Software Engineering Final Project –

Parking Drone

Design Document



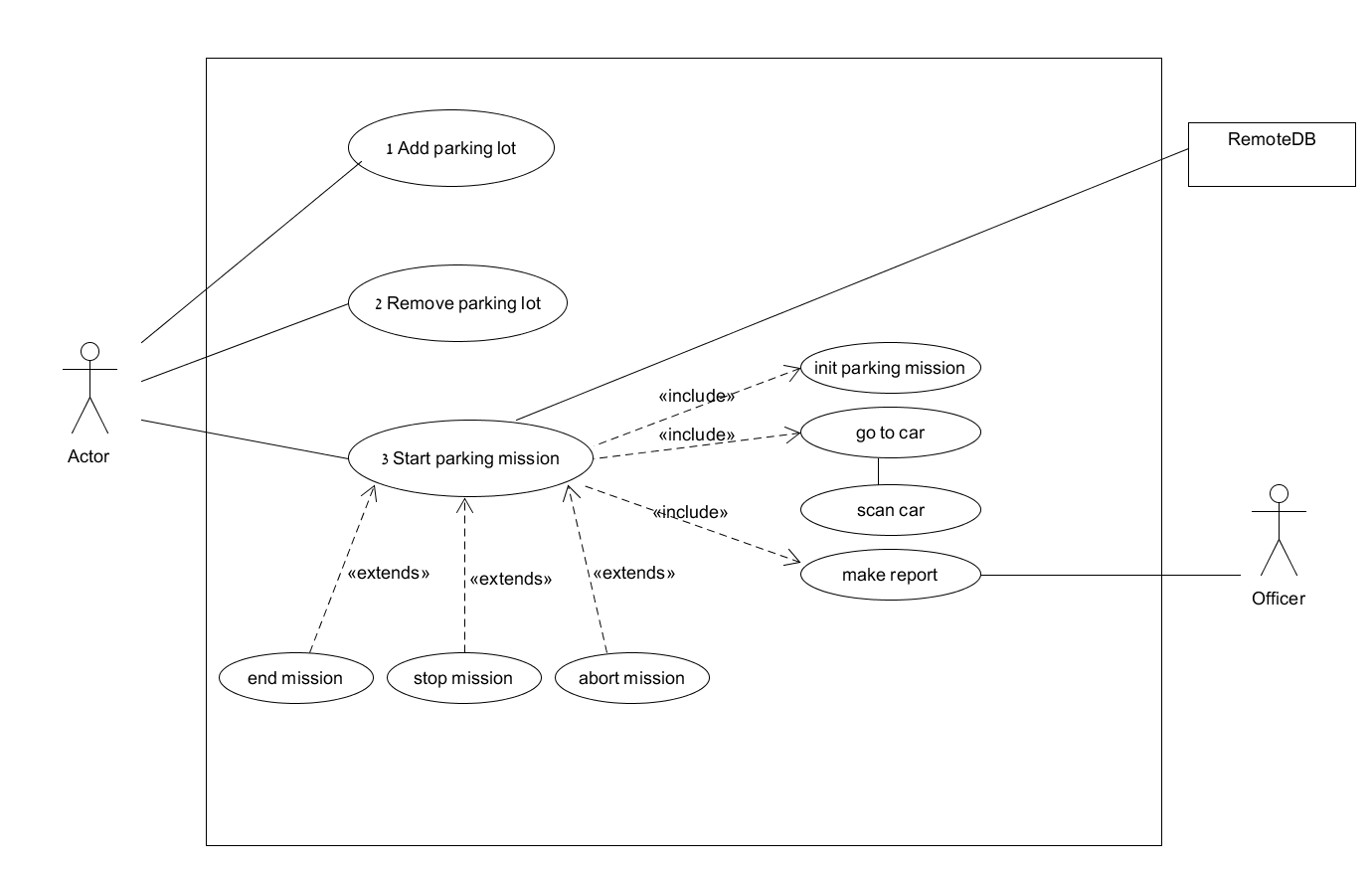
**Chapter 1- Use Cases:**

* 1. User Profiles –

Warden: a parking supervisor, authorized by the local enforcer, will use the system in order to detect unauthorized cars who park within the parking lot.

Official: Analyze the system output after each task.

Remote Database – Given by the owner of the parking lot, contains information about the permitted license plates.

1.2 Use cases –

1. Add Parking Lot

Description:

Warden adds parking spot.

Pre-Conditions:

Internet connection – in order to use google maps.

Google maps key – positive balance in google maps account.

Post-Conditions:

A new parking lot is added to the data base.

Main Success Scenario:

1. Warden press on create parking spot button.
2. Warden enters parking spot name.
3. Warden navigate to desired location in the map.
4. Warden marks the parking borders.
5. Warden press finish button.

Alternatives:

5.1 Parking spot name already exists.

5.2 The system alerts the Warden that scenario has failed.

1. Remove Parking Lot

Description:

Warden remove parking spot.

Pre-Conditions:

At least one parking spot exists.

Post-Conditions:

The desired parking spot has been removed from the data base.

Main Success Scenario:

1. Warden press remove parking spot button.
2. Warden picks a parking spot from a list.
3. Warden press on remove button.
4. Start Parking Mission

Description:

Warden choose a parking lot to scan, the system scans the desired parking lot and produce a report.

Pre-Conditions:

Internet connection – in order to use google maps.

Google maps key – positive balance in google maps account.

Drone connection – successful connection to the drone.

Drone calibration – sensor of the drone must be calibrated.

Drone Batteries – batteries must be charged.

Controller Batteries - batteries must be charged.

Bridge – Bridge software is running and communication with the server has been established.

DJI KEY – Bridge has successfully registered and authorized by DJI.

Visibility – daylight.

Remote DB – The server established a connection with the remote DB.

Post-Conditions:

Report has been sent to the official.

Main Success Scenario:

1. Warden selects parking lot.
2. Warden clicks on start.
3. The system pulls data from the remote DB.
4. The system will scan the parking lot for cars and calculate a course (UC 3.1).
5. The drone will get to the front of each car in the parking lot (UC 3.2).
6. The drone will scan the number plate of the cars (UC 3.3).
7. The system sends report to the Official and notifies the warden that mission ended (UC 3.4).

Extensions\Alternatives:

Warden clicks stop mission (UC 3.5).

Warden clicks end mission (UC 3.6).

Warden clicks abort mission. (UC 3.7).

At any point if error accrued the System will abort in a safe way. The warden will be notified.

* 1. Init parking mission

Description:

The drone will move to the middle of the parking spot, take a photo and analyse it in order to Find the cars in the parking lot.

Pre-Conditions:

GPS – connected and calibrated.

Camera – a high end camera is connected.

Drone – the drone has already taken off.

Post-Conditions:

Have a photo of the parking lot.

Main Success Scenario:

1. The mission manage will calculate the GPS coordinate the center of the parking lot based on the parking lot details.
2. The drone starts a way point mission to the desired GPS location.
3. The drone will rotate his gimbal’s camera until the lens of the camera will be horizontal to the surface.
4. The drone will take a photo and send it to the mission manager.

Extensions\Alternatives:

Inherited from UC 3.

* 1. Scan cars

Description:

The mission will run CV algorithm to detect the location of the cars.

Then the mission will calculate the course between the cars and execute UC. 3.3 multiple times.

Pre-Conditions:

Init parking mission (UC. 3.1) finished successfully.

Post-Conditions:

All cars in the parking lot are scanned.

Main Success Scenario:

1. The mission will run a CV algorithm to detect the location of the cars.
2. The CV algorithm return all location of cars.
3. The mission calculate course between cars.
4. Run UC 3.3 for each car.

Extensions\Alternatives:

3.2.1.a There are no cars in the parking lot.

3.2.1.b The algorithm fail to detect a car in the parking lot.

* 1. Scan single car

Description:

The drone will scan the car and the mission will extract information about the car.

Pre-Conditions:

The drone in the parking lot.

Post-Conditions:

The mission knows the car details.

Main Success Scenario:

1. Execute UC 3.4
2. The drone move back 3 meters.
3. The drone move to 4 meters height.
4. The Drone takes photo and send it to the mission manager.
5. The mission manager use CV algorithm in order to extract car details.

Extensions\Alternatives:

* + 1. The algorithm failed to extract car details.
    2. The Mission manager repeat stage 4, 5 after moving the gimbal.
       1. After 3 retries, the car plate will be declared as non-readable and will be emitted from future report.
  1. Go to car

Description:

The Drone will move to the middle of a specific car.

Pre-Conditions:

The car is present in the parking lot.

Post-Conditions:

The drone is located above the car.

Main Success Scenario:

1. The mission calculates the curse to the car.
2. The drone fly according to the mission instruction.
3. The drone take a photo and send to the server.
4. The mission verify the location according to the photo.

Extensions\Alternatives:

* + 1. The mission detect that the drone is not on the desired location
    2. The mission manager recalculate the curse and repeat stages 3,4
  1. Make report

Description:

Mission manager organize all the car details that was collected and produce a report.

Pre-Conditions:

The DB is connected.

Post-Conditions:

A report with all the unauthorized cars is produced.

Main Success Scenario:

1. The mission manager pulls all authorized car plate from DB.
2. The mission manager determine which cars are parking illegally.
3. The mission manager produces a report on which car should receive a ticket.
   1. Stop mission

Description:

The User press on the stop button at any time during a mission execution. The drone will stop at his current location and communicate only with the controller.

Pre-Conditions:

Drone – the drone has already taken off.

Communication- have communication with the drone.

A Mission is being executed.

Post-Conditions:

The Drone will hang in the air. The only communication with him will come from the controller.

Main Success Scenario:

1. User press the stop button.
2. The drone close the communication with the mission manager.
3. The Drone stop his mission and hover at his current location.
   1. End mission

Description:

The User press on the end mission button at any time during a mission execution. The drone will stop his mission and perform return to his take off position and land there.

