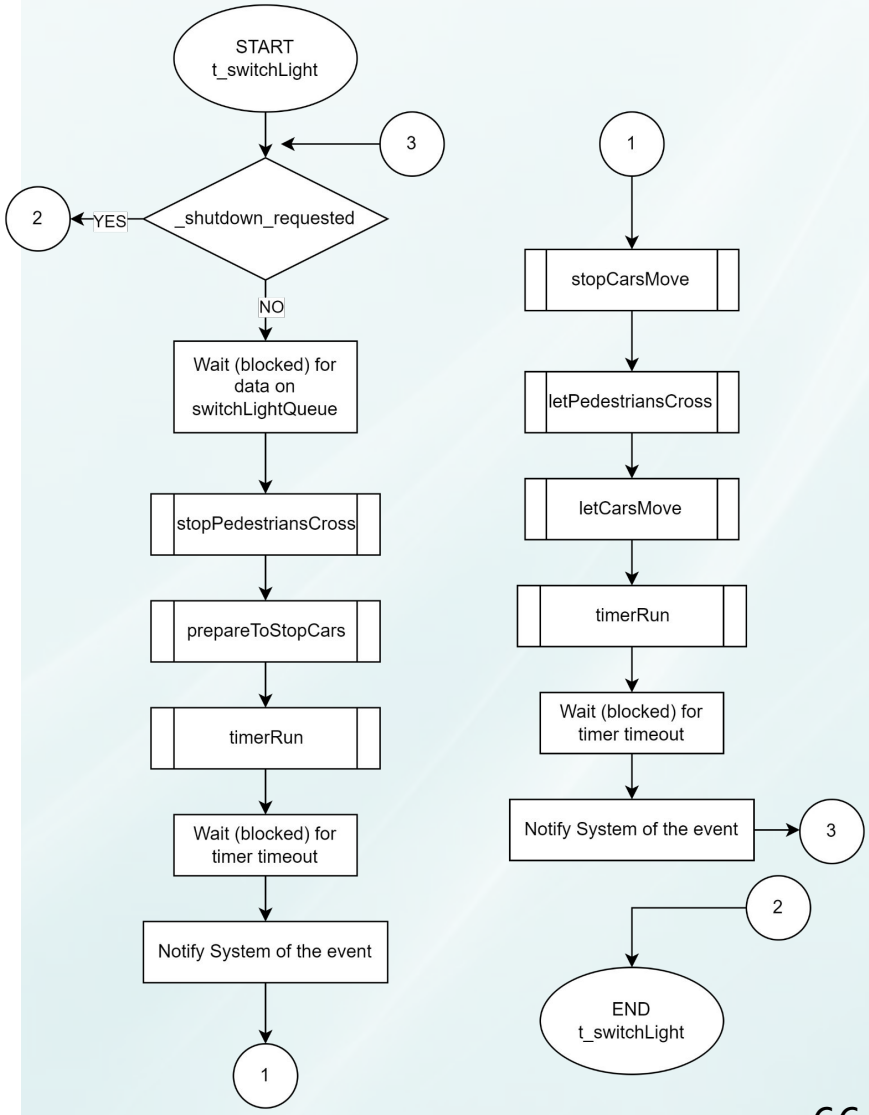
```
void TrafficControlSystem::consumer()
{
    Event data = eventQueue.receive();
    TrafficStrategy->controlOperation(this, data);
}
```

```
void* TrafficControlSystem::t_tcs(void* arg)
{
    const auto self = static_cast<TrafficControlSystem*>(arg);

    while (!_shutdown_requested.load())
    {
        self->consumer();
    }
    return arg;
}
```

Intersection Control Box

Switching Lights


```
void* TrafficControlSystem::t_switchLight(void* arg)
{
    auto self = static_cast<TrafficControlSystem*>(arg);

    SwitchLightsData&  switchingData = self->currentSwitchingData;
    while (!_shutdown_requested.load())
    {
        // Wait for switching data to be ready
        switchingData = self->switchLightQueue.receive();

        // Change Semaphores
        std::cerr << "PSEM OFF \n";
        self->stopPedestriansCross(switchingData.OFF_Crosswalk, false);

        std::cerr << "YELLOW \n";
        self->prepareToStopCars(switchingData.OFF_Tsem);

        self->timerSwitchLight.timerRun(YELLOW_DURATION);
        self->timerSwitchLight.timerWait();

        self->notify(nullptr, InternalEvent::YELLOW_TIMEOUT);

        self->stopCarsMove(switchingData.OFF_Tsem);

        self->letPedestriansCross(switchingData.ON_Crosswalk, false);
        self->letCarsMove(switchingData.ON_Tsem);

        std::cerr<<"GREEN: config "<< self->current_config_idx <<"  \n";

        self->timerSwitchLight.timerRun(switchingData.time);
        self->timerSwitchLight.timerWait();

        // Notify the system itself
        self->notify(nullptr, InternalEvent::LIGHTS_TIMEOUT);
        std::cerr<<"RED\n";
    }
    return arg;
}
```

