



Drone project

Students

Ofir zeilig

Natan sela

Supervisors

David Dovrat

Yaron Honen

PROJECT IDEA	3
USER MANUAL.....	4
SITL – FLIGHT SIMULATOR	4
CONTROLLING SERVO.....	9
ARDUPILOT	13
DRONEKIT	14
HOW TO USE OUR PROJECT.....	15
REAL SENSE.....	20
LIBRARIES THAT WE USED.....	22
HARDWARE WE USED	24
EXPLANATION ABOUT THE PROJECT.....	26
CREATING A FRAMEWORK FOR COMPUTER VISION ON A DRONE.....	26
DIFFICULTIES WE ENCOUNTERED AND INSIGHTS	30
WHAT HAVE WE LEARNT:	36
IN CONCLUSION:.....	46

Project idea

Drones can be used to various tasks and replace humans in many aspects in our lives. Our goal was to create a system that will enable to control a drone using computer vision that will enable it to fly autonomously. Eventually, we created a framework that wraps all the parts that control the flight functionality of the drone in a way that future projects could use it could focus only on computer vision.



SITL – FLIGHT SIMULATOR

In order to test our code before trying to launch a drone we used the Silt flight simulator.

SITL allows you to run ArduPilot on your PC directly, without any special hardware. It takes advantage of the fact that ArduPilot is a portable autopilot that can run on a very wide variety of platforms. Your PC is just another platform that ArduPilot can be built and run on.

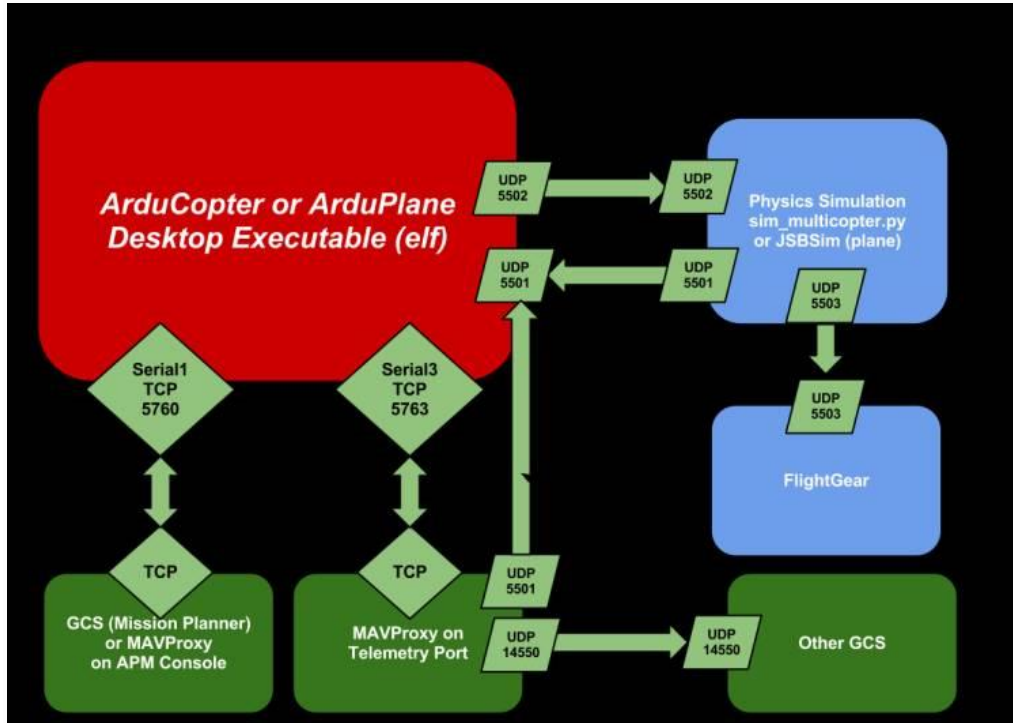
When running in SITL the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of vehicle simulators built in, and can interface to several external simulators. This allows ArduPilot to be tested on a very wide variety of vehicle types. For example, SITL can simulate:

- multi-rotor aircraft
- fixed wing aircraft
- ground vehicles
- underwater vehicles
- camera gimbals
- antenna trackers
- a wide variety of optional sensors, such as Lidars and optical flow sensors

Adding new simulated vehicle types or sensor types is straightforward.

A big advantage of ArduPilot on SITL is it gives you access to the full range of development tools available to desktop C++ development, such as interactive debuggers, static analyzers and dynamic analysis tools. This makes developing and testing new features in ArduPilot much simpler.

The sitl architecture is:



How to use:

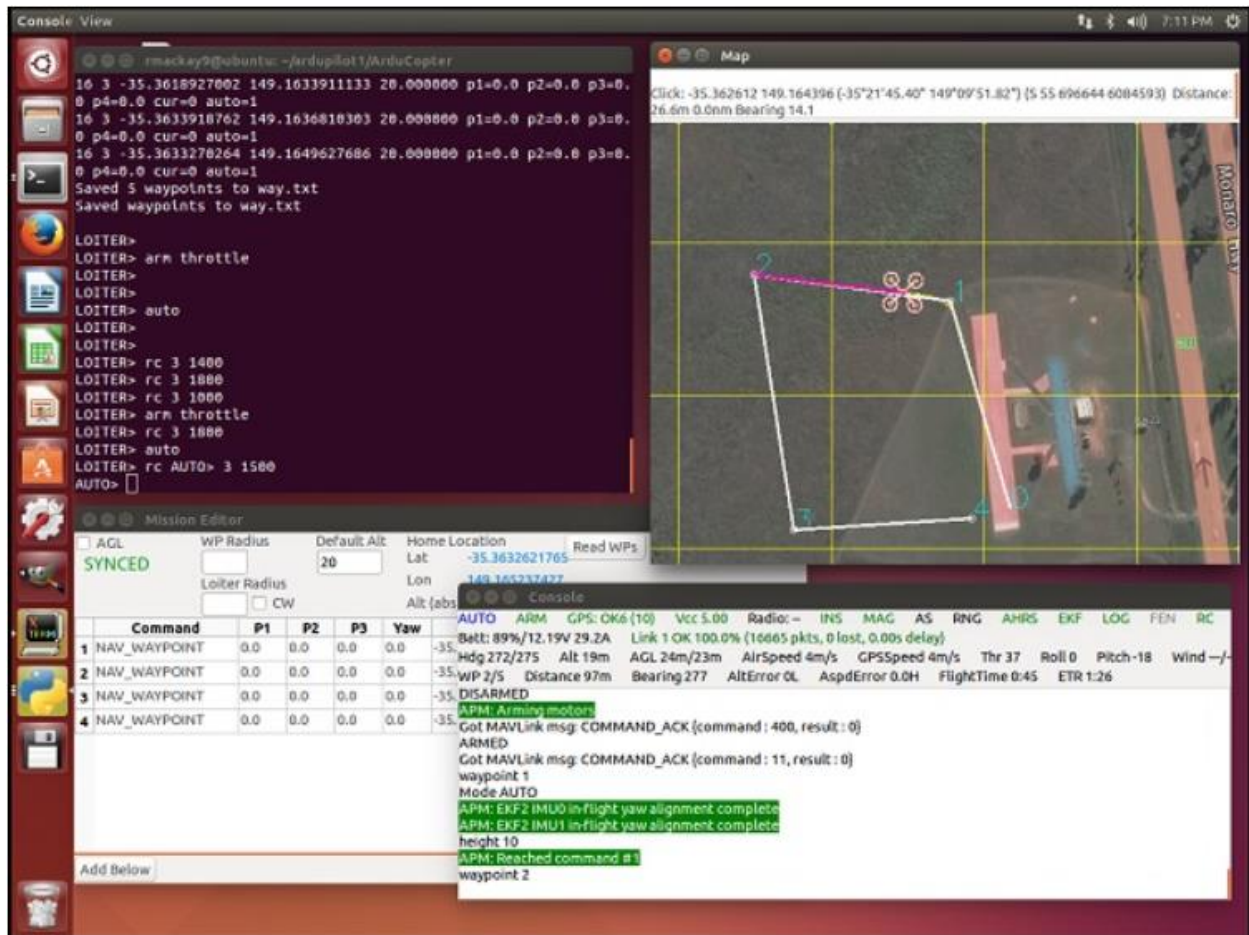
First, you need to setup the simulator using the guides that are provided by the framework. Then, to start your simulator open Cygwin64 and type:

```
Cd ~ardupilot
```

