Technical University of Denmark

Bachelor thesis

Exploring the use of Brain-Computer Interfacing in drone control

Carl Emil Elling - s164032
Aleksander Sørup Lund - s153827

Supervised by
Per Knudsen, Daniel H. Olesen & Sadasivan Puthusserypady
Units: 68.500
Spring 2019

DTU Space
National Space Institute

DTU Digital Health
Department of Health Technology

Exploring the use of Brain-Computer Interfacing in drone control

Authors

Carl Emil Elling, s164032@student.dtu.dk
Aleksander Sørup Lund, s153827@student.dtu.dk

Supervised by

Per Knudsen, Daniel H. Olesen & Sadasivan Puthusserypady

DTU Space
Technical University of Denmark
National Space Institute
Elektrovej buildings 327-328 & Ørsteds Plads building 348
2800 Kgs. Lyngby

DTU Digital Health
Technical University of Denmark
Department of Health Technology
Ørsteds Plads, Building 345C
2800 Kgs. Lyngby

Project information

Project period:	February 2019 - May 2019
ECTS:	15 each
Education:	B.Sc.
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	

Preface

This project was supported by DTU Space and DTU Digital Health. A special thanks go to our supervisors Daniel Olesen, Per Knudsen and Sadasivan Puthusserypady for helpful guidance and access to the required material and hardware.

Also thanks to Mikkel Bugge for allowing us to build upon his Bachelors Project.

Abstract

This thesis provides a modular framework for steering drones based on Brain-Computer Interfacing. This is done with the explicit purpose of exploring a new possible paradigm in drone control for research surveying.

Current BCI technology has proved its potential in providing aid to disabled people. These types of systems are largely developed for medical purposes, aiding locked-in patients and helping rehabilitate stroke victims. Although this is a worthy purpose, we establish here that it might not only be an aid exclusive to disabled people, but also a general control unit for other purposes.

In geophysical surveying, semi-automatic use is of interest. For this, a ground based platform is build using the ArduPilot software, mimicking the behavior and workflow of any vehicle with this architecture.

The system is proved functional through tests and is reproducible for later research or improvements. Sets of generic classifiers have been used for demonstration of the system. Demonstrations and tests show that the system developed is established to a degree where cognitive tasks explicitly can be converted into steering commands for drones. Distinguishing between 3 commands, we achieve an accuracy of 62.9%, concluding that the developed BCI technology might not be optimal for practical use as of now. Since the system is modular, it may in time provide a highly efficient steering paradigm.

Contents

List of abbreviations

BP	**B**and**P**ass
BCI	**B**rain **C**omputer **I**nterface
CAR	**C**ommon **A**verage **R**eference
CSP	**C**ommon **S**patial **P**atterns
DC	**D**irect **C**urrent
DTU	**T**echnical **U**niversity of **D**enmark
ECG	**E**lectro**c**ardio**g**raphy
EEG	**E**lectro**e**ncephalo**g**raphy
fMRI	**f**unctional **M**agnetic **R**esonance **I**maging
FMU	**F**light **M**anagement **U**nit
GND	**G**rou**nd**
GPS	**G**eneral **P**ositioning **S**ystem
ICA	**I**ndependent **C**omponent **A**nalysis
IMU	**I**nertial **M**anagement **U**nit
IO	**I**nput/**O**utput
LDA	**L**inear **D**iscriminant **A**nalysis
MEG	**M**agneto**e**ncephalo**g**raphy
MI	**M**otor **I**magery
PCA	**P**rincipal **C**omponent **A**nalysis
PLUTO	**P**latform for **L**andbased **U**nmaned **T**hought-controlled **O**bservations
PPM	**P**ulse **P**osition **M**odulation
PWM	**P**ulse **W**idth **M**odulation
RC	**R**emotely **C**ontrolled
SSVEP	**S**teady **S**tate **V**isually **E**voked **P**otentials
SVM	**S**upport **V**ector **M**achine
UAV	**U**nmaned **A**erial **V**ehicle
UGV	**U**nmanned **G**round-based **V**ehicle

Chapter 1

Introduction

1.1 Motivation

Drone engineering is a field in rapid development. Technology previously reserved for military operations is used today in everything from medical deliveries to industrial uses, and both professionals and hobbyists alike can easily learn to pilot unmanned vehicles. The increasingly complex manoeuvres demanded by equally complex uses requires sophisticated controls, and this is very much the case in the field of geophysical surveys.

When using drones and rovers as a research tool, the steering can require very advanced manoeuvring. Changing from automatic mission steering to manual steering whenever areas of interest need further examination, and vice versa. This can often instance a control scheme which is complicated, unintuitive and occasionally requires multiple people.

A recent sprout in Brain-Computer Interfacing (BCI) technology might be of interest in such situations, with the ability to provide hands-free control to these kinds of advanced drone systems. This type of technology is currently being developed for people with disabilities, where it can provide freedom for those not in control of their bodies. While this is a primary focus for current BCI research, the possibilities of BCI span much, much further.

1.2 Hypothesis

The Brain-Computer Interfacing technology developed today can be of use in many fields as a control unit. This paper will explore the use of BCIs in controlling drones by means of performed, or even *imagined* motory tasks. First, the noninvasive acquisition of electrical brain signals (electroencephalography)

will be used to collect MI evoked signals. These signals can then be processed and transformed into control commands for moving an unmanned vehicle in 2 or 3 dimensions. Such a system will be designed with geophysical surveys in mind, as it could introduce an alternative and intuitive way of handling drones in fieldwork.

1.3 Objectives

- Creating an intuitive interface between open source BCI software and open source drone controller software.
- Using already developed signal processing technology and state-of-the art electroencephalography (EEG) software to examine the viability of controlling unmanned vehicles using EEG based BCIs.
- Propose a way to interrupt planned automatic missions in drone surveys in order to overtake steering using BCI.
- Modularize each subsystem for the possibility of upgrading each part, without having to rebuild a completely new BCI system.
- Prove that BCI systems might be able to control a drone consistently and reliably.
- Provide a theoretical foundation for the interested reader to better understand BCI and drones.

Chapter 2

Theory

To explore the use of brain-computer interfacing as a means of control, we must first understand the underlying theory of both electroencephalography, which the BCI in this project is based on, and the intricacies of drone control.

2.1 Basic Neuroscience

First, it is necessary to understand why signals can be detected in the brain at all. On the cellular level, the brain consists mainly of fat cells, but the thing that makes it so unique are highly specialized cells called neurons. These can be seen as "a leaky bag of charged liquid" [[1] p.7], in the sense that they consist of an ionic liquid surrounded by a permeable membrane. This membrane is lying in another ionic liquid with a different chemical makeup, resulting in a small difference in electric potential of around -65 to -70mV between the inside and outside of the neuron. Neurons communicate by releasing chemicals via long branching synapses. These long branching connections release chemicals to the neuron, either causing it to open channels in the membrane and thus releasing the built-up potential, or increase the potential of the neuron. The release of potential causes a spike of voltage that can be measured.

Now that the basic mechanics of the neuron and how they communicate have been covered, we can begin to discuss what, in fact, they are communicating. For this, a macroscopic look on the brain and a rough sketch of its anatomy is in order.

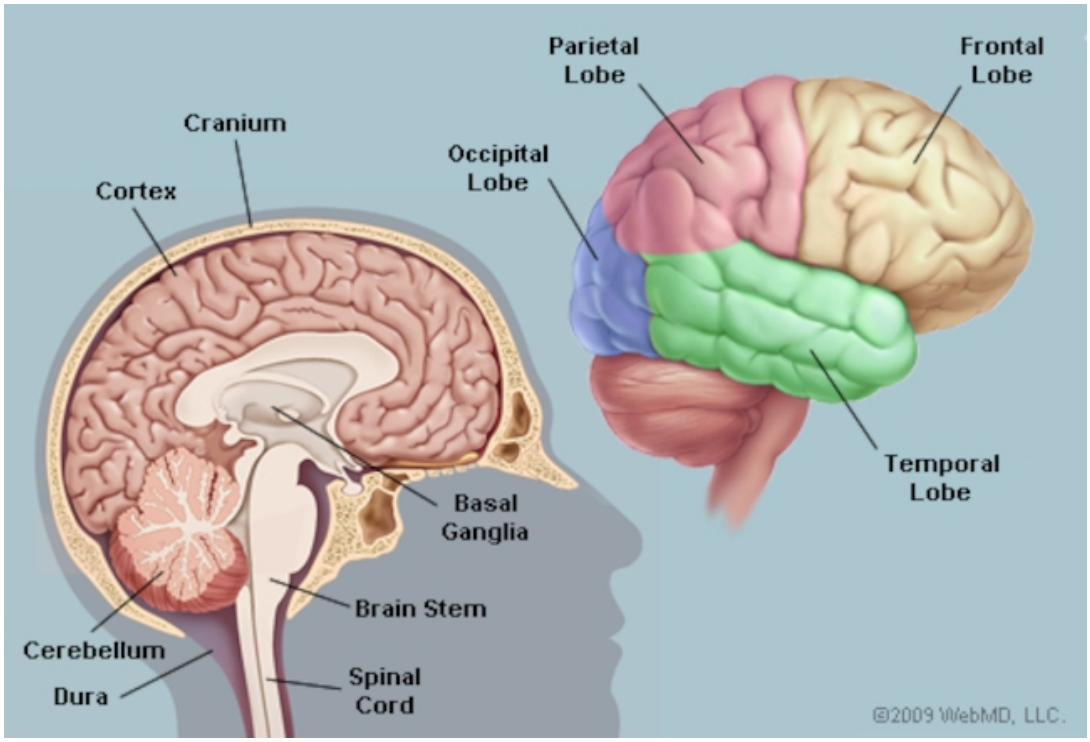


Figure 2.1: Overview of the brains anatomy. From https://webmd.com

The brain consists of three major parts: The *brain stem*, the *cerebellum* and the *cerebrum*. We will not discuss the cerebellum and brain stem in detail as they are not widely used in BCI applications. In short, the cerebellum mainly takes care of our subconsciousness, fine motor control and the filtering of information that comes from our senses. The brain stem is the connection from the brain to the rest of the body. The highlighted areas to the right in fig. 2.1 is an overview of the major parts of the cerebrum. Each area is associated with different tasks. Some tasks and sensory responses are located in highly specific sites, while others are more general.

2.1.1 The cerebrum

The cerebrum is what is most commonly thought of as "the human brain". It is the physically largest part of the brain and it is where the tasks that require higher levels of thought occur.

It consists of two halves - the right and left hemisphere. Each of these hemispheres is essentially a mirror of the other, with the left brain hemisphere being responsible for the right part of the body and vice versa.

The frontal lobe is the foremost part of the brain extending from the middle top of the brain and forward. It is responsible for motor functions, psychological state, motivation and deep thoughts. This is what we primarily think with, and this part of the brain is capable of planning actions and making important decisions.

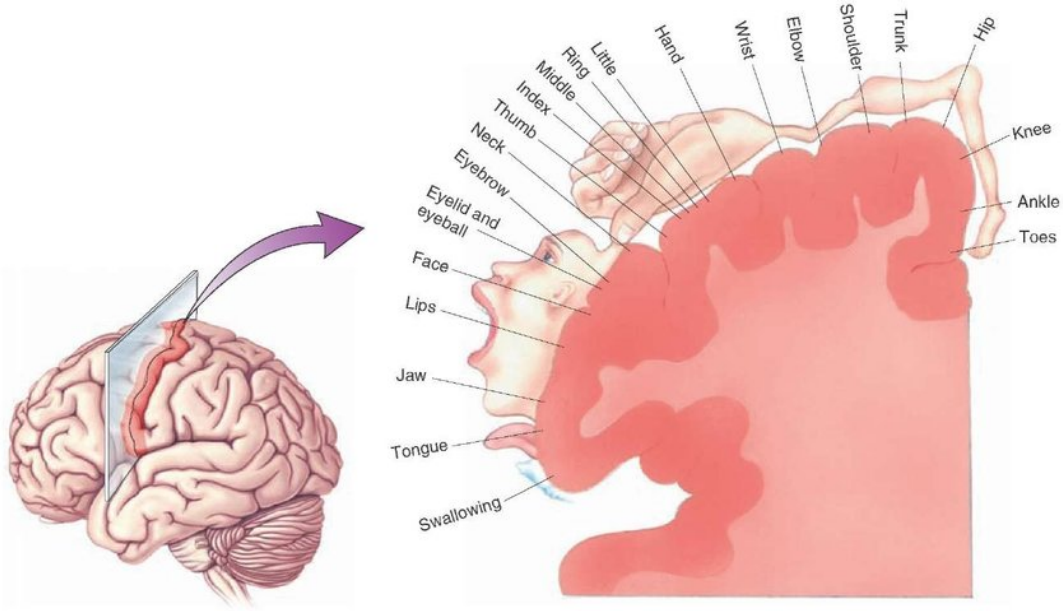


Figure 2.2: Overview of the motor cortex, visualizing where motor tasks origin in the brain. From researchgate.net

The motor cortex is a part of the frontal lobe, located in the back. This part is responsible for sending information about motor tasks to the rest of the body. It is a very distinct part of the brain, making it relatively is easy to derive signals from. The signals originating here are also unique, in that the activity of a very specific frequency band, the mu band (8-12Hz), corresponds to certain actions. All of this makes it very interesting in BCI applications. That information will be revisited later. In fig. 2.2 an overview of the motor cortex can be seen showing where different motory tasks are derived in the brain.

The Temporal lobe plays a major role in our memory, language and emotion. This part of the brain is where we store the information we have learned and processes sensory inputs for a higher level of cognitive use.

The Parietal lobe is the part placed right behind the frontal lobe, this is largely connected to the ability to interpret and understanding of the sensory inputs, this also has deep connections to the motory strip of the brain (placed in the back part of the frontal lobe.)

The Occipital lobe is placed in the far back of the brain. This is primarily responsible for the vision this is where you would acquire SSVEP signals for some BCI-systems.

2.2 Electroencephalography - EEG

When wanting to use brain signals as a mean on control, they first need to be extracted. This need not be as morbid as it sounds, although some very accurate signals can be measured by placing electrodes directly on the brain. This project is restricted to the use of non-invasive BCI systems. The most commonly used signal type for this application is the EEG-signal. This section tries to cover the basic and most important aspects of this signal-type in BCI-technology, while discussing its limitations and practicality.

EEG is the primary candidate for non-invasive BCIs. The signal is recorded by electrodes placed on the scalp and connected to an amplifier setup. The equipment varies, but the signal received will always be a voltage from each electrode over time. EEG signals are rather weak and oscillate in the range of -30 to 30 μV. As discussed, they are produced in the neurons in the outer layers of the brain, the neocortex, and have to penetrate both bone and skin to reach external electrodes, making signal processing a huge part of interpreting the data.

This is not the only signal that can be measured non-invasively, but this is by far the most researched, developed and thoroughly documented technology in this field. Other options consists of Magnetoencephalography (MEG) and Functional Magnetic Resonance Imaging (fMRI), but these can be dismissed in this project as they are highly impractical for everyday use.

2.2.1 EEG based BCI setup

One of the important factors in producing and developing BCIs is considering the setup needed to make it work. This is one of the biggest advantages of an EEG based BCI system (EEG-BCI). The common EEG-BCI setup paradigm is more or less portable and uses a system called 10-20 setup. This is an agreed-upon convention that assures the reproducibility of an EEG-BCI experiment/setup by making it largely standardized. The 10-20 refers to the placement of electrodes on the scalp (See Fig. 2.3). The setup time for a complete EEG-BCI system is approximately 30-40 minutes, but this can be reduced by designing a system that relies on fewer electrodes. While fewer electrodes make the practical setup easier the spatial resolution gets worse, which can be a bottleneck for some purposes.

