

Abstract

Unmanned Aerial Vehicles (UAVs) have a tremendous appeal. One can imagine a large number of applications such as search-and-rescue, traffic monitoring, aerial mapping, etc. Helicopters are particularly attractive due to their Vertical Take Off and Landing (VTOL) capabilities. The research on UAVs has shown rapid development in recent years, and offers a great number of challenges. This thesis is the result of a project which is a part of the Autonomous Remote Controlled Helicopter (ARCH) project at the Department of Computer and Information Science, Norwegian University of Science and Technology. The ARCH project has already gained public interest, when it was featured on a television program (Schrödingers katt, NRK. September 2004).

The object of this thesis is divided into three main sections. Firstly, it is to create and describe a remote control system for controlling the UAV in semi-autonomous mode, that will also enable the UAV to autonomously follow objects (pursuit-mode). Secondly, it is to create and describe a virtual cockpit which is to be used with the remote control system. Finally, it is to create and describe an image stabilization system, which can stabilize the visual information sent from the UAV to the ground and the virtual cockpit.

These three components have been combined and integrated into the client prototype called ARCH Groundstation. Together, these three components provides a platform for an operator to control the ARCH UAV in semi-autonomous mode.

Acknowledgements

I would like to thank my supervisors, Prof. Bjørn Olstad and Post.doc. Will Archer Arentz at IDI, NTNU, for tutoring me through this project. In addition, I would like to thank Jarle Anfinssen, Jo Vetle Aure Hansen, Fredrik Orderud and, from Peregrine Dynamics AS, Vegard Evjen Hovstein for their contribution.

Contents

Contents

List of Figures	iv
List of Tables	vi
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Objective	2
1.4 Structure of the document	3
2 Unmanned aerial vehicles	5
2.1 What is a UAV	5
2.1.1 Remote-controlled vehicle	6
2.1.2 Fly-by-wire	6
2.1.3 Autonomous vehicles	7
2.2 The usage of UAVs	8
2.2.1 Search and Rescue	8
2.2.2 Surveillance	9
2.2.3 Law enforcement	10
2.2.4 Inspection	10
2.2.5 Aerial photography and mapping	11
2.3 Display Systems	11

2.3.1	Head Up Displays	12
2.3.2	Helmet Mounted Displays	14
2.3.3	HMDs and the virtual cockpit	14
3	The ARCH UAV	17
3.1	ARCH- Autonomous Remote Controlled Helicopter	17
3.1.1	Hardware platform	17
3.1.2	Software platform	19
3.1.3	Assembly	20
3.2	OCU - Operator Control Unit	21
3.2.1	Hardware platform	22
3.2.2	Software platform	23
4	Theory	25
4.1	Digital Image Stabilizing	25
4.1.1	Motion estimation based on correlation methods	27
4.1.2	Motion estimation based on FFT methods	28
4.1.3	Motion estimation based on feature-based methods	30
4.1.4	Motion estimation based on graph theoretic methods	32
4.1.5	Motion correction systems	33
4.2	Drawing graphics	36
4.2.1	Translation	37
4.2.2	Scaling	37
4.2.3	Rotation	38
4.2.4	Shearing	39
5	Experiments, design and results	41
5.1	Remote Control System	41
5.2	Virtual Cockpit	46
5.3	Image stabilization	50
5.3.1	Motion estimation based on a correlation method	51
5.3.2	Motion estimation based on a FFT method	53
5.3.3	Bit-plane matching or phase correlation	54
5.3.4	Motion correction	60
5.4	Integrating it all into the Groundstation	61

6 Discussion	63
6.1 Remote control system	63
6.2 Virtual cockpit	64
6.3 Image stabilization	65
6.4 Test results	66
7 Conclusion and future work	69
A Glossary	i
B ARCH API and communication protocol specifications	v
B.1 ARCH API interface functions	v
B.2 ARCH protocol messages	ix
B.2.1 Description	ix
B.2.2 Protocol messages	xi
B.3 Unit parameters	xii
C ARCH GroundStation	xv
C.1 Functional description	xv
C.1.1 Connection tab	xvi
C.1.2 Waypoints tab	xvi
C.1.3 IMUData tab	xvii
C.1.4 PathFinder tab	xvii
C.2 Architectural description	xix
Bibliography	xxiii
Index	xxvii

List of Figures

2.1	Degree of autonomous behavior	5
2.2	Remote-controlled car	6
2.3	Planes with <i>fly-by-wire</i> technology	7
2.4	Autonomous vehicles	7
2.5	Search and Rescue	9
2.6	Patrolling	9
2.7	Chase	10
2.8	Inspection	10
2.9	Mapping	11
2.10	Head Up Displays	13
2.11	HUD projecting a FLIR video image	14
2.12	Helmet Mounted Display	15
3.1	The main components of the helicopter	18
3.2	The peripheral components of the helicopter	19
3.3	Inner container (illustrated)	20
3.4	Inner container	21
3.5	Helicopter (illustrated)	21
3.6	Helicopter	22
3.7	Components on the Operator Control Unit	22
4.1	Example of spatial-coordinate mapping	26
4.2	Block Matching Algorithm	29

4.3	An abstract view of generational replacement in a genetic algorithm.	33
4.4	Reading proper block of image, with moving object, into frame memory	34
4.5	Correlation measures	35
4.6	Affine Transform	36
4.7	Translate	37
4.8	Scale	38
4.9	Rotate	38
4.10	Shear	39
5.1	Testing in simulator	45
5.2	Pursuit-mode	46
5.3	Example of a Head Up Display	47
5.4	Drawing the artificial horizon	49
5.5	HUD Architecture	50
5.6	HUD (navigation-mode)	50
5.7	Example of generating bit-planes from gray-scale image	52
5.8	Inputs and output of phase correlation	55
5.9	test-images	56
5.10	1-D Gaussian distribution with mean 0 and standard deviation 1	56
5.11	Motion estimation on noisy images	58
5.12	Test results from motion estimation	59
5.13	Image pair used in performance test	59
5.14	Basic structure of the motion correction system	61
5.15	Illustration of image correction	61
5.16	The Groundstation GUI	62
C.1	Connection tab	xvi
C.2	Waypoints tab	xvii
C.3	PathFinder tab	xviii
C.4	Input-boxes for 'calculate path' and 'calculate total coverage'	xviii
C.5	The GroundStations' architecture (high level).	xix
C.6	The GroundStations' architecture (low level)	xx

List of Tables

3.1	Specifications for the Raptor 90	18
3.2	Software versions	23
4.1	Encoding of transformation into chromosome string	34
5.1	New messages from client to server	42
5.2	Information sent from helicopter to Groundstation	47
5.3	Technical specifications on the computer	59
5.4	Performance of motion estimators	60
B.1	Messages from client to server	xii
B.2	Messages from server to client	xii
B.3	Special messages	xiii
B.4	Parameters for the INS/GPS unit	xiii

Chapter 1

Introduction

ARCH (Autonomous Remote Controlled Helicopter) is a research project at the Norwegian University of Science and Technology (NTNU), Department of Computer and Information Science (IDI). The project aims to create an unmanned helicopter, which will eventually be able to fly autonomously based on Global Positioning System (GPS) coordinates, map data and sensor readings. The ARCH project has already gained public interest, when it was featured on a television program (Schrödingers katt, NRK, September 2004). This thesis is a subproject within ARCH.

The remainder of this chapter contains some general background information about earlier experiences with unmanned aerial vehicles, and a clarification of what this project work seeks to accomplish in concrete terms. Henceforth, the abbreviation UAV will be used for unmanned aerial vehicle.

1.1 Background

The emerging area of UAV research has shown rapid development in recent years, and offers a great number of challenges. Much previous work has focused on low-level control capability, with the goal of developing a controller, which enables autonomous flight from one waypoint to another. Historically, the greatest use of UAVs has been in the areas of intelligence, surveillance and reconnaissance (military). UAVs have increasingly become more important in recent conflict areas, and are predicted to become even more important in future conflicts [Doherty 2004]. While UAVs play an increasing role in these mission areas, civil access to these various UAV assets is now emerging and intelligent UAVs will in the future play an equally important role in civil applications. For both military and civil applications, there is a desire to develop more sophisticated UAVs with more intelligent capabilities. UAVs have tremendous appeal. One can imagine a large number of applications such as search-and-rescue, traffic monitoring, aerial

mapping, etc. Helicopters are particularly attractive due to their Vertical Take Off and Landing (VTOL) capabilities. Earlier projects completed within the ARCH include the creation of an operator control unit on the ground called the Groundstation as well as an API and communication protocol to support communication between the helicopter and Groundstation. An offline-path planner has also been created, which also enabled several helicopters to work together to achieve a desired degree of sensor-coverage (i.e. camera coverage).

1.2 Motivation

A reliable unmanned aircraft will open the door to numerous possible applications that would be impractical or impossible with a manned aircraft. There is, for instance, a logical limitation to how small a manned aircraft can be. This limits both flexibility and price. A miniature UAV would be able to operate in tight, inaccessible and highly dangerous environments. There are several areas where UAVs would be a natural choice.

1.3 Objective

The main goal of this project is to make it possible to pilot the ARCH UAV in semi-autonomous mode. This means that an operator should be able to gain control of the, normally fully autonomous, helicopter from a remote location e.g. the ground. The main goal is composed of four more specific subgoals:

- Create a remote control system (RCS) which enables an operator to control the UAV from a remote location. The operator controls the UAV through the UAV's onboard control system which stabilizes the UAV. This computer-assisted control will make it easier for an operator to control the UAV compared to having direct control of it.
- Move the visual information from the UAV to the remote location to enable an operator to pilot the UAV as if she or he were sitting inside the UAV. Also, to reduce the pilot's workload, place the primary flight information as a graphical overlay on top of the visual information to create what is known as a Head Up Display (HUD).
- Create a Digital Image Stabilizing (DIS) system. To stabilize the visual information sent from the UAV to the remote location.
- Enable the UAV to autonomously follow objects through the remote control system. There is an ongoing project within the ARCH project to develop a tracking system. This system will be able to identify and track objects on a video image. The RCS is to be able to use this system to make the UAV able to pursue objects autonomously.

The aim is to implement a prototype for controlling the UAV in semi-autonomous mode, with all of the mentioned subgoals integrated. The implemented prototype is intended to be refined into a full-featured software system for controlling the autonomous helicopter in future work.