Then :

```
./Tools/autotest/sim_vehicle.py -v ArduCopter --map --console --  
out=tcpin:0.0.0.0:14552
```

And the following screens should appear:



If the map isn't available, press view->service->googleMap

And now the drone is ready to fly, The way to control the drone is using MavProxy which is a fully-functioning GCS for UAV's. The intent is for a minimalist, portable and extendable GCS for any UAV supporting the MAVLink protocol (such as the APM).

- It is a command-line, console based app. There are plugins included in MAVProxy to provide a basic GUI.
- Can be networked and run over any number of computers.

- It's portable; it should run on any POSIX OS with python, pyserial, and select() function calls, which means Linux, OS X, Windows, and others.
- The light-weight design means it can run on small netbooks with ease.
- It supports loadable modules, and has modules to support console/s, moving maps, joysticks, antenna trackers, etc

Tab-completion of commands.

Mavproxy offers various commands which controls the vehicle:

MAVProxy Cheatsheet (V 1.5.7)

Link		Manual mode	mode manual
		Fly to specific location	mode guided LAT LON ALT
List all links	link list		
Set link n to primary	set link N		
Add new link	link add X	Relays	
Remove link	link remove N	Set relay	relay set N [0 1]
		Set output servo	servo set N PWM
Map			
Show waypoints	wp list	Terrain	
Show fence	fence list		
Enable geofence	fence enable	Get terrain height at location	terrain check LAT LON
Disable geofence	fence disable		
Arming			
Arm Vehicle	arm throttle		
Disarm Vehicle	disarm		
Force disarm (heli)	disarm force		
AUTO Flight			
Set current wp	wp set N		
Engage auto mode	auto		
Override auto speed (m/s)	setspeed N		
Parameters			
Get param value	param show X		
Set param	param set X N		
Download param defns	param download		
Get param help	param help X		
Flight modes			
Loiter mode	mode loiter		
Return to Launch	mode rtl		

Reference:

http://ardupilot.github.io/MAVProxy/html/_static/files/MAVProxyCheetsheet.pdf

The last thing you need in order to control the simulator using a code is the connection string that is required to send to the dronekit command: connect.

`connect(tcp:127.0.0.1:14552, wait_ready=True)` and a vehicle will return that is ready for instructions.

CONTROLLING SERVO WITH UP-BOARD USING PINS

Servomotor

A servomotor is a rotary actuator that allows for precise control of angular position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

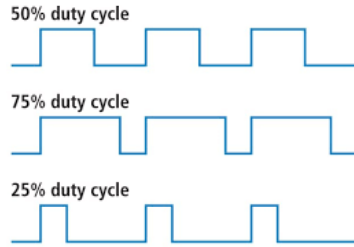
A servomotor is a closed-loop servomechanism that uses position feedback to control its motion and final position. The input to its control is a signal (either analogue or digital) representing the position commanded for the output shaft.



Pwm Signal

Pulse-width modulation (PWM), is a modulation technique used to encode a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially for motors.

The term duty cycle describes the proportion of 'on' time to the regular interval or 'period' of time. a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.



Pwm pins

Using an up-board we have 40 pins connected to our board, some of them support pwm. Each pwm pin has a physical number and a pwm number. For example, the pin we used is physical pin 32 which is pwm pin 0

UP - Pinout																																							
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40																				
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39																				
1 3V3	2 5V	3 GPIO0/I2C_SDA	4 5V	5 GPIO1/I2C_SCL	6 Ground	7 GPIO2/ADC-Input	8 GPIO15/UART_TX	9 Ground	10 GPIO16/UART_RX	11 GPIO3	12 GPIO17/I2S_CLK	13 GPIO4	14 Ground	15 GPIO5	16 GPIO18	17 3V3	18 GPIO19	19 GPIO6/SPI_MOSI	20 Ground	21 GPIO7/SPI_MISO	22 GPIO20	23 GPIO8/SPI_CLK	24 GPIO21/SPI_CS0N	25 Ground	26 GPIO22/SPI_CS1N	27 GPIO9/I2C0_SDA	28 GPIO23/I2C0_SCL	29 GPIO10	30 Ground	31 GPIO11	32 GPIO24/PWM0	33 GPIO12/PWM1	34 Ground	35 GPIO13/I2S_FRM	36 GPIO25	37 GPIO14	38 GPIO26/I2S_DATAIN	39 Ground	40 GPIO27/I2S_DATAOUT

Pin Number	Signal Name	Color Scheme 1 (Futaba)	Color Scheme 2 (JR)	Color Scheme 3 (Hitec)
1	Ground	Black	Brown	Black
2	Power Supply	Red	Red	Red or Brown
3	Control Signal	White	Orange	Yellow or White

Servo connection Color Coding

The pins we used are: 1 for Power supply, 6 for ground, and 32 for pwm0.

How to Control a servo via PWM on Ubuntu

Exporting a pin

Before sending any command, we need to open a new connection on whatever pin we've chosen. To do that, chose a pin supporting pwm, then through terminal write the pin number to a file named export situated in the pwm directory

- `echo <pwm number> > /sys/class/pwm/pwmchip0/export`

after exporting the pin, a new directory will be created named `pwm<pwm number>` containing several files.

Unexporting a pin

At the end, when the pin is not needed anymore, it need to be unexported.

- `echo <pwm number> > /sys/class/pwm/pwmchip0/unexport`

Setting pin's Direction

To begin using the pin, we first need to set it direction i.e. do we want to use the pin for sending information or for receiving it. Notice that after exporting the pin a new directory was created named `pwm<pwm number>` (for example `pwm0` in our case) containing several files. One of those files is named `direction` and need to be set to `out`

- `echo out > /sys/class/pwm/pwmchip0/pwm<pwm num>/direction`

Setting Period time and Duty Cycle

As explained, a pwm pin is controlled with the period time and duty cycle of the sent signal.