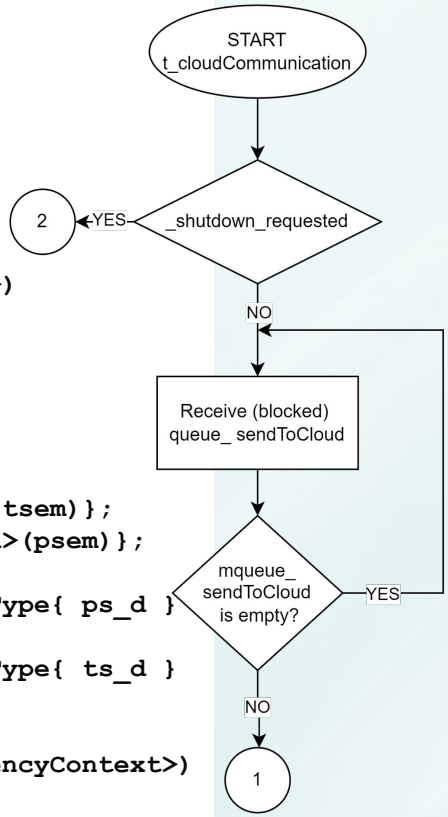


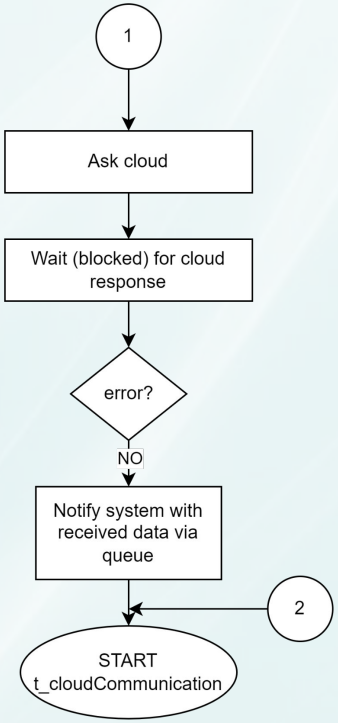
Intersection Control Box

CENTROALGORITMI

```
void* CloudInterface::t_cloud(void* arg)
{
   auto self = static_cast<CloudInterface*>(arg);

   self->cloudConnect();
   self->cloudSetUp();

   while (!self->_shutdown_request.load())
   {
       auto data = self->cloudSendQueue.receive();

       auto visitor = [&] (auto&& obj)
       {
           using T = std::decay_t<decltype(obj)>;

           if  constexpr (std::is_same_v<T, tx_cloud::Configure>)
           {
               json psem = self->query_database(obj.psem_table,
CONTROL_BOX_FK, self->controlboxID);
               json tsem = self->query_database(obj.tsem_table,
CONTROL_BOX_FK, self->controlboxID);

               rx_cloud::TSEM_data ts_d={std::make_shared<json>(tsem)};
               rx_cloud::PSEM_data ps_d = {std::make_shared<json>(psem)};

               self->mediator->notify(self, Event{ CloudReceiveType{ ps_d }
});
               self->mediator->notify(self, Event{ CloudReceiveType{ ts_d }
});
           }
           else if constexpr (std::is_same_v<T, tx_cloud::EmergencyContext>)
           {
               self->post_emergency_vehicle (self->tmcID,
self->controlboxID, obj.EmVehicleID,
                   obj.Origin, obj.Destination, obj.priority);
/* Continuation */
```

