Group Member NAMES: Ryan Sauer, Demetrius Johnson, Jonathan Schall & Olivia Pellegrini

Software Requirements Specification

Table of Contents

Table of Contents **1**
1.0 Introduction **1**
1.1 Goals and objectives 1
1.2 Statement of scope 1
1.3 Software context 3
1.4 Major constraints 3
2.0 Usage scenario **3**
2.1 User profiles 3
2.2 User stories 3
2.3 Special usage considerations 4
3.0 Data Model and Description **4**
3.1 Data Description 4
3.1.1 Data objects 4
3.1.2 Relationships 4
3.1.3 Complete Data Model 4
3.1.4 Data Dictionary (CRC: class, responsibility, collaboration) 4
4.0 Functional Model and Description **8**
4.1 Description for Function n 8
4.2 Software Interface Description 11
4.2.1 External Machine Interfaces 11
4.2.2 External System Interfaces 11
4.2.3 Human Interface 11
4.3 Use Case Diagrams 12
4.4 Sequence Diagrams 12
4.5 Communication Diagrams 12
5.0 Behavioral Model and Description **12**
5.1 Description for Software Behavior 12
5.1.1 Events 12
5.1.2 States 12
State diagrams are used to show the lifetime behavior of an object within our system as it reacts to external events. 18
5.3 Activity Diagram 18
6.0 Restrictions, Limitations, and Constraints **18**
7.0 Validation Criteria **19**
8.0 Appendices **21**

8.1 System traceability matrix 21
9.0 References **21**
Hardware and Software Materials and Requirements 21
Research Background Sources 24

1.0 Introduction

1.1 Goals and objectives

The main objective of our project is to develop a drone-car collaborative model that can be used to obtain data on drone-car collaboration. The model will use a drone camera to record the surroundings of the car and notify the car of potential hazards in real time. Some basic goals of our model are:

- Accurately detect a potential hazard in multiple environments using sensors from the car and drone, particularly the vision systems (camera).
- Minimize car stop-response-latency by optimizing the communication protocol between the drone and the car.
- Maximize drone battery life.
- Develop a working product that is open source and reproducible through our documentation to serve as a future reference/baseline for further research.

1.2 Statement of scope

General Requirements of the model

- An image recognition algorithm running on the drone for detecting a predefined object
- A manual/automatic control mechanism for the car
- A manual/automatic control mechanism for the drone
- A multiple communication protocols for use between the drone and car
- An ascii based log system to store data points about the drones battery life, bandwidth, and latency.
- A way to extract drone data logs from the raspberry pi
- A way to extract car data logs from the raspberry pi

1.3 Software context

We will be applying software to a drone and car for research purposes. It is not intended to be publicly accessible and only used by researchers for data creation and analysis. It is also meant to be used in a controlled environment with simple scenarios for gathering data.

1.4 Major constraints

The project needs to be completed by August 15th, 2023. This gives us approximately six months to complete the model. However, after the model is finished, we would like to use it to collect research data of our own. In order to do this the model would need to be finished before August.

Currently we are working with the PiCAR-x and the Clover Drone

We will be obtaining a few extra Raspberry-Pi’s in order to give everyone on the team the ability to work with one. Outside of this, there will be a very limited budget and it is unlikely the remaining budget can afford extra drones, cars, or expensive technology.

The Raspberry Pi for the PiCarX is coming in later than anticipated, preventing us from being able to work on it.

Additionally, drone flight regulations from University and state policies make finding and developing a more comprehensive test environment more challenging and limited.

2.0 Usage scenario

2.1 User profiles

There will be 3 main types of users:

Vehicle Administrators (Operators):

The vehicle administrators will manage the function of each vehicle, ensure its safety, and make any physical or software changes as requested by an experiment facilitator.

Data Analyst

The data analyst will recover information from the vehicles after an experiment, process the data, and create visualizations for the results.

Experiment Facilitator

The experiment facilitators will make decisions based on the data analysts findings on what parameters should be set for further experiments. They will ensure all the constants remain and variables are set and documented. The experiment facilitators should have contact with the vehicle administrators to make changes are applied to the vehicles.

2.2 User stories

The Vehicle Administrator will want to make adjustment to car parameters
The Vehicle Administrator will make adjustment to drone parameters
The Experiment Facilitator will run test flight using drone and car
The Data Analyst will recall logs from drone
The Data Analyst will recall logs from car
The Data Analyst will compare data from previous tests
The Vehicle Administrator will change communication protocol between drone and car

2.3 Special usage considerations

- The experiments run using the car and drone must be done in a controlled environment.
- All data retrieved will be for in house use and doesn't need to be sanitized.
- The parameters set for the car and drone along with the resulting data must be kept on record for accurate analysis.
- University policy is strict around drone use and we must adhere to those policies during testing and execution.

3.0 Data Model and Description

3.1 Data Description

The primary library used on the drone is the catkin and the **ROS** (Robot Operating System) (see references) libraries, which the **mavros** libraries depend on. From those libraries and the objects therein that we can interface with the easiest, we will develop two classes: **_Comm**, and **_Collect**. **_Comm** will be used to establish communications between the drone and the car. _Collect will be used to gather data streams from sensor information and redirect it to output files for logging. See section 9.0 (references) for more information.