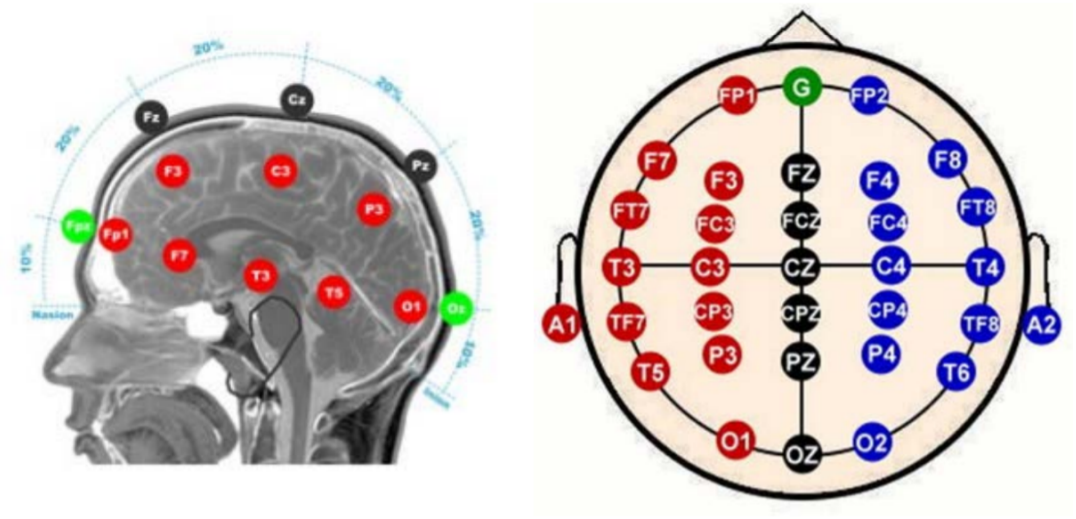


Figure 2.3: 10-20 setup of electrodes used in EEG [2]

Other setups, such as the commercially produced Emotiv BCI-Headsets, remove degrees of freedom by using fewer EEG channels in exchange for a gyroscope. This can be useful in applications for spatial control. For lab experiments, a cap is usually used with an arrangement of electrodes as shown in fig. 2.3, either with or without a conducting gel that reduces impedance between the scalp and the electrode.

2.2.2 Acquiring and interpreting the EEG signal

As a cognitive task is being done, thousands of neurons in a general area fire. The signal of the ones that are oriented with synapses pointing radially towards the scalp can be detected and acquired by the electrodes. The signals are very task-specific, as the areas measured are in charge of a rather distinct variety of cognitive tasks (See section 2.1). Despite this, the brainwaves do have some amount of stochastic nature. As the source of the EEG-signal is not spatially in the very position of the electrodes detecting the signal, the spatial resolution is poor. The voltage is emitted from the neuron synapses towards the detector of the scalp, and since this is a radial wave, the strength detected is proportional to the inverse of the distance squared - a geometric dilution of a signal known as the inverse square law. This causes the signal to be dominated by the outermost layers of neurons in the cerebral cortex. Signals also have to travel through the cerebrospinal fluid, the skull and the scalp, giving the resulting measurement accuracy only in the range of around $1cm^2$. The usefulness of EEG-signals thus comes from the high temporal resolution, as this is precise in the range

of milliseconds, which makes it fairly reliable to detect time-dependent signal types, such as P300.

The EEG-signals, unfortunately, are easily disturbed by electromagnetic signals originating from muscles during movement, caused by chewing, eye movement, blinking etc. (Electrocardiographic signals - ECG's) and other such artifacts. Much of this distortion can be filtered using a bandpass filter and a variety of data filtering methods, many of which are based on machine-learning models. It should be noted, that although this can take care of a good deal of the distortion, the subject using BCI-EEG should preferably be in a rested and completely still sitting state.

Other disturbances may come from psychological state of mind. This is a much harder disturbance to filter, as the signals caused by dopamine release in the brain, the response of boredom, stress and frustration is much more similar to those of ordinary stimulation in the brain than the artifacts mentioned earlier.

2.3 Brain-Computer Interface - BCI

When using EEG in a Brain-Computer Interface, only specific signals can be used. Contrary to popular belief, BCI technology cannot read minds. Only specific patterns firmly associated with specific cognitive tasks or events can be used as a means of control [3]. It is essential that the measured electric pattern must be reproducible, either by external stimuli or by some form of conscious internal action. This is, in fact, the basis for much of the field of BCI technology and also for this paper, since we here explore a method of control. BCIs could also be used in stimulating the brain, which might have relevance for many fields in the future, but that is not of relevance here.

Asides from being reproducible, the measured signal must be distinguishable from noise. Temporally oscillating brain signals called "brain waves" are well captured by EEG, making them suited to use in BCI applications. The amplitude of the spike in brain activity is also important since a larger amplitude makes for easier distinguishing.

Most common today are BCIs based on selective attention, on external stimuli (most commonly visual), and those based on motor imagery. Some of the first BCIs were visually based, most notably P300 and SSVEP (Steady State Visually Evoked Potentials). These both use flickering light as the external stimulus, but differ in a few key areas. The P300 system uses a command, that flickers once. After this flicker, if the command area is focused on, the brain will respond with a signal about 300ms after, therefore called P300. The P stands for positive potential spike, 300 for the number of milliseconds the brain take to respond. Other such signals exist, but this is the most pronounced.

On the other hand, SSVEP systems work by constant flashing. The frequency of the flashing can then be registered in the visual cortex, meaning that if light flashes at 20Hz, the resulting signal will also flash at 20Hz. This oscillatory nature makes for easy signal recognition.

BCIs based on selective attention have both advantages and disadvantages. They are usually very reliable, with the resulting signals being very well studied and recognizable. Besides, BCI systems using these, at least partially, have been extensively developed and are used in real life situations around the world today. As stated though, they require attention, making them unwieldy in situations where visual attention is needed elsewhere. Besides, especially the P300 systems have a rather large latency, making the information transfer rate low, naturally being bottlenecked by the 300 ms latency.

Another much used type of BCI uses *motor imagery*. Here, we return to the motor cortex with its distinct signals. Moving and even *imagining* moving limbs reduces activity in just this part of the brain, in a frequency band of brain waves from 8-12Hz called the *mu band* [[1] p. 45.]. Movement is convenient, since a given motor function can be easily repeated, and thus the EEG signal can be screened for activity relating to that particular function.

2.3.1 Signal Processing

Raw EEG-signals have to be processed before we can use it as a control input. This is done in order to extract the patterns produced by one of the various methods proven to produce extractable patterns, while also being easily repeatable. Such signal processing is often separated in 3 parts: *pre-processing*, *feature extraction* and *classification* (see fig. 2.4 showing the entire chain of signals in a typical BCI)

Temporal filtering is widely used in BCI applications. This method simply filters the signal so that certain frequencies pass through, or certain frequencies are excluded. This is especially useful in reducing known noise frequencies, such as the 50Hz signal from the power grid in Europe, or when only looking at a certain frequency band, e.g. the mu-rhythm in the 8-12Hz range. This consideration can go a long way, as multiple signals may be present simultaneously, band-passing a small range of frequencies will make it easier to classify the desired signal and performing data processing in real time. The range can't be narrowed too much though, as some critical frequencies might be missed.

Since non-invasive EEG-based BCIs record the activity of neurons in a large area, noise and artifacts become a problem [[3] s 306]. This includes noise from surroundings, other brain activity than the wanted signal, and importantly, noise from muscle movement. This, in particular, is a problem in EEG based BCIs

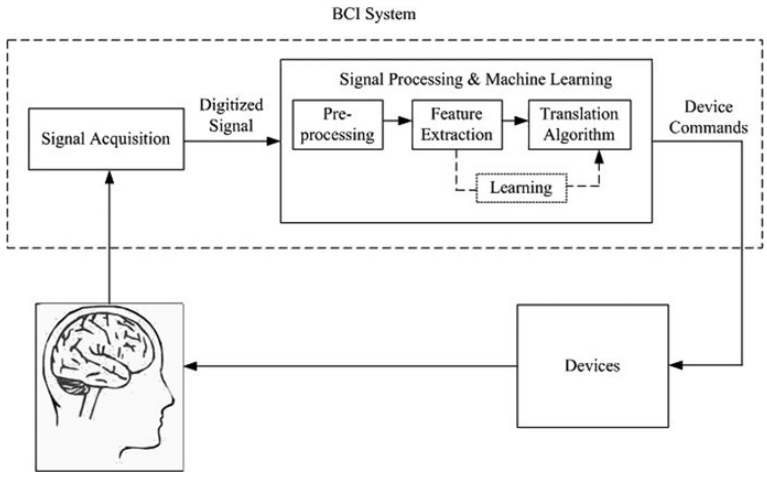


Figure 2.4: The framework of most EEG-based BCI systems. From Frontiers p.306

since the cranial muscles are directly in the way of the brain. This noise can be removed in pre-processing steps, where methods such as temporal and spatial filtering greatly improve signal quality with little to no change in the actual usable data.

In order to extract data from the exact place, a signal in the brain originates, spatial filtering is used. Common spatial filtering methods are mostly linear transformations, with the differences between them being the transformation matrix. One such method is the Common Average Reference (CAR), which subtracts the average signal of all electrodes from one [[3] p.309]. This reduces noise common to all electrodes, such as power sources, with the downside that noise from individual electrodes is not filtered, so further filtering would still be required. Others include the Laplacian reference, which subtracts the average signal from around one electrode, in order to reduce more region-specific noise, Principal Component Analysis (PCA) and Independent Component Analysis (ICA), both of which can be used to separate the EEG signal into components. This is useful when, like in most EEG-based BCIs, only certain brain signals are needed.

CSP

A specific solution to the spatial filtering problem much used in BCI applications is the Common Spatial Patterns (CSP) algorithm. It seeks to maximize variance between different classes, in order to extract the most information from each channel. The end results is a matrix of weights given to the channel data, in order to simulate recording the source data at the origin location. This can

be done in a few different ways, either as an optimization problem [4], as an eigenvalue problem [5] or even geometrically [6].

In essence, we need to create a weighted matrice W, so that

$$\mathbf{S} = \mathbf{W} \cdot \mathbf{X}$$

where S is the data at the source locations and $X \in R^{TxN}$ is the measured EEG data from all channels, T being the number of trials and N the number of channels.

Since the raw EEG data is bandpass filtered, there is no DC information in the signal, meaning we cannot distinguish the spatial origin of a signal from the mean signal power. Thus, the information needs to be found in the variance between channels. By using the classical 2-class CSP method developed by Ramoser et al. (1998) [5], covariance matrices from both classes between all channels. Normalizing with the number of trials, T_n we get the covariance matrix C for each class $k \in l, r$ (e.g. left hand clenched and right hand clenched) [7]:

$$\mathbf{C}_k = \frac{1}{T_n} \sum_{j=1}^{T_n} \frac{\mathbf{X}_j \cdot \mathbf{X}_j^T}{\text{trace}(\mathbf{X}_j \cdot \mathbf{X}_j^T)}$$

Here, the trace function calculates the diagonal sum in a matrix. A joint spatial covariance matrix, C_c can be found as a composite of the covariance matrices averaged over trials:

$$\mathbf{C}_c = \sum \overline{\mathbf{C}_k}$$

Now the eigenvector problem can be seen. Since we can extract an eigenvector and eigenvalue from this, we can transform the averaged covariance of each class, so that they have common eigenvectors.

$$\mathbf{C}_c = \mathbf{U}_c \lambda_c \mathbf{U}_c^T$$

First we whiten the eigenvalues:

$$\mathbf{P} = \sqrt{\lambda^{-1}} \mathbf{U}_c^T$$

This means that the variance of $\mathbf{C}_c$ in the P space is equalized and all eigenvalues equals 1. Transforming the covariance matrices:

$$\mathbf{S}_l = \mathbf{P}\overline{\mathbf{C}}_l\mathbf{P}^T \text{ and } \mathbf{S}_r = \mathbf{P}\overline{\mathbf{C}}_r\mathbf{P}^T$$

Then, since they share eigenvectors meaning

$$\mathbf{S}_l = \mathbf{B}\lambda_l\mathbf{P}^T \text{ and } \mathbf{S}_r = \mathbf{B}\lambda_r\mathbf{B}^T$$

and since all eigenvalues equal 1, we minimize all probabilites of one class while maximizing the other. This results in a projection matrix $\mathbf{W} = (\mathbf{B}^T\mathbf{P})^T$, which is used as the spatial filter.

It is a supervised learning algorithm, meaning when training, the algorithm needs the test subject to see cues on screen, while receiving timestamps and types of each cue.

After the pre-processing comes feature extraction. This step selects certain useful signals from the rest of the data, according to the event used as a trigger for the system. Currently, this is one of the major differences between the interfaces of different BCI systems, and no "correct" solution exists. This is where machine-learning techniques will assist.

2.3.2 Machine Learning

An important part of BCI is being able to find *meaning* in the data collected. While temporal filtering can remove the data that certainly does not contain any useful information, a hard-coded solution for extracting this meaning does not exist. This is where machine learning enters. It is a powerful and important tool in the process of establishing a BCI-interface.

Machine learning is a general term for applying statistics and algorithms to extract patterns. It is also the use of these to recognise a meaning from data, that often seems like gibberish to humans. There are two main branches of machine learning, *supervised learning* and *unsupervised learning*. They differ in the way we train the machine learning algorithm. Supervised learning requires the training set to be paired with a correct answer and the algorithm will then try to establish rules to mimic this answer. In unsupervised learning the answer is hidden and the machine learning algorithm itself will have to establish differences between datasets to derive meaning.

In the case of most BCIs, supervised learning is used to initially create a way to distinguish between cognitive tasks.

For extracting the meaning of the signals, we train a set of CSP filters and classifiers. A classifier is one of the aforementioned algorithms, capable of putting new data into a class. The class is simply a set of data that share properties, and by putting new data into a class, the algorithm has decided that this data belongs to that existing set. Thus, the new data has been classified.

The classifiers used later during the design process is a generic LDA classifier.

Linear Discriminant Analysis (LDA)

Although other classifiers do exist and are widely used, a classifier with a history of widespread use in BCI applications is the LDA.

This section is a rundown of a simple LDA. For the sake of simplicity let us assume one input variable to classify upon, this set of data being called X. This data we want to split into classes. In this example that is 2 classes $k = [1, 2]$. Where, for the specific example of MI-BCI, each class would correspond to the execution of an imagined task. We also need to establish that P_k denotes the probability that a data point is contained in class k, and ρ_k denotes a density function for the clusters defining the classes. If we assume that the classes are somewhat separated in the dataset X, we can write the probability of each data point being contained in a class using Bayes theorem.

$$P_k(X) = P(Y = k|X = x) = \frac{P_k \rho_k(x)}{\sum_{l=1}^{K} P_l \rho_l(x)} \tag{2.1}$$

If we only predict one thing, which is the case for the following applications, we can write out the predictor by assuming that the density function is normal distributed.

$$f_k(x) = \frac{\exp(-\frac{-(x-\mu_k)^2}{2\sigma^2})}{\sqrt{2\pi\sigma_k}} \tag{2.2}$$

Here μ_k is the average of all training observations in a class k. From this expression follows that the denominator term is just the square of the distance from the middle of the cluster. σ^2 is the variance of the data set.

Now the challenge is not to use these equations, but rather to figure out what the density functions, average value and variance should be. Then the LDA trains to get the appropriate values of μ_k and σ^2.

What we really want to do to make the classification as efficient as possible is to maximize the discriminant. The discriminant is defined by the equation

$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\rho_k) \tag{2.3}$$

This is a linear function, hence the name Linear Discriminant Analysis or LDA for short. A simple generic LDA algorithm classifying 2 classes uses the following results to generate a classifier.

Averaging all training data:

$$\mu_k = \frac{\sum_{i=1}^{f_{x_i}=k} x_i}{n_k} \tag{2.4}$$

The variance of the classes:

$$\sigma^2 = \frac{\sum_{k=1}^{K} \sum_{i=1}^{f_{x_i}=k} (x_i - \mu_k)^2}{n-k} \tag{2.5}$$

Note that this is the way an LDA classifier with two classes and only one-dimensional data would work. The method is not restricted to only 1 dimensional data though.