To set both values, use the same commands we have used so far,

- `echo <value> > /sys/class/pwm/pwmchip0/pwm<pwm num>/period`
- `echo <value> > /sys/class/pwm/pwmchip0/pwm<pwm num>/duty_cycle`

Enable the pin

Now that the pin is configured, the last thing to do is start sending the signal, to do that, we need to enable the pin – send 1 to the file “enable” located at the new directory that were created after exporting the pin.

- `echo 1 > /sys/class/pwm/pwmchip0/pwm<pwm num>/enable`

Writing Permission

While writing to these files they may be some permissions problems, to solve that you can set the permission correctly with `chmod` (note that after exporting the pin, new files will be created with permissions issues) or simply use these commands as root with `sudo`.

Working with our python class

We offer a new class named `Pwm`, which manage all pwm setting-up.

- `def __init__(self , p=0):` - create a new pwm controller with pin number p.
- `def setupPin(self , enable=True):` - setup all needed
- `def turnTo(self, degree):` - turn the servo to angle degree.

There are more specific functions that do every step mentioned before, but all of them are hidden in those methods.

ARDUPILOT

Ardupilot is the most advanced, full-featured and reliable open source autopilot software available. It has been developed over 5+ years by a team of diverse professional engineers and computer scientists. It is the only autopilot software capable of controlling any vehicle system imaginable, from conventional airplanes, multicopters, and helicopters, to boats and even submarines. And now being expanded to feature support for new emerging vehicle types such as quad-planes and compound helicopters.



Installed in over 1,000,000 vehicles world-wide, and with its advanced data-logging, analysis and simulation tools, Ardupilot is the most tested and proven autopilot software. The open-source code base means that it is rapidly evolving, always at the cutting edge of technology development. With many peripheral suppliers creating interfaces, users benefit from a broad ecosystem of sensors, companion computers and communication systems. Finally, since the source code is open, it can be audited to ensure compliance with security and secrecy requirements.

The software suite is installed in aircraft from many OEM UAV companies, such as 3DR, jDrones, PrecisionHawk, AgEagle and Kespry. It is also used for testing and development by several large institutions and corporations such as NASA, Intel and Insitu/Boeing, as well as countless colleges and universities around the world.

Ardu pilot documentation can be found [here](#)

DRONEKIT

DroneKit-Python allows developers to create apps that run on an onboard companion computer and communicate with the ArduPilot flight controller using a low-latency link. Onboard apps can significantly enhance the autopilot, adding greater intelligence to vehicle behaviour, and performing tasks that are computationally intensive or time-sensitive (for example, computer vision, path planning, or 3D modelling). DroneKit-Python can also be used for ground station apps, communicating with vehicles over a higher latency RF-link.

The API communicates with vehicles over MAVLink. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

API features

The API provides classes and methods to:

- Connect to a vehicle (or multiple vehicles) from a script
- Get and set vehicle state/telemetry and parameter information.
- Receive asynchronous notification of state changes.
- Guide a UAV to specified position (GUIDED mode).
- Send arbitrary custom messages to control UAV movement and other hardware (GUIDED mode).
- Create and manage waypoint missions (AUTO mode).
- Override RC channel settings.

A complete API reference is available [here](#).

HOW TO USE OUR PROJECT

Run the project

In order to start the project one will need to run the following commands:

- Open a simulator – SITL (you can just run the commands in the file `sitl_lunch.py` – not run the script! A python shell must be open with the simulator in it)
- Optional – Open a map module (In case you didn't succeed downloading the official map simulator) `python MapScriptListener.py`
- `Python __main.py__`

Problems...

- While trying to run the project, you can encounter few problems. To solve them try running the help script `DroneRequiements.sh`
- Opening an ssh connection might not work, the first line of the `DroneRequiements.sh` script might do the work.

•
`__main__`

To run our project, after lunching the simulator, you should run the main script with a simple python command.

The main script consists of setting up all the project, and letting him run by himself. For that few things are created on the main script:

- A config file is read, from which we can receive the wanted connection string and additional parameters.
- A flight object is created – any object inheriting from Flight module i.e. implements `addSafeCommands` and `run` methods, would do it. In our case, we use `DKFlight` (Dronekit) which connect and control the vehicle using dronekit library.
- An HLC object is created which is responsible for translating high level instructions received from the controller and prepare the real commands for the Flight module in a way that would be threads safe.

- Finally, the controller is created and the main call the controller's method run which will start the entire process and make it works.

Flight

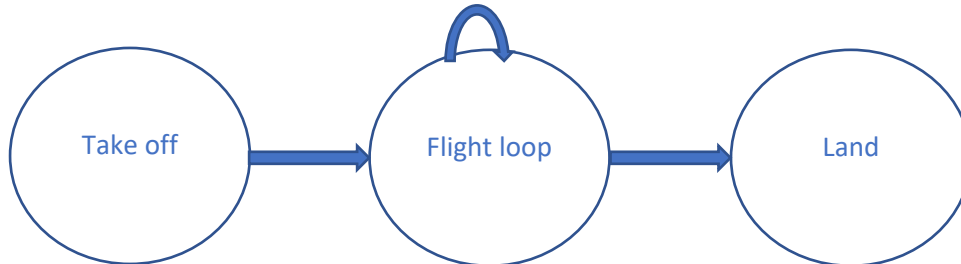
The flight module is the one responsible for talking to the drone consistently. Any object inheriting from Flight module must implements the following methods:

`def run(self):`

this method will automatically start the thread life, and make the drone begin flying.

DKFlight

The DKflight control the drone using dronekit library, that means it connect to the drone or simulator using dronekit connect function and control the drone using Mavlink messages.



After the method run is called, the new thread is created the thread begin and do 3 things sequentially:

- **Taking off** – the Flight will ask from the drone to take off to a certain height and maintain the hight.

- **Flight loop** – During this loop the Flight module will look up for the recent command in the shared safe command variable and send it via Mavlink to the drone and ask it for telemetry data which it will put in the safe Response shared variable so that the HLC could ask for the recent telemetry data anytime without waiting.
- **Land** – the Fight module will ask the drone to return to the home location and land.

After those three missions are done, the Flight thread will end.

HLC

The HLC module is responsible of receiving High level instructions (such as 'left', 'right', etc), preparing the real direction for the Flight module (as vector direction) and send it to him thread safely. To use correctly the HLC module one must use the following methods:

- **Takeoff** – with parameter wait which indicate whether the method will wait for the vehicle to take off or not.
- **hasTakenOff** - returns whether the vehicle has finished taking off or not
- **gotoDirection** – receive a high level direction as specified and send it to the Flight thread.
- **getTelemetry** – get the most recent telemetry data.
- **Finish** – when the controller want to finish the flight and land, it should call HLC's Finish method.

Controller

The Controller module is the one controlling the flight. It's flight decisions can be based on input coming from the user (keyboard) or sensors (camera). It is an abstract class, i.e. there is no any instance of controller in the project but a lot of classes inherits from controller and are implemented.

The controller init method first of all receive an HLC object so it could control the flight and send flight directions using it.

Any class inheriting from controller should implement init as it needs and receive the needed modules (such as camera, servo controller, etc.) and must implement the following method:

```
def start(self):
```

The method will automatically start the flight and after taking off start compute and send the wanted directions.

Hard Coded Controller

The hard coded controller is a demo controller which only take off, send 4 simple direction commands, and land.

Keyboard Controller

The keyboard controller is a controller which make its decisions based on user input. After taking off, the controller waits for user input and send flight commands according to the received direction.