3.1.1 Data objects

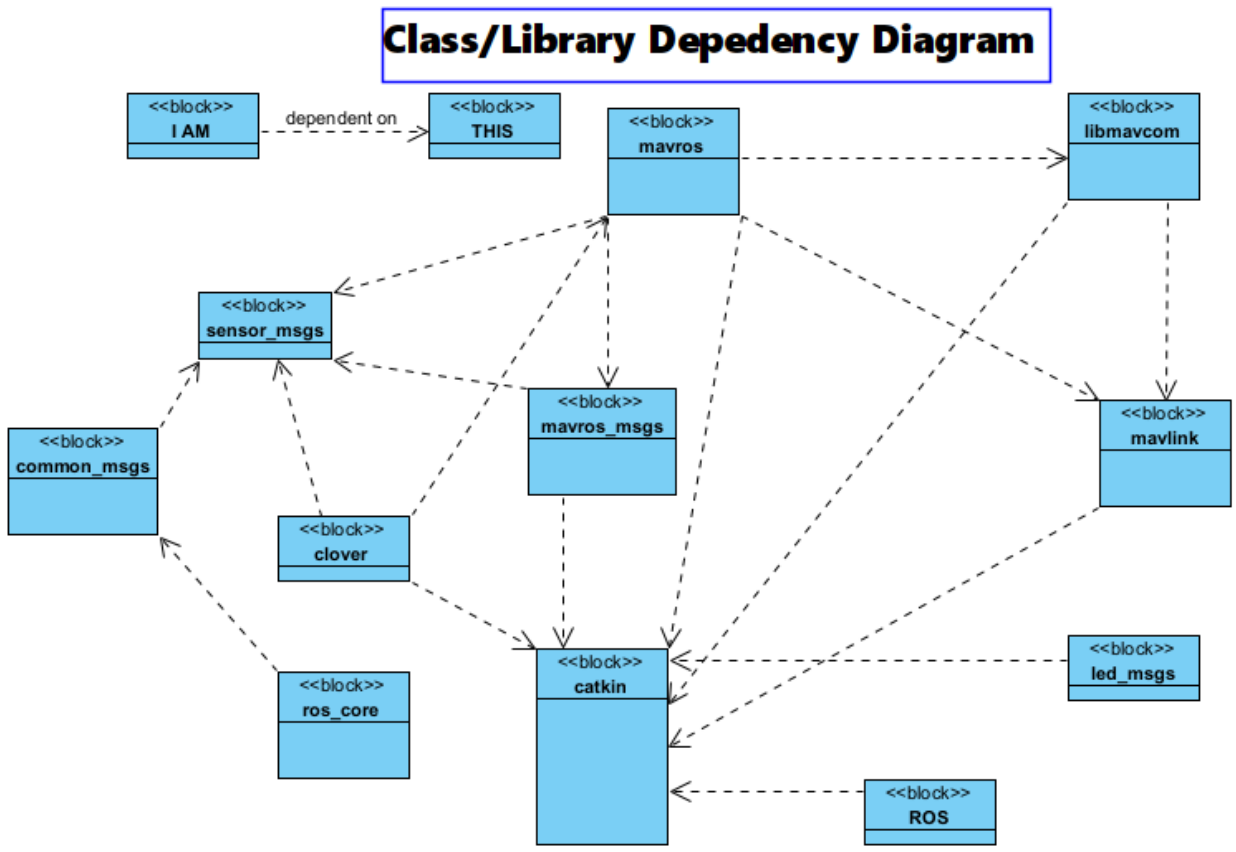
mavros_node
_Comm
_Collect

3.1.2 Relationships

Most of the **ROS** library depends on the **catkin** library. These two libraries are the primary code base for the **clover** class (used for the clover drone). From the catkin library, the various **mavros** libraries are written to stream messages from sensor data and other programmed messages through topics.

3.1.3 Complete Data Model

An UML Class model for the software is developed



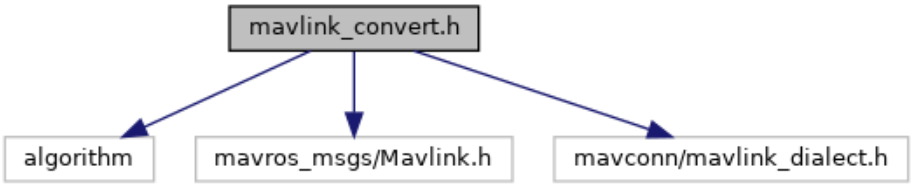
Mavlink_convert.h UML (convert ROS messages to mavlink)

mavlink_convert.h File Reference

Mavlink convert utils. More...

```
#include <algorithm>
#include <mavros_msgs/Mavlink.h>
#include <mavconn/mavlink_dialect.h>
```

Include dependency graph for mavlink_convert.h:



3.1.4 Data Dictionary (CRC: class, responsibility, collaboration)

Class	Description	General Overview of Responsibilities	Collaborators/ Relationships
_Comm	**Use this class to create communication objects that can connect across devices (i.e. drone and car) through socket programming**	**Interface with mavros_node object on the drone, and car_communication_object on the car to establish a connection**	**Interface with mavros_node objects and _Collect objects, car_node_objects**
_Collect	**Use this class to collect data streams from sensors and store them in a log file locally, or redirect the stream to a network socket.**	**Provide an interface to collect data using drone and car data structures/objects, and writing the data to a log file or other stream.**	**Interface with _Comm objects and mavros_msgs objects**
Mavros (mavros_node)	**Main communication protocol (mavlink protocol) used on the drone**	**Provides communications with the drone.**	**Interfaces with mavros_msgs and mavlink objects**
mavros_msgs	mavros_msgs defines messages for MAVROS	**Provide data streams for subscribers to sensor and other information topics of the drone.**	**Interfaces with mavros_node and mavlink objects**

sensor_msgs	**Use this class to obtain data streams for laser range finder, camera, battery, etc, that we will send to a log file**	**Provide data streams for subscribers to sensor and other information topics of the drone (like laser range finder, camera, etc.)**	**Interfaces with mavros_node, mavlink, and _Collect objects**
param_id[string]	**Used to set a parameter to a value**	**Use this parameter to select and change a parameter in the mavros_msgs object**	**mavros_msgs /ParamSet.srv**
libmavconn	**May be the interface by which we can drop our _Comm class to establish drone-car communications.**	**MAVLink communication library. This library provide unified connection handling classes and URL to connection object mapper. This library can be used in standalone programs.**	**Depends on mavlink library, and catkin library**
mavlink	**May need to use this to establish communication between the car and the drone; libmavconn library may be the better interface to implement our communications, however.**	**MAVLink message marshaling library. This package provides C-headers and C++11 library for both 1.0 and 2.0 versions of protocol. For pymavlink use separate install via rosdep (python-pymavlink).**	**Depends on catkin library**
catkin	**Catkin is included by default when ROS is installed. Catkin can also be installed from source or prebuilt packages. Most users will want to use the prebuilt packages, but installing it from source is also quite simple.**	**Low-level build system macros and infrastructure for ROS.**	

mavros_msgs Msg/Srv Documentation

Website	
Code API Documentation	
Message types	**Service types**
ADSBVehicle	CommandAck
ActuatorControl	CommandBool
Altitude	CommandHome
AttitudeTarget	CommandInt
BatteryStatus	CommandLong
CamIMUStamp	CommandTOL
CameraImageCaptured	CommandTriggerControl
CellularStatus	CommandTriggerInterval
CommandCode	CommandVtolTransition
CompanionProcessStatus	FileChecksum
DebugValue	FileClose
ESCInfo	FileList
ESCInfoItem	FileMakeDir
ESCStatus	FileOpen
ESCStatusItem	FileRead
ESCTelemetry	FileRemove
ESCTelemetryItem	FileRemoveDir
EstimatorStatus	FileRename
ExtendedState	FileTruncate
FileEntry	FileWrite
GPSINPUT	LogRequestData

GPSRAW	LogRequestEnd
GPSRTK	LogRequestList
GlobalPositionTarget	MessageInterval
HilActuatorControls	MountConfigure
HilControls	ParamGet
HilGPS	ParamPull
HilSensor	ParamPush
HilStateQuaternion	ParamSet
HomePosition	SetMavFrame
LandingTarget	SetMode
LogData	StreamRate
LogEntry	VehicleInfoGet
MagnetometerReporter	WaypointClear
ManualControl	WaypointPull
Mavlink	WaypointPush
MountControl	WaypointSetCurrent
NavControllerOutput	
OnboardComputerStatus	
OpticalFlowRad	
OverrideRCIn	
Param	
ParamValue	
PlayTuneV2	
PositionTarget	

RCIn	
RCOut	
RTCM	
RTKBaseline	
RadioStatus	
State	
StatusText	
TerrainReport	
Thrust	
TimesyncStatus	
Trajectory	
Tunnel	
VFR_HUD	
VehicleInfo	
Vibration	
Waypoint	
WaypointList	
WaypointReached	
WheelOdomStamped	

sensor_msgs
Website
Base class: common_msgs

ROS Message Types			ROS Service Types
BatteryState	Joy	NavSatStatus	SetCameraInfo
CameraInfo	JoyFeedback	PointCloud	
ChannelFloat32	JoyFeedbackArray	PointCloud2	
CompressedImage	LaserEcho	PointField	
FluidPressure	LaserScan	Range	
Illuminance	MagneticField	RegionOfInterest	
Image	MultiDOFJointState	RelativeHumidity	
Imu	MultiEchoLaserScan	Temperature	
JointState	NavSatFix	TimeReference	

4.0 Functional Model and Description

4.1 Description for Function n

Use Case: Running an Experiment
Actors: User, Car, Drone
Preconditions: The input will be a video frame captured from the drone. There is no real output, but after analyzing input a command may be sent to the car.
Triggers: The drone should always be capturing video frames and constantly be analyzing the frames.
Scenario Description: To successfully run an experiment, the drone must be able to capture video frames and analyze a frame. Once a target is detected, a command should be sent to the car to perform an action.
Post Description: We have limitations on the environment our group can work in. This is due to university and state policies about handling drones.