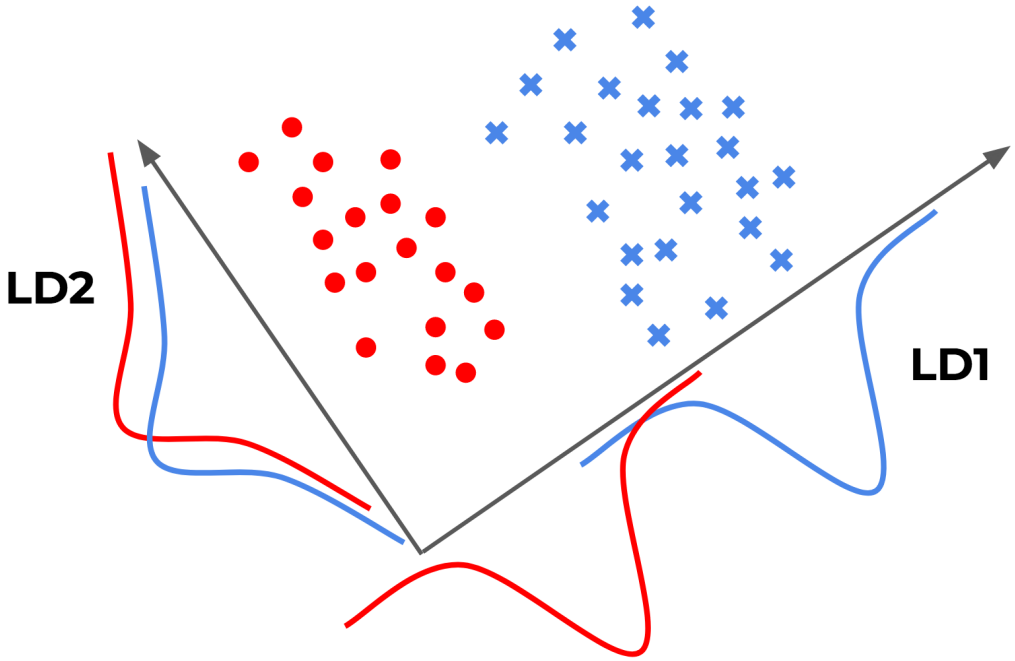


Figure 2.5: 2-dimensional visualization of the LDA problem solution

For the generalized vectorized version of the discriminant expression , a complete form for any dimensional dataset in vector notation would be

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{\mu_k^T \Sigma^{-1} \mu_k}{2} + log(\rho_k) \tag{2.6}$$

Where $\sum^{-1}$ denotes the covariance matrix and μ_k is the mean vector. Here we assume that the data is Gaussian distributed, which is the equivalent to normal

distribution in higher dimensions. The general properties of this equation are the same as the one-dimensional version and can be optimized using the same ideas and principals.

Modified versions of the LDA classification algorithm exists. Interestingly, an adaptive, unsupervised and multiclass version exists. However it is not widely used in BCI technology as of now, and conclusions of Vidaurre et al. (2012) [8] show that this would be most useful when the BCI is used non-task specifically.

2.4 Drone control

While other papers focus mainly on BCI, this project seeks to connect it to state-of-the-art drone technology. Due to the vast amount of different drone components, some important aspects of drone steering needs to be addressed.

2.4.1 Navigation

Drone navigation has many similarities with aerospace navigation. Essentially, there are 4 degrees of freedom in navigating 3D space. *Yaw*, *pitch* and *roll* correspond to rotating around the z, y and x-axis respectively, with the x-axis being forward. This is shown in fig. 2.6.

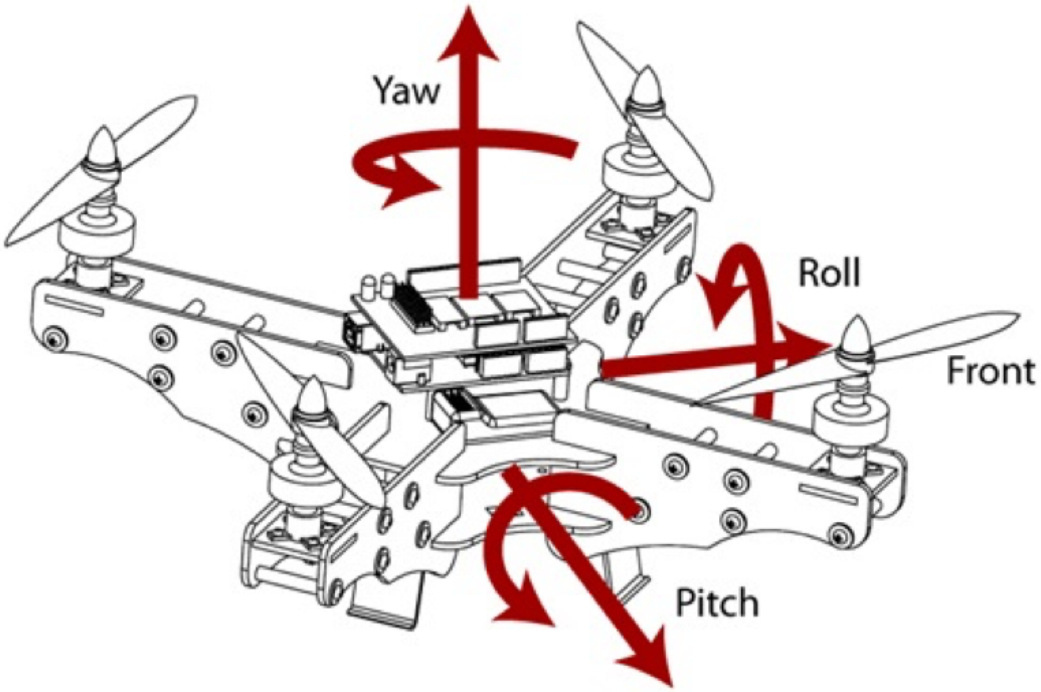


Figure 2.6: A stylized view of drone steering. From www.qualtre.com.

The 4th control is the thrust, All of these play together in an intricate dance as an unstable system balancing forces in the required direction. On ground-based vehicles, this steering simplifies to a left/right steering (essentially the yaw of aviation) and a thrust.

2.4.2 Hardware

Most drones, be they aerial or ground-based, consists of similar parts: A chassis to contain the internals, motors to create a torque and parts to use this torque as propulsion. In Unmanned Aerial Vehicles (UAVs) these are rotors, whereas remotely controlled (RC) rovers use wheels. The anatomy of a chassis, the inner workings of a motor and the optimal design of a rotor or wheel are large topics and since most people are aware of their uses and workings, they will not be discussed here.

Flight controller

Overlooking drone chassis, motors and rotors, some components are essential for every type of drone. One such component is the flight controller, especially

useful whenever automatic steering is required. A flight controller is the brain of any UAV. It manages signals from transmitters and sensors and steers the vehicle according to inputs both pre-programmed and real time.

In this project, the Pixhawk 2.1 flight controller is used to steer the remote controlled vehicle. This is done because of its open-source hardware, which is compatible with open-source radio controllers such as the FrSky Taranis also used in this paper. It is also well suited for use with ArduRover, an open-source auto-piloting system used in this paper.

The Pixhawk is a system with plenty of features, most of which are not used in this project. Thus, only essentials of the way the controller works are described here.

The controller is split in two processors: a Flight Management Unit (FMU) and an Input/Output (IO) unit. The FMU receives information from a pre-programmed path and sensors. Most notable is the GPS, which can be connected directly to the Pixhawk to give exact positioning of the unit. Integrated sensors are positioned on the Inertial Management Unit (IMU) board. Here, multiple accelerometers, gyroscopes and a magnetometer are used in tandem with an externally mounted GPS to orient the vehicle. It controls 6 Pulse Width Modulators (PWM). These modulate the average power sent from the battery through the motor controller in order to control the turning speed of the motors.

The IO-unit is directly controlled by a remote controlled signal, and works separately from the FMU, controlling 8 separate PWM's. [9]

This versatility, along with compatibility with many external modules, are some of the reasons for the controller's widespread use in drone piloting.

Radio transmitter

Since we wish to control the drone in real time, a radio transmitter is needed. In short, this is a system sending high-frequency signals (typically in the 2.4GHz band) modulated in a certain way, that a radio receiver can decode and use for inputs. These signals can be modulated in a wide variety of ways, but most commonly used in commercial UAV piloting is the PWM-paradigm.

In short, PWM consists of pulses with a fixed period. The pulses have varying length, and it is this pulse-width that contains all information.

To send remote signals overriding the Pixhawk autopilot, we use the transmitter Taranis FrSky Q X7 equipped with the open source software OpenTX.

2.4.3 Software

The hardware of a drone is only one aspect of a whole. Interpretation of signals, mission planning and steering is all controlled by software, most of which is implemented on the drone hardware itself.

ArduRover

Many drone steering protocols exist, all doing much the same thing. Instead of the flight controller receiving an input and the drone reacting accordingly, a microcontroller with drone firmware installed on the platform can provide new possibilities. While acting in symbiosis with the inputs from the flight controller, it can execute planned automatic routes and use sensor values in custom ways.

Although the firmware is open-source, the hardware is not necessarily. There are 29 different hardware setups that use this firmware, all developed by commercial companies independent of the software. The Pixhawk is one of the few hardware setups with open hardware.

The specific workflow of the program is also a topic of interest for this project. How does the signal then become a movement?

The ArduRover firmware initially starts up the main caller function. It starts continuously calling a code sequence, ApmRover2.cpp which is timed by a scheduler to handle all inputs, outputs and processes of the ArduRover software. After setup, the ApmRover script starts calling tasks from the task scheduler at a high frequency. This updates all information, reads the input data from the radio signal and plans what to do next. This is called in subfunctions. For this project, the command for sending PWM signals to the motor controller is of particular importance. Below, the workflow is illustrated as a flow chart, with relevant subfunctions.

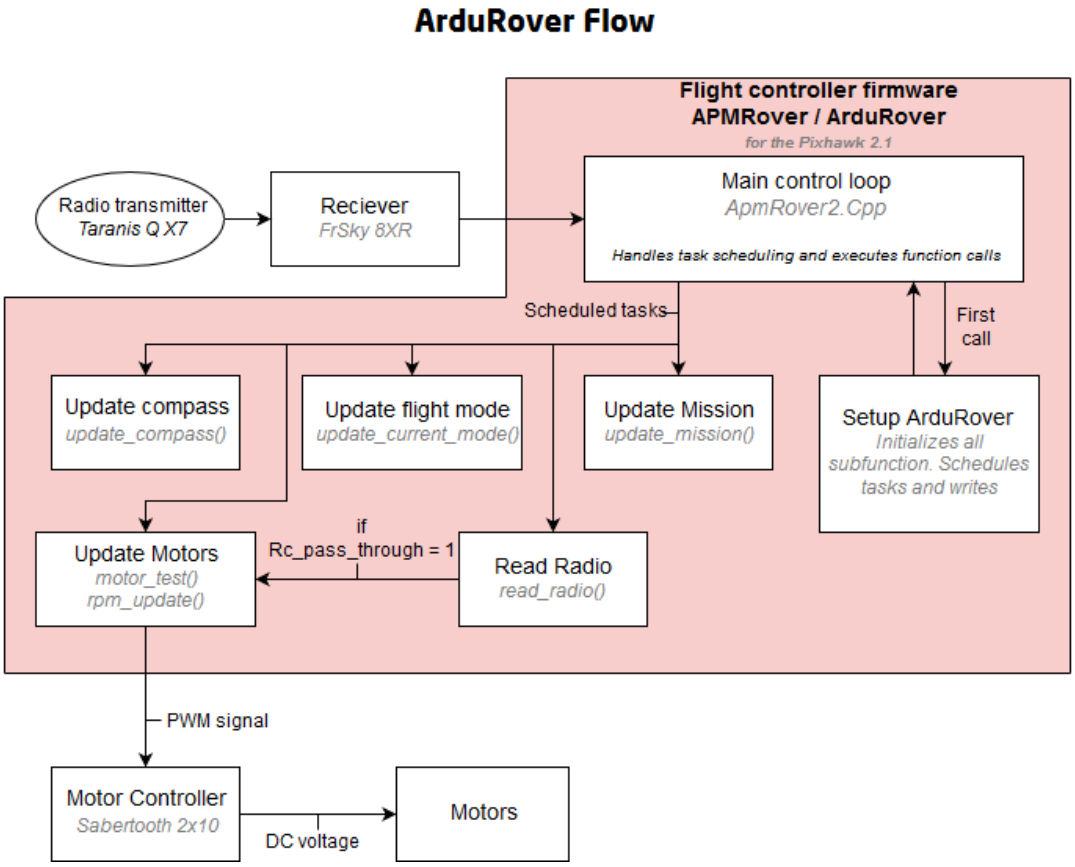


Figure 2.7: Workflow of ArduRover from input to motor reaction. The most relevant subfunctions have been chosen for this diagram

These subfunctions can be modified to suit the projects needs, which we will return to in Chapter 4.

Mission Planner

For the handling and setup of the ArduRover firmware, Mission Planner is used. This is an interactive interface which allows for planning automatic missions for the platform used. For a more detailed overview and insight see the user manual for PLUTO (See Appendix D) or the Mission Planner website [10].

Coupling from computer to drone

A vital part of the project is being able to control a drone directly from the computer. However, the serial connections offered by a Windows 10 cannot

communicate with the Taranis radio transmitter, let alone send direct radio signals. The transmitter itself can only receive PPM (Pulse Position Modulation) signals. These are signals of a fixed length sent in series with pauses of varying length between. The pauses carry the information needed, which in the case of steering is values from 0-100% thrust. Each new pulse signals the end of data sent to one channel and the beginning of the next (in chronological order).

An Arduino can output these signals, which is why an Arduino Mega 2056 is used, along with custom code from Bugge (2018)[11]. Windows 10 can communicate with the Arduino via a normal serial connection.

Chapter 3

State-of-the-Art

Research in BCI technology has seen a large growth just in the last decade. To find the current state of technology and especially to explore the obstacles faced today, a series of articles and other publications are examined in this section. Here, we will focus on projects concerning the control of unmanned vehicles with MI-BCI in order to get an overview of what is currently possible with state-of-the-art EEG technology.

MI BCI systems controlling drones have in recent years been getting more and more popular, and several setups has been tested.

Duarte (2017)[12] has documented a complete end-to-end BCI-controlled drone system. The system is developed in a simulation platform called Gazebo, using the open source tool ROS. Here, 2-class motor imagery is extracted using CSP and LDA achieving high accuracy (72.5%) and reliability during tests. However, he does not apply the system onto a real-life drone. There is reason to believe that he could attain similar results on an AR Parrot 2.0 Drone, which the system has been modelled after. A mean time of drone flights on a simulated track of 100m at 118 seconds is quite good, since the max. velocity of the drone is 1 m/s. This shows that reliable control of a drone using motor imagery is possible.

Another approach is taken by Shi et al. (2015)[13] regarding semi-autonomous steering controlled by MI. In an attempt to strike a balance between an intelligent system and low computational cost, a combined BCI and obstacle-avoiding system is developed. It is a semi-autonomous system in the sense, that it uses a laser range finder to extract information about the drone's surroundings. This information is used to determine feasible directions and avoid obstacles. The system does not take action itself, however, as it leaves this task to the user. This is done with a BCI system based on MI tasks, imagined movement of the left hand steers the drone left, vice versa with the right hand, and idling to move the drone forward. Data was collected over 5s, and features were extracted from 1-4s. This was done in series of continuous tasks. The latency of this system

is very high. This poses a major challenge for real-time BCI drone control, as quick reactions can be vital. The UAV is self-built with an ARM9-processor with Linux as the operating system.