Pre-Conditions:

Drone – the drone has already taken off.

Communication- have communication with the drone.

A Mission is being executed.

Post-Conditions:

The Drone will land at the same place he takes off.

Main Success Scenario:

1. User press the stop button.
2. The Drone stop his mission.
3. The Drone navigate by gps point to the takeoff location.
4. The Drone will land.
   1. Abort mission

Description:

The User press on the abort mission button at any time during a mission execution. The drone will stop his mission and land at his current location.

Pre-Conditions:

Drone – the drone has already taken off.

Communication- have communication with the drone.

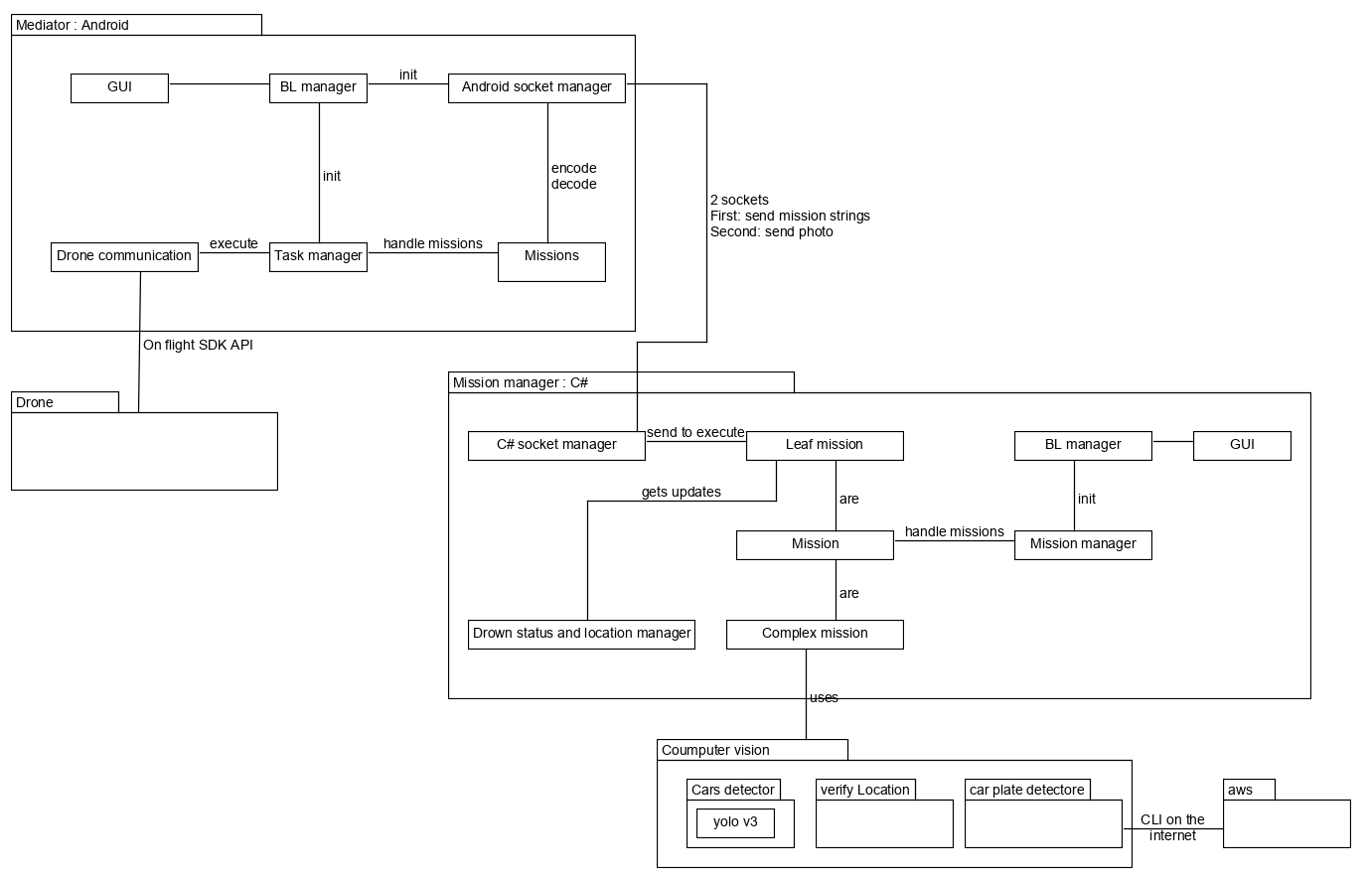
A Mission is being executed.

Post-Conditions:

The Drone will land at the current location.

Main Success Scenario:

1. User press the abort button.
2. The Drone stop his mission.
3. The Drone lands.

**Chapter 2- System Architecture:**

**The socket protocole between C# and Android for mission sending:**

Android side to C# -

* Move mission will send as the following string when finished: “move <index> Done”.
* Take off mission will send as the following string when finished: “takeOff <index> Done”.
* Confirm landing mission will send as the following string when finished: “confirmLanding <index> Done”.
* Start landing mission will send as the following string when finished: “startLanding <index> Done”.
* Go home mission will send as the following string when finished: “goHome <index> Done”.
* Move gimbal mission will send as the following string when finished: “moveGimbal <index> Done”.
* Go to GPS mission will send as the following string when finished: “goToGPS <index> Done”.
* Take photo mission will send as the following string when finished sending the picture: “takePhoto <index> Done”.
* Stop mission will send as the following string when finished: “stop <index> Done”.
* Get status mission will send as the following string when finished: “getStatus <index> Done <num>”.
* Get location mission will send as the following string when finished: “getLocation <index> Done <longitude> <latitude> <altitude> <bearing>”.
* Get height mission will send as the following string when finished: “getHeight <index> Done <height>”.

C# side to Android -

* Move mission will send as the following string to execute by the drone: “move <index> <left/right/up/down/forward/backward/rotate> <distance/degree>”.
* Take off mission will send as the following string to execute by the drone: “takeOff <index>”.
* Start landing mission will send as the following string to execute by the drone: “startLanding <index>”.
* Confirim landing mission will send as the following string to execute by the drone: “confirimLanding <index>”.
* Move gimbal mission will send as the following string to execute by the drone: “moveGimbal <index> <VerDegree> <HorDegree>”.
* Go to GPS mission will send as the following string to execute by the drone: “goToGPS <index> <longitude> <latitude> <altitude>”.
* Take photo mission will send as the following string to execute by the drone: “takePhoto <index> “
* Stop mission will send as the following string to execute by the drone: “stop <index>”
* Get status mission will send as the following string to execute by the drone: “getStatus <index>”.
* Get location mission will send as the following string to execute by the drone: “getLocation <index>”.
* Get height mission will send as the following string to execute by the drone: “getHeight <index>”.

**The socket protocol between C# and Android for picture sending:**

Send the photo with end of file.

**The protocol between C# and cars detector:**

Send a path of photo.

**The protocol between cars detector( yolo3) and C#:**

For each car:

<type of vehicle> \t<precentage>\t<margin-left>\t<margin-top>\t<width>\t<height>/n

**The protocol between C# and verify location:**

Send: <path of base photo> <current photo path> <base photo height> <current photo height>

**The protocol between verify location and C#:**

For each prediction:

<margin-left> <margin-top>/n

**The protocol between C# and car plate detector:**

Send a path of photo.

**TODO**

**The protocol between car plate detector and C#:**

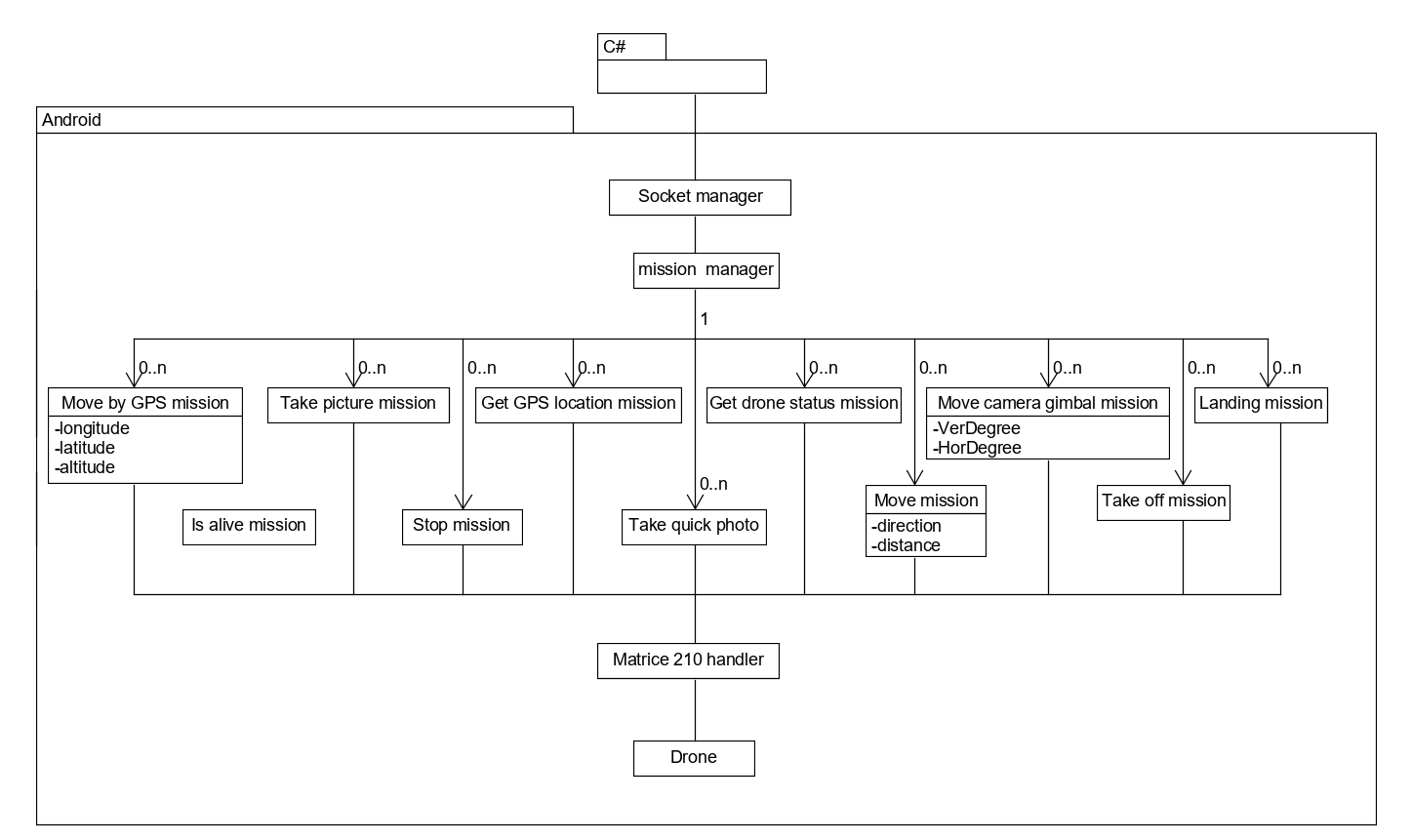
Send a path of photo.