Exceptions: Our drone and car should be able to function in all circumstances and constantly collect metrics and analyze captured video frames.

Use Case: Change Car and Drone Parameters
Actors: User, Car, Drone
Preconditions: The input is parameters from the user. There is no real output, but the drone and car will receive these parameters.
Triggers: In order for this use case to take place, a user must intervene and change the parameters using the command line.
Scenario Description: The user will change parameters using the command line to run experiments with the drone and car.
Post Description: Commands must be correctly formatted to properly function.
Exceptions: System should only take valid parameters.

Use Case: Retrieve Data
Actors: User, Car, Drone
Preconditions: The input will be metrics received from the drone and the output is the information that will be recorded in the files.
Triggers: Log files will be updated when actions are completed by the drone and car. Data files will collect updates about the battery life of the drone and the network status.
Scenario Description: Metric should be taken from the drone and car constantly. Data will be sent to the log file when an action is executed by the drone and car.
Post Description: Due to limited battery capacity of the drone, the drone may need to choose between longer flight time and the accuracy and timeliness of the data.
Exceptions: To avoid draining the drone’s battery, our project aims to offload the time-insensitive tasks to car-mounted computers to avoid draining the drone’s battery.

4.2 Software Interface Description

4.2.1 External Machine Interfaces

Clover Drone:

The clover drone will be accessed in order to run the required programs for flight, vision analysis, and networking. We will be using QGroundControl and SSH to communicate with the drone. QGroundControl will be covered in External System

Interfaces. SSH is installed on the drone ahead of time and will allow us to place our programs onto the drone as well as recover data and log files.

PiCarX:
The PiCarX can also be connected to via SSH for program changes and log file recovery.

For both the Clover drone and PiCarX, we can also plug in an external monitor, but doing so may require some disassembly.

4.2.2 External System Interfaces

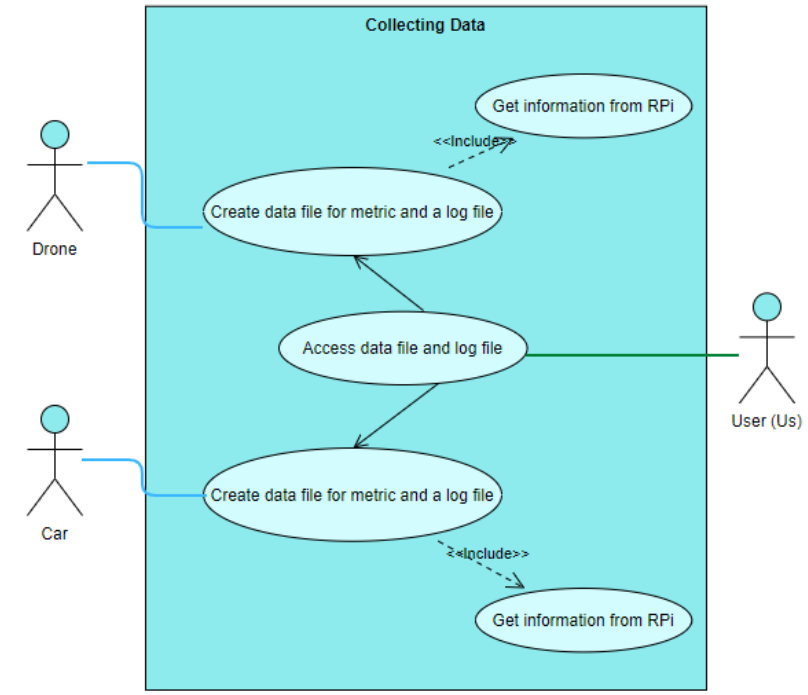
QGroundControl:
This is a program that allows us to connect to the Clover drone to monitor, control, calibrate, and configure the flight controller. It is a GUI based program for most systems which we will use to control the drone.

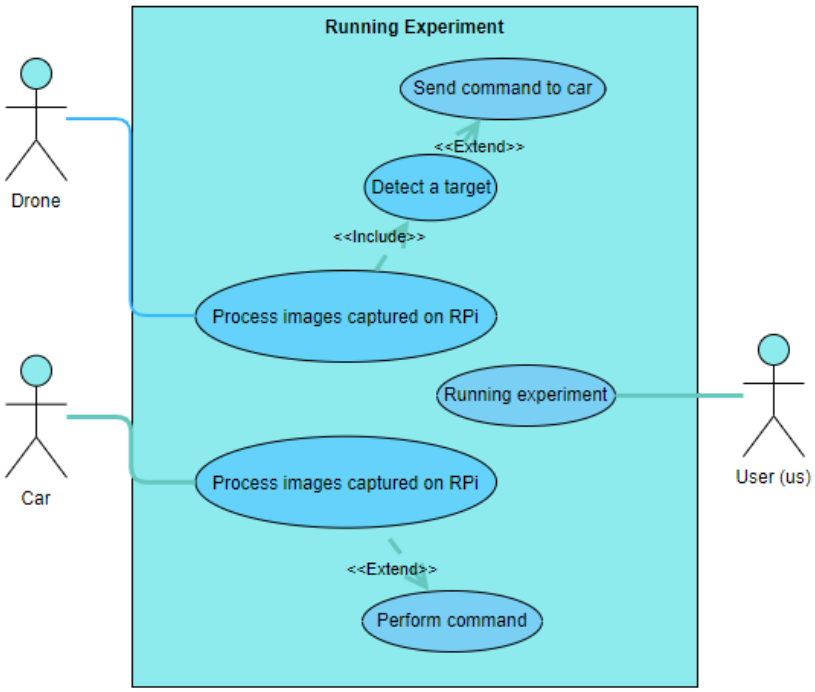
Dnsmasq:
Dnsmasq is a piece of software we will be utilizing on the PiCarX to act as a router for the connection point for the machines. It will be controlled through the command line.

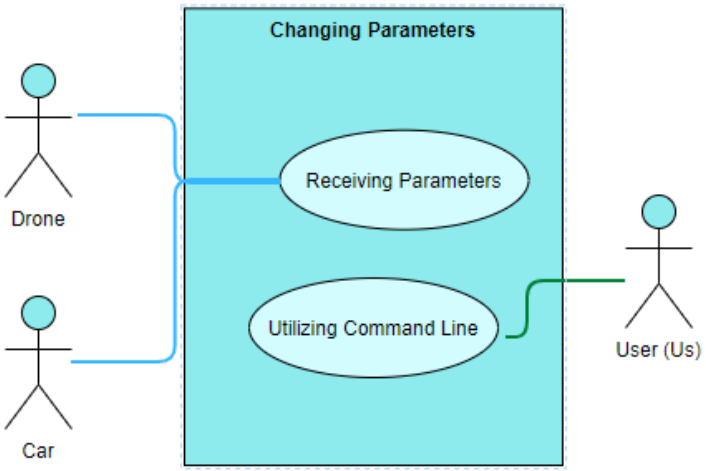
4.2.3 Human Interface

Our software will have little to no human user interface. Any debugging interface will be done through command line input. All changing of values will occur in files