1.4 Structure of the document

The next chapter provides an introduction to unmanned aerial vehicles. It describes what they are and what they can be used for. It also describes some of the display systems found in today's conventional aircrafts. This provides some background information for the display system developed in this thesis. In chapter 3 the platform developed for the ARCH project is presented. Chapter 4 gives a brief summary of the theory necessary in order to understand the background and motivation for some of the choices made in this thesis. In chapter 5 the results of this thesis work is presented along with a short description of how they were tested. A discussion regarding various relevant aspects of the thesis will be given in chapter 6 and a summary presented in the concluding chapter.

Chapter 2

Unmanned aerial vehicles

Unmanned aerial vehicles, commonly referred to as UAVs, are defined as powered aerial vehicles sustained in flight by aerodynamic lift over their flight path and guided without an onboard crew. They may be expendable or recoverable, and can fly autonomously or be piloted remotely. A number of UAVs presently exists. Their capabilities in terms of payload (weight carrying capability), accommodations (volume), mission profile (altitude, range, duration) and onboard systems (control and data acquisition) vary significantly. Historically, the greatest use of UAVs have been in the areas of intelligence gathering, surveillance and reconnaissance (military). For more information concerning UAVs see [UAV Web]. While UAVs play an increasing role in these mission areas, civil access to these various UAV assets is now emerging. The following two subsections will examine what a UAV is, and how it can be put to good use. The last subsection will describe the display systems currently used in piloted aircrafts. In order to effectively pilot an unmanned aircraft these display systems must be moved from the aircraft to the remote location i.e. groundstation.

2.1 What is a UAV

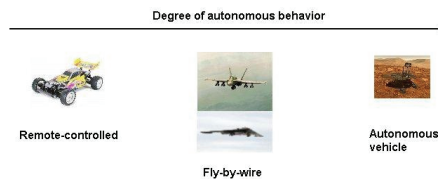


Figure 2.1: Degree of autonomous behavior

As mentioned UAVs can either fly autonomously or be piloted remotely. The word autonomous is often used for a wide range of vehicles with different levels of autonomy, meaning the vehicle's ability to solve tasks without intervention from an operator. A typical autonomous vehicle is able to assist the operator by performing a number of sub-tasks on its own, which makes the operator able to focus on the main task or mission. The focus is on a vehicle's ability to perform autonomous navigation and, when considering aerial vehicles, the ability to maintain a fixed position or heading by itself. Unmanned vehicles can therefore be categorized by their degree of autonomy, as illustrated in figure 2.1.

2.1.1 Remote-controlled vehicle



Figure 2.2: Remote-controlled car

(copied from [RC Web])

Remote-controlled vehicles are vehicles without any form of built-in navigation or control. Examples of such are remote-controlled cars (figure 2.2), planes and helicopters. The vehicle is completely dependent on the operator's constant input. A remote-controlled helicopter would crash to the ground if not for continuous monitoring and handling by the operator. Some types of remote-controlled vehicles, for example helicopters, demands a great deal of skill from the operator.

2.1.2 Fly-by-wire

On some vehicles, the operator's inputs are converted into the correct settings for the engine and controls. The principle is known as *fly-by-wire*. More on *fly-by-wire* can be found at [NASA Web]. The vehicle has a higher degree of autonomy than a remote-controlled vehicle, but is still dependent on the operator's or pilot's input. The technology is known from fighter-planes, as the ones shown in figure 2.3, where *fly-by-wire* makes the plane stable. A modern fighter-plane is deliberately constructed unstable to make it as maneuverable as possible. This means that it's impossible for a human pilot to keep the plane stable in the air without assistance from computers. Another example is the B-2 *stealth* bomber-plane which is constructed without a tail-rudder to lower the radar-reflection. The bomber turns with only a combination of bank- and



(a) F-18 Hornet (b) B-2 Stealth Bomber

Figure 2.3: Planes with *fly-by-wire* technology

(copied from [FAS2 Web, FAS3 Web])

height-rudders. These properties are hidden from the pilot by the plane's *fly-by-wire* system, which makes the plane handle like a conventional airplane with a tail-rudder.

A more simple version of *fly-by-wire* can be found in modern cars today. We know them as Anti-Spin-Control (ASC), Anti-lock Braking System (ABS) and Electronic Stabilisation Programme (ESP). When the driver presses the brake or accelerator pedal, the inputs are sent to the car's computer, where the optimal brake or engine power is calculated.

There is no clear distinction between *fly-by-wire* systems and autonomous systems. Fighter planes are, for instance, capable of leveling themselves if the pilot should pass out because of G-forces or for other reasons be incapable of controlling the plane.

2.1.3 Autonomous vehicles



(a) Mars Rover (b) Tomahawk missile

Figure 2.4: Autonomous vehicles

(copied from [Mars Web, FAS Web])

Autonomous vehicles are generally defined as vehicles that are capable of intelligent motion and action without requiring either a guide to follow or the

constant input of an operator.

The well known Mars-Rover, from NASA's Mars exploration project shown in figure 2.4 (a), had a built-in intelligent navigation and control system. The distance (487 million kilometers) and the time delay (27 minutes) between the vehicle on Mars and the operator on Earth made the rover impossible to operate by hand. Instead, NASA¹ had to make it operate on its own. The rover was able to autonomously navigate to a desired position and perform the necessary tasks. More on the Mars-rover can be found at [Mars Web].

Another example is the tomahawk missile, figure 2.4 (b), which is a so-called fire- and-forget weapon system (more on the tomahawk missile can be found at [FAS Web]). This means that the missile will find its way to the target by itself. The missiles are able to cover great distances over mountains and cities before hitting a specific target with high precision. Modern missiles are capable of circling the target and compare what it sees with stored photos to positively identify the target before hitting it.

A less advanced form of an autonomous system is the autopilot system used in commercial and large private aircrafts, where the pilot's only job between take-off and landing is to monitor the system. Common to all these autonomous systems are the high demands that are put on the onboard hardware and software. In return, the demands to the operator lessens, often to the extent that the entire vehicle can be operated through simple commands and instructions.

2.2 The usage of UAVs

There is a lot of ongoing research on autonomous or unmanned aircrafts. The reason for this is that a reliable unmanned aircraft will open the door to numerous possible applications that would be impractical or impossible with a manned aircraft. There is, for instance, a logical limitation to how small a manned aircraft can be. This limits both flexibility and price. A miniature UAV would be able to operate in tight, inaccessible and highly dangerous environments. There are several areas where UAVs would be a natural choice. The following examples are based on UAVs in the form of remote-controlled helicopters, because this project work focuses on such a vehicle.

2.2.1 Search and Rescue

Vision-guided UAVs can quickly and systematically search a very large area to locate victims of an accident or a natural disaster, as illustrated in figure 2.5. They can then visually lock on to objects at the site or at stranded victims, to guide rescue forces to the scene. This would help focus the efforts of search and

¹National Aeronautics and Space Administration



Figure 2.5: Search and Rescue

(copied from [AHP Web])

rescue crews to the rescue operation instead of on the time-consuming search operation.

UAV's can more easily be deployed in such weather conditions, which would normally prevent human piloted search and rescue operations. They may be sacrificed, and thus could be used in very dangerous conditions to attempt saving human lives. For example, they could fly close to a forest fire to look for stranded individuals, search in contaminated areas, and identify potential radioactive leaks after a nuclear reactor accident.

2.2.2 Surveillance

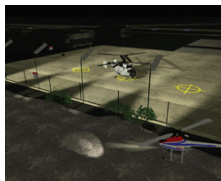


Figure 2.6: Patrolling

(copied from [AHP Web])

UAVs can patrol an area and report interesting or unusual activity. They can perform a variety of surveillance operations ranging from around-the-clock border patrol to looking for potential danger on the battlefield. They could keep watch on an area non-stop by automatically landing and refueling from ground stations in or near the area, as illustrated in figure 2.6. They are also capable of automatically locating and identifying suspicious activity, and visually lock-on to objects or persons involved until ground forces arrive.

2.2.3 Law enforcement



Figure 2.7: Chase

(copied from [AHP Web])

Besides being unmanned, small UAVs would be relative inexpensive. This makes them suitable for being placed in harm's way. UAVs could fly overhead to aid the police in dangerous high-speed chases or criminal search operations, as illustrated in figure 2.7. Stationed on top of buildings in urban areas, they can be dispatched in in seconds and relay images from trouble spots. This real-time imagery could assist the tactical assessment of the situation by human experts who dispatch police units to the scene.

2.2.4 Inspection

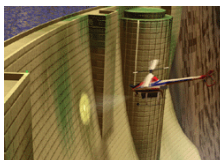


Figure 2.8: Inspection

(copied from [AHP Web])

It is cost effective to have one or more small UAVs in the air compared to conventional aircrafts. This makes them a good alternative when installations need to be inspected. UAVs could inspect high voltage electrical lines in remote locations. They could also inspect large structures such as bridges, dams and oil-riggs effectively and at a small cost, as illustrated in figure 2.8. The fact that they are unmanned, and therefore expendable would make them useful for inspecting buildings and roads for potential damage after an earthquake. Or they can locate hazardous materials in waste sites by providing aerial imagery to

human experts or by automatically identifying waste containers and materials using on-board vision systems.

2.2.5 Aerial photography and mapping

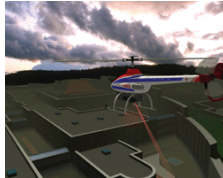


Figure 2.9: Mapping

(copied from [AHP Web])

UAVs are able to fly and maneuver a lot closer to the ground than what could be considered safe for a conventional aircraft. This makes the UAVs able to build more accurate topological maps than conventional aircraft, thus achieving substantial cost savings. Unlike airplanes, they can fly close to the ground while carrying cameras or range sensors to build high resolution 3D maps, as illustrated in figure 2.9. They can also fly in smaller and more constrained areas to build highly detailed elevation maps.

2.3 Display Systems

The cockpit display systems provide a visual presentation of the information and data from the aircraft sensors and systems to the pilot in order to enable the pilot to fly the aircraft safely and carry out the mission. They are thus vital to the operation of any aircraft as they provide the pilot with:

- Primary flight information
- Navigation information
- Engine data
- Airframe data
- Warning information.

Often there is a wide array of mission specific information to view. The pilot is able to rapidly absorb and process substantial amounts of visual information,

but it is clear that the information must be displayed in such a way that it can easily be assimilated, and unnecessary information must be filtered out to ease the pilot's task in high work load situations. A number of technologies have been developed to improve the pilot-display interaction, and this is a continuing activity as new technology and components become available. Examples of these developments are:

- Head Up Displays (HUDs)
- Helmet mounted displays (HMDs)
- Multi-function colour displays
- Digitally generated colour map displays

This subject can readily fill several books. Attention has, therefore, been concentrated on providing an overview and explanation of the basic principles involved in the topics most relevant to this project. For a more in-depth look at flight display systems see [Collinson 2002], which is the basis for this overview.