**TODO**

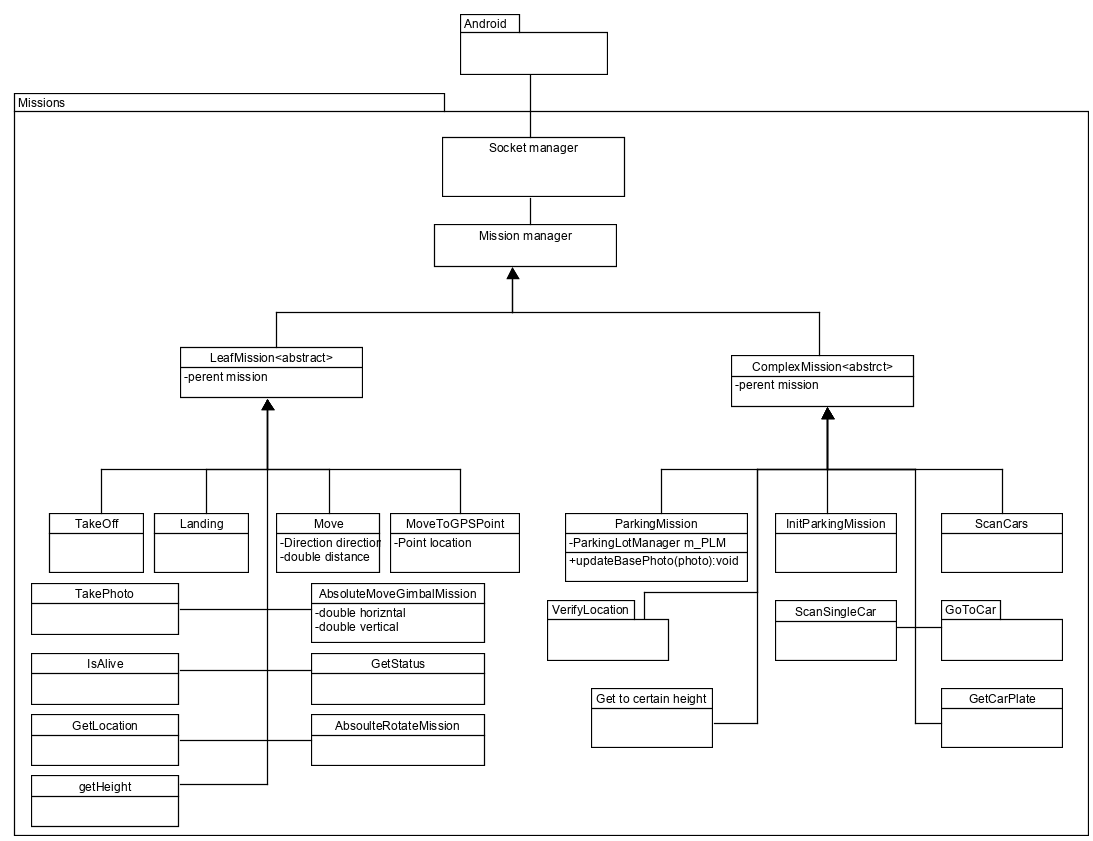
**Chapter 3 - Data Model:**

The main data objects in our mediator (Android) will be:

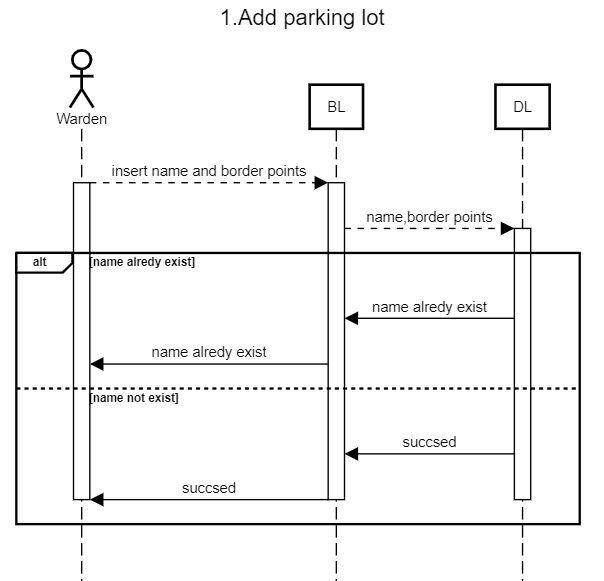
* Socket manager: will send the result after finish executing the task, and get the missions the drone need to execute.
* Mission manager: Contains a pool of missions that need to be execute.
* Missions: Each missions hold the data needed for his execution.
* Matrice 210 handler: Hold the data of specific drone API

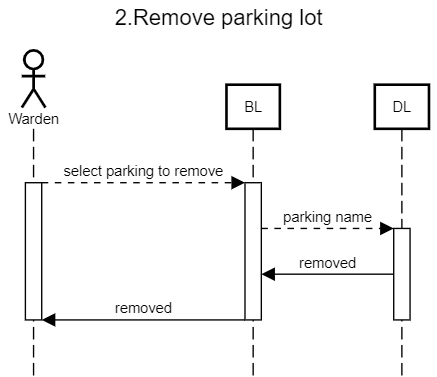


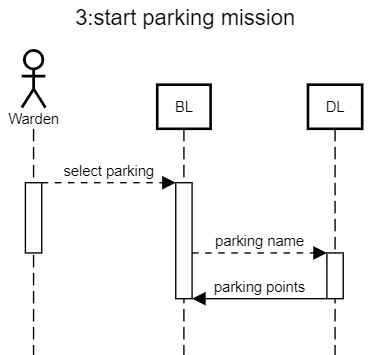
The main data objects in our mission manager (C#) will be:

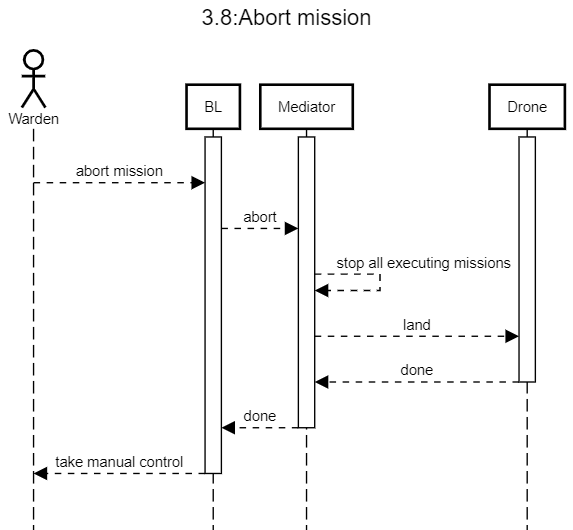
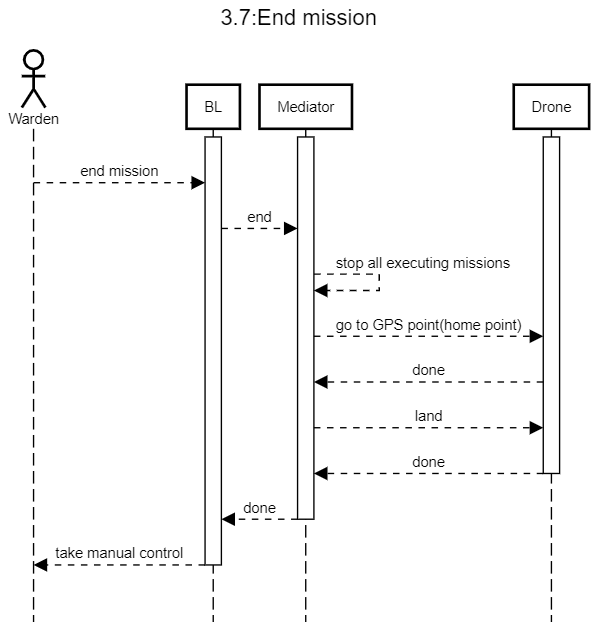
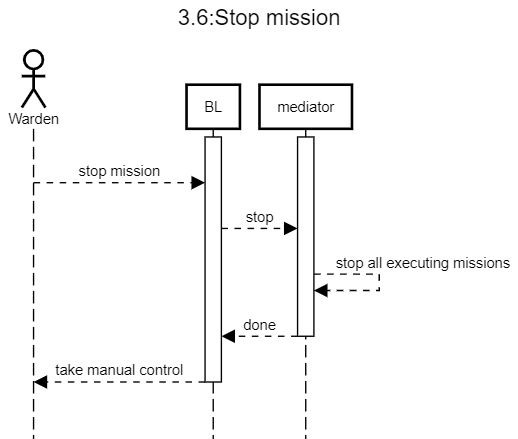
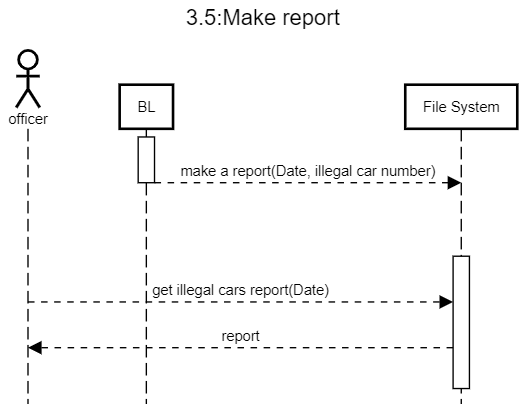
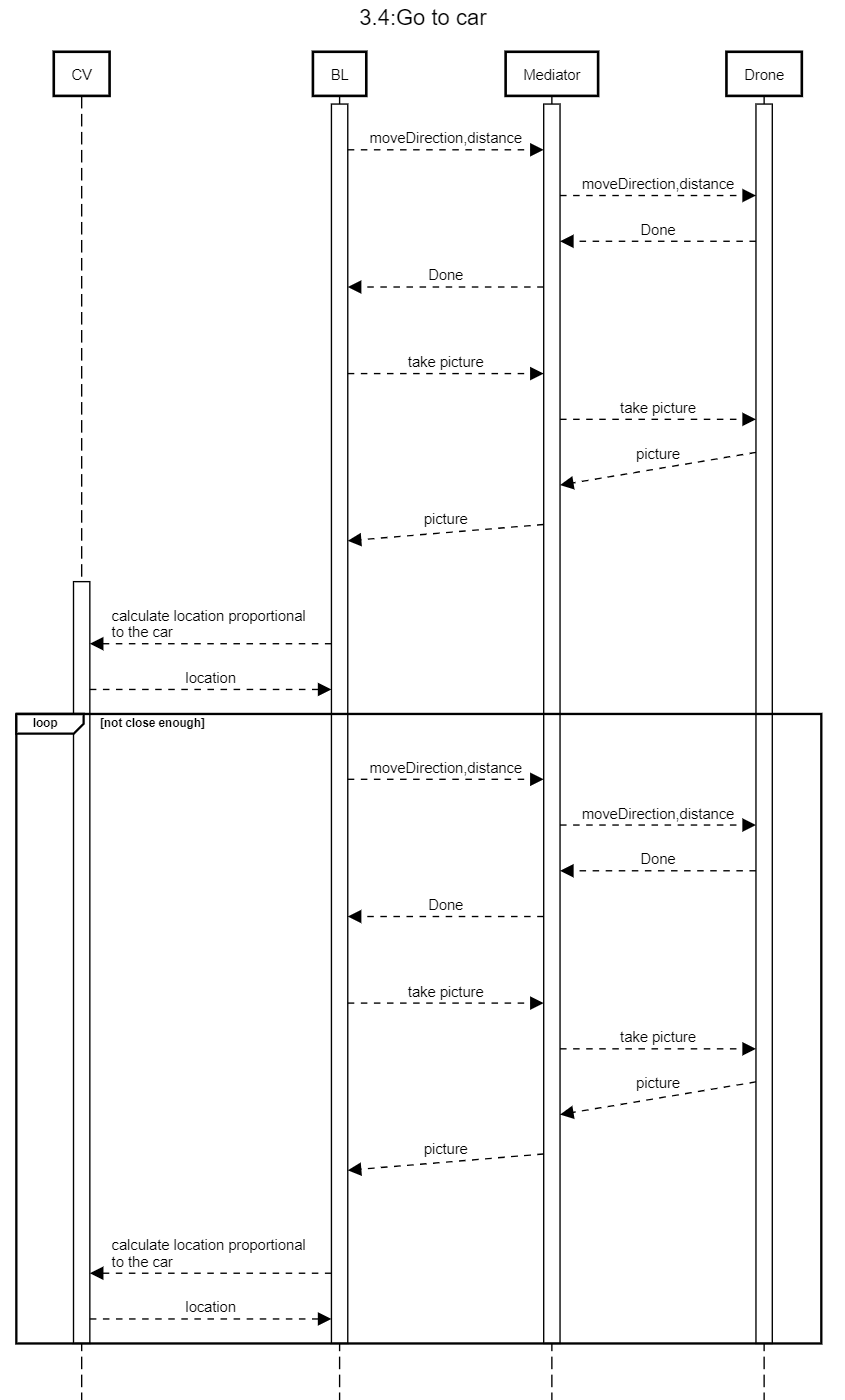
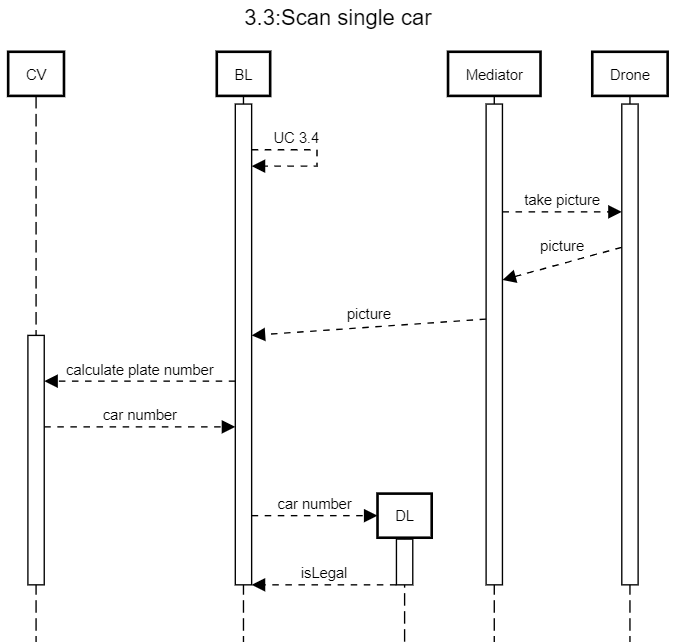
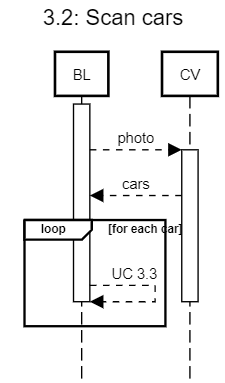
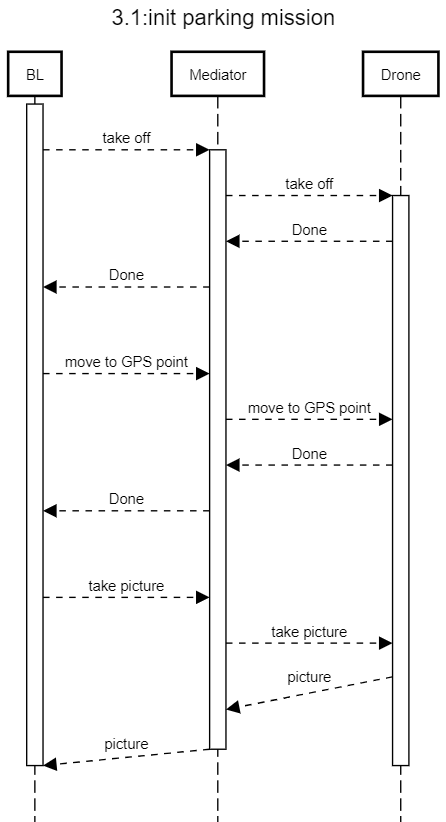
* Socket manager: will send a simple mission to the mediator to execute, and wait for the missions result.
* Mission manager: Contains a pool of missions that need to be execute.
* Missions: Each missions hold the data needed his execution.