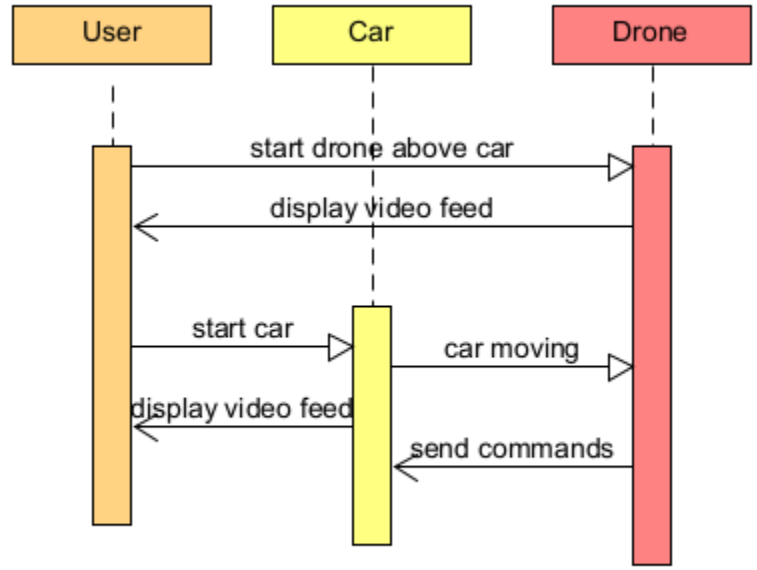
4.3 Use Case Diagrams

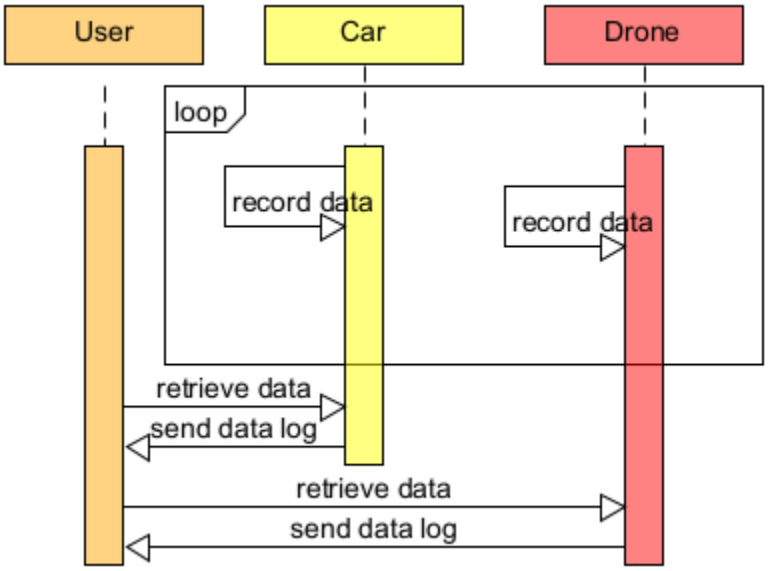


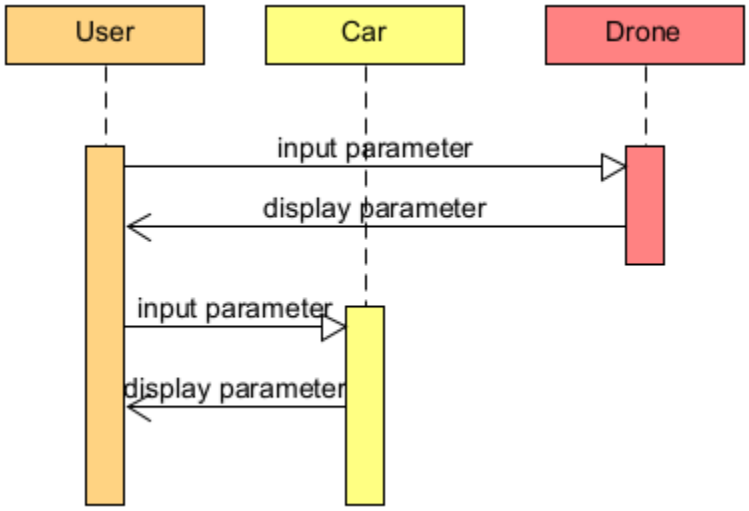



Changing Parameters
Drone
Car
Receiving Parameters
Utilizing Command Line
User (Us)