CV Controller

This is a stab controller, i.e. it is not ready to use and must be implemented!

```
while True:
    frames = self.camera.getFrames()

    #####
    ##### insert your code here #####
    #####

    sleep(2) # do some image processing

    #####
    #####

    #send directions
    direction = 'stay'
    self.hlc.gotoDirection(direction)
```

This Controller is ready to implement while the only missing code is given recent frames, after image processing, decide the next flight direction and other commands (such as rotating the camera for example.)

Safe Command and Safe Response

These are two class representing a thread protected variable i.e. a variable that when accessing it, the Safe class is responsible of acquiring a lock before and releasing it after it.

Camera Handler

This class is an abstract class responsible of connecting to a camera and receiving frames from it. Any class inheriting from Camera Handler must implement the following method:

```
def getFrames(self):
```

in which the inheriting module will ask for frames from the camera and return them as cv matrices.

RSCamera

The RSCamera is a module connecting to a realsense 2 camera. To do that, the module open a connection with the camera and when asked for, it get the three recent frames – color frame, I.R frame, Depth frame.

PWM

This class knows how to control servos using pwm signals. It assumes the connected servo is a 180-degree servo. To more information about controlling servos via PWM pins look up for Controlling Servo with Up-Board using pins

REAL SENSE

Intel RealSense Technology, is a suite of depth and tracking technologies designed to give machines and devices depth perceptions capabilities that will enable them to “see” and understand the world. There are many uses for these computer vision capabilities including autonomous drones, robots, AR/VR, smart home devices amongst many others broad market products. RealSense technology is made of Vision

Processors, Depth and Tracking Modules, and Depth Cameras, supported by an open source, cross-platform SDK called librealsense that simplifies supporting cameras for third party software developers, system integrators, ODMs and OEMs.

D435



The D435 Camera

By introducing the Intel® RealSense™ Depth Camera D435 into the Intel RealSense™ product lineup, Intel continues our commitment to developing cutting-edge new vision sensing products. Placing an Intel module and vision processor into a small form factor results in a combined solution ideal for development or productization. Lightweight, powerful, and low- cost, this complete package pairs with customizable software to enable the development of next-generation sensing solutions and devices that can understand and interact with their surroundings.

For more information about Pyrealsense enter [here](#)

Pyrealsense library

To use the intel realsense camera programmatically, we use the library Pyrealsense.

The library allows depth and color streaming, and provides intrinsic and extrinsic calibration information. The library also offers synthetic streams (pointcloud, depth aligned to color and vise-versa), and a built-in support for record and playback of streaming sessions.

For more information about Pyrealsense enter [here](#).

LIBRARIES THAT WE USED

Dronekit:

DroneKit-Python allows developers to create apps that run on an onboard companion computer and communicate with the ArduPilot flight controller using a low-latency link. Onboard apps can significantly enhance the autopilot, adding greater intelligence to vehicle behaviour, and performing tasks that are computationally intensive or time-sensitive (for example, computer vision, path planning, or 3D modelling). DroneKit-Python can also be used for ground station apps, communicating with vehicles over a higher latency RF-link.

The API communicates with vehicles over MAVLink. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

pyrealsense2

The library allows depth and color streaming, and provides intrinsic and extrinsic calibration information. The library also offers synthetic streams (pointcloud, depth aligned to color and vice-versa), and a built-in support for record and playback of streaming sessions.

opencv

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point

clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching streetview images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

HARDWARE WE USED

Px4:

PX4 is powerful open source autopilot flight stack.

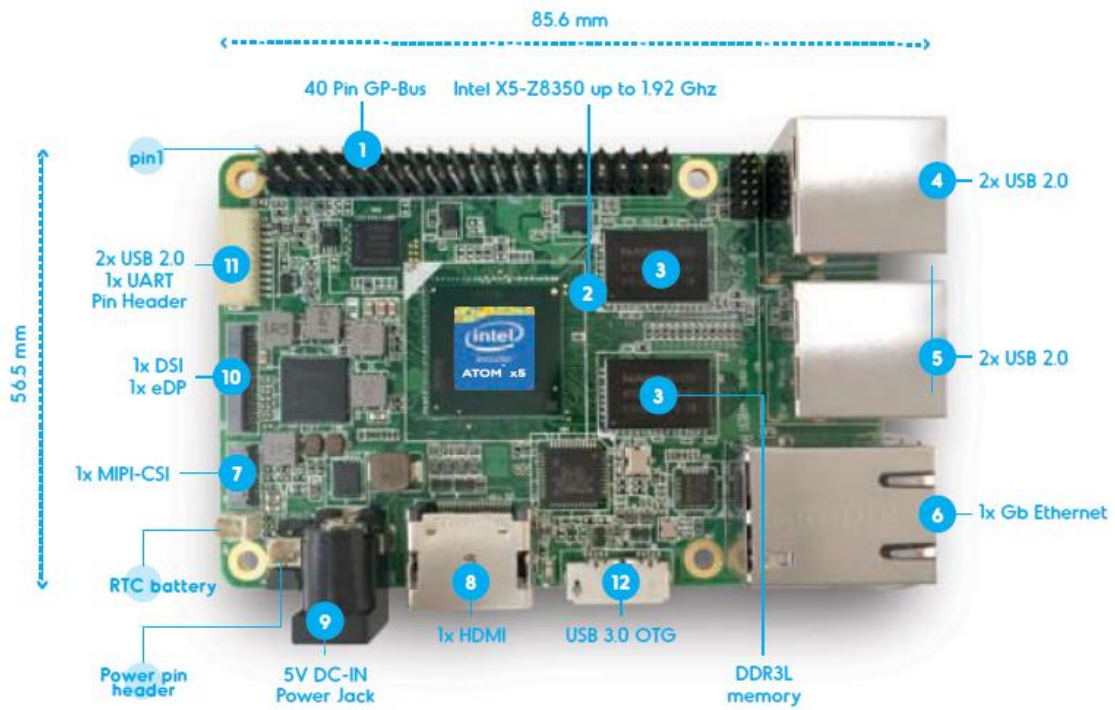
Some of PX4's key features are:

- Controls many different vehicle frames/types, including: aircraft (multicopters, fixed wing aircraft and VTOLs), ground vehicles and underwater vehicles.
- Great choice of hardware for vehicle controller, sensors and other peripherals.
- Flexible and powerful flight modes and safety features.

upBoard:

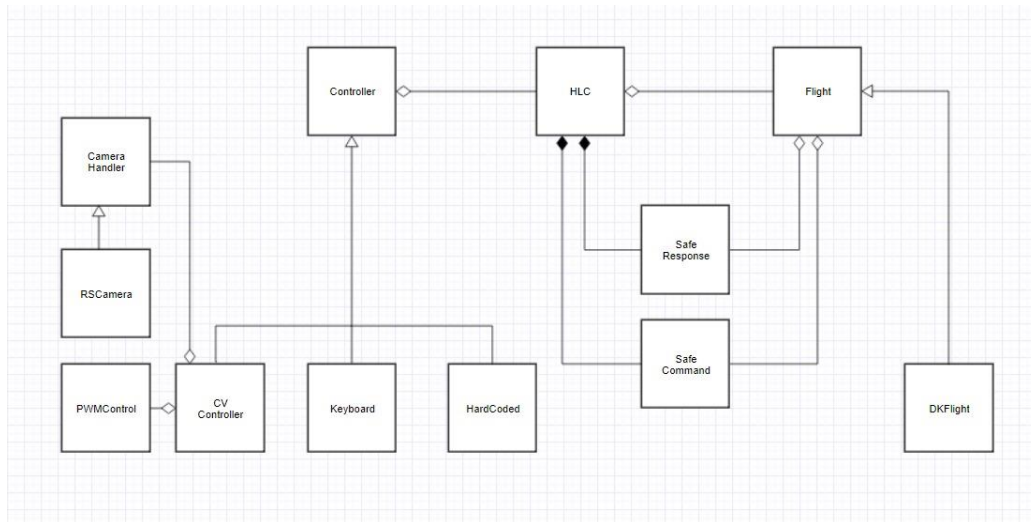
UP is a credit card size board with the high performance and low power consumption features of the latest tablet technology: the Intel® Atom™ x5 Z8350 Processors (codename Cherry Trail) 64 bits up to 1.92GHz. The internal GPU is the new Intel Gen 8 HD 400 with 12 Execution Units up to 500MHz to deliver extremely high 3D graphic performance. UP is equipped with 1GB/2GB/4GB DDR3L RAM and 16GB/32GB/64GB eMMC.

UP has 40-pin General Purpose bus which provides the freedom to makers to build up their shield. There are more interfaces available, such as 4x port USB2.0 on connectors, 2x port USB2.0 + 1x UART on header, 1x USB 3.0 OTG, 1x Gbit Ethernet (full speed), 1x DSI/eDP port, 1x Camera (MIPI-CSI), 1x HDMI, RTC.



* The information is from the upboard spec

CREATING A FRAMEWORK FOR COMPUTER VISION ON A DRONE



Here we can see the class diagram of our code, the main modules of our code are the controller and the flight modules.

The Controller is a module responsible for making the decision and sending flight commands, based on hard coded data, user input, or live image processing.

The Flight module is an object running on a separate thread, and constantly sending flight control commands to the drone or simulator.

The HLC module (High Level Controller) is responsible of translating high level instructions received from the controller, (such as left, right, etc.) to instructions the Flight module will understand. In addition, since the Flight object runs on a second thread the HLC module is responsible for the communication between those threads

Communication between the two threads is done with shared data. Two objects named Safe command and Safe Response are shared between the threads. These objects contain a lock and some variables, and know how to read the variables and write to them threads safely using locks. Using those variables, the HLC

module can pass flights commands (as vector directions) and ask for flight telemetry data.

Modularity – all our code is very modular, it means every part of it can be changed in the future in order to fit with other requirements:

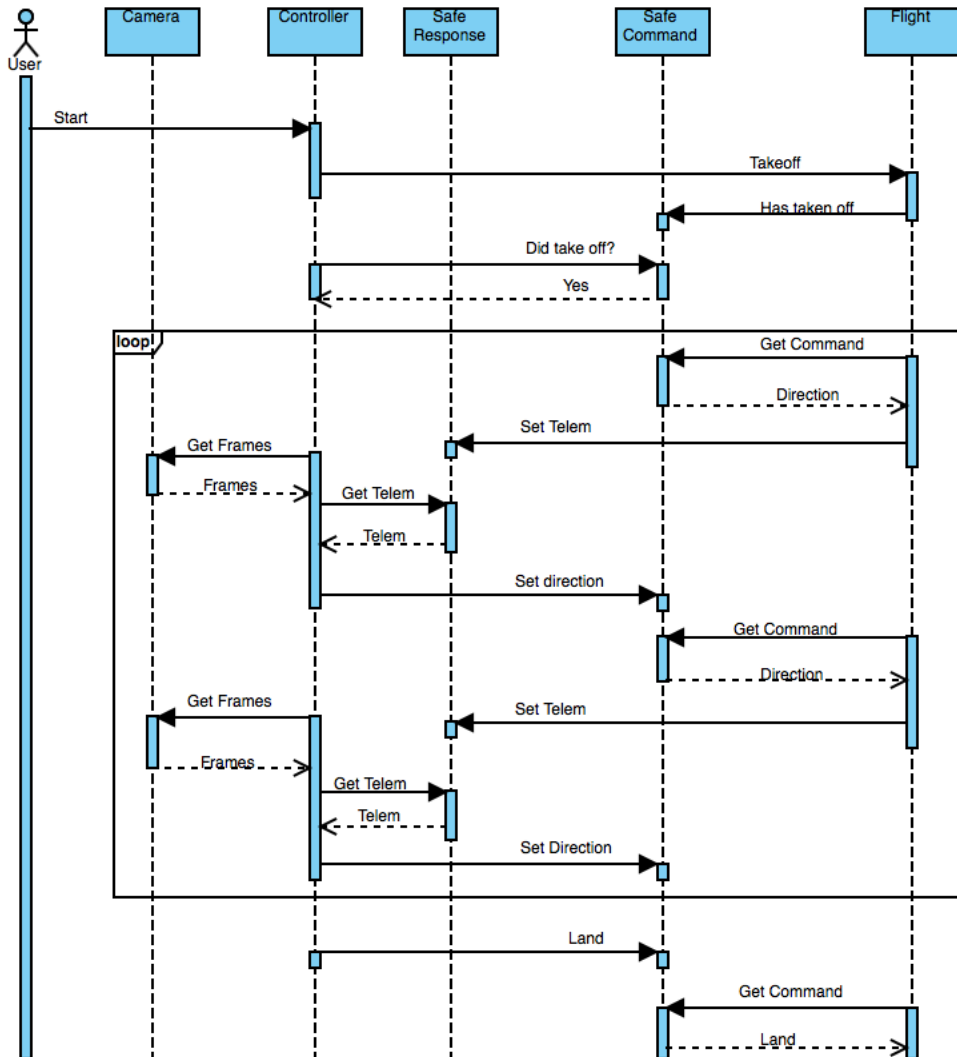
- another controller can be written immediately and control it differently using other inputs or other decision lines
- another Flight module – for example one which doesn't use Dronekit (it can use Parrot's library for example)
- while using our CV Stab one can always change the camera by using another camera module inheriting from Camera Handler.

Design Tradeoffs

- Threads – to enable sending continuous flight commands to the vehicle and letting the controller process images without thinking about it. Theoretically we could have used only one thread which will do some image processing, decide on the next command and then send it to the vehicle but in the reality, that means letting the vehicle with no commands for too long what can even cause a disconnection. Even without fearing of disconnection the idea of letting the vehicle without repeatedly telling it what to do is not correct, and the thread's separation of the code s.t one thread only do image processing and the other constantly send commands to the vehicle is more right.
- HZ vs FPS – the rate in which we send commands to the vehicle is called HZ, and the rate in which we receive images from the camera and process them is called FPS. In an ideal world, we would have wanted both to be very high, but unfortunately, this is not possible. This is not possible since it would cause the processor to over power and both threads would fight on the communication between them what will lead to a decrease in one of their rate. This is when we should decide which thread would have more resource power. By increasing the Drone communication rate (HZ) we can make sure the drone is more controlled and not leaved alone for unsupervised commands, and by increasing the controller frame rate of image processing we can obtain more adjusts commands.