**Chapter 4 - Behavioral Analysis:**

**4.1 Sequence Diagrams:**

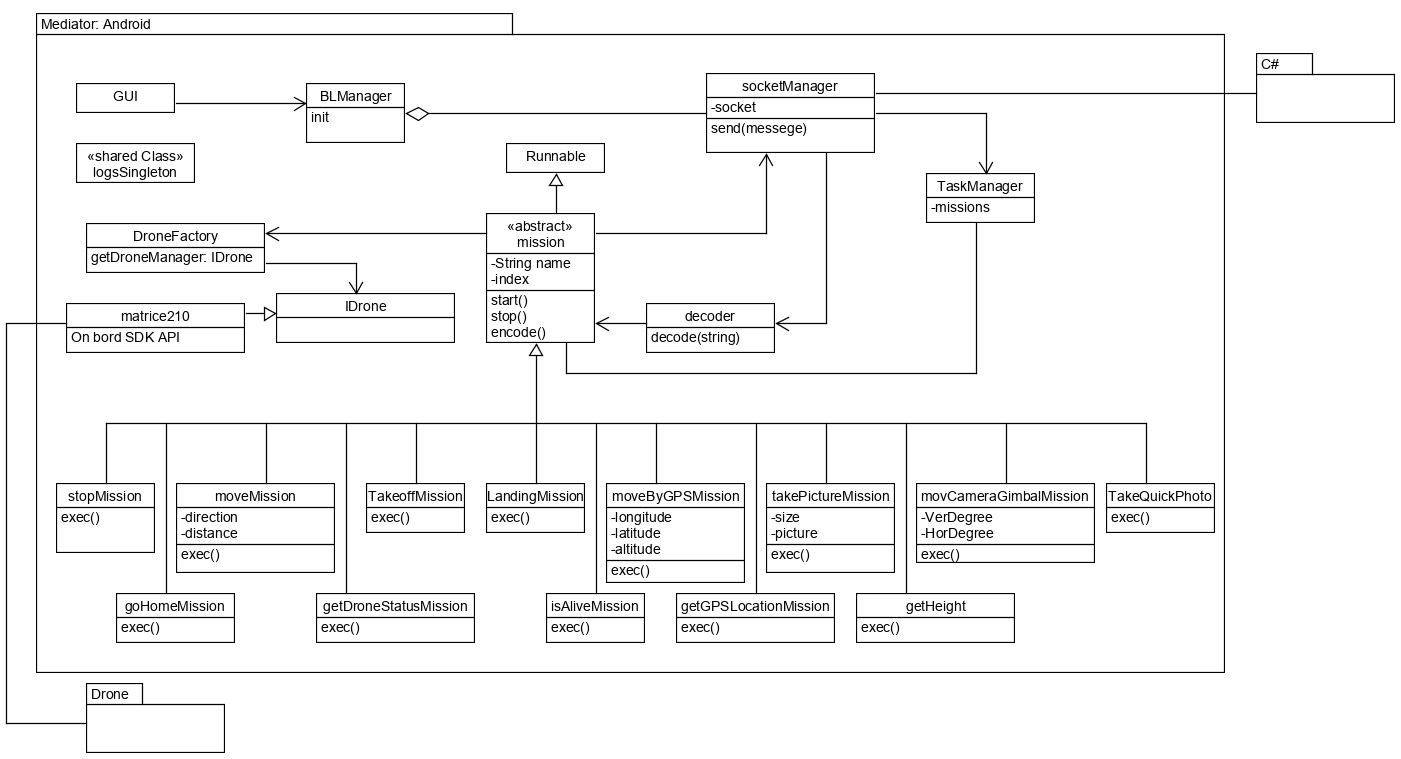
****

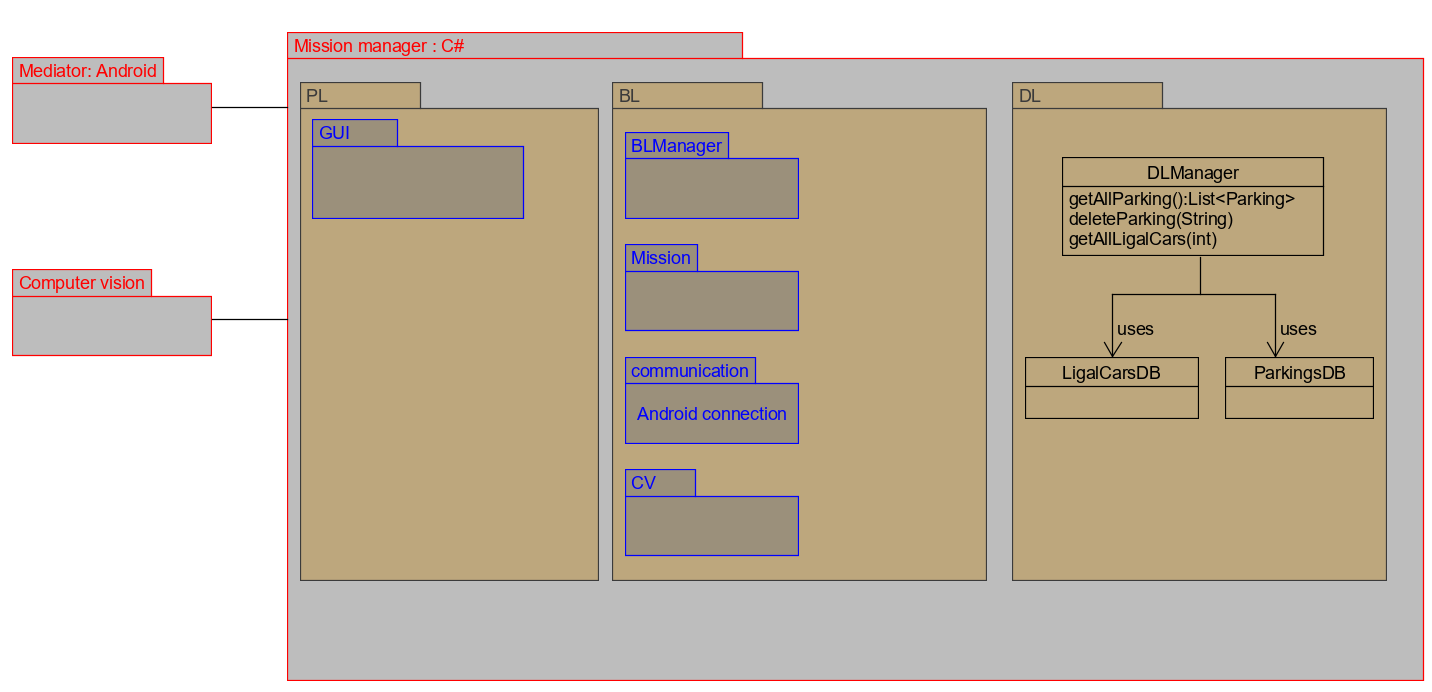


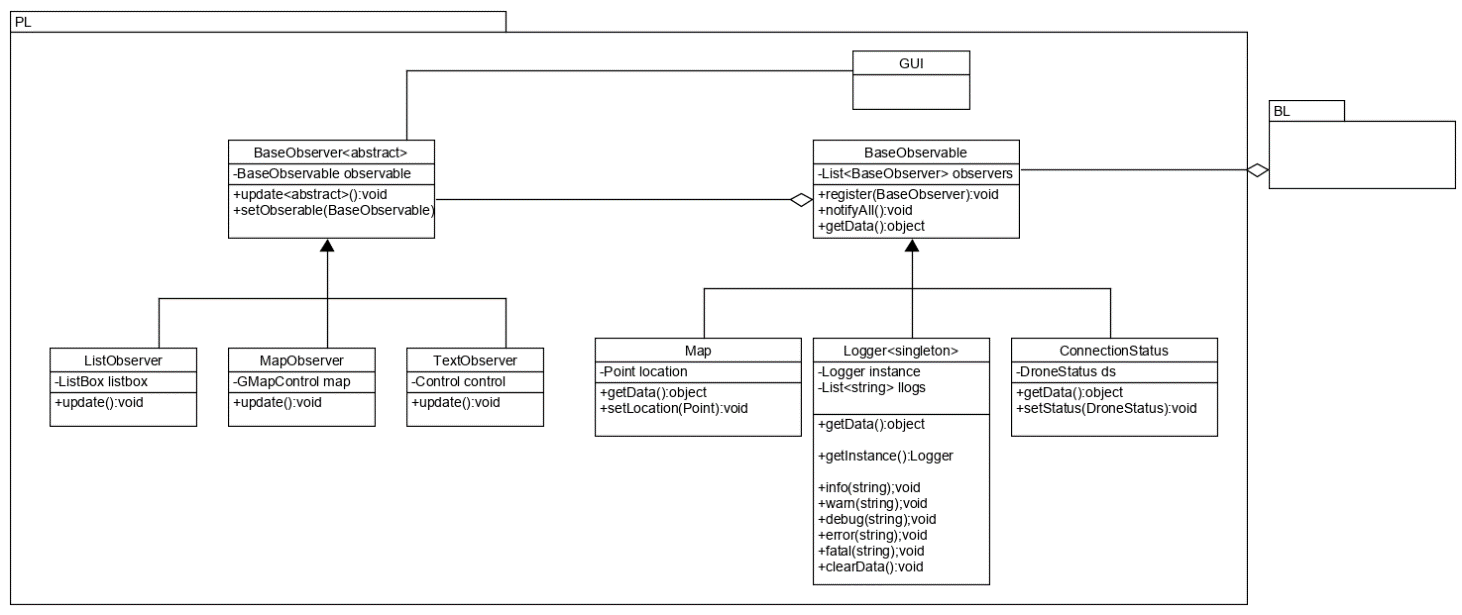


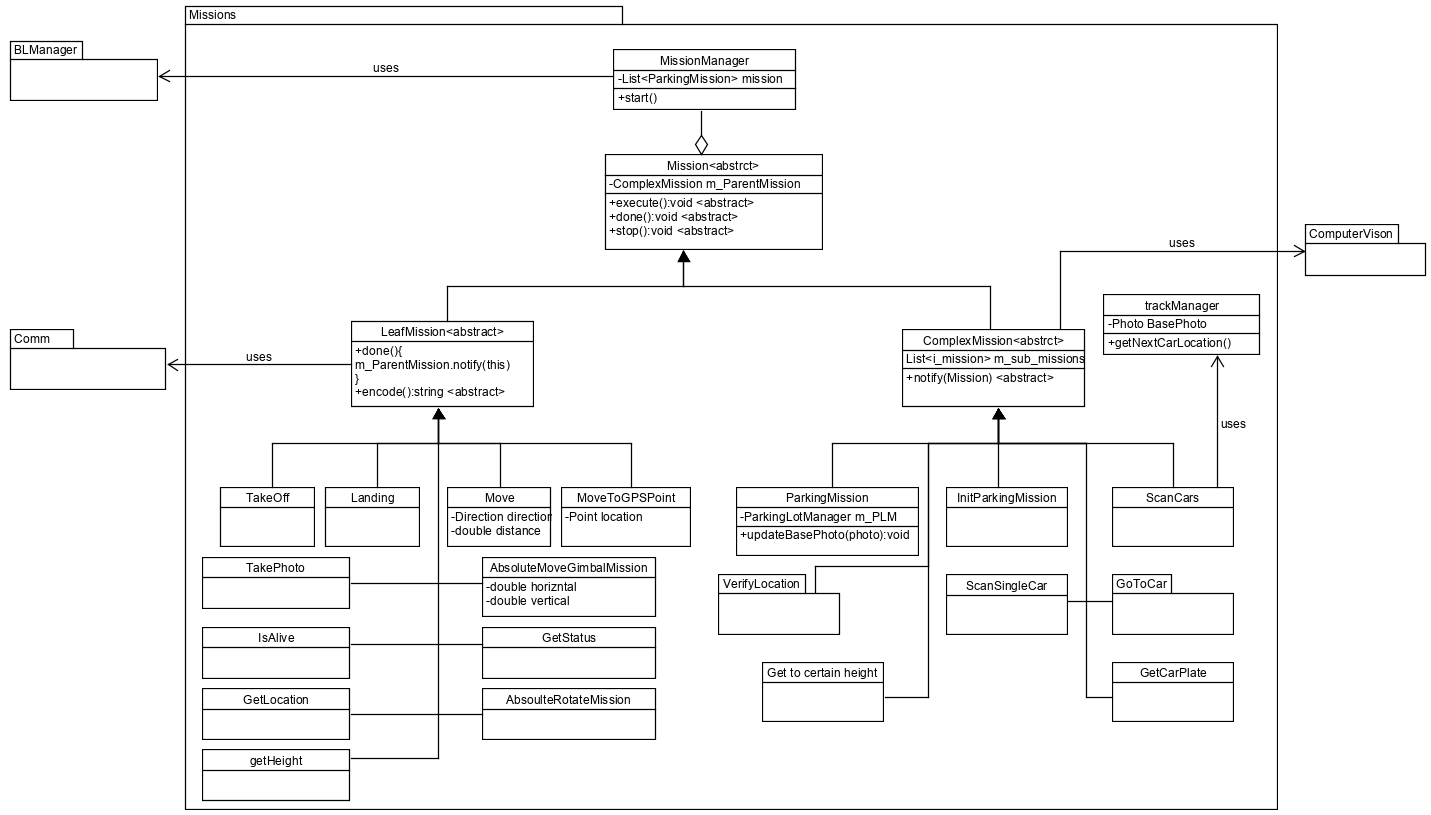
**Chapter 5 - Object-Oriented Analysis:**