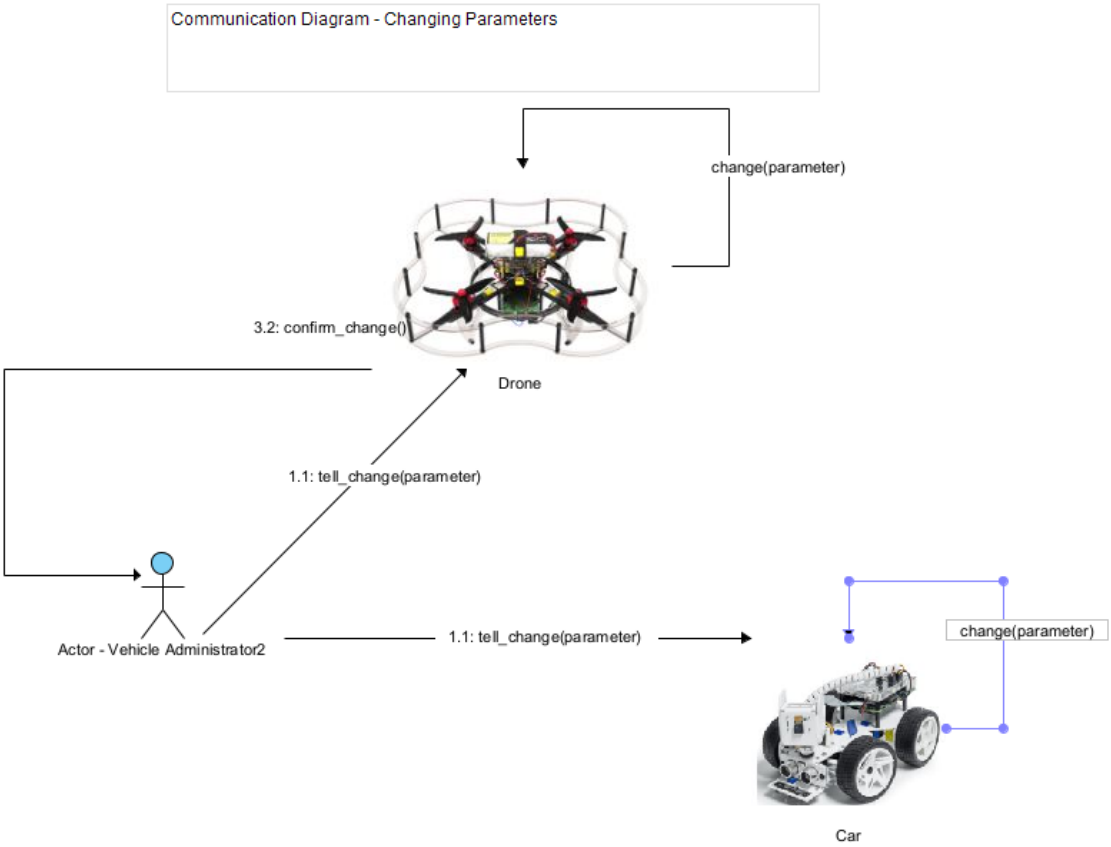
4.4 Sequence Diagrams

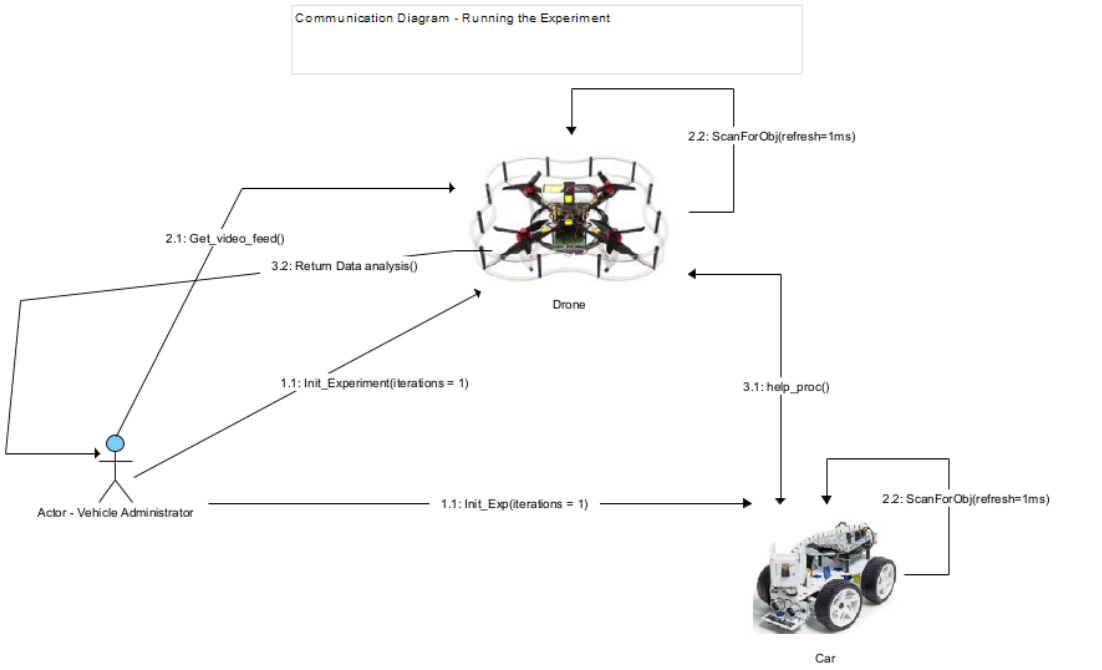


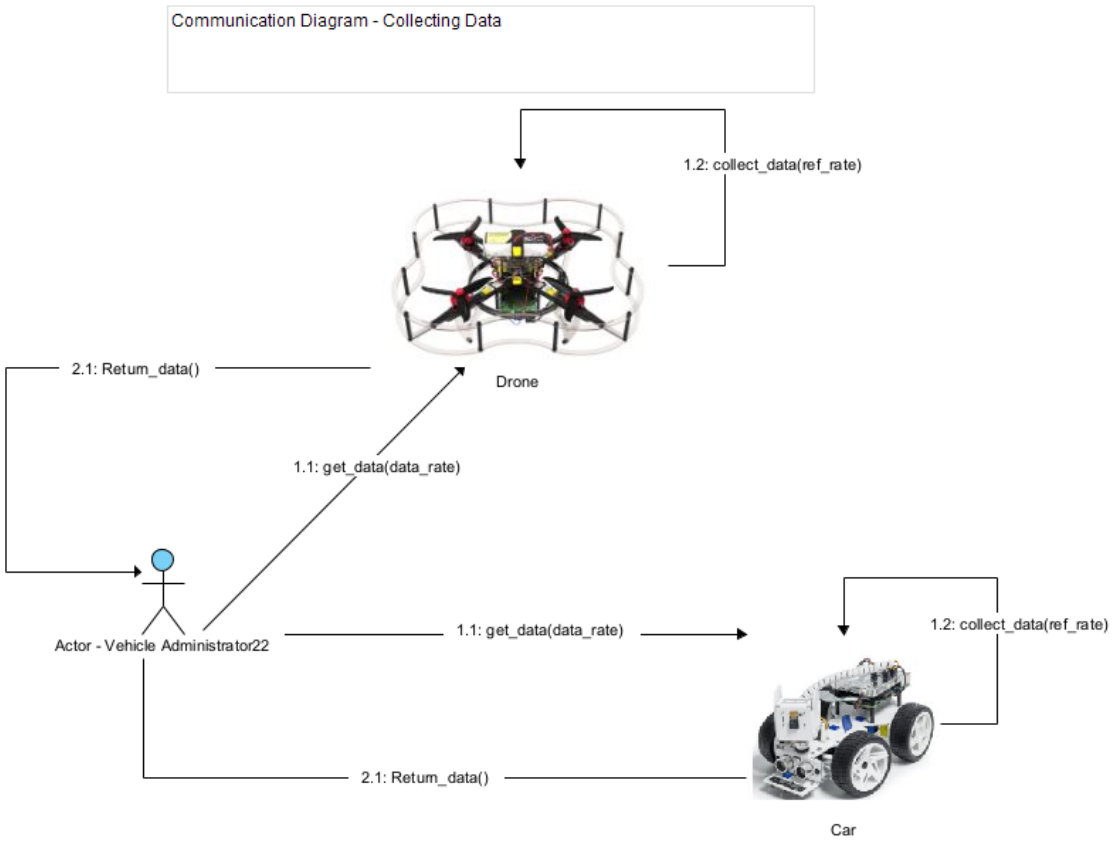




4.5 Communication Diagrams




Communication Diagram - Running the Experiment
2.2: ScanForObj(refresh=1ms)
2.1: Get_video_feed()
3.2: Return Data analysis()
Drone
1.1: Init_Experiment(iterations = 1)
3.1: help_proc()
Actor - Vehicle Administrator
1.1: Init_Exp(iterations = 1)
2.2: ScanForObj(refresh=1ms)
Car


Communication Diagram - Collecting Data
1.2: collect_data(ref_rate)
2.1: Return_data()
Drone
1.1: get_data(data_rate)
Actor - Vehicle Administrator22
1.1: get_data(data_rate)
1.2: collect_data(ref_rate)
2.1: Return_data()
Car

5.0 Behavioral Model and Description

5.1 Description for Software Behavior

5.1.1 Events

A listing of events (control, items) that will cause behavioral change within the system is presented.

External:

- The drone detects a red object.
- The car's sensor senses objects around it like walls or other things in its way.

Internal:

- Images received from drones are processed.
- Collect data from a drone to create data files.

5.1.2 States

A listing of states (modes of behavior) that will result as a consequence of events is presented.

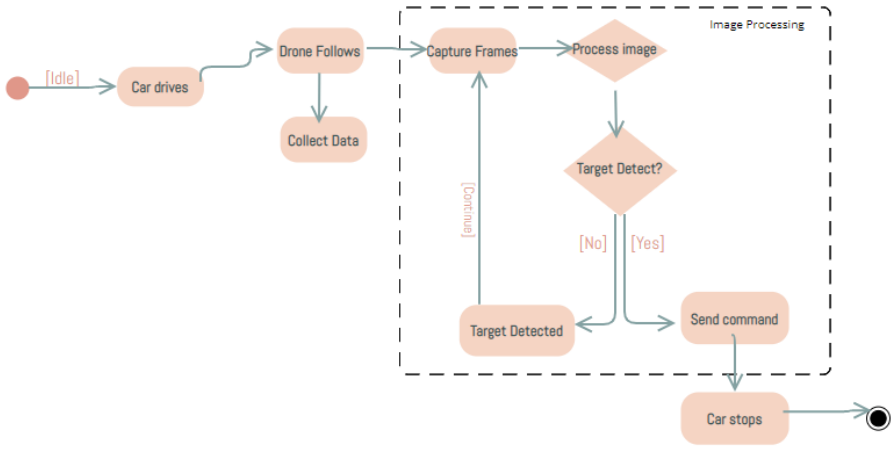
- As the car begins to drive, the drone will follow. As a result, the drone begins to capture images of the car's surroundings to detect a red object.
- The drone follows the car resulting in data collection which includes information about latency, bandwidth, battery life, etc.
- As the drone captures video frames of the car's surroundings, if a target is found, then the car will receive a command to stop from the drone.
- As the car drives, if it detects it is too close to an object it can stop.