When evaluating a system, choosing the right test subject matters. A short article on this subject was published by North et al. (2018)[14] for the use on a tech exposition. They set up a simple BCI controlled drone using the Emotive EPOC+ and parrot AR-Drone 2.0. They asked a number of participants on the exposition to try out the system, evaluating a couple of questions. This was compared also to a pretest in a similar fashion, establishing the mental state before trials. The result showed that 50% had a hard time producing the necessary flight commands, but this is a skill that subjects improve over time. They concluded that some of it had to do with poor connection, and that the stress level has a significant impact on the ability to control the drone. The number of participants was few, and therefore the conclusion doesn't stand as strong.

To combat this, subjects can be trained over a long period of time. LaFleur et al. (2013)[15] conducted a series of tests on a steering a quadcopter in a virtual framework. The subjects were readied for online tests over a period of 3 months in a rigorous training scheme. They used a 3 class classification. The classifiers were trained one class at a time against the rest of the classes. The accuracy was evaluated based on tasks subjects had to perform. Evaluation was done in multiple aspects with high scores (82.6% tasks were at least partially correctly performed, and 66% of participants performed correctly) but the classification performance is not presented. The reliability was gauged to be 25.8% of that of keyboard inputs. Very little information on the BCI system is given.

All of these papers present a very small sample size, so significance of the accuracies is not guaranteed.

High accuracy have been achieved for controlling drones using other BCI paradigms such as SSVEP[16] and hybrid systems[17].

An overview of the information can be found in table 3.1.

Article	BCI Paradigm	Classes	BCI Platform	Drone	Pre-processing	Feature extraction	Classification	Online/offline	Accuracy
Duerte (2017)[12]	MI	2	OpenBCI	AR Parrot 2.0	BP [8-30Hz]	CSP	LDA	Online	72.5% validation -
North et al. (2018) [14]	MI	-	Emotiv EPOC+	AR Parrot 2.0	BP [0.2-43Hz]	Not specified	Not specified	Online	Concluding, that statistically, some people are worse at controlling BCIs.
Shi et al. (2015)[13]	MI	2 (and idle)	Custom. Little information	Custom. Linux flight controller	BP [0.5-30Hz]	Cross-correlation & CSP.	Linear regression	Online	High (no metrics)
LaFleur et al. (2013)[15]	MI	4	BCI2000	AR Parrot	Not specified	None	Not specified	Offline+online	25.8% reliability compared to keyboard.
Royer et al. (2010)[18]	MI	4	BCI2000	Virtual	Not specified	Minimum-Norm Estimate in the Frequency Domain	Autoregressive	Online	>0.99 cross correlation in short flight route

Table 3.1: List of related State-of-the-Art projects in MI-BCI

Chapter 4

Materials & methods

To judge whether a BCI-EEG system can be used for advanced 2-3 dimensional control we want to propose a system bridging already existing open source software in both BCI and drone tech. Although similar systems have been tested before, they use equipment like the Emotiv EPOCH headset and the AR Parrot 2.0 drone, which is not research-grade. This isn't suited for in depth research, as the quality and reliability may vary.

The system in this project should be able to be developed independently alongside the development of cutting edge technology in both BCI and drone technology. The clarity of our conclusion largely depends on how well this system fulfils the potential of current technology in the area. Below is a list of requirements for the system developed in this project, categorised as *necessities*, *preferences* and *optional features*.

Necessities

- Real-time functionality of the entire of the system.

  Real-time functionality, however unstable, is one of thee main foci of this project. This is perhaps the most important requirement, as a system not running in real-time would defeat the purpose of steering the drone in real-time.
  An offline system might have applications in other BCI research, but that will not be examined in this study.

- BCI-EEG based control interface.

  As described in section 2.2, EEG based BCI systems arethe most well-described type of system and also the most widely available solution. We will therefore be able to draw from a great host of publications, just like

any replications or likewise studies can draw parallels to this paper. This is required to make the system viable for further development.

- Complete multi-directional control of the unmanned vehicle, and throttle.

  This goes without saying. If you have a drone that can't be controlled in all directions, then it is not to any help in field surveys or flight in general. This is why full freedom of movement for the control of the vehicle is absolutely essential.

- Implementation using technology developed for research purposes.

  We *could* replicate a system that uses AR Parrot drone 2.0 and an Emotiv BCI-EEG headset, as this has been documented in articles and publications before. For evaluation of this project, though, the setup needs to be dependent on the systems that scientific drone surveys rely on. DTU Space has proposed the ArduPilot framework, as this is what is currently used to conduct research at their facilities. This clearly allows seeing the potential of BCIs being implemented as a control unit in scientific surveys.

- A Motor Imagery based BCI system. For the purpose of controlling a drone, it is wanted to be able to follow the drone's movement with your eyes, which can be a hard task using SSVEP. This should be opposed by a possible MI-based paradigm for future drone control. Using MI to control drones is a challenge, which is why this project adds something new to the literature.

- Can be used alongside automatic missions. This would be an aid in geophysical surveys where an automatic path for the drone is programmed. This could be incorporated by sending a terminate mission command and starting the BCI-control system in a seamless way.

- Must be reproducible For examination and improvement in later studies.

Preferences

- Intuitive interface for the BCI. To be able to aid in general purposes a system that couples an intuitive thought to the respective action should be preferred. Example: Think about moving the left arm up, and the drone yaws counterclockwise.

- A sufficiently low latency to be able to evaluate direct cause and effect line from thought to movement of vehicle. The time from a command to action should be low enough to be able to without a doubt be linked together. Otherwise, we might end up evaluating the effect of the previous command during tests. This is a high priority feature.

- Direct compatibility of other likewise systems. In the case that further development is of interest to later studies or projects, a system that has a high degree of design compatibility is preferred.

Optional features

- Low latency (<500 ms.)

  This feature would make the system reliable in just about any relevant situation. However, this is not a feature that's easy to apply from the perspective of this project, as it is largely dependent on the hardware, computing power available and the number of links from thought to movement. So this is an evaluation factor, but can't necessarily be a choice incorporated in the design.

- High accuracy (>70%)

  A truly reliable BCI system should have an accuracy of about 70-80%. This would set the system at the same baseline as cutting edge BCI-technology.

- Semi-mobile setup

  It is prioritised that the system can be setup outside a lab for real life testing. However this may come down to future work.

4.1 Design of RC rover - PLUTO

For the purpose of extensive and thorough testing, the chosen platform is a remotely controlled rover which is later to be used for topological surveys by DTU Space. This vehicle is built from the ground up during the project, and the full user manual is found in the appendix, although the most important parts are included here.

This is PLUTO, the special platform build for this project.
Platform for **L**andbased **U**nmanned **T**houghtcontrolled **O**bservations

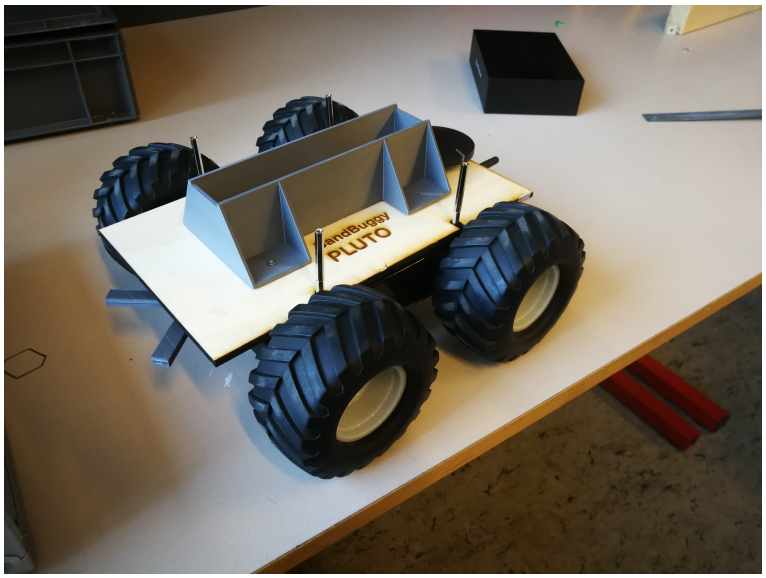


Figure 4.1: Pluto

PLUTO has the major advantage of not needing any external surveillance from an employee of DTU Space during tests, while also being significantly sturdier and cheaper than a research-grade drone. You could argue that this platform is not a drone in the *strictest* sense of the word, but the controls are easily translated between the RC vehicle and drones[1]. This is due to the implementation of the Pixhawk 2.1 controller, which can be used for both vehicle types, and was covered in chapter 2.4.2. Also, the controls for a ground-based vehicle have fewer degrees of freedom, making the controls a subset of the controls of an aerial vehicle. The method of sending motor-commands to the drone and PLUTO is the

[1]The drones that DTU Space has available for research are Tarot T960 hexacopters, where forward and left/right can be translated to yaw and pitch

same, with the exception that no pitch, roll and elevation is required. Instead, the steering of the RC vehicle simplifies to motions of thrust and yaw.

PLUTO uses skid steering, where, instead of turning the wheels to steer left or right, a difference in speed between the left and right pair of wheels create a turning motion.

The motors consist of 4 DC motors, which is coupled in pairs of right and left through a motor controller (Sabertooth 2x10 V.1.03). This translates the PWM signal from the PixHawk into DC currents driving the motors. This only has two outputs as opposed to the number of wings for the drone. This provides the same complete freedom and the output can easily be extended for use in high-end drones, simply by changing a series of parameters in the PixHawk.

4.1.1 ArduRover

As stated in the goals of this project, the firmware of the ArduRover platform has been modified to allow for semi-automatic steering. In essence, this means that the rover operator should be able to stop the automatic mission by changing modes, then steering with BCI and then changing back to automatic steering. When doing this, it should remember the spot the automatic mission had been interrupted at, in order to return there later.

We added a new mode, mode_BCI(), which was essentially the same as the manual steering mode. The only difference is that when switching to this mode, ArduRover records the current position to be used as a waypoint. This is done by overwriting the global _destination variable with the already used current_loc variable. When exiting this mode, another variable has been created to let the automatic steering mode know if there has been any BCI steering. If that is the case, the auto mode sets its next location to the recorded position. In addition, several headers had to be updated with information regarding the new mode. The code for this is included in Appendix C.

Unfortunately, due to a known bug where internal and external compasses disagree, the semi-automatic steering could not be tested in real conditions.

4.2 BCI Interface

The absolute goal for the project is to implement a MI interface for handling the drone. MI invoked signals are specifically detectable in the mu frequency band (and vanishingly in the beta band). Thus we want to apply a BP filter around 8 to 30 Hz. MI evokes an event related desynchronisation of the brain waves in this very band. This means that the power spectrum of the brain waves in the

mu and beta rhythms will show an attenuation, which should be consistently detectable. Thus, besides CSP, the final feature transformation is obtaining the power of the signal. For the interface to be intuitive, the gestures to be classified needs to be considered. For the experiments conducted right hand clenching is chosen as the class for steering right, and the left hand clenching is chosen as the class for steering left. For the forward command, the gesture of moving both arms up and away from the body was used. Test subjects found this rather intuitive and easy to imagine.

4.2.1 OpenViBE

As with many areas of science, there is a variety of solutions available and the field of BCI is no exception to this. Searching through the internet presents primarily MatLab-powered applications e.g. EEGLAB, BioSig and BCILAB, alongside the seemingly most popular solution, BCI2000. However, these solutions has very low to no documentation of use in real-time signal processing, and use very computational heavy algorithms (as a tradeoff for precision). This is mainly due to this software being developed for BCI research, which requires high precision and often only offline analysis. Some research groups have had some success with interacting from these software solutions to a real-time application, but for this application, another platform was desired.

Further delving down the informational ocean uncovered another open-source solution, OpenViBE.

OpenViBE is a platform for acquiring and processing data from BCIs, developed by the Inria Hybrid Team in France. The documentation of the program is quite comprehensive and available on their website [2]. OpenViBE supports a lot of EEG acquisition solutions, including the g.USBAmp used in this thesis. It also supports prototyping in a variety of programming languages, including MatLab and Python, which allows for easy real-time processing in an otherwise single threaded application.

OpenViBE is, in its core, a pipeline for data. This pipeline can be manipulated through modules inserted in a user-friendly graphical interface (See 4.2).

The implementation of a three-class motor imagery BCI system is implemented in partial programs and has to be followed step by step in order to approach a practical use of this paper as a tool.

- Step 1: Data acquisition for training purposes.
- Step 2: Training the CSP (Common Spatial Pattern) filters.

[2] http://openvibe.inria.fr/

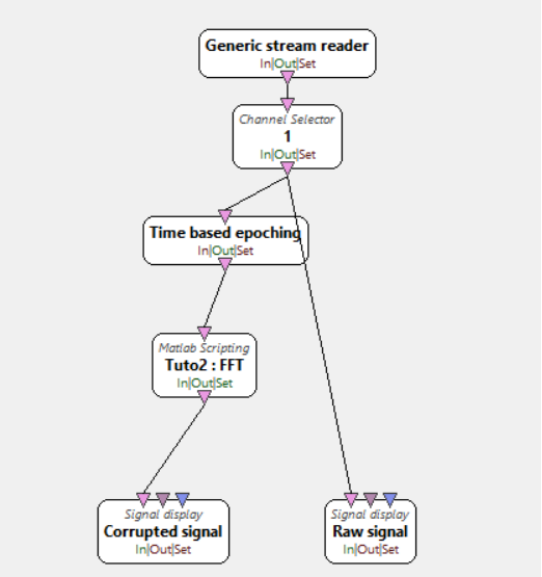


Figure 4.2: Overview of MatLab scripting tutorial found in OpenViBE.

- Step 3: Training the Classifiers.
- Step 4: Running online classification on incomming EEG-stream.
- Step 5 (optional): Evaluating performance.

This procedure of setting up and running the training easily takes approximately an hour. Furthermore, the trained classifiers and CSP filters may only be viable for at brief period of time, and exclusively for a single person, due to the signal frequency and origin varying with time and from person to person. We experienced reliability in at least two hours, with little performance difference.

4.2.2 Setup and Laboratory Procedure

The following instructions is very specific to the instruments available at the Hearing Lab at DTU and any replications might need slight adjustments based on the available equipment.

Prerequisites

The following instruments and equipment was used.

- 1 x g.USBamp

- 1 x g.GAMMAsys - Preamp
- 16 x g.LADYbird - Active Electrodes
- 1 x g.LADYbirdGND - Active Electrode
- 1 x g.LADYbird - Reference Active Electrode
- 1 x g.BCIgel - Conductive Gel
- 1 x Windows 10 Computer - with min. 2x USB 3.0 ports, OpenViBE installation and driver g.USBamp. Hardware requirements unknown.
- 1 x PLUTO - Groundbased Vehicle.
- 1 x Taranis - Drone Controller
- 1 x Arduino - With a compiled version of Mikkels Project.

First setup the electrodes and the g.GAMMAsys box with the following configuration.

Channel:	1	2	3	4	5	6	7	8	9
Placement:	C3	CP5	FC5	FC1	FZ	AFZ	FC2	C4	FC6
Channel:	10	11	12	13	14	15	16	GND	REF
Placement:	CP6	P2	P1	CP3	C1	C2	CP4	Ear	NZ

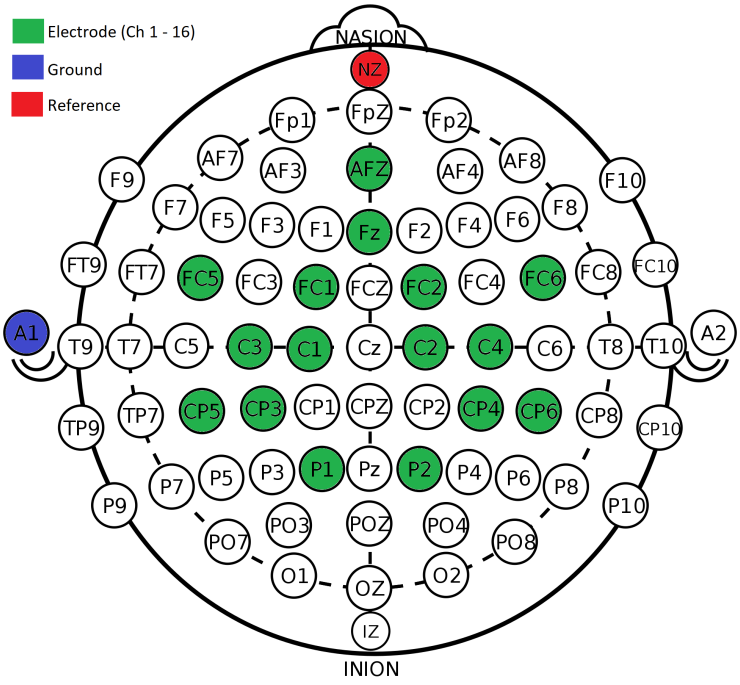


Figure 4.3: Electrode Placement on 10-20 system

Once this is done apply the cap on your subject and apply the conductive gel using a syringe. Now connect the g.GAMMAsys to the g.USBamp and the g.USBamp to the computer. Turn on everything and start up OpenViBE, and the OpenViBE acquisition client.