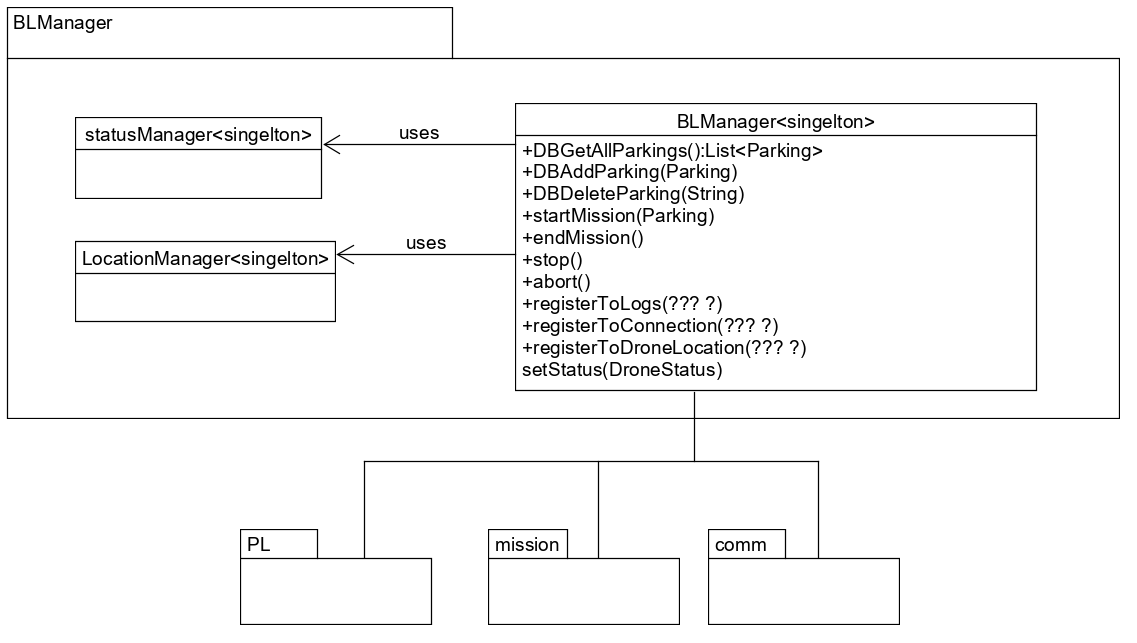
**5.1 Class Diagrams:**

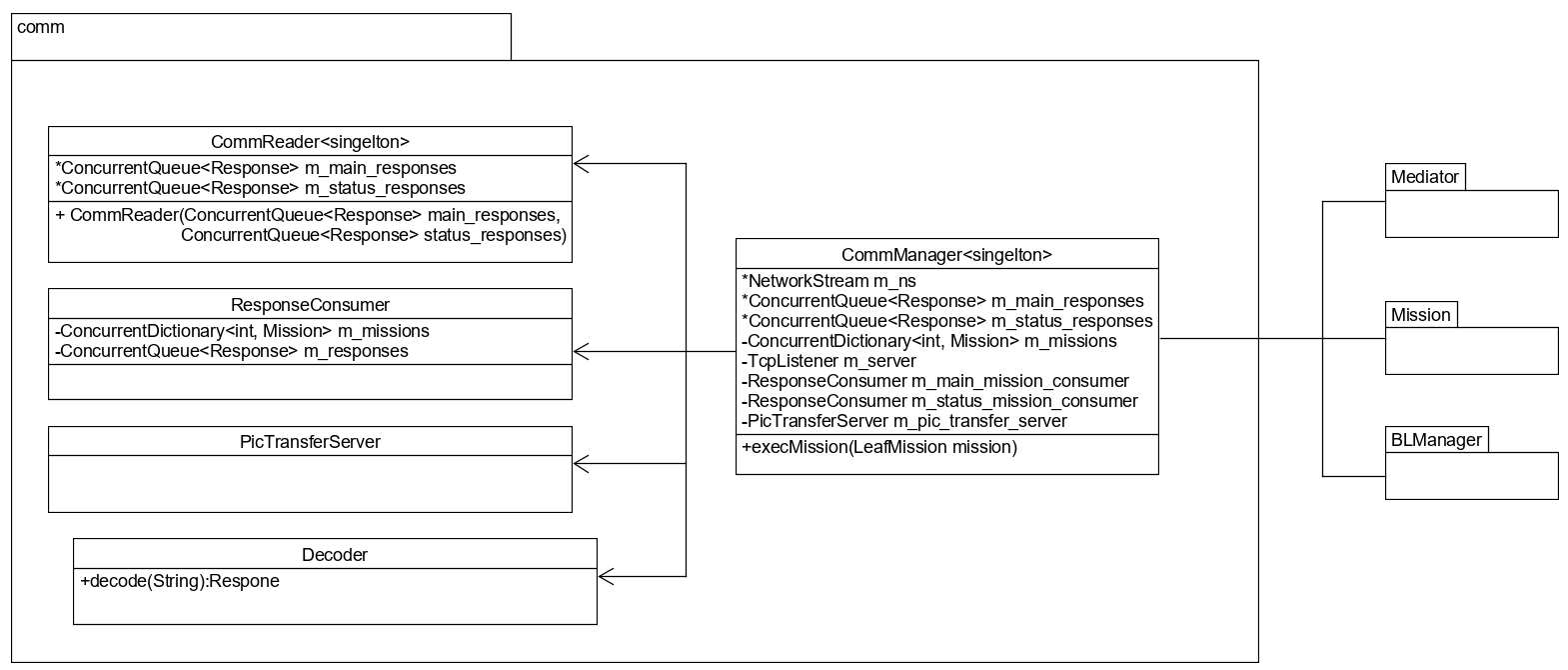
Mediator component:

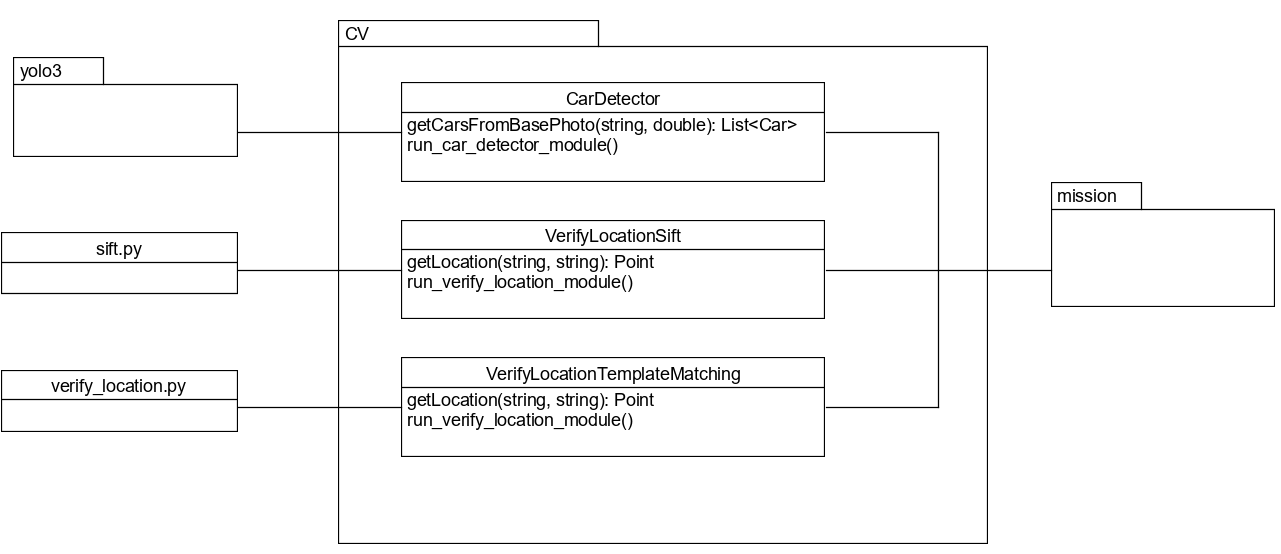
Mission manager component over view:

PL:

Mission component:

BL manager component:

Comm component:

CV:

**5.2 Class Description:**

**Mediator component:**

**Missions:**

Mission is an abstract class that many classes implements him. The main purpose of this class is to execute the specific mission and notify the next mission.

**DroneFactory & IDrone:**

Drone Factory return the drone that is being connected to the system. Let us change more easily different kinds of drones that implements IDrone interface.

**M210Manager:**

Implements IDrone, manages the communication between the drone and the mediator.