5.2 State Transition Diagrams

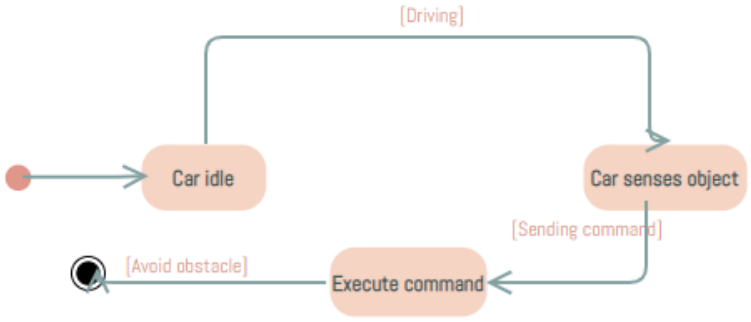
Depict the manner in which the software reacts to external events.

State diagrams are used to show the lifetime behavior of an object within our system as it reacts to external events.

Detection of red object:



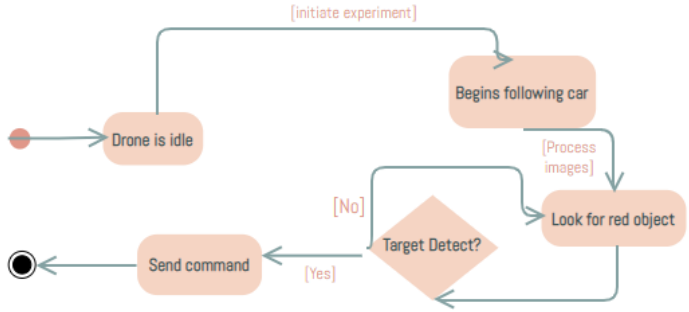
Car sensing object:



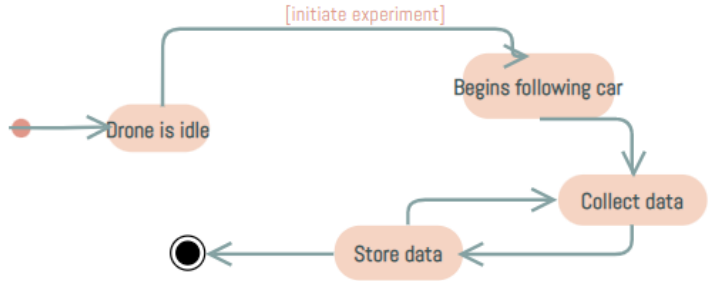
5.3 Activity Diagram

Depict the manner in which the software reacts to internal events.
Activity diagrams show how the system reacts to internal events, which in our case is image processing. It focuses on the flow of events, and the reaction of our system to these internal events.

Processing image:



Collect data:



6.0 Restrictions, Limitations, and Constraints

Time:
We only have two semesters to get the software, hardware, and tests done from scratch with little to no experience with the subject matter beforehand. We have a lot of possible

additions to the scope, but we may be limited by time. One of these additions is the addition of autonomous flight by the drone.

Expertise:
We are working with a lot of hardware and new interfaces/libraries we aren't familiar with. We will have to use up time to learn how to use and work with these interfaces.

Access:
We have limited hardware (computing and vehicles) as well as limited access to them. We can work hands on with the drone and car when on campus and when we are given access to the testing facility, so we must be intentional about when we can be on campus and work with the hardware together.

7.0 Validation Criteria

The tests we are running are focused on black box testing. They are aimed to test the functionality of the system with less emphasis on the performance. Performance is the goal fo the project/research, so the testing does not require a certain level of performance above being functional.

Test Case	Test Name	Preconditions	Steps	Expected Output
1	Car Test 1	Car constructed	1. Construct the car 2. Turn on car 3. Control via controller	Car drives via manual control
2	Car Test 2	Car Test 1 Dnsmasq installed Python program installed	1. Begin dnsmasq 2. Run python program 3. Turn on car 4. Drive car via controller 5. Send message to car via python program and TCP/IP	Car stops driving after receiving packet
3	Drone Test 1	Drone constructed	1. Construct the drone 2. Turn on drone 3. Control via controller	Drone flies via manual control
4	Drone Test 2	Drone Test 1 Sensors calibrated	1. Turn on drone 2. Connect the drone to wifi 3. Go to the drones IP on port 8080/main_camera/image_raw	Camera view shows on
5	Drone Test 3	Drone Test 1 Drone connected to a router	1. Turn on the drone 2. Run server program on a computer 3. Run the client program on the drone through ssh 4. Send message to computers IP	The server receives the message
6	Program Test 1	OpenCV installed Program written	1. Run the program on a computer with a camera 2. Place a red object in the view of the camera	Program identifies red object
7	Drone x Program Test 1	Drone Test 2 Program Test 1 Program placed on drone	1. Turn on the drone 2. Run program on drone via ssh 3. Place a red object in view of the camera	Program identifies red object
8	Drone x	Drone x Program	1. Run a server program	Computer receives

	Program Test 2	**Test 1 Altered program with TCP/IP**	**on a computer 2. Turn on the drone 3. Run the program on the drone via ssh 4. Place a red object in view of the camera**	**a message from the drone after seeing red object**
9	**Drone x Program Test 3**	**Drone x Program Test 2**	**1. Turn on the drone 2. Run the program on the drone via ssh 3. Control via controller 4. Fly over a red object**	**Drone flies while program is running and identifies the red object**
10	**Drone x Car Test 1**	**Drone Test 3 Car Test 2**	**1. Turn on car 2. Car dnsmasq initialized 3. Turn on drone 4. Drone connects to car via wifi 5. Run server program on car 6. Run client program on drone 7. Send message to car**	**Car receives the message from the drone**
11	**Drone x Car Test 2**	**Drone x Car Test 1 Drone x Program Test 3 Car program created and installed**	**1. Turn on the car 2. Car dnsmasq initialized 3. Turn on drone 4. Drone connects to car via wifi 5. Run program on car 6. Run program on drone 7. Control via controller 8. Fly drone over red object**	**Car receives a message from the drone and stops taking inputs.**