Step 1: Data Acquisition

Now the subject has to be seated in front of the screen in a rested position. The signal can be monitored to check that nothing is wrong using the Mi_signal_-Monitor.xml program(See Appendix A.1). When satisfied with the signal, open the 1Mi_signal_-acquisition.xml program, (See Appendix A.2). When starting this the subject is shown 3 arrows pointing right, left or up. This corresponds to the cue for imagining clenching right hand, clenching left hand or moving both arms up towards the screen. This will repeat for in a random sequence for 7 minutes.

Step 2: Training the CSP filters

Now that the training data has been acquired, start up the program for training CSP - 2Mi_csp_trainer.xml (See Appendix A.3) while fast forwarding. Remember to follow instructions in the program, and change file names. This will train the CSP filters.

Step 3: Training the classifiers

The LDA-Classifiers can now be trained in the same fashion as the CSP-filters. Open the 2Mi_classifier_trainer.xml (See Appendix A.4). Again, remember to change file names. This is the last step in setting up the BCI.

Step 4: Steering the drone

Now you will need to connect the Arduino to the computer and set up the Taranis so that it connects to PLUTO. The Arduino must be connected to the trainer port of the radio transmitter, which must be set up so that it sends commands to PLUTO on cues from the trainer port. Open the online processing program in OpenViBE 4Mi_online.xml (See Appendix A.5).

Now the subject should be able to steer PLUTO using the same imagined gestures as trained on. If you find that it reacts on too small probabilities, you might want to change the threshold values for sending steering commands in the MatLab processing script for (See Appendix B.2).

Step 5 (optional): Evaluating performance

If wanted, the performance of the steering can be evaluated. This is done similar to the procedure of the training session. A dataset is acquired by showing cues for the three classes in a random order, either while steering the drone or as a standard data acquisition (see Step 1.). This time, the stimulations recorded serve as a correct answer for the classifiers to compare to later. When data has been acquired for evaluation it is played back and the classifiers will output probability values to a MatLab script in 5MI_evaluation.xml (See Appendix B.4 and A.6), which this time also takes in the recorded stimulations. Now MatLab compares the predictions from the three classifiers, then calculates the class with the highest probability as the correct answer, and saves the results in a confusion matrix. This confusion matrix is a measure of how accurate our classifiers perform.

4.3 Overview of workflow while steering

While the theory behind all subsets of this steering paradigm is covered, the actual workflow from brain signals to motion in PLUTO might seem a little complicated. This chapter is then meant to give a better overview and recap of what has been designed.

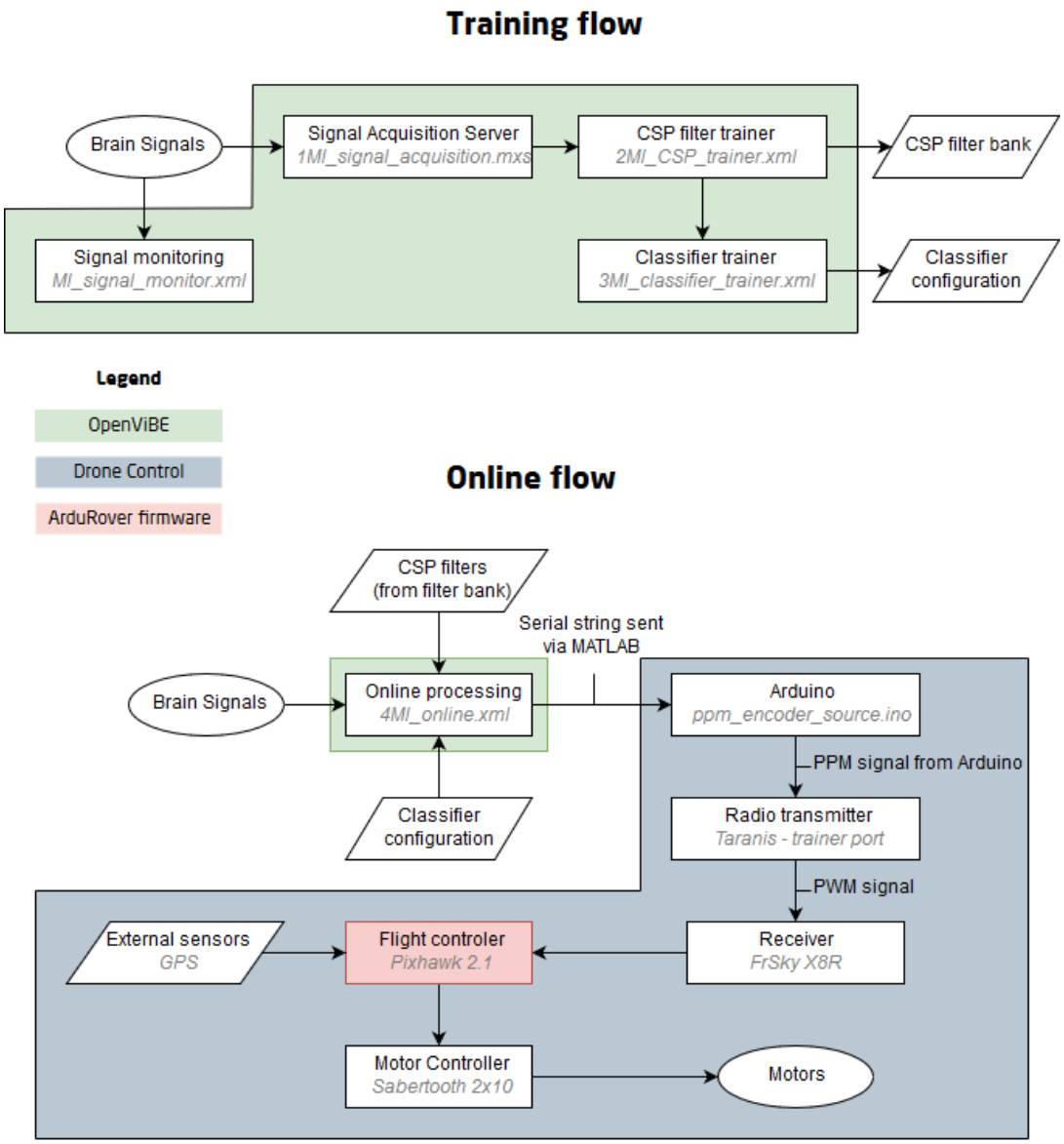


Figure 4.4: Overview of workflow

The flow chart in fig. 4.4 is split in training and online flow. When steering the drone using this paradigm, we collect the data using g.Tec instruments. OpenViBE then records the data using the implemented acquisition client and script. The training data is used to train both CSP filters and classifiers for use in the online script.

Here, brain signals are acquired directly by the acquisition script. Using the trained filters and classifiers, the data is processed real-time, during which MatLab sends a string to the Arduino. This string is then decoded and sent as a PPM signal to the Taranis radio transmitter. It sends a PWM signal to a

receiver which forwards the signal to the Pixhawk flight controller. Using this and external sensor data, it controls the motor controller, which in turn signals for Pluto to perform an action.

Chapter 5

Results and discussion

After conducting tests with the lab procedure described in section 4.2.2, we concluded with a functional setup for steering PLUTO. One of the authors of this paper acted as test subject. Instead of motor imagery, motor execution was used, as this seemed easier to focus on while having nearby disturbances. A video of the tests can be seen on YouTube.[1]

The tests concluded that a real-time link from intention to drone control is possible and could be achieved with very low latency. The core of the general control interface felt perfectly intuitive to all subjects used throughout the design procedure. The implementation of a control switch that can alternate between automatic and BCI controlled steering is by principle implemented, but currently, a firmware issue makes it hard to test. A C++ function containing the handling of this switch between these functionalities (See appendix C). This functionality is built upon the source code for the ArduRover firmware which can be found on Github [19].

After steering the drone, an evaluation for measuring the accuracy of the classifiers was conducted. The results of these tests can be seen on fig. 5.1 on the next page. Here 3 datasets have been acquired and have been individually used to train 3 different sets of classifiers. During the training of the classifiers, we conducted a 5 fold cross-validation of the training data. This is the initial measurement of the performance expected from each classifier. These are included in the diagonal plots of fig. 5.1 for each subset of classifiers (Right/Left, Right/Up and Left/Up). In the off-diagonal figures, each dataset have been tested on each of the other classifiers. The diagonal of each confusion matrix is a measure of how often the classifiers collectively classified the correct outcome, while the off-diagonal squares are a measure of how often they collectively did not classify correctly.

[1] https://www.youtube.com/watch?v=wYRMMQU5Fu8

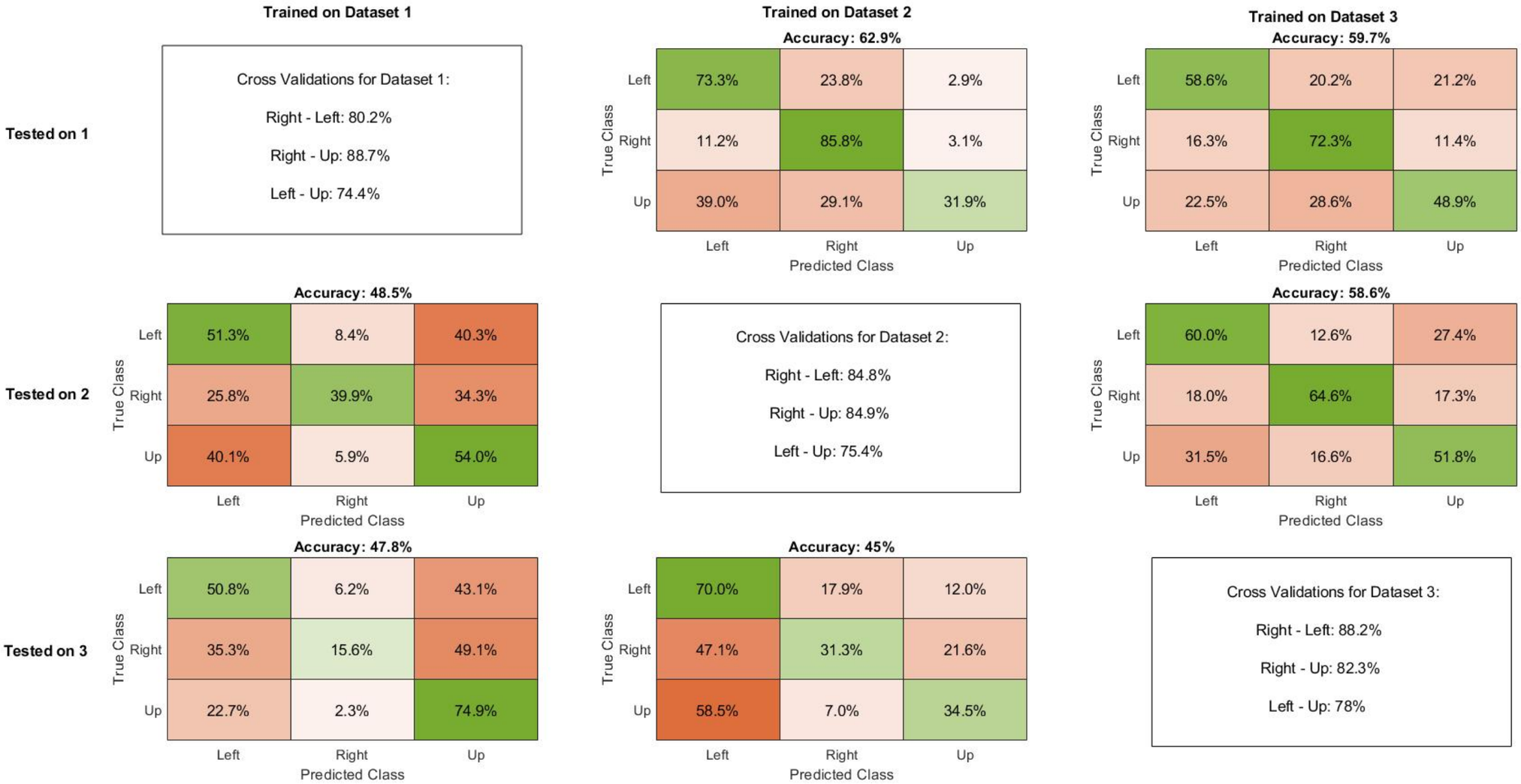


Figure 5.1: Confusion Matrixes and Cross Validation for 3 different classifiers trained on 3 different datasets. Each row corresponds to a different

From these results, we can see that the classifiers definitely aren't just guessing the result since that would give an accuracy around 33.3%. Conducting a quick χ^2 test on our data shows a less than 0.0001% chance that this would have been the result of random guessing, which firmly states that the steering is a product of intention. It is also seen that the classifiers perform differently on different datasets. This might be because it's hard to copy the exact same cognitive task, or maybe that something has changed slightly between data collection runs. That could be EEG cap placement, environment, or maybe a more fundamental change of the test subjects state of mind.

Chapter 6

Conclusion

As a whole, this project shows that a link can be established between current state-of-the-art software and hardware in both drone control and BCI. It is demonstrated that standard CSP algorithms and unmodified LDA classifiers, in a 3-class classification system used with open source drone control software, can be used to send (admittedly, somewhat unstable) directional controls to a drone.

The motivation for this project was to test if BCI would work as an extension for drone control in field surveys. This could help by freeing up hands to take notes or further examine an area of interest without the need of an external controller. Since only a rudimentary steering has been achieved, this would not be ideal yet.

From looking at 5.1 we see that the accuracy ranged from 45% to 62.9%, which is somewhat satisfactory for this project. The result is not ground-breaking in the BCI field by any means, but that has never been the intention of this project. Proving that the accuracy of the controls is statistically significant, means that the primary goal of implementing MI-BCI in real-time for steering a drone is a resounding success.

It is also seen that some classifiers are heavily weighted towards one class. For instance, the classifier trained on dataset 2 classifies *left* above the other classes (about 50% of the time), and the one trained on dataset 1 classifies *up* more often (about 46% of the time), and very rarely classifies right. This might be an indicator that one of the classes is being significantly classified by the part of the LDA looking for the other two classes. The classifier trained on dataset 3 performed particularly well since it decently classified all classes correctly in an even manner. Because of this, it would be the most desired classifier to use for steering a vehicle, even though all classifiers perform similarly in overall accuracy. Seeing that the different datasets might produce consistently wrong classifications means that it is crucial for this type of BCI system to gather the best possible training-data.

Accuracy and inconvenience of the equipment are some of the most noticeable obstacles before the technology will appeal to everyday users. The best results are also achieved when focused and relaxed, a state few achieve in a steady work day, let alone when in the field with multiple distractions every second. Before it will be feasible, either rigorous

focusing training will have to be undertaken by the user, or machine learning algorithms capable of better distinguishing intent will have to be implemented. The first option undermines one of the primary goals of the project, ease-of-use, while the second is very much still in development.

A goal of the project was to achieve modularity with open source software and hardware, giving an easy-to-use way to build upon the steering paradigm. This has been achieved in a very satisfactory manner, connecting widely used drone controllers with custom semi-automatic steering firmware to a research grade EEG sensor system, and actually outputting steering commands. Each individual part can be replaced, mostly without an impact on the rest of the pipeline.