**Socket manager:**

Handle the communication between the mediator (Android) and mission manager (C#). Encode the data from the mission manager and transfer it to specific mission, then send the mission to the task manager.

**Task manager:**

Handle the drone main mission, status, sub mission and critical missions (stop, abort, end).

**Mission manager component (C#):**

**Missions:**

Mission is an abstract class that can be either a leaf mission (simple) or a complex mission.

**Complex mission:**

It is a mission that makes sub missions. Meaning for each complex need to be executed few missions before notify the parent mission. Complex mission cannot be execute.

**Leaf mission:**

It is a mission that can be sent to execute by the mediator. When finished notify the parent mission.

**Communication manager:**

Manage encode and decoding massages that are being transfer to and from the mediator. Moreover, after decode the communication manager notify the mission that been finished.

**Location manager:**

Responsible to keep track of the drone real-time location. And handle limits of no fly zones.

**BL manager:**

Initialize the data base connection, log server, execute the critical missions (stop, end, abort), get map permission to show in the GUI, get initial location and status and start a parking mission(the first complex mission).

**Parking:**

An object that represent a parking lot. This object obtain the name of the parking lot, its borders (by points) and the base position(the latitude, longitude and altitude) from which the drone can capture all the parking lot from the camera.

**DL manager:**

Manage connections to the data bases. Fetch parking object, check if name exist, remove parking, check if a specific can are permitted to park.

**Verify location:**

Manage the algorithm that will check the location of the drone in the parking.

**Car detector:**

Uses trained yolo3 algorithm to detect cars in a picture.

**Car plate detector:**

Uses AWS deep learning algorithm to detect numbers in a picture.

**5.3 Packages:**

**Mediator Project:**

**-BL package**: Hold all the business layer logic of the project.

* Drone package: Responsible of getting the instance Drone with the privileges.
* Mission package: Have all the missions that the mediator can handle.

**-Shared Classes package**: includes configuration files, assertion class, logger and more global functionality and objects.

**-GUI package**: responsible of the UI.

**Mission manager:**

**-PL**: responsible of the UI.

* Observers: responsible to show loges to the user.

**-BL**: Hold all the business layer logic of the project.

* Comm: manages the communication between the mission manager(C#) and mediator (Android).
* Mission: Have all the missions that the mission manager can execute.
  + Complex mission: mission that transfer to few leaf missions.
  + Leaf mission: mission that can be send to the mediator to execute.

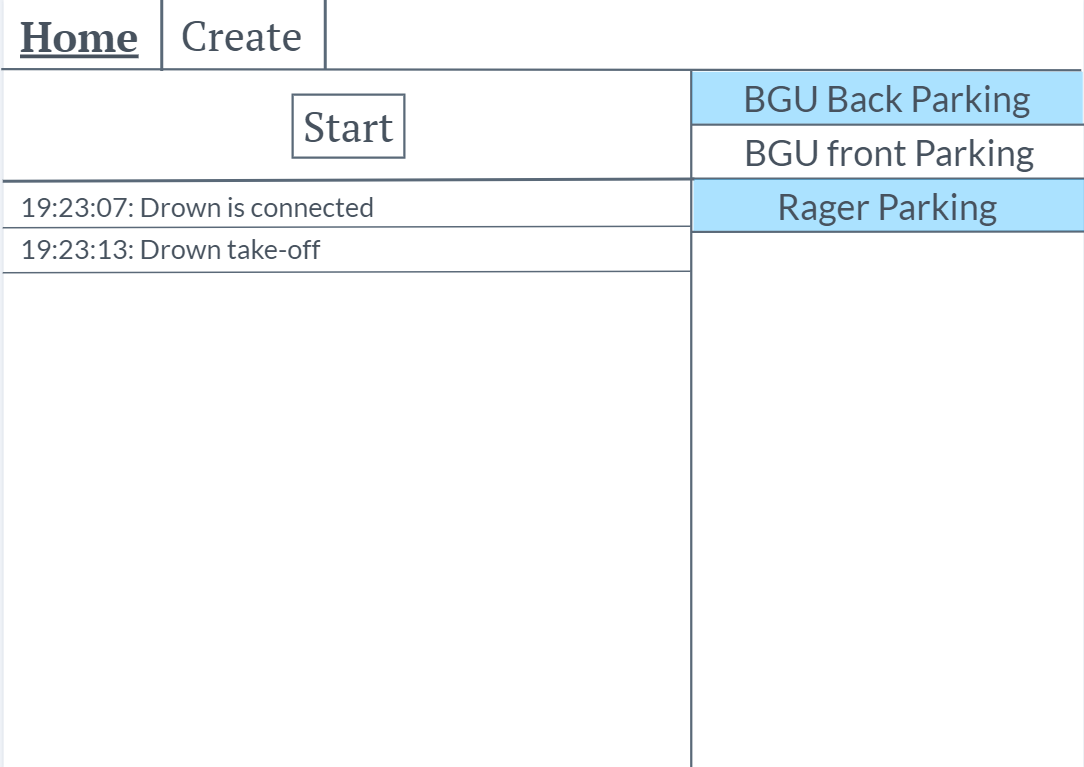
**-DL**: Data base handling.

**-Shared classes**: includes configuration files, assertion class, logger and more global functionality and objects (Like Point, and Parking).

**-CV:** computer vision executors.

**Chapter 6 – User Interface Draft:**

**6.1 Home page –**



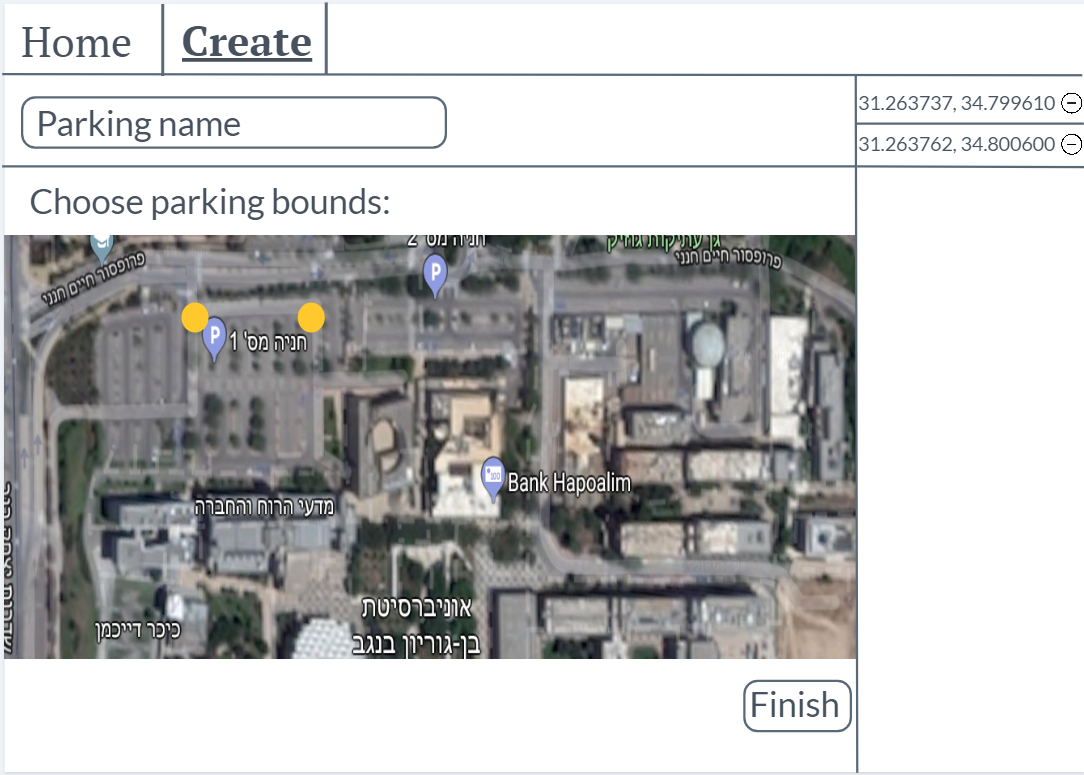
At the right side of the GUI we can see the predefined parking lots.

The button start will start scan of the selected parking lots.

We can see the system logs beneath the start button.

Press create will is for creating a new parking lot.

**6.2 create parking page –**

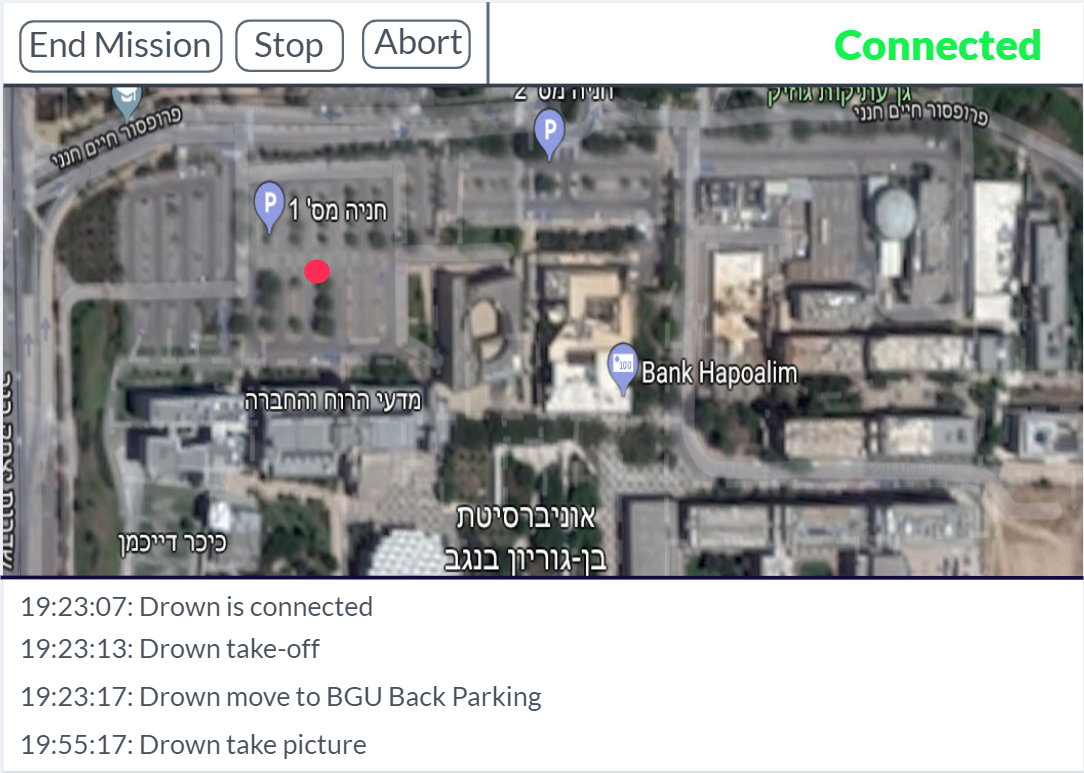


The user chooses the parking name and press the map in order to choose parking bounds.