2.3.1 Head Up Displays

HUDs and the progressive development of these are without a doubt the most important advance in the visual presentation of data to the pilot in the past forty years or so. The first production HUDs went into service in 1962 in the BUCCANEER strike aircraft in the UK. The HUD has enabled a major improvement in man-machine interaction (MMI) to be achieved, as the pilot is able to view and assimilate the essential flight data generated by the sensors and systems in the aircraft whilst keeping hers or his head up and maintaining full visual concentration on the outside world.

A HUD basically projects a collimated display on a transparent screen which is positioned in the pilots forward line of sight, so that she or he can view both the displayed information and the outside world scene at the same time. Because the display is collimated, that is focused at infinity (or a long distance ahead), it overlays the outside world scene. The pilot is thus able to observe both distant outside world objects and displayed data at the same time without having to change the direction of gaze or re-focus the eyes.

The advantages of head up presentation of essential flight data such as the artificial horizon, pitch angle, bank angle, flight path vector, height, airspeed and heading can be seen in figure 2.10. The figure shows a typical HUD as viewed by the pilot. The pilot is thus free to concentrate on the outside world during fight manoeuvres and does not need to look down at the cockpit instruments or head down displays. For instance in combat situations, it is essential for survival that the pilot is head up and scanning for possible threats from all directions. The



Figure 2.10: Head Up Displays

(copied from [F-16 Web])

ability to remain head up in combat has made the HUD an essential system on all modern combat aircraft. The HUDs have also been widely retro-fitted to earlier generation fighters and strike aircraft. It should be noted that there is a transition time of one second or more to re-focus the eyes from viewing distant objects to viewing near objects a metre or less away, such as the cockpit instruments and displays and adapt to the cockpit light environment.

Using a Forward Looking Infra-Red (FLIR) sensor, an electro-optical image of the scene in front of the aircraft can be overlaid on the real world scene with a raster mode HUD. The TV raster image generated from the FLIR sensor video is projected on to the HUD and scaled one to one with the outside world enabling the pilot to fly at low altitude by night in fair weather. This provides a realistic night operation capability to the fighters. HUDs are also being installed in civil aircrafts for reasons such as:

1. Increased safety while landing the aircraft in conditions of severe wind shear using the HUD to provide a flight path director display which allows for the effects of wind shear. The flight path is computed from the flight path vector derived from the INS², airspeed and height from the air data system and the aircraft's aerodynamic characteristics.
2. To display automatic landing guidance to enable the pilot to land the aircraft safely in conditions of very low visibility due to fog, as a back up and monitor for the automatic landing system.
3. Enhanced vision using a raster mode HUD to project a FLIR video image of the outside world from a FLIR sensor installed in the aircraft. As shown in figure 2.11.

²Inertial Navigation System, this system is described in section 3.1.2.



Figure 2.11: HUD projecting a FLIR video image

(copied from [F-16 Web])

2.3.2 Helmet Mounted Displays

The advantages of HUDs have been described in the previous section. The HUD, however, only presents the information in the pilot's forward field of view, and this is somewhat limiting. Significant increases in this field of view are not practicable, because of the cockpit geometry constraints. Especially with helicopters the pilot requires visual information head up, when she or he is looking in any direction. This requirement can only be met by a helmet mounted display (HMD). A HMD can provide, in effect, a "HUD on the helmet", as shown in figure 2.12. This can display all the information to the pilot which is normally shown on a HUD, but with the pilot able to look in any direction. The HMD also enables a very effective night/poor visibility viewing system to be achieved by displaying the TV image from a gimballed³ infrared sensor unit which is slaved to follow the pilot's line of sight. The pilot's line of sight with respect to the airframe is measured by a head positioning system. Such a helmet can also incorporate night viewing goggles which are integrated into the HMD optical system.

2.3.3 HMDs and the virtual cockpit

The concept of a "virtual cockpit", where information is presented visually to the pilot by means of computer generated 3D imagery, is being very actively

³Gimbals: An appliance for permitting a body to incline freely in all directions, or for suspending anything, as a barometer, ship's compass, chronometer, etc., so that it will remain plumb, or level, when its support is tipped, as by the rolling of a ship. [Gimbals Web].



Figure 2.12: Helmet Mounted Display

(copied from [Airforce Web])

researched in a number of establishments both in the USA and the UK. The increasing use of remote piloted vehicles (RPVs) and their control from a “parent” aircraft or ground station is another future application for HMDs and virtual cockpit technology. It should be noted that RPVs can include land vehicles or underwater vehicles as well as airborne vehicles.

A correctly designed binocular HMD (BHMD) is a key component in such systems, because it is able to present both a display of information at infinity and also stereo images so that the pilot sees a 3D image. The ability to generate 3D displays opens up entirely new ways of presenting information to the pilot (and crew), the objectives being to present the information so that it can be visually assimilated more easily and in context with the mission. When head up, the pilot views the outside world directly, or indirectly by means of a TV display on the HMD from a sensor unit. When looking down at the instrument panel, the virtual cockpit computer system recognises the pilot’s head down sight line and supplies this information to the display generation system. The display generation system then generates a stereo pair of images of the appropriate instrument display on the panel which corresponds to the pilot’s line of sight. Thus, looking down into the cockpit at the position normally occupied by a particular instrument display will result in the pilot seeing a 3D image of that instrument display appearing in the position it normally occupies - i.e. a virtual instrument panel display.

Novel ways of presenting information to the pilot by means of the BHMD include displaying a 3D “pathway in the sky” as a flight direction display, which can be overlaid on the normal outside scene or on a computer generated outside world scene created from a terrain data base.

Chapter 3

The ARCH UAV

ARCH (Autonomous Remote Controlled Helicopter) is a research project at NTNU. The project aims to create an unmanned helicopter which will eventually be able to fly autonomously based on GPS coordinates, map data and sensor readings. This chapter contains information about the different components that are currently developed in the ARCH project. Both the aerial vehicle itself and the operator control unit on the ground.

3.1 ARCH- Autonomous Remote Controlled Helicopter

The first two sections contains information about the components that the ARCH helicopter is presently comprised of. The final section describes how the different hardware components were mounted on the helicopter.

3.1.1 Hardware platform

The ARCH helicopter is originally Thunder Tiger's¹ Raptor 90 remote controlled helicopter, which is commercially available, the specifications for the Raptor 90 can be found in table 3.1. It is augmented with a mini-ITX board running a Linux version called Gentoo. This board constitutes the heart of the control system. Mini-ITX is an ultra-compact (17×17 cm.) mainboard form-factor developed by Via Technologies Inc. There is also an on-board Inertial Measurement Unit (IMU) from Rotomotion (Rev 2.4 6DOF IMU Kit). This is a 6 degrees of freedom (6DOF) inertial measurement unit. The unit measures rotation and acceleration about/along the x, y and z axis. Both the mini-ITX

¹Thunder Tigers web-page is available at: <http://www.tiger.com.tw/>

Item	Value
Full length of fuselage	55.5"
Full width of fuselage	7.48"
Total height	18.75"
Main rotor diameter	62.2"
Tail rotor diameter	10.24"
Gear ratio	8.45:1:4.6 (90)
All up weight	approx. 4.76 kg+

Table 3.1: Specifications for the Raptor 90

board, the IMU and the Raptor 90 helicopter are shown in figure 3.1. The most important instruments of the IMU are:

- Three *gyroscopes*, which measure rotational values without reference to external coordinates.
- Three *accelerometers*, which measure acceleration in three directions of a body-centered coordinate system.
- A *magnetometer*, which measures the heading toward the magnetic north pole.

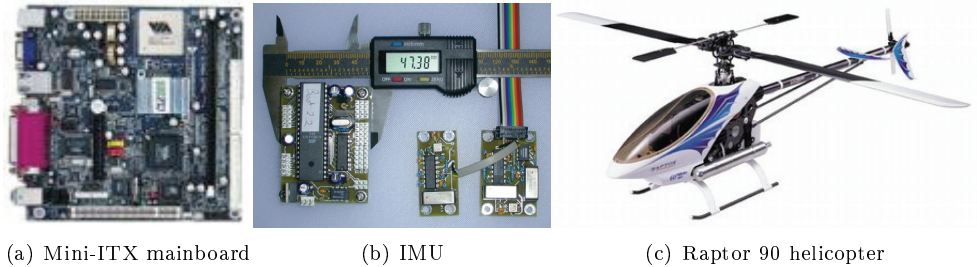


Figure 3.1: The main components of the helicopter

The magnetometer, which is a digital compass, is together with a Global Positioning System (GPS) unit used to calculate the position of the UAV. The GPS is a worldwide radio-navigation system formed from a constellation of 24 satellites and their ground stations. GPS uses these "man-made stars" as reference points to calculate positions accurate to a matter of meters. In fact, with advanced forms of GPS it is possible to make measurements to better than a centimeter. The GPS receiver used on the helicopter is from U-blox, shown in

figure 3.2 (a), its size is 71 x 41 x 11 mm and with a precision of about 2,5 meters.

There is also a small camera mounted on the helicopter to provide live, high-quality video from the helicopter to the ground-station. The camera is a Pro X2 from Hicam, as shown in figure 3.2 (b), which is a Charge Coupled Device (CCD) camera which requires a 4.8 volt power source. This is conveniently the same power-level which the helicopter operates on. It also has its own transmitter and receiver which enables it to transfer the video to the Groundstation without affecting the rest of the onboard system. At the Groundstation the signal is received as a normal video-in signal. There is of course also an onboard power-supply to power all of these components. This power-supply was originally a battery for a laptop computer. The platform is still evolving, and additional sensors will be added in the future.

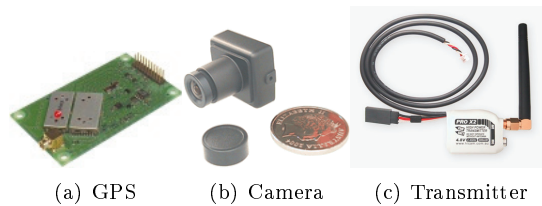


Figure 3.2: The peripheral components of the helicopter

3.1.2 Software platform

As mentioned in the previous section, the mini-ITX board runs a Linux operating system. The heart of the control system is the Inertial Navigation System (INS). The INS calculates orientation and position from gyro and accelerometer readings. The real world data from these sensors are fed into a *Kalman filter*. A Kalman filter is a filter which is used to estimate the state of a system from measurements which contain random errors. Estimating variables such as position and velocity for aerial vehicles typically requires a Kalman filter. A more in-depth description of the Kalman filter can be found in [Kalman 60]. Many INS implementations exist, most of these are only available commercially. An open source project called Autopilot is used as a basis for the ARCH project. Autopilot is a complete control system for unmanned helicopters. It provides a three-axis Electronic Flight Instrument System (EFIS). The system provides the instrumentation for attitude, engine and position. The entire design and all software is available as Free Software, licensed under GPL². The goal of the autopilot project is to provide a do-it-yourself autopilot kit. For more information about the autopilot project see [Autopilot Web].