- HLC doesn't create the Flight module – the other option could have been simpler and could give the possibility of knowing the HLC from the Flight module what would let the option of sending messages from the Flight module to the HLC. Since it was less modular we have chosen not to design it that way.
- Who initiate events. As explained one of the design options was to let the Flight module knowing the HLC. In that way, the Flight module could have initiated messages by himself, for example, in case of very low battery the Flight module could have decided that it send immediately a message to the HLC and so on to the controller letting him know that the battery is critical so that the controller could have acted correctly. In our project, there are no such urgent and real time messages and so we preferred letting the controller polling the flight module once a while and knowing about necessary messages.

Sequence Diagram



the sequence diagram describes the objects and threads interactions arranged in times sequences.

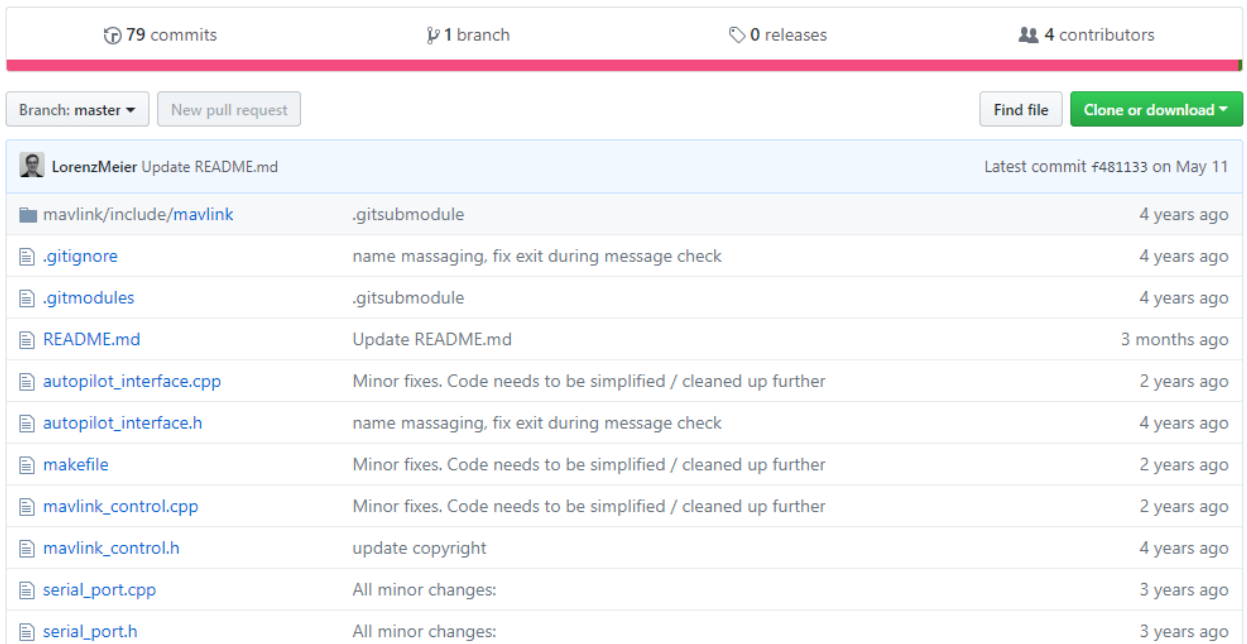
When the user asks from the project to start the controller will be created automatically along with the Flight modules and the safe variables. Then, the controller will ask from the Flight to take off and wait until the take-off is complete.

DIFFICULTIES WE ENCOUNTERED AND INSIGHTS

While working on the project we encountered many difficulties which delayed our progress for weeks at a time, here are some of the difficulties:

- Trying to control the drone using c++:

In most of the semester we worked with the conclusions of the previous project, which claimed that using python is slow and that we should work only with c++. Therefore, we searched the web for ways to communicate with the drone using c++ and found ways to connect and receive messages. We found the following git project:



The screenshot shows a GitHub repository page for 'mavlink/c_uart_interface_example'. At the top, it displays '79 commits', '1 branch', '0 releases', and '4 contributors'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Find file', and 'Clone or download'. The main content is a list of files and folders with their commit messages and dates. The files listed are:

File/Folder	Commit Message	Time Ago
mavlink/include/mavlink	.git submodule	4 years ago
.gitignore	name massaging, fix exit during message check	4 years ago
.gitmodules	.git submodule	4 years ago
README.md	Update README.md	3 months ago
autopilot_interface.cpp	Minor fixes. Code needs to be simplified / cleaned up further	2 years ago
autopilot_interface.h	name massaging, fix exit during message check	4 years ago
makefile	Minor fixes. Code needs to be simplified / cleaned up further	2 years ago
mavlink_control.cpp	Minor fixes. Code needs to be simplified / cleaned up further	2 years ago
mavlink_control.h	update copyright	4 years ago
serial_port.cpp	All minor changes:	3 years ago
serial_port.h	All minor changes:	3 years ago

https://github.com/mavlink/c_uart_interface_example

This project is suppose to control the drone using c++ via mavlink messages. After a lot of research and code reading of the projects who runs a few threads parallely and few different redundant objects which complicated the task severely. we were enable to extract from this repository the relevant code that is needed to establish a connection and created the following code:

```

int main(int argc, const char * argv[]){
    const char *uart_name = (char*)"/dev/tty.usbmodem1";
    int baudrate = 57600;

    //open the connection
    int fd = open(uart_name, O_RDWR | O_NOCTTY | O_NDELAY);

    uint8_t cp;
    mavlink_status_t status;
    uint8_t msgReceived = false;
    mavlink_message_t message;

    bool received_all = false;
    while (!received_all) {
        int result = read(fd, &cp, 1);

        if (result > 0)
        {
            // the parsing
            msgReceived = mavlink_parse_char(MAVLINK_COMM_1, cp, &message, &status);
            positionMessage(message);
        }
    }
    return 0;
}

```

The problem was that the messages weren't parsed correctly because it wasn't read in the right way- for example, it should have read the size of the message according to the second byte, but it didn't do it.

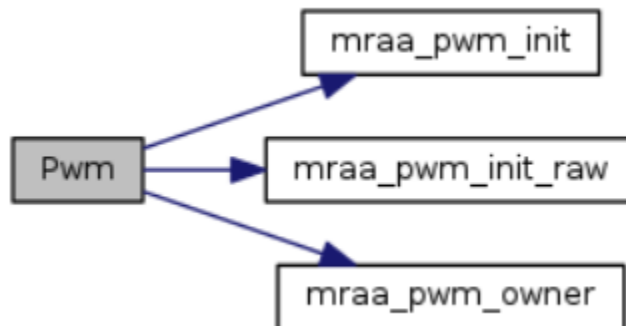
* Important insight: after we learnt about correct usage of git, we now know that we shouldn't have wasted so much time on this repository because it is clearly under developed- as we can infer from the number of branches and commits.

- Trying to bypass the file system in order to control the pwm:

Another assumption we worked with was that using the file system (the standard way) to control the gimble, is too slow. So, we searched for other ways to do it. We found alternative library called mraa:

<http://iotdk.intel.com/docs/master/mraa/python/>

Their main purpose is exactly what we needed, and supplies a convenient interface for that purpose:



* This is the call graph of the constructor of the class, helps conveying the complexity of it. Reference:

https://iotdk.intel.com/docs/master/mraa/classmraa_1_1_pwm.html

But after we were enabled to use it we checked the code source and found out that they are also using the file system.