Chapter 7

Further improvements

Even though the results from experiments were somewhat satisfactory, the system proposed is by no means ready to make a commercial adaption yet. It might be proof that motor imagery based BCIs do have some potential for aviation and vehicle control. The actual imagery part presented a steep learning curve for the user, which is why actual movement was used for demonstration. The cognitive task of keeping an intent on moving but not performing the movement does feel hard to do consistently as we aren't trained by default to do this, but as other studies have shown, practice helps[15][14].

A major issue of the project is the BCI output not being consistent enough. This is, in fact, a main focus in most contemporary BCI research. The methods used to produce the experiments presented are also not the state of the art. The CSP filter is a generic one, based on the original algorithm and the LDA classifier has pros and cons as well. It might be interesting to look into adaptive versions of the classifiers as well to reduce setup time and ease the workload of training them, while simultaneously increasing accuracy. The lack of plug-and-play implementation of better algorithms in OpenViBE leaves this to further improvements of the program.

The experiments did show slight issues with MatLab clogging the pipeline of OpenViBE in order to synchronise the data flow. It makes it difficult to consistently reach full real-time processing with this basic sort of scripting. Fortunately, it is fixable by implementing all algorithms used in MatLab in C++ which OpenViBE is based on, although could easily become a considerable piece of work.

The setup used during the experiments could be improved as well. One of the improvements the future might provide is using a portable setup. The g.Tec setup is quite bulky and tethers the user to a seated position, but this may require some sacrifice towards the quality of the signals.

As for drone control, a more intricate steering paradigm might be in order, since 3 degrees of freedom is too little in most cases.

7.1 Use cases

For the possible implementations of systems like the one proposed only the imagination is limiting. A few of these are listed here

- **General use cases**
- Handicap assistance with BCI - The primary case of interest to BCI researchers. It can be used for disabled as wheelchair control, communication systems or just allowing various control abilities for the user. Tariq et al. (2018) show that the wealth[20] of BCI protocols being developed just for movement assistance is advancing rapidly, and predicts great feasibility in the field.
- Smart-home control - The use of BCI could possibly extend control of devices being used in our daily lives. This might be used similarly to a light switch triggering on claps.
- Body extensions - Extending body functionality, adding an extra limb or assistive robot-devices might be in the realm of possibilities for BCI technology in the near future. This could make dangerous or heavy tasks more convenient and help safety in specialized work environments, such as construction.
- Manned aircrafts - The applications of this project is not exclusive for unmanned vehicles, but similar paradigms could be extended to manned aircraft or the general transport sector, freeing hands for safer transport.

7.2 Future improvements

In perspective, many things need improvement, as this is not a project with closed ends. Both BCI and drone technology are largely unstandardized fields. Presented here is a list of things that could build upon this project directly in order to improve it.

- Multiclass spatial filters - the fact that the spatial filters used are only capable of handling two classes as ready-to-use algorithms in OpenViBE is an issue for doing optimal three-class classifiers, which is also the primary reason that three class classification had to be done by the concatenation of 3 two-class classifiers. This could improve the classification being oddly weighted. Such a project would require some extensive C or C++ programming to create a box algorithm directly in OpenViBE.
- Sophisticated classifiers - In this project, a very basic and barebones classifier have been used. It is very likely that a more sophisticated and perhaps adaptive classifier algorithm would perform much better.

- Full implementation and improvements on semiautomatic steering - As this project is mainly focussed on establishing a firm link between BCI and drones, the semi-automatic steering and specific implementation for our use-case is left untested. Another benefit of using the ArduPilot architecture is that it is equally as adaptable to boats, tractors, planes and autonomous robots as it is with rovers and drones. It would definitely be possible to implement the system in different applications for further testing the possibilities in the symbiosis between BCI and drone tech.

- Fully open source the project - Currently the g.tec hardware is the only part of this project not being open source. An open source amplifier would create more possibilities for customization and optimization.

- Extend to 3 dimensional (or maybe more) steering - It would be interesting to combine different kinds of BCI systems for making N-class systems that could control things with very high degrees of freedom. a direct extension of this project would be to have full 3-dimensional steering in an aerial drone.

- Reduce number of links - The setup for this project is somewhat redundant. It would definitely be possible to send signals directly from a computer to the drone without the use of Taranis and Arduino. This might require some external hardware to produce the PWM signals, and most likely a very high amount of programming. With a reduction in computing demands for signal processing, it might even be possible to skip the computer entirely and create a truly portable system.

An improvement of the system is already being conducted by another group of students on DTU. It is our hope that in time, integration of the principles discussed here will become more widely used.

References

[1] Rajesh P. N. Rao. *Brain-Computing Interfaces: An Introduction*, volume 1. Cambridge University Press, 2013.

[2] Alessandra Ibba. *Introduction to brain-computer interface systems - Internship report*. DTU Elektro.

[3] Bernhard Graimann, Brendan Allison, and Gert Pfurtscheller. *Brain-Computer Interfaces: A Gentle Introduction*. Springer-Verlag Berlin Heidelberg.

[4] Motoaki Kawanabe, Carmen Vidaurre, Simon Scholler, and Klaus-Robert Muuller. Robust common spatial filters with a maxmin approach. *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009.

[5] H. Ramoser, J. Müller-Gerking, and G. Pfurtscheller. Optimal spatial filtering of single trial eeg during imagined hand movement. *IEEE Trans. Rehab. Eng*, 8:441–446, 1998.

[6] Guido Dornhege, Benjamin Blankertz, Gabriel Curio, and Klaus-Robert Müller. Increase information transfer rates in bci by csp extension to multi-class. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 733–740. MIT Press, 2004.

[7] Master's thesis.

[8] C. Vidaurre, A Schlögl, B. Blankertz, M. Kawanabe, and K.-R. Müller. Unsupervised adaption of the lda classifier for brain-computer interfaces. 2012.

[9] ProfiCNC and Pixhawk®. Pixhawk v2 feature overview. `http://www.hex.aero/?page_id=236`. Last visited: 01.03.2019.

[10] Michael Oborne. Mission planner documentation. `http://ardupilot.org/planner/`. Last visited: 06.05.2019.

[11] Mikkel Bugge. External uav control. B.Sc. Thesis, Technical University of Denmark, July 2017.

[12] Rafael Duarte. Low cost brain computer interface system for ar.drone control. Master's thesis, Federal University of Santa Catarina, https://www.researchgate.net, June 2017.

[13] Tianwei Shi, Hong Wang, and Chi Zhang. Brain computer interface system based on indoor semi-autonomous navigation and motor imagery for unmanned aerial vehicle control. 2015.

[14] Josh Cooper Adnan Rashied Eric Rawls Jason Walters Utku "Victor" Sahin Kade Randell Sarah North, Ahmad Alissa and Cheyenne Sancho. Performance analysis of brain control interface in drone applications. 2018.

[15] Karl LaFleur, Kaitlin Cassady, Alexander Doud, Kaleb Shades, Eitan Rogin, and Bin He. Quadcopter control in three-dimensional space using a noninvasive motor imagery-based brain–computer interface. *Journal of Neural Engineering*, 2013.

[16] Andrei Chiuzbaian. Drone control in 3d using brain-computer interface (bci). Master's thesis, Technical University of Denmark, June 2017.

[17] Kristian Moltved and Peter Flintenborg-Simonsen. A cvep-based hybrid bci system using eye tracking for drone control in 3d. Master's thesis, Technical University of Denmark, May 2018.

[18] Audrey S. Royer, Alexander J. Doud, Minn L. Rose, and Bin He. Eeg control of a virtual helicopter in 3-dimensional space using intelligent control strategies. *IEEE transactions on neural system and rehabilitation systems*, 2010.

[19] ArduPilot. Github directory for ardurover firmware. https://github.com/ArduPilot/ardupilot/tree/master/APMrover2. Last visited: 15.05.2019.

[20] Madiha Tariq, Pavel M. Trivailo, and Milan Simic. Eeg-based bci control schemes for lower-limb assistive-robots. *Frontiers in Human Neuroscience*, 12:312, 2018.

Appendices

Appendix A

OpenViBE Programs

A.1 Mi_signal_Monitor.xml

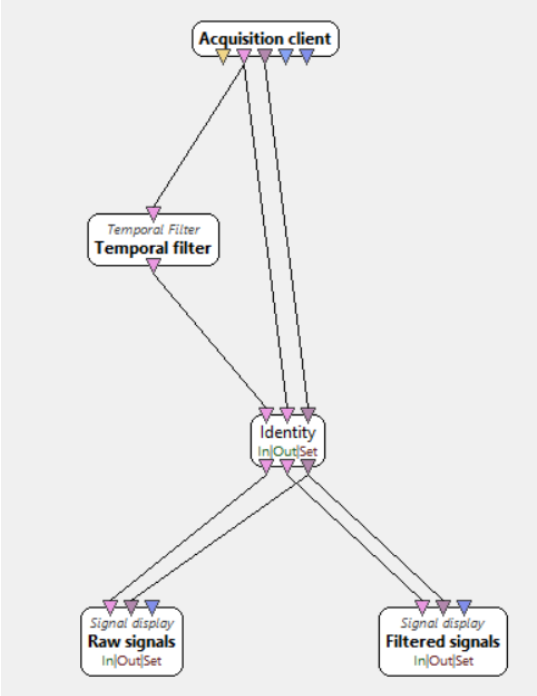


Figure A.1: Signal monitor program

This program monitors raw EEG data real time, filtered and unfiltered. This is used to make sure all electrodes works properly. The program acquires data from the OpenViBE acquisition server, bandpass filtering it between 8-30Hz and displays both filtered and unfiltered signals.

A.2 1Mi_signal_acquisition.xml

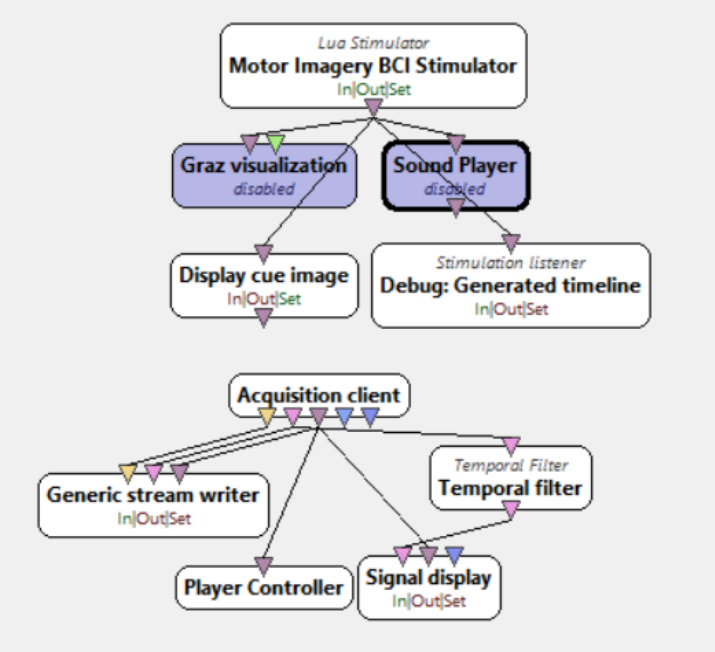


Figure A.2: Signal acquisition

Script for data acquisition. The top block creates random stimulations from a Lua script, choosing between 3 different stimulation types (OVTK_GDF_Right, OVTK_GDF_Left and OVTK_GDF_Up). It then sends these stimulations to either the "Graz visualization" box, which supports up to 4 different cues (up, right, left and up) using the Graz BCI paradigm, or the "Display cue image" box, which supports custom images up to n classes. The sound player box receives stimulations from the Lua script in order to warn a user that a cue is about to appear. This box only works if OpenAL is installed.

The bottom block receives signals from the OpenViBE Acqusition server (a separate program), relaying it to the "Generic Stream Writer" box, which saves the data. The raw data is also displayed, and the "Player Controller" box stops the program when no more stimulations are sent.

A.3 2Mi_Csp_trainer.xml

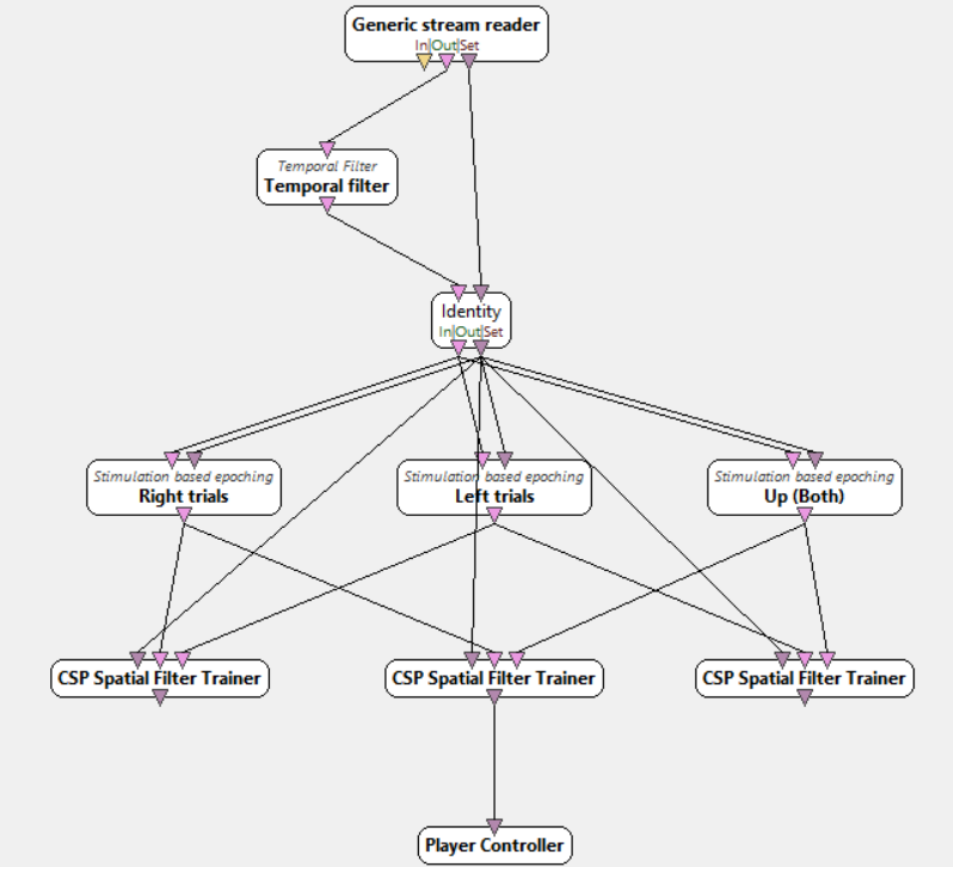


Figure A.3: CSP Trainer Program

The CSP trainer trains a filter-bank of 3 different spatial filters on the recorded data. It firsts epochs the data into time-chunks of the same stimulation, then trains. After training, the program is ended and filters are saved.