²General Public License, see <http://www.gnu.org/copyleft/gpl.html>

Since Autopilot is an open source project there is a risk that the codebase will not remain backwards compatible. Therefore it was deemed necessary to implement an own Application Program Interface (API) for the ARCH project. All communication through and from the autopilot control system goes through this API. Software units using this API to communicate with the UAV control system are called clients, clients can be either local (i.e. running on-board the UAV) or remote. Details on this API can be found in appendix B.

3.1.3 Assembly

The components described in section 3.1.1 are all bought “off the shelf”. However, a great deal of time and engineering has been invested in mounting the different components on to the helicopter. The mini-ITX board, IMU, GPS and power-supply are all secured to a container called the inner container as illustrated in figure 3.3 and 3.4. The inner container is basically a plastic case with a lid, which enables quick and easy access to the components.

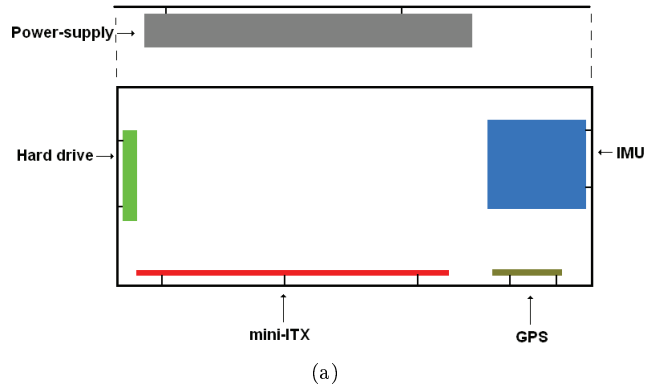


Figure 3.3: Inner container (illustrated)

When the helicopter is in flight, the inner container is secured inside an outer container. The outer container is similar to the inner container, only bigger, and it is fixed to the helicopter itself. The inner container is placed inside the outer container but it is not in direct contact with the outer container. There is a layer of bubble-wrap in between the inner and outer container to minimize the vibrations on the inner container. Besides containing the inner container, the outer container acts as a undercarriage to the helicopter. Therefore, a training gear has been fixed to the container. A training gear is basically four poles mounted to the helicopter in a way which makes it virtually impossible to make the rotor-blades hit the ground. The poles on this helicopter are mounted in an angle to absorb most of the impact in any “not so soft” landings. Figure 3.5 illustrates where the different components are mounted on the helicopter. And

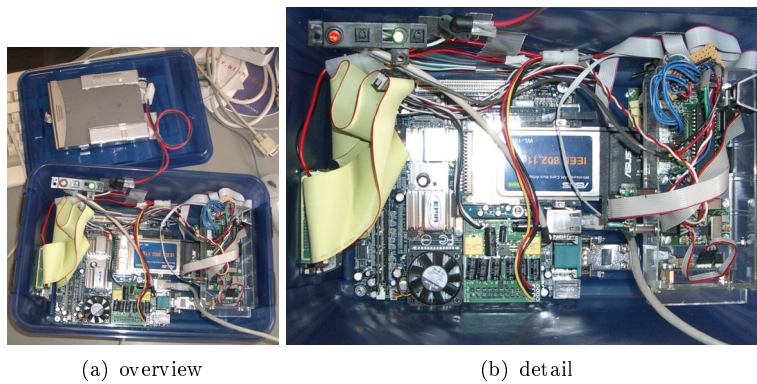


Figure 3.4: Inner container

Figure 3.6 shows the helicopter and undercarriage, the white casing behind the helicopter was placed there to pick up spills from the exhaust and is not part of the helicopter.

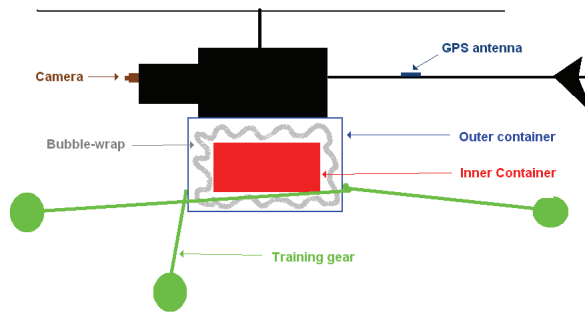


Figure 3.5: Helicopter (illustrated)

3.2 OCU - Operator Control Unit

The Operator Control Unit (OCU) is the collective term of all the components, software and hardware, that combined enables an operator to control the UAV. This section contains information about the components that constitute the OCU. Both the actual hardware parts and the software applications are described.

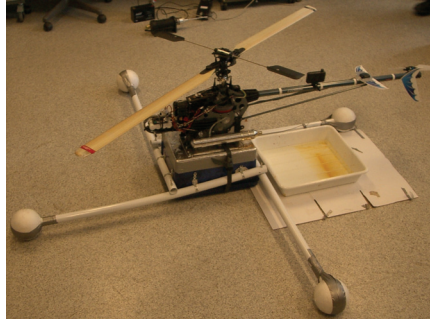


Figure 3.6: Helicopter

3.2.1 Hardware platform

The OCU is a laptop, which is used to, among other things, send high-level commands such as waypoints to the helicopter. In order to pilot the helicopter from the OCU there must also be a joystick connected to it. Any joystick which is configured and calibrated on the OCU should do, although it would be most convenient with a joystick with four axes. On the ARCH project the Logitechs Freedom 2.4 joystick, figure 3.7 (a), is used. In order for an operator to see where she or he is flying the OCU must also be able to receive the video signal sent from the helicopter. This is done by connecting the cameras receiver described in section 3.1 to an analog to digital video decoder. On the ARCH project a decoder from Hauppauge is used, figure 3.7 (b). This was chosen because it connects to the OCU through USB and it is supported on most operating systems.



(a) Logitech Freedom 2.4 (b) Hauppauge WinTV PVR USB2

Figure 3.7: Components on the Operator Control Unit

3.2.2 Software platform

In order to test new applications the ARCH GroundStation prototype was developed. The purpose of the GroundStation is to enable people to monitor and control the helicopter's actions. It is meant to provide an interface to UAVs, where a user can plot flight paths, designate areas for UAVs to cover and have direct control of an UAV. The GroundStation is connected to the helicopter through a wireless LAN. Although the helicopter could function well on its own, it is assumed that the helicopter will always be connected to the GroundStation. This means that the reach of the wireless LAN, or the lack thereof, implies a serious restriction on the reach of the helicopter. This is something that will have to be changed or fixed before the ARCH helicopter can perform tasks of significance beyond research and development. For more information concerning the Groundstation see appendix C. To make the GroundStation able to run on any platform, the entire application is written in the programming language Java. Because there is no native support for polled input, such as joysticks, in Java an open source project called Jinput has been used in the ARCH project. The JInput project hosts an implementation of an API for game controller discovery and polled input. The API itself is pure Java and presents a platform-neutral complete portable model of controller discovery and polling. It can handle arbitrary controllers and returns both human and machine understandable descriptions of the inputs available. The implementation also includes plug-ins to allow the API to adapt to various platforms. These plug-ins often contain a native code portion to interface to the host system. This allows the GroundStation to stay platform-neutral by applying different plug-ins on different platforms. For more details on the JInput project see [Jinput Web]. In order to view and manipulate the video sent from the helicopter to the Groundstation, the Java 2 Platform, Standard Edition (J2SE) has been extended with an optional package called Java Media Framework API (JMF). JMF enables audio, video and other time-based media to be added to applications and applets built on Java technology. It enables a developer to capture, playback, and stream multiple media formats by providing a powerful toolkit to develop scalable, cross-platform technology. For more information on JMF see [JMF Web]. Table 3.2 shows the different versions of the packages and the programming language used.

Software	Version
Java	J2SE 5.0
JMF	2.1.1e
JInput	1.5

Table 3.2: Software versions

Chapter 4

Theory

This chapter contains the theory on which the proposed solutions, in this thesis, are based upon. The first section describes the theory behind motion estimation and motion correction. It also presents four different approaches to motion estimation. The second section describes the underlying theory used when drawing computer graphics. This theory is utilized when implementing the Head Up Display.

4.1 Digital Image Stabilizing

Image stabilizing is defined as the process of generating a compensated video sequence, where image motion by the camera's undesirable shake or jiggle is removed [Kinugase et al. 1990]. Recent Digital Image Stabilizing (DIS) systems are realized using digital image processing techniques instead of mechanical motion detection techniques using gyros or fluid prism [Uomori et al. 1990]. The image stabilization task may be subdivided into two basic systems, namely: the motion estimation system and the motion correction system.

In general, the motion estimation system generates several local motion vectors from sub-images in different positions of the frame. The generation of these motion vectors is known as image mapping or image registration. Image registration may be defined as a spatial mapping between two images. If we define these images as two 2D arrays of a given size denoted by I_1 and I_2 where $I_1(x, y)$ and $I_2(x, y)$ each map to their respective intensity (or other measurement) value, then the mapping between images can be expressed as:

$$I_2(x, y) = I_1(f(x, y)) \quad (4.1)$$

where f is a 2D spatial-coordinate transformation, i.e., f is a transformation

which maps two spatial coordinates, x and y , to new spatial coordinates x' and y' , (also illustrated in figure 4.1):

$$(x', y') = f(x, y) \quad (4.2)$$

The image registration problem is to find the optimal spatial transformations so that the images are matched for the purpose of determining the parameters of the matching transformation. These parameters will then become the motion vectors, which will be used in the motion correction system.