* Important insight: looking back we understand that our efforts were futile because in linux everything is a file, thus trying to bypass will lead to dead ends.

- Deciding which mavlink command is the best to control the movement of the drone:

There are many mavlink commands that dronekit supports which can control elements in the drone, some are deprecated, some don't work well. We tried various ways, one of them directly controlled the thrust:

```
def set_attitude(roll_angle = 0.0, pitch_angle = 0.0, yaw_rate = 0.0, thrust = 0.5, duration = 0):
    """
    Note that from AC3.3 the message should be re-sent every second (after about 3 seconds
    with no message the velocity will drop back to zero). In AC3.2.1 and earlier the specified
    velocity persists until it is canceled. The code below should work on either version
    (sending the message multiple times does not cause problems).
    """

    """
    The roll and pitch rate cannot be controlled with rate in radian in AC3.4.4 or earlier,
    so you must use quaternion to control the pitch and roll for those vehicles.
    """

    # Thrust > 0.5: Ascend
    # Thrust == 0.5: Hold the altitude
    # Thrust < 0.5: Descend
    msg = vehicle.message_factory.set_attitude_target_encode(
        0, # time_boot_ms
        1, # Target system
        1, # Target component
        0b00000000, # Type mask: bit 1 is LSB
        to_quaternion(roll_angle, pitch_angle), # Quaternion
        0, # Body roll rate in radian
        0, # Body pitch rate in radian
        math.radians(yaw_rate), # Body yaw rate in radian
        thrust # Thrust
    )
    vehicle.send_mavlink(msg)

    start = time.time()
    while time.time() - start < duration:
        vehicle.send_mavlink(msg)
        time.sleep(0.1)
```

Reference: https://github.com/dronekit/dronekit-python/blob/master/examples/set_attitude_target/set_attitude_target.py

But the response was too slow and we couldn't get the sensitivity we needed.

* Important insight: controlling and stabilizing a drone requires much more than just pushing the throttle, it requires a PId controller (proportional–integral–derivative controller) which is a control loop feedback mechanism widely used in industrial control systems and a variety of other

applications requiring continuously modulated control. A PID controller continuously calculates an error value as the difference between a desired setpoint and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms. This is the reason that we need to use the PX4 , so trying to control the thrust directly wont work.

- Decide which operating system will be the best for our project:

In the early stages of the project we needed to choose the operating system to install on our upboard. We needed an operating system that supports control of 40 pins and supports the realsense camera. Because it was the beginning of the project and we didn't know anything about operating systems, or realsense, or PWM, we had difficulties to fully understand the requirements of the operating system. So we chose the most common OS that we found for upboard which is Ubuntu 16.04.

Reference: <https://wiki.up-community.org/Ubuntu>

* Important insight: looking back, it would have been better to choose an operating system that doesn't have a UI because it requires much more processing power than we needed, which can cost much more overhead.

- Enabling the Realsense camera on the upboard:

When we wrote our code which controlled the realsense camera it worked well on our private computer, but when we tried to test it on the drone the image didn't appear. First we thought that the problem was that we didn't use the usb3 port, so we needed to order a cable that is compatible to the port. But after the cable didn't fix the problem we researched some more we discovered that it was because the realsense needed to update its drivers.

Reference:

<https://www.intel.com/content/www/us/en/support/articles/000023694/emerging-technologies/intel-realsense-technology.html>

WHAT HAVE WE LEARNT:

While working on the project we learnt many new things here are some of them:

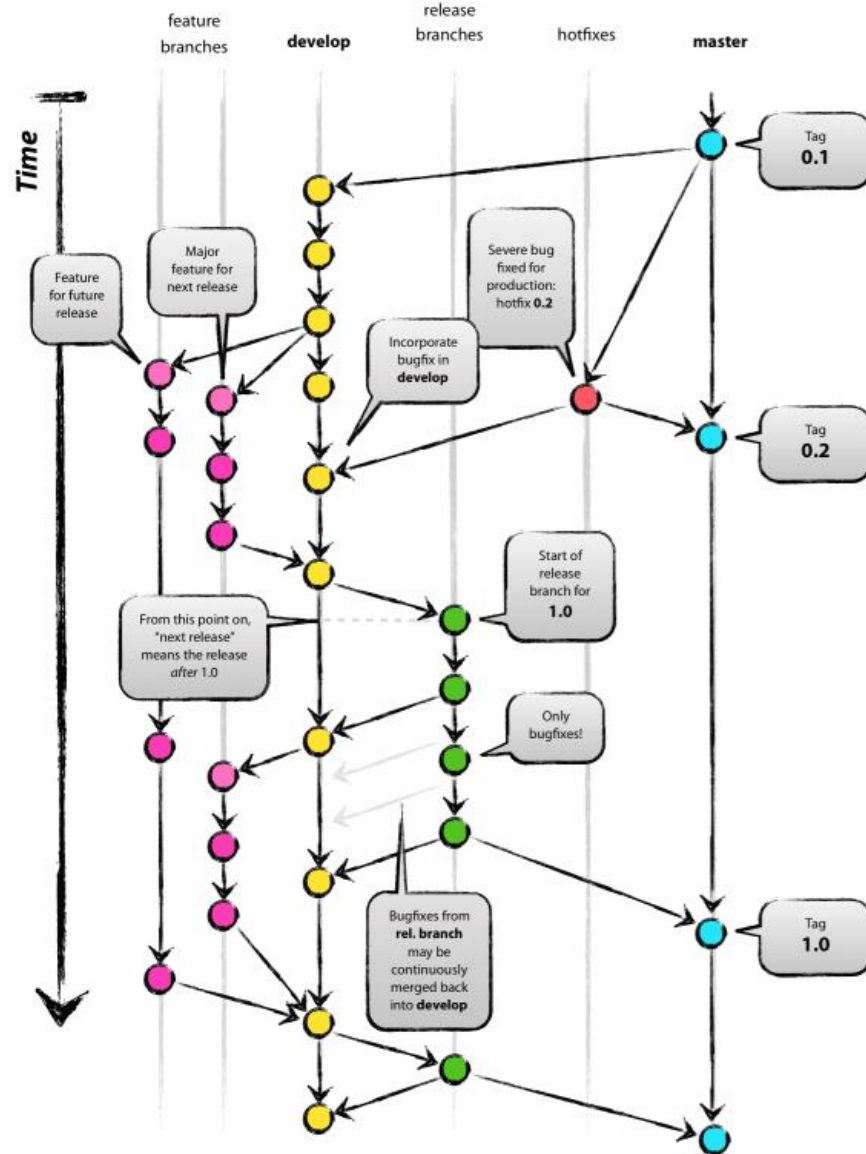
- Git best practices:

We learnt how to use git correctly in team projects. The main idea is to create many branches and promote the project only be working on the branches and merging it to the main branch. The main branch is called master, it is promoted only by merging with the version branches. It should always be documented, tested, is abled to compile and have a version number. The main purpose of the branch is to contain the latest most stable version of the project, that can be distributed.