When finish he can press finish and the parking lot will be created.

At the right side he can see the parking bound he already choose.

**6.3 Mission page –**

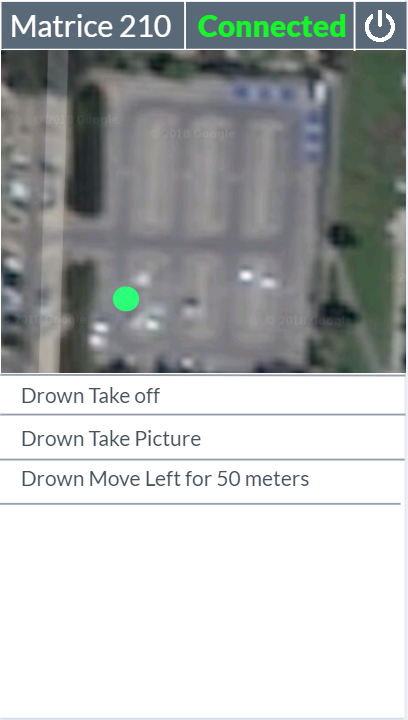


During an execution of a mission the user will see this page

Where he can see the current location of the drone, and press “End Mission”, “Stop”, “Abort” at any time.

The user can also see the logs and the drone status.

**6.4 Android GUI –**



The user can see the drone location at any time and the logs of the drone.

**Chapter 7 – Testing:**

**7.1 Unit Tests:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component Name | Goal | Input | Expected Result | Actual Result | Pass / Fail |
| Verify location | With car in the photo | Base photo and small photo with car | Width: 455  Height: 317 | Width: 455  Height: 317 | Pass |
| Cars detector | Photo with cars | Photo with cars |  |  | Pass |
| Cars detector | Photo without cars | Photo without cars |  |  | Pass |
| Cars detector | Different height photo | Different height photo |  |  | Pass |
| Cars detector | Different place photo | Different place photo |  |  | Pass |
| Pixel converter | Check a good pixel from certain height | 500 ,500 | 21 width  23 height |  | Pass |
| Pixel convertor | Different height taken | 500, 500 | 43 width  45 height |  | Pass |
| Decoder | For all the possible leaf mission |  |  |  | Pass |
| Point | Check if 2 point close enough | Close points |  |  | Pass |
| Point | Check if 2 point close enough | Far point |  |  | Pass |
| Point | Check if 2 point close enough | Close points with acceptable error distance |  |  | Pass |
| Point | Check if 2 point close enough | Far points even with acceptable error distance |  |  | Pass |
| Point | Get moves between 2 points | 2 good point |  |  | Pass |
| Point | Get moves between 2 points | Bad point |  |  | Pass |
| Parking | Good parking | 4 ok points for parking lot |  |  | Pass |
| Parking | Small parking | To close point |  |  | Pass |
| Parking | Big parking | To far points |  |  | Pass |
| Parking | Get base point work | From a good border get base point |  |  | Pass |
| Car | Good car |  |  |  | Pass |
| Car | To big car |  |  |  | Pass |
| Car | To small car |  |  |  | Pass |
| Car | Get the middle of the car |  |  |  | Pass |
| DB | Simple add parking | 4 Points, parking name | inserted |  | Pass |
| DB | Add parking without border | Parking name | Not inserted |  | Pass |
| DB | Add parking without name | 4 Points | Not inserted |  | Pass |
| DB | Add parking with name already existing in the DB | 4 Points, name | Not inserted |  | Pass |
| DB | Is exist parking | Name | Existed |  | Pass |
| DB | Is exist parking | Name | Not existing |  | Pass |
| DB | Is exist parking on empty data base | Name | Not exist |  | Pass |
| DB | Is exist parking without name |  | Not exist |  | Pass |
| DB | Simple delete parking | Parking name | Deleted |  | Pass |
| DB | Delete parking which is not exist | Parking name | Nothing happened |  | Pass |
| DB | Delete parking without name |  | Nothing happened |  | Pass |
| DB | Delete parking on empty DB | Parking name | Nothing happened |  | Pass |
| DB | Get all allow car plats |  | All car plates |  | Pass |
| Report manager | Simple make report | 2 Cars, base photo | Report is produce |  | Pass |
| Report manager | Make report with a lot of cars | 250 Cars, base photo | Report is produce |  | Pass |
| Report manager | Make report without cars | Base photo | Report is produce |  | Pass |
| Report manager | Make report without base photo | 2 cars | Report is not produce |  | Pass |
| Circle math | Left or right | 50,30 | Right |  | Pass |
| Circle math | Left or right | 220,0 | Left |  | Pass |
| Circle math | Left or right | -10,30 | Left |  | Pass |
| Circle math | Left or right | 120,30 | Right |  | Pass |
| Circle math | Left or right | 179,0 | Right |  | Pass |
| Circle math | Left or right | 180,0 | Right |  | Pass |
| Circle math | Left or right | 181,0 | Left |  | Pass |
| Latitude longitude | Get barring between merging point simple | Point(0,0,0),  Point(10,10,0) | 135 |  | Pass |
| Latitude longitude | Get barring between more than 90 degree | Point(0,0,0),  Point(-10,10,0) | 135 |  | Pass |
| Latitude longitude | And more |  |  |  | Pass |

**7.2 Functional Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement number | Component | Input | Pass / Fail |
| 1 | DroneServerAceptanceTests.ParkingAcceptanceTests | Good parking, bad parking | Pass |
| 2 | DroneServerAceptanceTests.ParkingAcceptanceTests | Remove existing parking | Pass |
| 3 | - DroneServerAceptanceTests .FullScenarioTests  - IntegrationTests. parkingTest | - Choose a parking to execute and execute.  - choose random parking to execute | Pass |
| 4 |  | Manual Test | Pass |
| 5 |  | Manual Test | Pass |
| 6 |  | Manual Test | Pass |
| 7 | - AndroidAccepanceTests. MoveToGPSTest  - DroneServerAceptanceTests .FullScenarioTests  -IntegrationTests. InitParkingTest | - check that move by GPS work  -check that whole cenario work.  -check init parking mission including go to the middle of the parking. | Pass |
| 8 | - DroneServerAceptanceTests .FullScenarioTests  - IntegrationTests. goToCarTest  - IntegrationTests .scanSingleCarTest | - parking and base photo  - car location  -car location | Pass |
| 9 | DroneServerAceptanceTests .FullScenarioTests | parking and base photo | Pass |
| 10 | DroneServerAceptanceTests .FullScenarioTests | parking and base photo | Pass |
| 11 | - IntegrationTests. verifyLocationTest | Check the location of the drone.  Also Visual check at the GUI | Pass |
| 12 |  | Manual check | Pass |
| 13 |  | Manual check | Pass |
| 14 |  | Supporting |  |
| 15 | DroneServerAceptanceTests .ParkingAcceptanceTests | Don’t letting to add parking that the drone will need to go over 100 meters | Pass |
| 16 | - DroeServerAceptanceTests. StartLanding  - IntegrationTests .LandingTest | Multiple landing situations | Pass |
| 17 | DroneServerAceptanceTests .ParkingAcceptanceTests | Don’t letting to add parking | Pass |

**7.3 Non-Functional Requirements**

|  |  |  |
| --- | --- | --- |
| Request number | Input | Pass / Fail |
| Performance 1 | Manual check | Pass |
| Performance 2 | Manual check | Pass |
| Performance 3 | Configure in the system | Pass |
| Performance 4 | Manual check | Pass |
| Performance 5 | Manual check | pass |
| Performance 6 | Not taking over 1 minute | Pass |
| Performance 7 | Takes about 40 ms. | Pass |
| Performance 8 | Manual check – 90%+ | Pass |
| Performance 9 | Manual check – 90%+ | Pass |
| Performance 10 | Manual check – 80%+ | Pass |
| Performance 11 | Manual check – 80%+ | Pass |
| Performance 12 | Manual check for 5 cars |  |
| Reliability 1 | Manual check | Pass |
| Reliability 2 | Manual check | Pass |
| Safety 1 | Manual check | Pass |
| Safety 2 | Check the drone location | Pass |
| Safety 3 | Hardware description | Pass |

**7.4 End to End / Acceptance Test**

Adding acceptance test project written in C#.

Mediator acceptance test:

To test the mediator we will simulate the mission manager and send missions to be execute and check the response. This test will run in different loads.

Pre-condition for that test:

* The drone have to be connected to a computer with DJI Matrice assistant 2 will the simulator is running.
* Drone have enough battery power.
* The controller have enough battery power.
* An android devise with the mediator app installed.
* The android devise granted the app with the necessary permission.
* The aircraft is connected.

Mission manager acceptance test:

To test the mission manager including the mission handling and the CV algorithms we build a simulator that simulate the mediator.

In the tests of the mission manager we run the whole scenario with different number of cars and different parking lots.