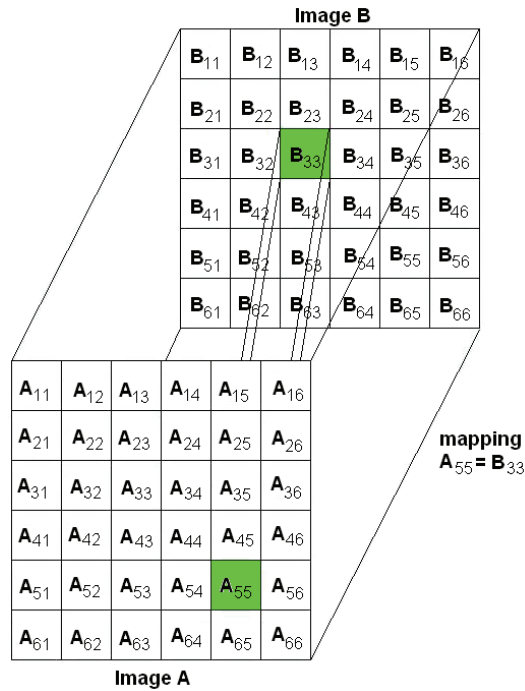


Figure 4.1: Example of spatial-coordinate mapping

Motion estimation methods may be loosely divided into the following four classes:

- Algorithms that use image pixel values directly, e.g., correlation methods [Barnea et al. 1972].
- Algorithms which use the frequency domain, e.g., Fast Fourier Transform(FFT)-based methods [De Castro et al. 1987].
- Algorithms which use low-level features such as edges and corners, e.g., feature-based methods [Brown et al. 1992].

- Algorithms which use high-level features such as identified (parts of) object, or relation between features, e.g., graph-theoretic methods [Rao et al. 2004].

The motion correction system determines the global motion of a frame by appropriately processing these local motion vectors, and decides whether the motion of a frame is caused by undesirable fluctuation of the camera or intentional panning. The stabilized image is generated by reading out the proper block of fluctuated image in the frame memory [Paik et al. 1992].

The following four subsections presents an overview of the theory behind the different classes of motion estimation, by introducing four different methods, one from each class, (this brief overview is based on the excellent presentation in [Brown et al. 1992].) Finally a brief description of the theory behind a motion correction system will be given.

4.1.1 Motion estimation based on correlation methods

Cross-correlation is the basic statistical approach to image registration. It is often used for template matching or pattern recognition where the location and orientation of a template or pattern is found in an image. By itself, cross-correlation is not a image registration method. It is a similarity measure or match metric, i.e., it gives a measure of the degree of similarity between an image and a template. However, there are several image registration methods for which it is the primary tool. These methods are generally useful for images which are misaligned by small rigid or affine transformations. For example translation which is the most relevant in image stabilizing. For a template T and image I , where T is small compared to I , the two-dimensional normalized cross-correlation function measures the similarity for each translation:

$$C(u, v) = \frac{\sum_x \sum_y T(x, y) I(x - u, y - v)}{\sqrt{\sum_x \sum_y I^2(x - u, y - v)}} \quad (4.3)$$

If the template matches the image exactly, at a translation of (i, j) , the cross-correlation will have its peak at $C(i, j)$. (See [Rosenfeld et al. 1982] for a proof of this using the Cauchy-Schwarz inequality.) Thus, by computing C over all possible translations, it is possible to find the degree of similarity for any template-sized window in the image. Notice that the cross-correlation must be normalized since local image intensity would otherwise influence the measure.

The cross-correlation measure is directly related to the more intuitive measure which computes the sum of differences squared between the template and the image at each location of the template:

$$D(u, v) = \sum_x \sum_y (T(x, y) - I(x - u, y - v))^2 \quad (4.4)$$

This measure decreases with the degree of similarity since, when the template is placed over the image at the location (u, v) for which the template is most similar, the differences between the corresponding intensities will be smallest. The template energy, defined as $\sum_x \sum_y T^2(x, y)$, is constant for each position (u, v) that is measured. Therefore, one should normalize as before, using the local image energy $\sum_x \sum_y I^2(x - u, y - v)$.

A related measure, which is advantageous when an absolute measure is needed, is the correlation coefficient:

$$\frac{\text{covariance}(I, T)}{\sigma_I \sigma_T} = \frac{\sum_x \sum_y (T(x, y) - \mu_T)(I(x - u, y - v) - \mu_I)}{\sqrt{\sum_x \sum_y (I(x - u, y - v) - \mu_I)^2 \sum_x \sum_y (T(x, y) - \mu_T)^2}} \quad (4.5)$$

where μ_T and σ_T are mean and standard deviation of the template and μ_I and σ_I are mean and standard deviation of the image. This statistical measure has the property that it measures correlation on an absolute scale ranging from $[-1, 1]$. Under certain statistical assumptions [Brown et al. 1992], the value measured by the correlation coefficient gives a linear indication of the similarity between images. This is useful in order to quantitatively measure confidence or reliability in a match and to reduce the number of measurements needed when a prespecified confidence is sufficient [Svedlow et al. 1987]. In motion estimation the template or pattern is a sub-image of the previous frame in the video.

The most common correlation method is known as the Block Matching Algorithm (BMA). The full search BMA under the Mean Absolute Difference (MAD) and Mean Square Error (MSE) criteria can be considered as an optimal solution for motion estimation [Musmann et al. 1985]. However, the full search BMA requires large amount of computations, which causes time delay [Liu et al. 1993, Gharavi et al. 1990]. In motion estimation it is possible to reduce the computation time by exploiting the fact that, usually, the image will only have moved a limited amount of space from one frame to another. (As shown in figure 4.2.) This will make it possible to achieve satisfactory results using a limited search space. The limited search space will enable the motion estimation to be performed in real-time.

4.1.2 Motion estimation based on FFT methods

The methods that fall into this class register images by exploiting several useful properties of the Fourier Transform. Translation, rotation, reflection, and scaling all have their counterpart in the Fourier domain. Furthermore, the transform can be efficiently implemented in either hardware or using the Fast Fourier Transform. These methods differ from the methods in the previous section, because they search for optimal match according to information in the *frequency domain* as opposed to in the image domain. By using the frequency

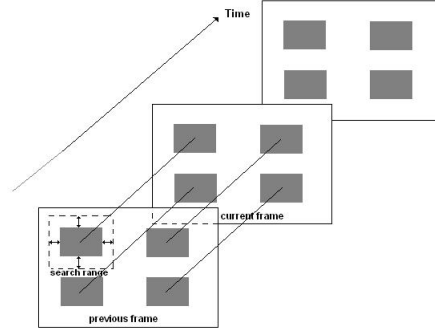


Figure 4.2: Block Matching Algorithm

domain, the Fourier methods achieve excellent robustness against correlated and frequency-dependent noise.

The most useful method for motion estimation and the most basic one that uses Fourier analysis is called Phase Correlation. It can be used to register images which have been shifted relative to each other. In order to describe this method one needs a few of the terms used in Fourier Analysis. The Fourier transform of an image $f(x, y)$ is a complex function; each function value has a real part $R(\omega_x, \omega_y)$ and an imaginary part $I(\omega_x, \omega_y)$ at each frequency (ω_x, ω_y) in the frequency spectrum:

$$F(\omega_x, \omega_y) = R(\omega_x, \omega_y) + iI(\omega_x, \omega_y) \quad (4.6)$$

where $i = \sqrt{-1}$. This can be expressed alternatively using the exponential form as:

$$F(\omega_x, \omega_y) = |F(\omega_x, \omega_y)| \exp^{i\phi(\omega_x, \omega_y)} \quad (4.7)$$

where $|F(\omega_x, \omega_y)|$ is the magnitude or amplitude of the Fourier transform and where $\phi(\omega_x, \omega_y)$ is the phase angle. The square of the magnitude is equal to the amount of energy or power at each frequency of the image and is defined as:

$$|F(\omega_x, \omega_y)|^2 = R^2(\omega_x, \omega_y) + I^2(\omega_x, \omega_y) \quad (4.8)$$

The phase angle describes the amount of phase shift at each frequency and is defined as:

$$\phi(\omega_x, \omega_y) = \tan^{-1} \left[\frac{I(\omega_x, \omega_y)}{R(\omega_x, \omega_y)} \right] \quad (4.9)$$

Phase correlation relies on the translation property of the Fourier transform, sometimes referred to as the Shift Theorem. Given two images f_1 and f_2 which differ only by a displacement (d_x, d_y) , i.e.,

$$f_2(x, y) = f_1(x - d_x, y - d_y) \quad (4.10)$$

their corresponding F_1 and F_2 will be related by:

$$F_2(\omega_x, \omega_y) = \exp^{-i(\omega_x d_x + \omega_y d_y)} F_1(\omega_x, \omega_y) \quad (4.11)$$

In other words, the two images have the same Fourier magnitude, but a phase difference directly related to their displacement. This phase difference is given by $\exp^{i(\phi_1 - \phi_2)}$. If one compute the cross-power spectrum of the two images defined as:

$$\frac{F_1(\omega_x, \omega_y) F_2^*(\omega_x, \omega_y)}{|F_1(\omega_x, \omega_y) F_2^*(\omega_x, \omega_y)|} = \exp^{i(\phi_1 - \phi_2)} \quad (4.12)$$

where F^* is the complex conjugate of F , the Shift Theorem guarantees that the phase of the cross-power spectrum is equivalent to the phase difference between the images. Furthermore, if we represent the phase of the cross-power spectrum in its spatial form, i.e. by taking the inverse Fourier transform of the representation in the frequency domain, we will have an impuls function. Such a function is approximately zero everywhere, except at the displacement position, which is needed to optimally register the two images. The Fourier registration method for images, which have been displaced with respect to each other, therefore involves determining the location of the peak of the inverse Fourier transform of the cross-power spectrum phase. Since the phase difference for every frequency contributes equally, the location of the peak will not change if there is noise that is limited to a narrow bandwidth, i.e., a small range of frequencies. Thus this technique is particularly well suited for images with this type of noise.

There are several extensions to this method available (i.e. [De Castro et al. 1987]) which enables the method to register images which are both translated and rotated with respect to each other. However, a real-time motion estimation system is primarily interested in translative displacement, therefore these extensions will not be a part of this brief overview.

4.1.3 Motion estimation based on feature-based methods

One of the more known image registration algorithms that employs feature-based methods is the Morphological Pyramid Image Registration (MPIR) algorithm [Zhongxiu et al. 2000]. The MPIR algorithm uses the low level shape

features to determine the global affine transformation model along with the radiometric¹ changes between the images. The images are represented by a Morphological Pyramid (MP)², as the MP's have the capability to eliminate details and to maintain shape features. The Levenberg Marquardt non-linear optimization algorithm is employed to estimate the matching parameters. This algorithm is capable of measuring, to sub-pixel accuracy, the displacement between images subjected to simultaneous translation, rotation, and shearing.