8.0 Appendices

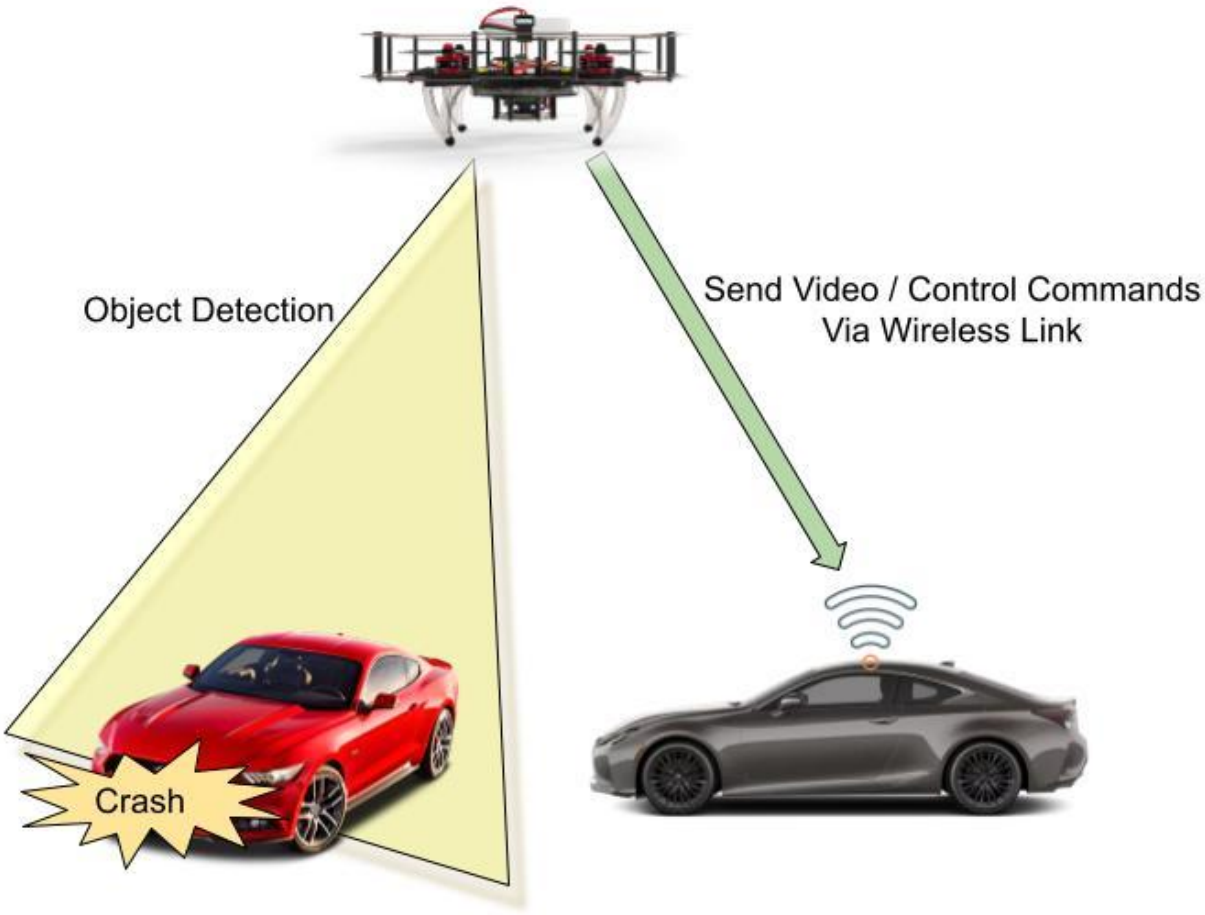
Presents information that supplements the Requirements Specification

8.1 System traceability matrix

A matrix that traces stated software requirements back to the system specification.

ID	Use Case	Requirements	Priority	Depends on (ID)
1	**Run an Experiment**	• **Car driving** • **Drone video** • **Drone following car**	**HIGH**	
2	**Retrieve and Analyze Data**	• **Develop API to collect data** • **Route the data to another API for analysis**	**HIGH**	**1,4**
3	**Change Car and Drone Parameters**	• **Develop API to change variable values**	**MEDIUM**	**1**
4	**Collect/track battery voltage of car and drone**	• **System voltage measurement interface**	**MEDIUM**	**1**

8.2 Diagram: Drone-car Collaboration setup



9.0 References

Hardware and Software Materials and Requirements

- **Raspberry Pi model 4B**
  - https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
  - Used as the onboard computing device for both the car and the drone (each has their own Raspberry Pi computing device) for autonomous driving/flying by taking input from the sensors and processing the information, and also for communications (networking) with between devices (vehicles), such as wireless communications, i.e. WIFI.
- **Raspberry Pi Ai Car Kit (PiCar-X) for Intermediate**
  - https://www.sunfounder.com/products/picar-x
  - https://docs.sunfounder.com/projects/picar-x/en/latest/introduction.html
  - Used as the physical device for the car including motors, frame, driving mechanism, and sensors.
  - The Raspberry Pi OS imager should be used to image the sd card that will serve as the nonvolatile memory unit the Raspberry Pi computer of the car:
    - https://www.raspberrypi.org/software/

    - Here is the Repository that is cloned onto the Raspbian OS image of the Raspberry Pi; it will contain all of the installation files needed to program and control the car:
        - https://github.com/sunfounder/robot-hat
    -

- **Clover Drone 4.2**
    - https://clover.coex.tech/en/
    - Used as the physical device for the drone including motors, frame, propellers, sensors, Electronic Speed Controllers (ESC), GPS, etc.
    - Includes Pixracer R15 Mini Pixracer Autopilot Xracer FMU V4 V1.0 PX4 Flight Controller
        - https://docs.px4.io/main/en/flight_controller/pixracer.html
- **Python Programming Language**
    - https://www.python.org/
    - We will use the Python programming language for both the car and the drone.
- **OpenCV-Python Library**
    - https://pypi.org/project/opencv-python/
    - This is a Python vision analysis library that has been adapted from a library originally written for C++
    - We will use it to analyze vision data collected from cameras on both the car and the drone.
- **Raspbian OS builds with Linux Kernel**
    - https://www.kernel.org/
    - Both the car and the drone have their own onboard computing device (Raspberry Pi model 4B) with a custom modified version of the Raspbian Operating System image that uses the Linux Kernel.
    - Additionally, the Clover 4.2 Drone uses the ROS robotic framework used for advanced robotic distributed systems.
        - https://wiki.ros.org/
    - Here is the image used for the Clover 4.2 Drone Raspberry Pi computer:
        - https://github.com/CopterExpress/clover/releases/tag/v0.23
        - Image features:
            - Raspbian Buster
            - ROS Noetic
            - Configured networking
            - OpenCV
            - Mavros
            - Periphery drivers for ROS (GPIO, LED strip, etc)
            - Aruco_pose package for marker-assisted navigation
            - Clover package for autonomous drone control
- **Q Ground Control**
    - https://docs.qgroundcontrol.com/master/en/
    - This is an open source software used to communicate with and calibrate and configure a drone's flight controller firmware. We will use this to calibrate the drone and manage the flight controller's parameters and how the flight system of the drone uses and responds to sensor data.
    - Here is the firmware image used for our flight controller:
        - https://github.com/CopterExpress/Firmware/releases/tag/v1.8.2-clover.13
- **Clover Drone Simulation virtual machine (VM) image**

- https://github.com/CopterExpress/clover_vm
- https://github.com/CopterExpress/clover_vm/releases/tag/v1.3
- https://github.com/CopterExpress/clover_vm/releases
  - Link to actual vm .ova vm image file
- This is the virtual machine image (using Virtual Box) used to run the simulation software used to simulate programmed autonomous flights for the Clover 4.2 Drone.
- Image contains:
  - Ubuntu 20.04 Focal.
  - ROS Noetic.
  - PX4 autopilot, QGroundControl.
  - Preinstalled Clover and Clover simulation packages.
  - Shortcuts for running Clover simulator.
  - VSCode.
  - Useful robotics-related software.