The second most important branch is the develop branch, it is the main branch that the develop team uses and is created from the master branch. Mostly, we don't commit directly on this branch because the development happens in the features branches and after adding a new feature, we merge it to the develop branch. The minimum requirement of this branch is that it should compile at all time.

Feature branch is a branch that each developer works on a feature, it can be in any condition because usually only one person works on it and doesn't share it with other developers- until it branches with the develop. The feature branches are created from the develop branch, each developer takes a snapshot of the current state of the project and add a new feature. Version branch is a branch that is created when we decide to publish a new version, all the features that were merged into the develop branch before the creation of the version will be inserted to the current version and the other wont. In this branch most of the work is bug fixes, the developers make sure that the version that is about to be published and there are minimum number of bugs, and the merges are bug fixes. At all time this branch should be able to compile and have a version number.

Bugfix Branch are branches that are created from the version branch, as said earlier, their purpose is to fix bugs in the version branch. All time should contain a version number.



References:

<https://nvie.com/posts/a-successful-git-branching-model/>

- Controlling PWM based hardware:

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a modulation technique used to encode a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to internal] loads such as motors. In addition, PWM is one of the two principal algorithms used in photovoltaic solar battery chargers, the other being maximum power point tracking.

The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

The PWM switching frequency must be much higher than what would affect the load (the device that uses the power), which is to say that the resultant waveform perceived by the load must be as smooth as possible.

The rate (or frequency) at which the power supply must switch can vary greatly depending on load and application, for example

Switching has to be done several times a minute in an electric stove; 120 Hz in a lamp dimmer; between a few kilohertz (kHz) and tens of kHz for a motor drive; and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies.

The term duty cycle describes the proportion of 'on' time to the regular interval or 'period' of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.

The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle.

PWM has also been used in certain communication systems where its duty cycle has been used to convey information over a communications channel.

References:

<https://learn.sparkfun.com/tutorials/pulse-width-modulation>

https://en.wikipedia.org/wiki/Pulse-width_modulation

● UML Diagrams

While planning our design we were introduced with UML 2 and the various diagrams that it offers. UML defines the notation and semantics for the following domains:

- The User Interaction or Use Case Model - describes the boundary and interaction between the system and users. Corresponds in some respects to a requirements model.
- The Interaction or Communication Model - describes how objects in the system will interact with each other to get work done.
- The State or Dynamic Model - State charts describe the states or conditions that classes assume over time. Activity graphs describe the workflows the system will implement.
- The Logical or Class Model - describes the classes and objects that will make up the system.
- The Physical Component Model - describes the software (and sometimes hardware components) that make up the system.
- The Physical Deployment Model - describes the physical architecture and the deployment of components on that hardware architecture.

There are a few types of UML diagrams:

Structure diagrams

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems. For example, the component diagram describes how a software system is split up into components and shows the dependencies among these components.

Behavior diagrams

Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. As an example, the activity diagram describes the business and operational step-by-step activities of the components in a system.

Interaction diagrams

Interaction diagrams, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled. For example, the sequence diagram shows how objects communicate with each other regarding a sequence of messages.

References:

https://www.sparxsystems.com.au/resources/uml2_tutorial/index.html

https://en.wikipedia.org/wiki/Unified_Modeling_Language

- Mavlink protocol:

In the semester we learnt a lot about the mavlink protocol, what is the structure of the packets, how to generate new messages and various types of messages.

The packet structure is the following:

Field name	Index (Bytes)	Purpose
Start-of-frame	0	Denotes the start of frame transmission (v1.0: 0xFE)
Payload-length	1	length of payload (n)
Packet sequence	2	Each component counts up their send sequence. Allows for detection of packet loss.
System ID	3	Identification of the SENDING system. Allows to differentiate different systems on the same network.
Component ID	4	Identification of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
Message ID	5	Identification of the message - the id defines what the payload "means" and how it should be correctly decoded.
Payload	6 to (n+6)	The data into the message, depends on the message id.
CRC	(n+7) to (n+8)	Check-sum of the entire packet, excluding the packet start sign (LSB to MSB)

It is possible to generate a message using xml in the following pattern:

```
<?xml version="1.0"?>

<mavlink>

    <include>common.xml</include>

    <!-- NOTE: If the included file already contains a version tag, remove
the version tag here, else uncomment to enable. -->

    <!--<version>3</version>-->

    <enums>

    </enums>

    <messages>

        <message id="150" name="RUDDER_RAW">

            <description>This message encodes all of the raw rudder
sensor data from the USV.</description>

            <field type="uint16_t" name="position">The raw data from
the position sensor, generally a potentiometer.</field>

            <field type="uint8_t" name="port_limit">Status of the
rudder limit sensor, port side. 0 indicates off and 1 indicates that the limit is
hit. If this sensor is inactive set to 0xFF.</field>

            <field type="uint8_t" name="center_limit">Status of the
rudder limit sensor, port side. 0 indicates off and 1 indicates that the limit is
hit. If this sensor is inactive set to 0xFF.</field>
```

```

        <field type="uint8_t" name="starboard_limit">Status of
the rudder limit sensor, starboard side. 0 indicates off and 1 indicates that the
limit is hit. If this sensor is inactive set to 0xFF.</field>

</message>

</messages>

</mavlink>

```

And generating the message using mavgen:

```

usage: mavgen.py [-h] [-o OUTPUT]
               [--lang {C,CS,JavaScript,Python,WLua,ObjC,Swift,Java,C++11}]
               [--wire-protocol {0.9,1.0,2.0}] [--no-validate]
               [--error-limit ERROR_LIMIT] [--strict-units]
               XML [XML ...]

This tool generate implementations from MAVLink message definitions

positional arguments:
  XML                    MAVLink definitions

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        output directory.
  --lang {C,CS,JavaScript,Python,WLua,ObjC,Swift,Java,C++11}
                        language of generated code [default: Python]
  --wire-protocol {0.9,1.0,2.0}
                        MAVLink protocol version. [default: 1.0]
  --no-validate         Do not perform XML validation. Can speed up code
                        generation if XML files are known to be correct.
  --error-limit ERROR_LIMIT
                        maximum number of validation errors to display
  --strict-units       Perform validation of units attributes.

```

References:

<https://mavlink.io/en/>
<http://qgroundcontrol.org/mavlink/start>
<https://en.wikipedia.org/wiki/MAVLink>

- System design

As we described earlier we encountered many dilemmas while designing our system. Each decision has a tradeoff between each possibility and each possibility has its own pros and cons. For more information about our specific design look at previous chapter. The main thing we learnt is that if the design is planned correctly and thoroughly, most of the decisions that are made would be easy, because if we consider all of our goals and constraints, many times there is only one logical solution.

- Working in linux based environment:

In the beginning, we had no experience with linux operating system and using open source projects, but during the past months we gained much experience with when we needed to update drivers and changing things using the file system. Also, during our struggles for searching the best ways to implement things, we found many open source projects that we downloaded and integrated in our system.

In Conclusion:

We offer a framework which can incorporate Computer Vision in drone missions, which we hope the GIP lab could utilize in the future. The framework covers everything you need in order to control the drone from end to end. We believe that it will be very beneficial for any computer vision purpose, and every developer that wishes to create computer vision-based missions will find it very easy to understand it and enhance its functionality.