Mathematical morphology is a set-theoretic approach to image analysis [Zhongxiu et al. 2000]. The morphological filters, such as Open and Close, can be designed to preserve edges or shapes of objects, while eliminating noise and details in an image. Successive application of morphological filtering and sub-sampling [Morales et al. 1995] can construct the Morphological Pyramid (MP) of an image:

$$I_L = [(I_{L-1} \circ K) \bullet K] \downarrow d \quad L = 0, 1, 2, \dots, n \quad (4.13)$$

Where K is a structuring element, d is a down sampling factor, \circ and \bullet are open and close filters. Thus the image at any level (pyramid level) L can be created. The spatial-mapping function and parameters in MPIR are described by a global affine transformation. The global affine transformation [Alliney et al. 1986, Brown et al. 1992], includes translation (tx, ty) , rotation (Θ) , scaling (sx, sy) , and shearing (shx, shy) and can be described as

$$\begin{aligned} \begin{pmatrix} p \\ q \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ shy & 1 \end{pmatrix} \begin{pmatrix} 1 & shx \\ 0 & 1 \end{pmatrix} \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \\ &\times \begin{pmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{pmatrix} \begin{pmatrix} r \\ c \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix} \end{aligned} \quad (4.14)$$

where (p, q) and (r, c) are points in respectively the first and second gray scale image. With six parameters, the above equation can be simplified to:

$$\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_4 & a_5 \end{pmatrix} \begin{pmatrix} r \\ c \end{pmatrix} + \begin{pmatrix} a_3 \\ a_6 \end{pmatrix} \quad (4.15)$$

The affine transformation can accommodate shearing in addition to scaling, translation, and rotation. In most practical cases, consideration of illumination changes is sufficient, but it may also be necessary to compensate for brightness and contrast between images, which are caused by radiometric variation in imaging. The intensity-mapping function and parameters in MPIR take care of the changes in brightness and contrast and are expressed by:

¹While the arrangement of pixels describes the spatial structure of an image, the radiometric characteristics describe the actual information content in an image. Every time an image is acquired on film or by a sensor, its sensitivity to the magnitude of the electromagnetic energy determines the radiometric resolution.

²A pyramid is a set / sequence of images with decreasing resolution.

$$g_2 = a_7 g_1 + a_8 \quad (4.16)$$

where g_1 and g_2 are the gray scale images. The Levenberg-Marquardt (LM) algorithm is used to estimate the transformation parameters iteratively. The LM nonlinear optimization algorithm is well suited for performing image registration based on least-squares criterion. Combining the spatial mapping and the intensity mapping functions, the complete relationship between the two input images is achieved:

$$g_2(r, c) = [a_7 g_1(p, q) + a_8] + n(r, c) \quad (4.17)$$

where $n(r, c)$ is due to noise existing in both images, and the eight transformation parameters a_k , $k = 1, 2, \dots, 8$ are unknown. They are estimated using the intensity-based method for matching, since image registration methods based on initial intensity values can make effective use of all data available. The parameters (a_1 to a_8) are estimated by the procedure similar to the one in [Keller et al. 2002].

4.1.4 Motion estimation based on graph theoretic methods

These methods use high level features such as identified parts of an object or relationships between features. One of the more commonly known algorithms that employs these methods is image registration using Genetic Algorithms (GAs). GAs have been known to be robust for search and optimization problems. Image registration can take advantage of the robustness of GAs in finding the best transformation between two images.

GA was formally introduced by John Holland and his colleague [Holland. 1975]. It is based on the natural concept that diversity helps to ensure a populations survival under changing environmental conditions. GAs are a simple and robust methods for optimization and search and have intrinsic parallelism. GAs are iterative procedures that maintain a population P of candidate solutions encoded in form of chromosome string. The initial population can be selected heuristically or randomly. For each generation, each candidate is evaluated and is assigned the fitness value that is generally a function of the decoded bits contained in each candidates chromosome. These candidates will be selected for the reproduction in the next generation based on their fitness values. The selected candidates are combined using the genetic recombination operation crossover. The crossover operator exchanges portions of bit string hopefully to produce better candidates with higher fitness for the next generation. The mutation is then applied to perturb the string of chromosome as to guarantee that the probability of searching a particular subspace of the problem space is never zero [Zheng et al. 1993]. It also prevents the algorithm from becoming trapped on

local optima [Xie et al. 2003], the reproduction process is illustrated in figure 4.3. Then, the whole population is evaluated again in the next generation and the process continues until it reaches the termination criteria. The termination criteria may be triggered by finding an acceptable approximate solution, reaching a specific number of generations, or until the solution converges.

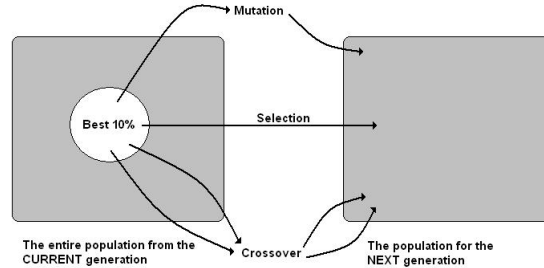


Figure 4.3: An abstract view of generational replacement in a genetic algorithm.

(This process is repeated many times, once for each iteration requested. The goal should be that each generation is better than the previous one, or at least no worse.)

Instead of linearly searching the whole search space, GA-based search techniques are used to selectively explore the huge search space. Unlike traditional linear search, the GAs adaptively explore the search solution space in a hyper-dimension fashion [Holland. 1975, Goldberg. 1989]. Consider for instance the search solution space of possible 2^{32} solutions which, in sub-pixel image registration, could be the combination of rotation and x-y translations. It is obvious that searching the whole space using linear search is impractical. Even sampling this search space by a fraction of 10^6 still leaves over 4,000 possible solutions. To solve image registration by GA, the transformation from one image to the next needs to be expressed in the form of a chromosome. An example of how this can be done is given in table 4.1. Table 4.1 shows three transformations, rotation and x-axis and y-axis translations, encoded in the 32-bit chromosome string. Using a bit encoding scheme for the chromosome string, a 12-bit field is used to represent possible relative rotation of the input image to the reference image. Likewise, 10 bits are used to express translation in x-axis and 10 more for the y-axis. After the transformation from one image to the next is expressed in the form of a chromosome, image registration can be solved by applying a standard GA algorithm as explained above.

4.1.5 Motion correction systems

A motion correction system needs to be able to cope with irregular conditions, such as moving objects and intentional panning that degrade the performance of

12 bits	10 bits	10 bits
Rotation	Translation X	Translation Y

Table 4.1: Encoding of transformation into chromosome string

the DIS system. The motion correction system determines the global motion of a frame by appropriately processing local motion vectors, and decides whether the motion of a frame is caused by undesirable fluctuations of the camera or intentional panning. The stabilised image is generated by reading out the proper block of fluctuated image in the frame memory, as illustrated in figure 4.4.

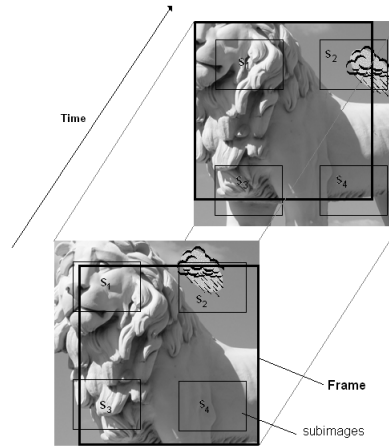


Figure 4.4: Reading proper block of image, with moving object, into frame memory

Various algorithms have been developed to estimate the global motion of a frame from local motion vectors [Liu et al. 1993, Musmann et al. 1985, Okada. 1996, Kinugase et al. 1990]. Most of these algorithms are complicated and computation will accordingly bring about a high level of costs.

In an image with motion, some sub-images with moving objects can produce motion vectors which are significantly different from the other motion vectors. Figure 4.4 shows an image that has moving objects in some sub-images. Figure 4.5 shows the correlation measure calculated using equation 4.3 from sub-image S_1 which has no moving objects and sub-image S_2 with moving objects respectively. In figure 4.5, for display, the correlation measures are normalized using $\frac{C_i(m,n)}{C_{max}}$ where C_{max} is the maximum $C_i(m,n)$ within the search range. It is shown that there does not exist a distinct maximum correlation value in figure 4.5 (b).

In general, motion vectors from the sub-images with moving objects are not

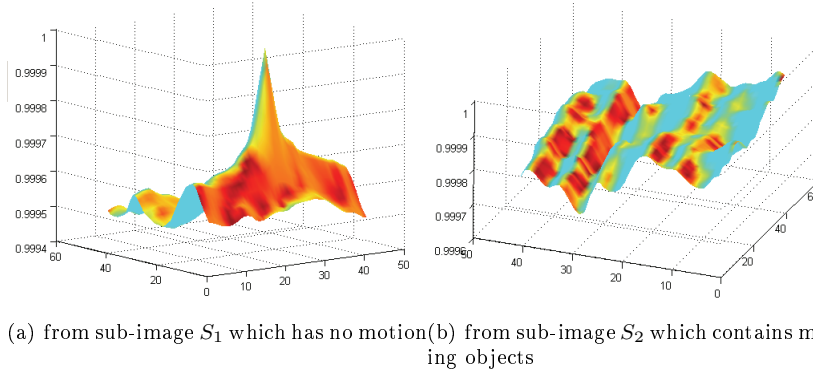


Figure 4.5: Correlation measures

reliable and should be excluded from the global motion decision process. Moreover, since the movement, from the camera shaking, is relative slow compared to the frame rate of the video camera, two successive frames fluctuated by the camera shaking should have similar global motion.

Based on these properties of the camera's movement [Ko et al. 1998] propose a simple and robust motion correction scheme where global motion decision is performed using current local motion vectors ($V_1^t, V_2^t, V_3^t, V_4^t$) and the previous global motion vector V_g^{t-1} . In the proposed algorithm, the global motion vector is obtained by:

$$V_g^t = \text{median} \{V_1^t, V_2^t, V_3^t, V_4^t, V_g^{t-1}\} \quad (4.18)$$

Local motion vectors affected by undesirable conditions such as moving objects can be viewed as impulses. It is known that the median filter is very effective in eliminating impulses. Therefore, the median-based method in equation 4.18 can exclude such abrupt local motion vectors and produce a global motion vector similar to the previous one.