- **Drone Simulation Environment (Using Gazebo software)**
  - The simulation environment is based on the following components: Gazebo, a state-of-the-art robotics simulator;
    - http://gazebosim.org/
  - PX4, specifically its SITL (software-in-the-loop) components;
    - https://px4.io/
  - sitl_gazebo package containing Gazebo plugins for PX4;
    - https://github.com/PX4/sitl_gazebo
  - ROS packages and Gazebo plugins
  - **Note:** all of the above components are installed on the Clover Drone Simulation VM in order to do simulation programming without the Raspberry Pi on board computing device of the drone. This allows for programming and simulation without needing the physical drone present.
- **Etcher - Flashing Software**
  - https://www.balena.io/etcher
  - This is the software used to flash the micro-SD card with the respective OS (drone or car) used as the nonvolatile memory unit for the Raspberry Pi computers.
- **What is ROS (Robot Operating System)?**
  - http://wiki.ros.org/
  - *ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.*
- **MAVROS (package)**
  - http://wiki.ros.org/mavros
  - MAVROS -- MAVLink extendable communication node for ROS with proxy for Ground Control Station.
  - This package provides a communication driver for various autopilots with MAVLink communication protocol. Additional it provides UDP MAVLink bridge for ground control stations (e.g. QGroundControl).
- **MAVLINK (package)**
  - http://wiki.ros.org/mavlink?distro=noetic
  - MAVLink message marshaling library. This package provides C-headers and C++11 library for both 1.0 and 2.0 versions of protocol. For pymavlink use separate install via rosdep (python-pymavlink).

    - This package provides a communication library for various autopilot systems. This package contains both C-headers and pymavlink.
    - **mavros_msgs/Mavlink Message**
        - http://docs.ros.org/en/api/mavros_msgs/html/msg/Mavlink.html
            - The above link is the header file to the mavrov_msgs class which is used as a message transport class to convert between ROS and MAVLink message types.
            - Used to transport mavlink_message_t via ROS topic
        - **Here is the documentation for the mavros_msgs class:** http://docs.ros.org/en/api/mavros_msgs/html/index-msg.html
        - mavros_msgs defines messages for MAVROS
        - And here is another important class **sensor_msgs**: http://wiki.ros.org/sensor_msgs?distro=noetic
    - At least this package uses its own bundled (or installed by pip) mavlink headers or pymavlink: mavlink_ros, rospilot, roscopter, autopilot_bridge, px4-ros-pkg.
- **ros_core**
    - Here is the documentation for **ros_core** class**:** http://wiki.ros.org/ros_core?distro=noetic
    - A metapackage to aggregate the packages required to use publish / subscribe, services, launch files, and other core ROS concepts.
- **libmavcon**
    - http://wiki.ros.org/libmavconn?distro=noetic
    - MAVLink communication library. This library provide unified connection handling classes and URL to connection object mapper. This library can be used in standalone programs.
- **catkin library**
    - http://wiki.ros.org/catkin?distro=noetic
    - Low-level build system macros and infrastructure for ROS.

Research Background Sources

1. S. A. Hadiwardoyo, E. Hernández-Orallo, C. T. Calafate, J. -C. Cano and P. Manzoni, **"Evaluating UAV-to-Car Communications Performance: Testbed Experiments,"** 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 2018, pp. 86-92, doi: 10.1109/AINA.2018.00025.
    - **Abstract:** Vehicular networks are gradually emerging due to the expected benefits in terms of enhanced safety and infotainment services. However, outside main metropolitan areas, little infrastructure currently deployed, which may hinder these services. To mitigate this problem, Unmanned Aerial Vehicles (UAVs) are envisioned as mobile infrastructure elements, supporting communications when fixed infrastructure is missing. This way, in emergency situations, UAVs can offer services to vehicles including broadcasting alerts or acting as message relays between ground vehicles. Our work attempts to be a first step in this direction by presenting experimental measurement results regarding communications quality between cars and UAVs. In particular, we varied the altitude of the drone and its antenna orientation, and the car's antenna location to assess their impact on performance. Based on the experimental results achieved, we find that UAVs communicating in the 5 GHz band using IEEE 802.11 technology are able to deliver data to moving cars

within a range of more than three kilometers, achieving more than 0.5 of packet delivery ratio up to 2.5 kilometers under the optimal configuration settings.
    - **URL:** https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8432227&isnumber=8432202
2. J. Yoon, I. Kim, W. Chung and D. Kim, **"Fast and accurate car detection in drone-view,"** 2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia), Seoul, Korea (South), 2016, pp. 1-3, doi: 10.1109/ICCE-Asia.2016.7804775.
    - **Abstract:** With the development of drones, aerial images are used in a variety of applications. We propose a way to detect cars in drone-view fast and accurately. For this purpose we propose a feature called G-ORF for effective feature description. Also we designed a pose classifier and bin-specific weighted Linear Discriminant Analysis (wLDA) classifier for pose classification and binary classification of each pose respectively. Our method showed real-time performance in HD (1280×720) video in a PC environment with high detection accuracy.
    - **URL:** https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7804775&isnumber=7804716
3. Yildiz, Melih, Burcu Bilgiç, Utku Kale, and Dániel Rohács. 2021. **"Experimental Investigation of Communication Performance of Drones Used for Autonomous Car Track Tests"** Sustainability 13, no. 10: 5602.
    - **Abstract:** Autonomous Vehicles (AVs) represent an emerging and disruptive technology that provides a great opportunity for future transport not only to have a positive social and environmental impact but also traffic safety. AV use in daily life has been extensively studied in the literature in various dimensions, however; it is time for AVs to go further which is another technological aspect of communication. Vehicle-to-Vehicle (V2V) technology is an emerging issue that is expected to be a mutual part of AVs and transportation safety in the near future. V2V is widely discussed by its deployment possibilities not only by means of communication, even to be used as an energy transfer medium. ZalaZONE Proving Ground is a 265-hectare high-tech test track for conventional, electric as well as connected, assisted, and automated vehicles. This paper investigates the use of drones for tracking the cars on the test track. The drones are planned to work as an uplink for the data collected by the onboard sensors of the car. The car is expected to communicate with the drone which is flying in coordination. For the communication 868 MHz is selected to be used between the car and the drone. The test is performed to simulate different flight altitudes of drones. The signal strength of the communication is analyzed, and a model is developed which can be used for the future planning of the test track applications.
    - **URL:** https://doi.org/10.3390/su13105602
4. Barbeau, Michel, Joaquin Garcia-Alfaro, and Evangelos Kranakis. 2022. **"Research Trends in Collaborative Drones"** Sensors 22, no. 9: 3321.
    - **Abstract:** The last decade has seen an explosion of interest in drones—introducing new networking technologies, such as 5G wireless connectivity and cloud computing. The resulting advancements in communication capabilities are already expanding the ubiquitous role of

drones as primary solution enablers, from search and rescue missions to information gathering and parcel delivery. Their numerous applications encompass all aspects of everyday life. Our focus is on networked and collaborative drones. The available research literature on this topic is vast. No single survey article could do justice to all critical issues. Our goal in this article is not to cover everything and include everybody but rather to offer a personal perspective on a few selected research topics that might lead to fruitful future investigations that could play an essential role in developing drone technologies. The topics we address include distributed computing with drones for the management of anonymity, countering threats posed by drones, target recognition, navigation under uncertainty, risk avoidance, and cellular technologies. Our approach is selective. Every topic includes an explanation of the problem, a discussion of a potential research methodology, and ideas for future research.

- **URL:** https://doi.org/10.3390/s22093321

5. Textbook: **Software Engineering A PRACTITIONER ' S APPROACH EIGHTH EDITION, by Roger S. Pressman, Ph.D. and Bruce R. Maxim, Ph.D.**
6. **Limitations of Camera and Radar Based ADAS**
   - https://www.cepton.com/twitter/limitations-of-camera-and-radar-based-adas
   - This is a link to a project that contains a threaded discussion about the failures of camera, lidar, and other sensors on Tesla vehicles as Tesla races towards making autonomous vehicles a reality. We can use this to help us keep track of some of the difficulties leading industries like Tesla faces in computer vision systems for autonomous vehicles.
7.