A.4 3Mi_classifier_trainer.xml

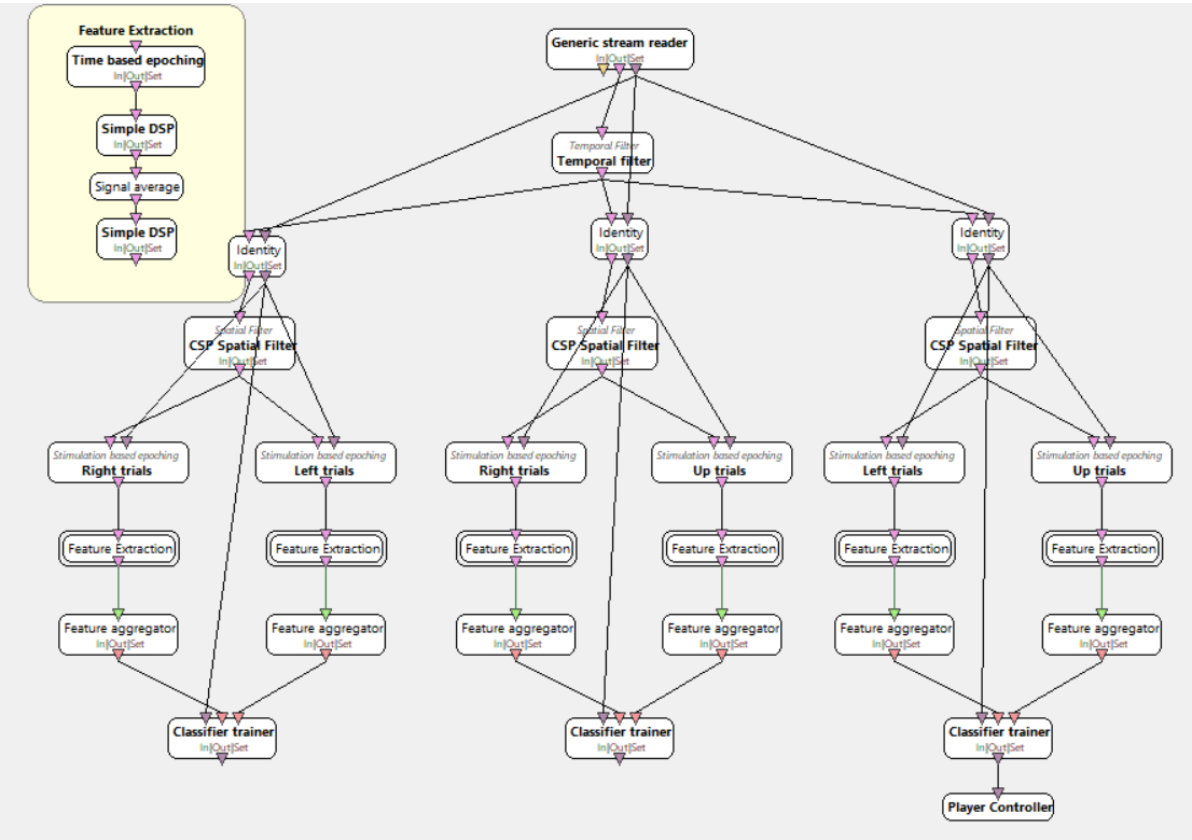


Figure A.4: LDA Trainer Program

The classifier trainer trains 3 classifiers on recorded data. First it applies already trained CSP filters, then does some simple signal processing (finding the signal power), before aggregating features. The classifiers (LDA) are then trained on the aggregations.

A.5 4Mi_online.xml

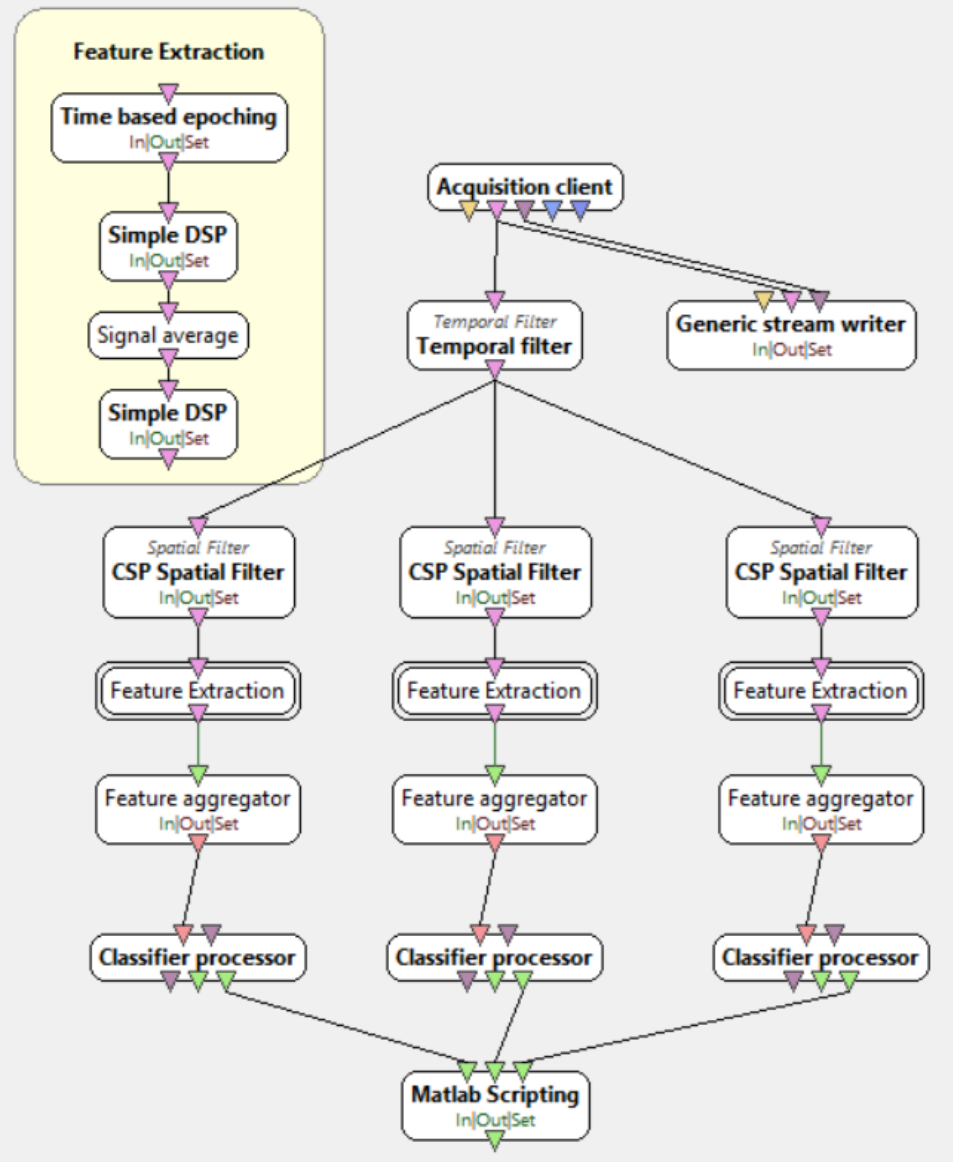


Figure A.5: Online Classification

The online script acquires data from the OpenViBE Acquisition Server. It saves the data via the "Generic Stream writer" box. It applies a band pass filter (8-30Hz 5th order Butterworth), applies trained CSP filters does feature extraction as a simple signal processing (finding the signal power), and classifies the data in pairs of 2 classes. It outputs probability values, which are then processed by a MATLAB scribt (see appendix B). In this script, signals are sent to an Arduino, which in turn send controls to the drone.

A.6 5Mi_Evaluation.xml

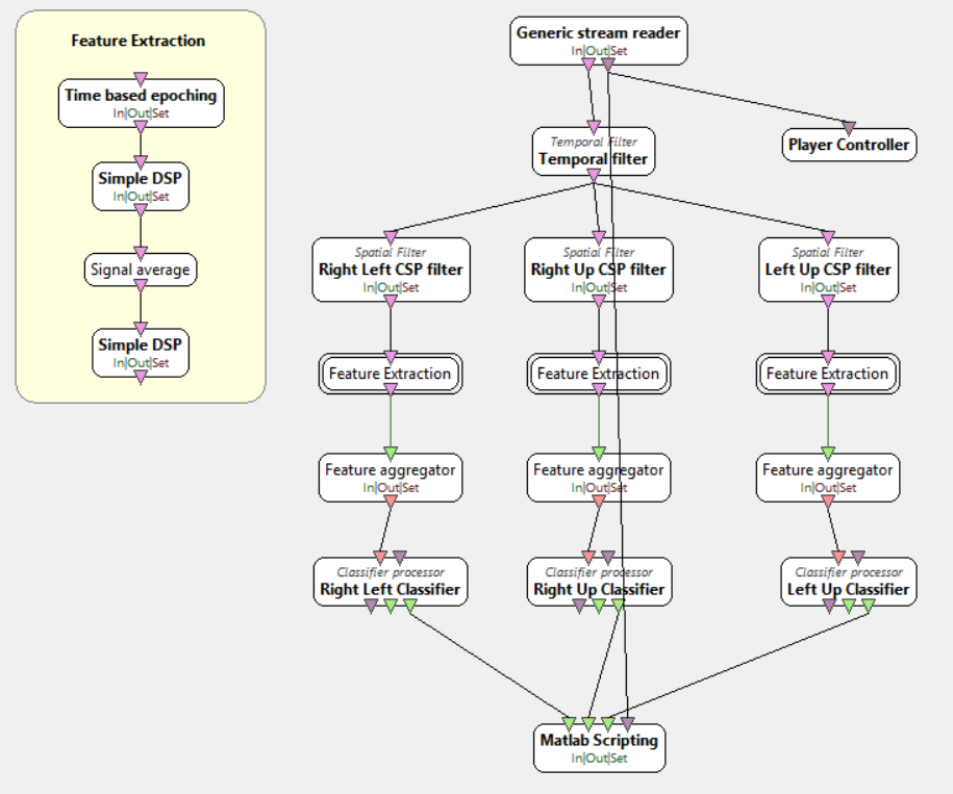


Figure A.6: Evaluation Script

This script is very similar to the online script, the major difference is that is plays back a dataset acquired earlier and instead of outputting an output sequence for the arduino to handle, it compares the classifications being estimated by the LDA classifiers and compares it to the actual performed cue. Then it creates a confusion matrix from the results.

Appendix B

MATLAB Scripts

B.1 threeclass_probability_Matrix_Initialize.m

```
% threeclass_probability_Matrix_Initialize.m
% ------------------------------
% Author : Aleksander Lund & Carl Emil Elling
% Date   : May 2019
%
% Initialize function for Online MI-BCI in OpenViBE.
% Sets box settings.
%
% Inputs: box_in: 3 streamed matrixes of probability values from 3 separate classifiers, classifying 3 classes.
%
% Outputs: box_out: information sent to process script, including settings and user data

function box_out = threeclass_probability_Matrix_Initialize(box_in)

        % we display the setting values
    disp('Box settings are:')
  for i=1:size(box_in.settings,2)
    fprintf('\t%s : %s\n',box_in.settings(i).name, num2str(box_in.settings(i).value));
    end

  % adding user data to (in process function) determine if output header and arduino output is set
    box_in.user_data.is_headerset = false;

```

```
24      box_out = box_in;
25
26  end
```

B.2 threeclass_probability_Matrix_Process.m

```
1   % threeclass_probability_Matrix_Process.m
2   % -------------------------------
3   % Author : Aleksander Lund & Carl Emil Elling
4   % Date   : May 2019
5   %
6   % Process function for Online MI-BCI in OpenViBE. Based on script by
7   % Laurent Bonnet (INRIA).
8   %
9   % Inputs: box_in: 3 streamed matrixes of probability values from 3
    ↪  separate
10  % classifiers, classifying 3 classes.
11  %
12  % Outputs: box_out: Normalized probability values
13  %          Serial output sending commands to custom Arduino scripts
14
15  function box_out = threeclass_probability_Matrix_Process(box_in)
16
17          if(~box_in.user_data.is_headerset)
18              %Set output channels
19              box_in.outputs{1}.header = box_in.inputs{1}.header;
20              box_in.outputs{1}.header.nb_channels = 3;
21        box_in.outputs{1}.header.channel_names = {'R','L','B'};
22              box_in.user_data.is_headerset = 1;
23              % Print the header in the console, so as to see what is
                ↪  being
24              % sent
25              disp('Input header is :')
26              box_in.inputs{1}.header
27              disp('Output header is :')
28              box_in.outputs{1}.header
29
30              %Setup of Arduino output. Some info will have to be entered
31              %manually here due to constraints in MATLAB
32              %CHANGE THIS:
33              box_in.user_data.port = "COM6"; %Name of COM port where
                ↪  Arduino is connected.
```

```
34
35              %Do not change this. Initializing Arduino output
36              box_in.user_data.s =
                ↪  serial(box_in.user_data.port,'BaudRate',9600);
37              fopen(box_in.user_data.s);
38              box_in.user_data.ArduinoOutput = '<E047T050R050A050>';
39          end
40      % Test if input lengths are equal
41      if (OV_getNbPendingInputChunk(box_in,1) ==
        ↪  OV_getNbPendingInputChunk(box_in,2)) &&
        ↪  (OV_getNbPendingInputChunk(box_in,1) ==
        ↪  OV_getNbPendingInputChunk(box_in,3))
42          % Iterating over the pending chunks on input 1
43          for i = 1:OV_getNbPendingInputChunk(box_in,1)
44              % we pop the first chunk to be processed, note that box_in
                ↪  is used as the output variable to continue processing
45              [box_in, start_time1, end_time1, Mat1] =
                ↪  OV_popInputBuffer(box_in,1);
46              [box_in, start_time2, end_time2, Mat2] =
                ↪  OV_popInputBuffer(box_in,2);
47              [box_in, start_time3, end_time3, Mat3] =
                ↪  OV_popInputBuffer(box_in,3);
48
49              %Adding probabilities for each class together.
50              % These input probabilities assume:
51              % first channel is right vs left classified
52              % second channel is right vs forward classified
53              % third channel is left vs forward classified
54              A = Mat1(1) + Mat2(1); %Right
55              B = Mat1(2) + Mat3(1); %Left
56              C = Mat2(2) + Mat3(2); %Forward
57              % 3 classifiers will have probabilities adding up to 300%.
58              % Normalizing with the sum of all probabilities gives a max
59              % probability of 66.67% for each class.
60              O = [A B C]/sum([A,B,C]);
61
62              %Steering
63              if O(1)>=0.5 %threshold percentage is set empirically
64                  tmpArduinoOutput = '<A060>'; %Go right
65
66              elseif O(2)>=0.5
67                  tmpArduinoOutput = '<A040>'; %Go left
68              elseif O(3)>=0.5
69                  tmpArduinoOutput = ' <E060> '; %Go forward
```

```
            else %If no threshold is exceeded
                tmpArduinoOutput = '<E047T050R050A050>'; %stand still
            end
            % If Arduino output changes, update the value - otherwise keep going the intended direction
            if ~strcmp(tmpArduinoOutput,box_in.user_data.ArduinoOutput)
                box_in.user_data.ArduinoOutput = tmpArduinoOutput;

                fwrite(box_in.user_data.s,box_in.user_data.ArduinoOutput);
            end


            box_in = OV_addOutputBuffer(box_in,1,start_time1,end_time1,0);

        end
    end
    % Pass the modified box as output to continue processing
    box_out = box_in;
end
```

B.3 threeclass_probability_Matrix_Uninitialize.m

```
% threeclass_probability_Matrix_Uninitialize.m
% --------------------------------
% Author : Aleksander Lund & Carl Emil Elling
% Date   : May 2019
%
% Uninitialize function for Online MI-BCI in OpenViBE. If needed,
% statistics can be extracted in this script. (eg. command history and the
% likes)
%
% Inputs: box_in: inputs from threclass_probability_Matrix_Process.m script
% This includes probability values
%
% Outputs: box_out: Normalized probability values
%
function box_out = threeclass_probability_Matrix_Uninitialize(box_in)
    disp('Uninitializing the box...')