After determining the global motion vector, the motion correction system decides whether the motion of a frame is caused by the camera shaking or intentional panning. For this decision, the global motion vector of a frame is integrated with a damping coefficient, and the integrated motion vector designates the final motion vector of a frame for motion correction. The integrated motion correction vector V_α for estimating intentional panning is given by:

$$V_\alpha^t = D_1 V_\alpha^{t-1} + V_g^t \quad (4.19)$$

where V_g^t is a global motion vector and $D_1 (0 < D_1 < 1)$ is a damping coefficient for smooth panning.

4.2 Drawing graphics

When drawing computer graphics, especially when drawing on top of other graphics, it is useful to be able to transform the objects or the position of the objects that are drawn. A *transformation* changes the positions of points in the plane. A set of points, when transformed, may as a result acquire a different shape. The transformations that move lines into lines, while preserving their intersection properties, are special and interesting, because they will move all polylines into polylines and all polygons into polygons. These are the *affine transformations*. Every affine transformation can be expressed as a transformation that fixes some special point (the "origin") followed by a simple translation of the entire plane. These point-fixing transformations are the linear ones. For more details on affine transformations see [Angel 2000], which this short presentation is based on.

Transforms are used most commonly to adjust a view's drawing area by translating the view's origin to some prescribed location on the screen. However, it is also possible to use transforms to arbitrarily scale or rotate shapes within their view. Because transformations occur relative to the origin of the local coordinate system, several transforms may need to be concatenated together to generate the proper effect. For example, Figure 4.6 shows a rectangle and the result of rotating it 45 degrees. To rotate the rectangle around its origin, you need to translate that origin to the center of its coordinate system, apply the rotation, and then translate the rectangle back to its original location.

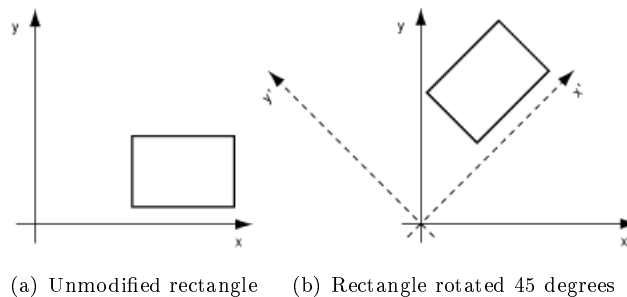


Figure 4.6: Affine Transform

Affine transforms are usually represented using homogeneous coordinates: given a point (x,y) in the traditional plane, its canonical homogeneous coordinate is $(x,y,1)$. The Affine transforms are represented in Homogeneous coordinates because the transformation of point A by any Affine transformation can be expressed by the multiplication of a 3×3 Matrix and a 3×1 Point vector.

The above property is not trivial. For example, a translation in normal Cartesian space results in the addition of a Point vector to the Point vector to transform

while in Homogeneous space, the same translation transformation results in a matrix/vector multiplication. There are four commonly used Affine Transformations: translation, scaling, rotation and shearing, which will be described in further detail in the following subsections.

Rather than applying separate transforms to translate, scale, and rotate a single shape, it is possible to combine a group of transforms into a single transform and apply only that transform. Therefore, it is possible to build up a complex transform by invoking the individual rotate, scale, shear, and translate methods in the proper sequence.

4.2.1 Translation

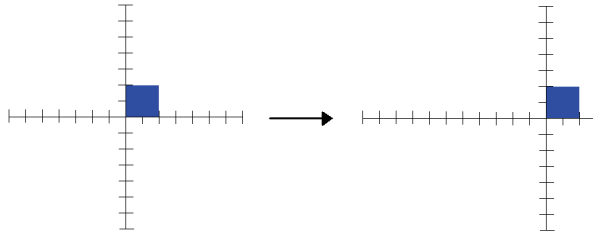


Figure 4.7: Translate

Translation matrix:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (4.20)$$

To translate a point in the xy -plane to a new place by adding a vector $\langle t_x, t_y \rangle$, it is not difficult to see that between a point (x, y) and its new place (x_{trans}, y_{trans}) , we have $x_{trans} = x + t_x$ and $y_{trans} = y + t_y$. Figure 4.7 illustrates the concept of translation. And equation 4.21 shows an example of the translation of point (x, y) by multiplication with a translation matrix in homogeneous space.

$$\begin{bmatrix} x_{trans} \\ y_{trans} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.21)$$

4.2.2 Scaling

Scaling matrix:

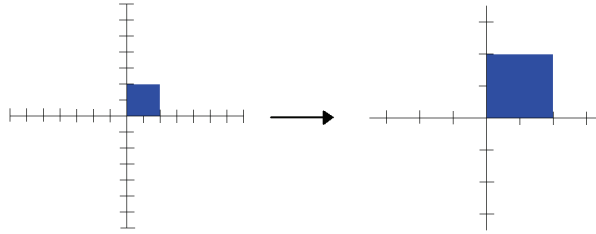


Figure 4.8: Scale

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.22)$$

Scaling transformations stretch or shrink a given object and as a result, change lengths and angles. The meaning of scaling is making the new scale of a coordinate direction n times larger. In other words, the x coordinate is "enlarged" n times, as illustrated in 4.8.

Scaling can be applied to all axes, each with a different scaling factor. For example, if the x - and y -axis are scaled with scaling factors S_x and S_y , respectively, the transformation matrix is:

$$\begin{bmatrix} x_{scale} \\ y_{scale} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.23)$$

4.2.3 Rotation

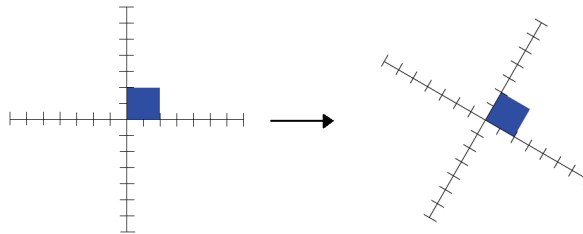


Figure 4.9: Rotate

Rotation matrix (with Θ measured in radians)

$$R(\Theta) = \begin{bmatrix} \cos \Theta & \sin \Theta & 0 \\ -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.24)$$

A rotation transformation simply rotates the reference system, as illustrated in figure 4.9. If a point (x, y) is rotated an angle Θ about the coordinate origin to become a new point (x_{rot}, y_{rot}) , the relationships can be described as follows:

$$\begin{bmatrix} x_{rot} \\ y_{rot} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \Theta & \sin \Theta & 0 \\ -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.25)$$

4.2.4 Shearing

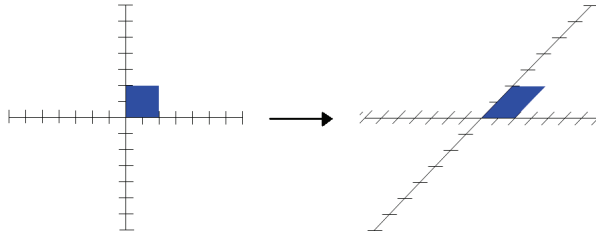


Figure 4.10: Shear

Shearing matrix:

$$SH = \begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.26)$$

Shear is the translation along an axis, e.g. the x-axis, by an amount that increases linearly with another axis, y-axis. It produces shape distortions as if objects were composed of layers that are caused to slide over each other, as illustrated in figure 4.10.

How far a direction is pushed is determined by a shearing factor. On the xy-plane, one can push in the x-direction, positive or negative, and keep the y-direction unchanged. Or, one can push in the y-direction and keep the x-direction fixed. The following is a shear transformation in the x-direction with shearing factor α :

$$\begin{bmatrix} x_{shear} \\ y_{shear} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.27)$$

The shear transformation in the y-direction with shearing factor β is the following:

$$\begin{bmatrix} x_{shear} \\ y_{shear} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.28)$$

Chapter 5

Experiments, design and results

This chapter contains a presentation of the three different components developed for this thesis. Each component is described in its own section. The final section describes how these three components are combined and integrated into the Groundstation.

5.1 Remote Control System

A computer based Remote Control System (RCS) which enables a pilot to control the helicopter through the Groundstation with a joystick has been developed. There are several scenarios where it is preferable to have the option of letting a human pilot take control of the autonomous aircraft. Whether it is because the task is still too complex to be performed by a computer or it needs a human's judgement. To reference some of the examples given in chapter 2 of possible usages of UAVs, if a UAV was assisting the police in a chase and it suddenly was engaged by the ones it was chasing, then a human pilot could quickly take control of the helicopter and perform the necessary evasive actions. Another example is when using a UAV for inspecting a large building for damage, e.g. after an earthquake, a human expert could manually navigate the UAV to look at all the things necessary to establish whether or not the building would be safe for humans to enter.

Operating a UAV in a semi-autonomous mode requires communication between the helicopter and the Groundstation. During the autumn of 2004 Jarle Anfinson and this author developed, as a subproject for ARCH, a protocol for communication between the helicopter and the Groundstation and an API for the server (helicopter). The work was part of the course TDT4715 Algorithm

Construction and Visualization at NTNU. The intention of the course was to act as an introduction and preparation for the post-graduate thesis. The details of this communication protocol can be found in appendix B. However, this protocol was designed first and foremost for a helicopter in fully autonomous mode. An autonomous helicopter is usually controlled with simple messages i.e. fly to a given waypoint. Therefore, in the existing communication protocol, whenever a message is received on the helicopter it must send a response back to the sender to confirm that the message is received. In semi-autonomous mode, however, there is a need to continuously send control signals (messages) to the helicopter. To have the helicopter send a response back to the sender each time it receives a control signal would not only be pointless, it would also put an unnecessary amount of traffic on the communication line. Therefore, there was a need to extend both the ARCH API and the communication protocol so that it would be more suitable for semi-autonomous mode.

In the API a new function for receiving commands, which does not need to be acknowledged, from the Groundstation has been added. The new API interface function is provided here with its function signature accompanied with a description of its functionality and intended usage.

```
void ARCH_client_perform_task(const int unit_id, const vector<ARCH arg> params) -
    This function is used for sending control signals when piloting the
    helicopter through semi-autonomous control. This function first sets
    the parameters to be performed on the helicopter, and then performs
    them.
```

Table 5.1 shows the new message added to the list of messages used between the ARCH server and clients. In the table, the reference to the corresponding interface function in the API is given. The code for the protocol message is also given, together with a representation of the succeeding arguments. The different arguments in the message are separated with the '%' character. In this representation, the operator + is used to indicate one or more repetitions of its operand. The term *bytes* indicates an array of bytes. In the current version of the system, all of the units treat all of their parameter values as doubles.