Cloud Communication





Intersection Control Box

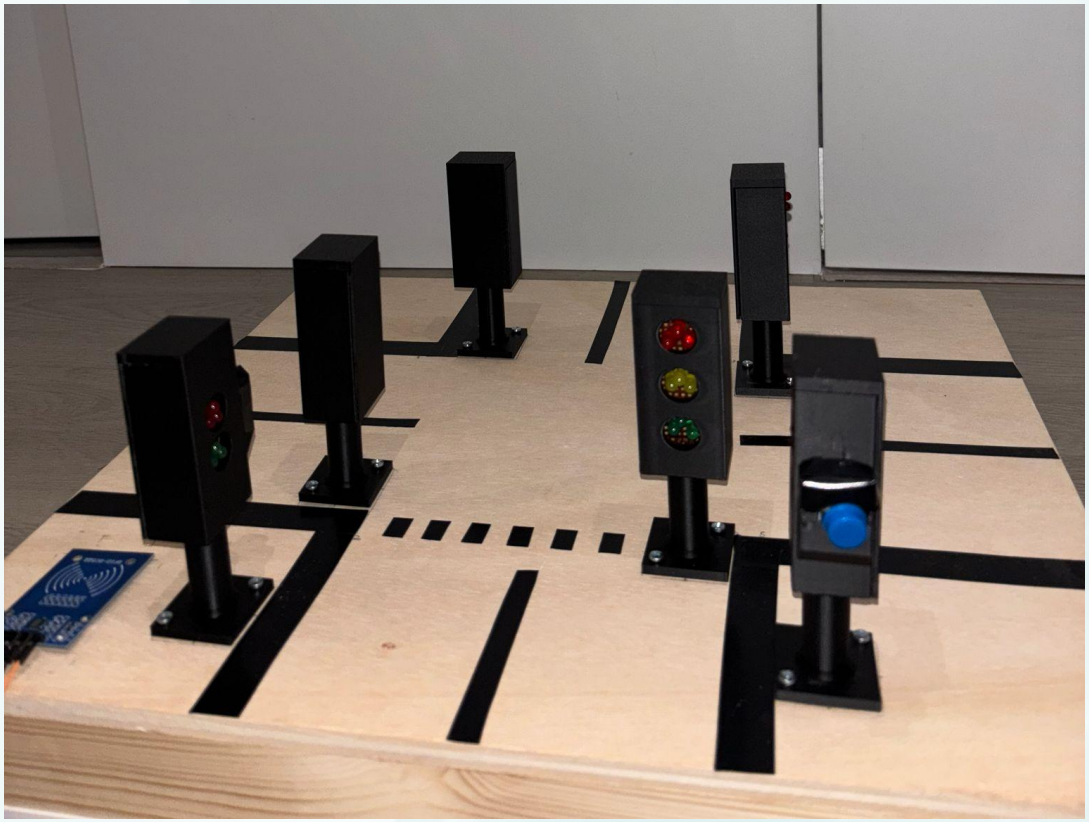
Test Cases

CENTROALGORITMI

Test Case	Test Description	Expected Result	Real Result
Login in cloud	Verify authentication with valid credentials	System returns success, user authenticated and connected to Cloud	Success
Button Responsiveness	Test if buttons respond to user input	Press event detected (pressCount incremented) and sent to system	Success
Semaphore State Change	Change semaphore state from one colour to another	Traffic/Pedestrian Light changes to another colour, duration respected, state updated	Success
RFID Card Reading	Bring valid RFID card close to reader	RFID_Reader detects card, reads CardID and returns data correctly	Success
Buzzer Activation	Pedestrian semaphore turns GREEN	BuzzerDriver activates buzzer, audible sound emitted (with a specified frequency rate, ioctl commands executed successfully)	Success
Traffic/Pedestrian Light Duration	Configure specific duration for green light (e.g., 30s)	Semaphore maintains GREEN for exactly 30 seconds before changing	Success
Invalid Login Attempt	Attempt login with invalid credentials	System rejects authentication, returns error, access denied	Success
DDS Responsiveness	Receive message from DDS communication	Receive message with emergency vehicle ID and direction	Success
Emergency Mode Activation	Activate System Mode EMERGENCY_ACTIVE	Semaphores change to emergency state, normal operation suspended	Success
DDS Subscriber creation	Access to DDS topic	Receive message from correct topic	Success
Extended Passage	Person with disability passed card on RFID	Extend time on that configuration, on the next time it happens	Success
Early Passage	Count of press buttons greater than threshold in a few seconds	Pedestrian light turns GREEN before its time	Success
Pin Association through cloud	Cloud attributes a pin for semaphore light	Pin attributed is available for the effect and is configured with the right settings	Success
Invalid RFID cardID	Person without disability scans card on RFID module	GREEN light is not extended	Success
Configuration Set Up on boot	Get data from cloud and form the correct configurations	Data is retrieved and configurations are performed effectively	Success

Intersection Control Box

Hardware Implementation



Practical Demonstration

Limitations

- Ethernet Connection;
- On demonstration, the connection time required for the emergency vehicle, since *boot*.

Future Work

- Accept different priorities from emergency vehicles;
- Integrate other vehicles (enable full V2X);
- Integrate IEEE norms;
- Integrate 5G technology.

CENTROALGORITMI

QUESTIONS?