```

```
18      %Closes serial connection to Arduino
19    fclose(box_in.user_data.s);
20      %disconnects and deletes all instrument objects
21      instrreset;
22      box_out = box_in;
23  end
```

B.4 Eval_Matrix_process.m

```
1   % Eval_Matrix_Process.m
2   % -------------------------------
3   % Author : Aleksander Lund & Carl Emil Elling
4   % Date   : May 2019
5   %
6   % Process function for performance evaluation of online MI-BCI in
    ↪  OpenViBE. Based on script by
7   % Laurent Bonnet (INRIA).
8   %
9   % Inputs: box_in: 3 streamed matrixes of probability values from 3
    ↪  separate
10  % classifiers, classifying 3 classes. 1 stimuli used to compare
11  % classification to.
12  %
13  % Outputs: box_out: Normalized probability values
14
15
16  function box_out = Eval_Matrix_Process(box_in)
17
18          if(~box_in.user_data.is_headerset)
19              %Set output channels
20              box_in.outputs{1}.header = box_in.inputs{1}.header;
21              box_in.outputs{1}.header.nb_channels = 3;
22        box_in.outputs{1}.header.channel_names = {'R','L','B'};
23              box_in.user_data.is_headerset = 1;
24              % Print the header in the console, so as to see what is
                ↪  being
25              % sent
26              disp('Input header is :')
27              box_in.inputs{1}.header
28              disp('Output header is :')
29              box_in.outputs{1}.header
30
```

```
31              % Saves
32              box_in.user_data.CMatrix = zeros(3);
33              box_in.user_data.Stim = 0;
34              box_in.user_data.LastStim = 0;
35
36          end
37      % Test if input lengths are equal
38      if (OV_getNbPendingInputChunk(box_in,1) ==
        ↪  OV_getNbPendingInputChunk(box_in,2)) &&
        ↪  (OV_getNbPendingInputChunk(box_in,1) ==
        ↪  OV_getNbPendingInputChunk(box_in,3))
39          % Iterating over the pending chunks on input 1
40
41          for i = 1:OV_getNbPendingInputChunk(box_in,1)
42              % we pop the first chunk to be processed, note that box_in
                ↪  is used as the output variable to continue processing
43              [box_in, start_time1, end_time1, Mat1] =
                ↪  OV_popInputBuffer(box_in,1);
44              [box_in, start_time2, end_time2, Mat2] =
                ↪  OV_popInputBuffer(box_in,2);
45              [box_in, start_time3, end_time3, Mat3] =
                ↪  OV_popInputBuffer(box_in,3);
46
47              %Actual stims
48              [box_in, start_time4, end_time4, Stim_Set] =
                ↪  OV_popInputBuffer(box_in,4);
49
50              %Adding probabilities for each class together.
51              % These input probabilities assume:
52              % first channel is right vs left classified
53              % second channel is right vs forward classified
54              % third channel is left vs forward classified
55              A = Mat1(1) + Mat2(1); %Right
56              B = Mat1(2) + Mat3(1); %Left
57              C = Mat2(2) + Mat3(2); %Forward
58              % 3 classifiers will have probabilities adding up to 300%.
59              % Normalizing with the sum of all probabilities gives a max
60              % probability of 66.67% for each class.
61              O = [A B C]/sum([A,B,C]);
62              maximum = max(O);
63              Prediction = O==maximum;
64              Certainty = ((maximum/0.66)*Prediction);
65
66              if(numel(Stim_Set) >= 3) % at least one stim in the set.
```

```
67                  box_in.user_data.Stim = Stim_Set(1);
68              end
69              %770 = right, 769 = left, 780 = up, 800 = End of trial
70              if((box_in.user_data.Stim == 770) || (box_in.user_data.Stim
                ↪  == 769) || (box_in.user_data.Stim == 780) ||
                ↪  (box_in.user_data.Stim == 800))
71                   box_in.user_data.LastStim = box_in.user_data.Stim;
                      ↪  %Added to keep checking the last stimulation
72              end
73
74              if (box_in.user_data.LastStim == 770)
75                          box_in.user_data.CMatrix(1,:) =
                            ↪  box_in.user_data.CMatrix(1,:) + Prediction *
                            ↪  1;
76              elseif (box_in.user_data.LastStim == 769)
77                          box_in.user_data.CMatrix(2,:) =
                            ↪  box_in.user_data.CMatrix(2,:) + Prediction *
                            ↪  1;
78              elseif (box_in.user_data.LastStim == 780)
79                          box_in.user_data.CMatrix(3,:) =
                            ↪  box_in.user_data.CMatrix(3,:) + Prediction *
                            ↪  1;
80              end
81
82              % Save workspace for further analysis. CMatrix has Targets
83              % vertical and predictions Horizontal.
84              save('workspace');
85
86              %for outputting confusion matrix
87              %box_in =
                ↪  OV_addOutputBuffer(box_in,1,start_time1,end_time1,(box_in.user_data.C
88          end
89      end
90      % Pass the modified box as output to continue processing
91      box_out = box_in;
92  end
```

Appendix C

Semi-automatic steering with C++

```
////////////////////////////////////////////////////////////////////
//////////////////* New mode: Mode_BCI.cpp: *////////////////////////
////////////////////////////////////////////////////////////////////
#include "mode.h"
#include "Rover.h"

#define AUTO_GUIDED_SEND_TARGET_MS 1000

bool ModeBCI::enter(){
    _destination = rover.current_loc;  // Sets target location for the rover
}

void ModeBCI::update(){
    /* executes same steering commands as Mode_manual. */

    float desired_steering, desired_throttle, desired_lateral;
    get_pilot_desired_steering_and_throttle(desired_steering, desired_throttle);
    get_pilot_desired_lateral(desired_lateral);

    // if vehicle is balance bot, calculate actual throttle required for balancing
    if (rover.is_balancebot()) {
        rover.balancebot_pitch_control(desired_throttle);
    }

    // set sailboat mainsail from throttle position
    g2.motors.set_mainsail(desired_throttle);

```

```
28      // copy RC scaled inputs to outputs
29      g2.motors.set_throttle(desired_throttle);
30      g2.motors.set_steering(desired_steering, false);
31      g2.motors.set_lateral(desired_lateral);
32  }
33
34  void ModeBCI::_exit()
35  {
36      // stop running the mission
37      FromBCI = 1;
38      // clear lateral when exiting manual mode
39      g2.motors.set_lateral(0);
40  }
41  ////////////////////////////////////////////////////////////////////
42  ////////////////* Addition in mode_Auto: *//////////////////////////
43  ////////////////////////////////////////////////////////////////////
44
45  bool ModeAuto::_enter() {
46
47      // init location target
48      if (~fromBCI) {
49          set_desired_location(rover.current_loc);
50      } else {
51          set_desired_location(_destination);
52      }
53
54  }
55
56  ////////////////////////////////////////////////////////////////////
57  ////////////////* Addition in mode.h: *////////////////////////////
58  ////////////////////////////////////////////////////////////////////
59  // Creates variable to tell Mode_auto, that the rover has been in BCI
     ↪  mode,
60  // Also has to be added to all Mode_xxxx_exit() Functions
61  bool FromBci = 0;
62
63  // in enum Number{}, creates a mode index for the BCI mode.
64  BCI = 17;
65
66  //
67  class ModeBCI : public Mode
68  {
69  public:
70
```

```
71      uint32_t mode_number() const override { return BCI; }
72      const char *name4() const override { return "BCImode"; }
73
74      // methods that affect movement of the vehicle in this mode
75      void update() override;
76
77      // attributes for mavlink system status reporting
78      bool has_manual_input() const override { return true; }
79      bool attitude_stabilized() const override { return false; }
80
81      // manual steering does not require position or velocity estimate
82      bool requires_position() const override { return false; }
83      bool requires_velocity() const override { return false; }
84
85  protected:
86
87      void _exit() override;
88  };
89
90  ////////////////////////////////////////////////////////////////
91  ////////////////* Addition in mode.cpp: *////////////////////////
92  ////////////////////////////////////////////////////////////////
93
94  // Under Mode *Rover::mode_from_mode_num(const enum Mode::Number num)
    ↪  add:
95  case Mode::Number::BCI:
96          ret = &mode_BCI;
97          break;
```

Appendix D

PLUTO Documentation

SandBuggy – PLUTO
Documentation and user manual

Introduction

This manual documents the setup and use of an remotely piloted (RP) vehicle setup made for DTU Space. Its purpose of doing geophysical surveys along beaches and other difficult terrain, providing a safe and stable platform for data gathering. The manual contains a setup guide for similar types of vehicles and a user manual to get you started using the platform.

The hardware and software used is as follows;

Hardware:

PixHawk 2.1 – Cube.

Sabertooth 1.03 – Motor controller

Here 2.0 – GPS module

Taranis Q X7 – Telemetry based controller

FrSky X8R – Radio Telemetry Reciever

Software:

MissionPlanner[1]

First time setup

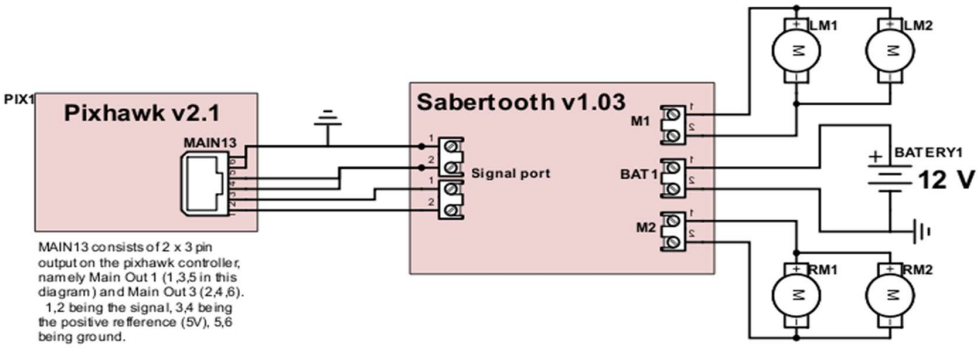
When setting up the UGV, the ArduPilot framework is used. For reference, this manual uses a software as interface called Mission Planner. Note that other software is available and any software that is compatible with the Pixhawk is usable with this vehicle.

Motor Controller.

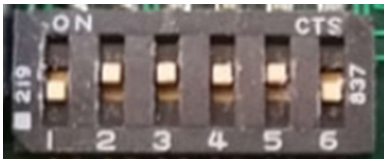
As the Pixhawk flight controller can't send direct signals to the motors, a link is needed. For this, the Sabertooth Motor controller has been chosen. The purpose of the Sabertooth board is to convert analog PWM signals from the Pixhawk to specific voltages, corresponding to motor function. This is handled by and inputted in the signal ports. This signal is then processed and is passed to the motor with the support of a

[1] Download - http://ardupilot.org/planner/docs/mission-planner-installation.html

separate 9-15 volt battery. The exact wiring can be seen in the schematics diagram below. Connection between the Sabertooth and Pixhawk is done using Futaba cables cut on the Sabertooth end.



Now that the Pixhawk is connected to the Sabertooth, the mode of the motor controller needs to be chosen. This is done on 6 dip-switches as seen on this picture:



This sets the motor to PWM signal input with simple steering of right and left motors. Now the Sabertooth setup is done.

The purpose of the Sabertooth controller is to convert the PWM (Pulse Witdth Modulation) signal, into a DC input for the motors.

FrSky X8R Radio telemetry.

For the radio receiver (FrSky X8R) connection to the Pixhawk, you will need to connect the RC IN port on the Pixhawk to the S-BUS port on the FrSky unit. This unit establishes the radio link to your flight controller. For this guide the Taranis Q X7 was used. It is important that both receiver and transmitter use corresponding software (EU or US)

Pixhawk peripherals.

For the rest of the peripherals, the wiring is straight forward. A guide can be found in the quickstart guide for the PixHawk on the website:

http://www.hex.aero/wp-content/uploads/2016/09/PIXHAWK2-Assembly-Guide.pdf

Software Setup.

There are two recommended software packages for accompanying the Pixhawk, APM planner and Mission Planner. The PLUTO platform is developed using the Mission Planner software. To install this, download it from their website[1] . When this is installed you should connect PLUTO to the computer that Mission Planner is installed on. Now you should be able to connect with the button on the top right of the screen, just choose the correct com port.

In the case that no firmware is installed in the Pixhawk you can browse to the initial setup and run through the setup wizard available.

Pairing with the Taranis.

If you are running this with a Taranis unit not already paired with PLUTO you can do this by browsing the

Taranis to Model Setup -> Binding. Now initiate the binding process and let Taranis search for telemetry signals to pickup. While this is done you need to hold down the button on the FrSky 8XR unit.

It is recommended to follow along a Youtube tutorial[2], while making sure the X8R and the Taranis both have corresponding (EU or US) software

Setting up the actual manual controls is tricky and would be difficult to describe in text. I recommend checking out Painless 360 on Youtube to achieve this.[3]

When doing this, make sure the trainer port is used as controls for the channels chosen in Mission Planner as steering.

[2] https://www.youtube.com/watch?v=UGSRWx_JWUg&t=639s
[3] https://www.youtube.com/watch?v=agjHu-WhCJw

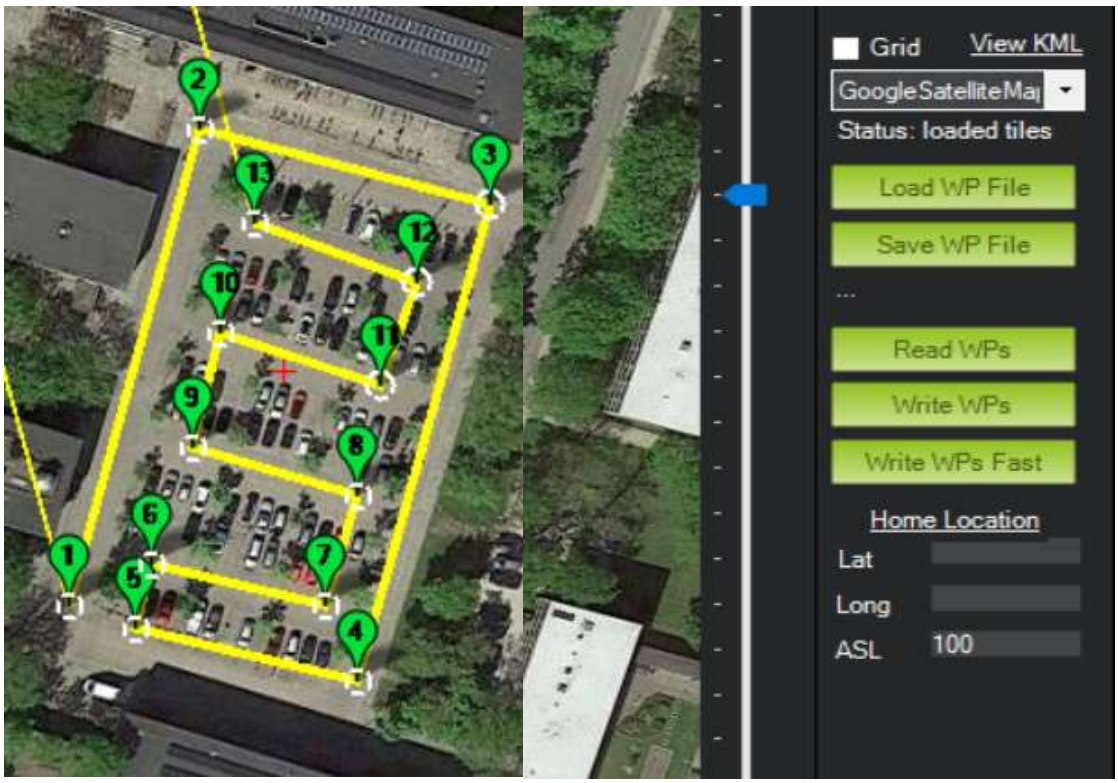
Use an arbitrary switch to control changing between modes. This can be done by setting a specific flight mode channel as a control on the Taranis and changing certain Mission Planner parameters (MODE_CH, which by default should be set to CH5) to the corresponding value.

The steering needs to be set to skid steering, which can be done in Mission Planner parameters. Here, SERVO1_FUNCTION should be set to 73 and SERVO3_FUNCTION to 74. This should be done by default.

After changing parameters, remember to use the “Write Params” button to upload to PLUTO.

User manual

To use PLUTO you will need to connect it to Mission Planner and arm it. In the program, you may also want to upload a series of waypoints that it has to follow along during the auto mode. This is done by choosing flight plan on the left corner in Mission Planner. Now you should be able to plan a route and upload it. The pictures show an example of a route and the buttons to read and write the route to the Pixhawk.



For the manual setup you should be able to steer using the two primary sticks on the Taranis if it has been configured to these inputs, and with PPM signals sent to the trainer port if you configure it to use these.