Interface function	Code	Arguments
ARCH_client_perform_task	303	<i>int%(int%bytes%)+</i>

Table 5.1: New messages from client to server

A control signal (message) sent from the Groundstation to the helicopter would normally look something like:

303%9%0.0%10%0.21%11%0.14%2%-5

The first word, *303*, is the code corresponding to the interface function `ARCH_client_perform_task` of the API. Message identification codes are always interpreted as integers. The second word, *9*, is also interpreted as an integer, this is the identification code for the pitch attitude angle of the helicopter. The third word, *0.0*, is the helicopter's desired pitch angle, in radians. The same follows for the remaining three pairs where *10* is the code for the bank attitude angle, *11* is the code for the yaw attitude angle and *2* is the code for the altitude. The altitude is specified in meters and uses a negative axis which makes *-5* five meters above the ground.

Before the control signals (messages) can be sent to the helicopter, they need to be obtained from the pilot. These signals are obtained through a joystick connected to the Groundstation. As described in chapter 3 the entire Groundstation is written in the programming language Java. Because there is no native support for polled input such as a joystick in Java runtime, an open source project called JInput has been utilized to connect the joystick to the Groundstation. When in semi-autonomous mode the joystick is polled and its values are converted into suitable values for the helicopter before they are transmitted to it from the Groundstation. To allow the onboard control system some time to work, in order to keep the helicopter stable, a control message is only sent to the helicopter when the input signals from the joystick differ from the last time the joystick was polled. Algorithm 1 describes the actions taken each time the joystick is polled.

Algorithm 1 Poll joystick

```

1. Every tenth millisecond do
2.  $v \leftarrow j$  // Read values from joystick
3. If  $v \neq v_{old}$  do // If new values differs from old values do
4.    $send(convert(v))$  // Send message to helicopter, containing only the values
   // that are different
5.    $v_{old} \leftarrow v$  // Set new values = old values
```

When piloting the helicopter in semi-autonomous mode the pilot does not have direct control of the helicopter. The system can be compared to the *fly-by-wire* systems described in chapter 2, where the onboard system assists the pilot with flying the aircraft. Since there are very few who are capable of piloting a helicopter, the computer assisted mode is preferable because it makes it possible for anyone to pilot the helicopter. In semi-autonomous mode the angle of the pilots' joystick is proportional with the attitude angle of the helicopter. For instance if the pilot moves the joystick forward, the helicopter will pitch forward, which again makes the helicopter move forward. But the helicopter will maintain the current altitude, roll angle and yaw angle. If the pilot was to let go of the joystick, or move it to the neutral position, the helicopter would go into a hover-mode at its current position.

There are three different flight-modes in the current remote control system.

There is one called absolute altitude, one called relative altitude and one called pursuit-mode which will be described later on. When piloting in absolute altitude mode, the altitude lever on the joystick will correspond directly to an altitude of the helicopter. For instance at the minimum position of the altitude lever the helicopter would be two meters above the ground, and at the maximum position the helicopter would be ten meters above the ground. In relative altitude flight-mode, however, the altitude lever on the joystick controls the *rate of climb* of the helicopter. When the altitude lever is at neutral position, the helicopter will maintain its altitude. The more the altitude lever is moved towards its maximum position, the faster the helicopter will gain height, and vice versa. Both of these flight-modes have their own advantages, in absolute altitude mode it is virtually impossible to crash the helicopter and it is very easy to move fast between two set heights. In relative altitude mode, one has significantly greater capabilities due to the fact that there are no restrictions on where to fly.

The pursuit-mode separates itself from the other flight-modes, because the operator does not control the helicopter. Instead, the helicopter follows an object by itself. There is an ongoing subproject within the ARCH project where the goal is to develop a system, which can recognize and “lock on” to objects in a video image. This system uses the *Condensation* algorithm (*Conditional Density Propagation*), which enables it to track agile moving objects, in the presence of dense background clutter. For more information on the *Condensation* algorithm see [Condensation Web]. The pursuit-mode in the RCS was developed under the assumptions that this tracking system is available on the Groundstation. The ARCH tracking system is scheduled to be finished June 2005, and it should be just a matter of plugging it into the Groundstation to enable the RCS to use it for following objects. In pursuit-mode the RCS controls the helicopter based on the inputs from the tracking system. The input is in the form of a bounding box¹. The first bounding box received from the tracking system is set as the reference bounding-box. The RCS pilots the helicopter to make all of the following bounding boxes as close to the reference bounding box as possible. For instance, if the new bounding box is smaller than the reference bounding box, the helicopter moves forward. And the smaller it is compared to the reference, the faster the helicopter will fly towards it.

As described in chapter 3 the helicopter’s control system is an open source project called Autopilot. Unfortunately, their current control system has a somewhat limited flight envelope². The consequence of this is that if the helicopter surpasses a certain speed, the control system will no longer be able to maintain a stable flight. The helicopter will start to oscillate more and more until it crashes. To avoid this from happening when piloting the helicopter in semi-autonomous mode some restrictions have been placed on the remote control system. When the pilot moves the joystick to one of its extremities,

¹A bounding box, or minimum bounding box, is the smallest possible rectangle completely enclosing the object.

²Flight Envelope = Normal operating environment, within which an aircraft is safe to fly.

the pitch or roll angle of the helicopter will never surpass twelve degrees. This ensures that the helicopter will never gain enough speed to move outside the control system's flight envelope. With these restrictions in place a pilot is free to navigate the helicopter in semi-autonomous mode wherever and in whatever way she or he chooses. The helicopter will remain stable and with predictable flight characteristics. Should the pilot for some reason become incapable of piloting the helicopter, she or he would simply have to let go of the joystick and the control system would level the helicopter and go into a hover-mode.

At present date the ARCH helicopter is not yet fully operational. Therefore, it has not been possible to test the remote control system in real life. However, it has been tested extensively in the simulator. The simulator is part of the Autopilot project and was designed to test the onboard control system. Figure 5.1 shows a screen-shot of one of the testing runs in the simulator. For more information concerning the simulator and on the autopilot project in general see [Autopilot Web].

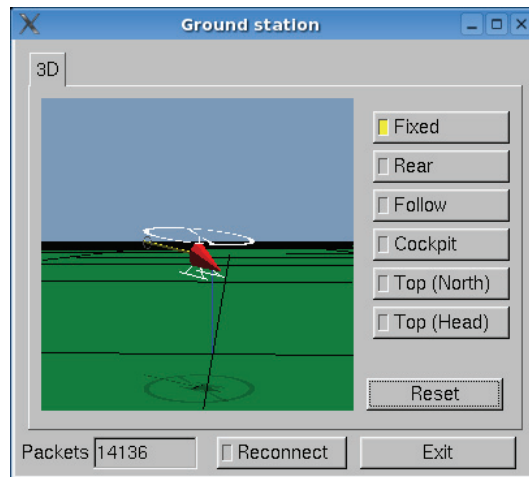


Figure 5.1: Testing in simulator

Because the tracking system is not yet available in the Groundstation, the pursuit-mode had to be tested by using a custom developed test system which was integrated into the Groundstation. The test system allowed for the positioning of a bounding box inside the *virtual cockpit*, which will be described in the following section. The position and size of the bounding box were sent to the RCS as input, and one could observe the helicopter follow the "virtual" rectangle in the simulator. A screen-shot from the virtual cockpit in pursuit-mode is shown in figure 5.2.

Piloting the helicopter, in semi-autonomous mode, from the Groundstation obviously has some significant limitations. One of the most obvious limitations is

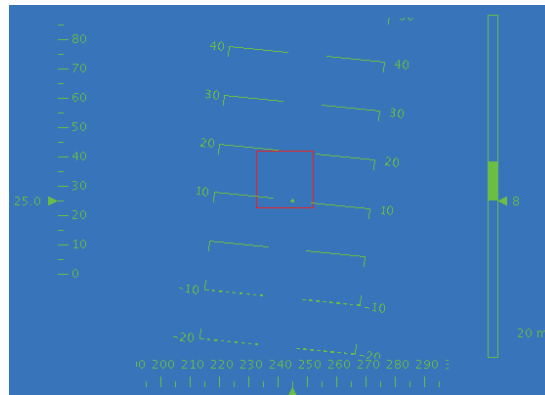


Figure 5.2: Pursuit-mode

the situation were one does not know the helicopters position and are not able to see what the helicopter sees. The proposed and implemented solution is the *virtual cockpit* which will be described in the following section.

5.2 Virtual Cockpit

One of the challenges of an UAV, in semi-autonomous mode, is to move the pilot physically from the aircraft to the ground, but at the same time keep his/hers “eyes” in the air. To be able to operate an UAV, in a semi-autonomous mode, the pilot needs to see what the UAV is seeing. The result of this is a *virtual cockpit*. The goal of the virtual cockpit is to recreate the important parts of the cockpit on the ground, or rather the information the pilot could acquire in the actual cockpit. As for a pilot of a standard aircraft there is a significant amount of data a pilot of an UAV needs to assimilate. It has become common to put primary flight information on top of the visual information. This function is known as Head Up Display (HUD) and is described in chapter 2. HUDs are used to cut down on the pilots workload.

The most important part of the virtual cockpit is the visual information from the helicopter. The camera mounted on the helicopter is equipped with its own transmitter which transmits the video signal to the ground. On the ground the signal is received with the corresponding receiver and the signal is sent to the Groundstation through an analog-to-digital converter. This equipment is described in chapter 3. In order to enable the Groundstation to display this visual information, a package called Java Media Framework has been used. With all of this working together a pilot using the virtual cockpit in the Groundstation would be able to see what the UAV is seeing.

In order to put the primary flight information on top of the visual information

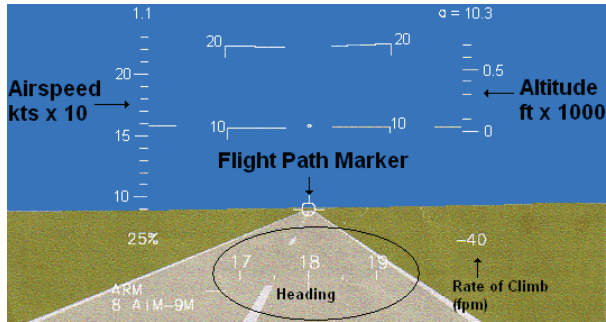


Figure 5.3: Example of a Head Up Display

the flight information must first be sent from the helicopter to the Groundstation. The ARCH API and communication protocol, described in appendix B, already supports this functionality. The Groundstation simply subscribes to the desired information, and the helicopter sends this information continuously to the Groundstation until the Groundstation unsubscribes. The information sent from the helicopter to the Groundstation in semi-autonomous mode is shown in table 5.2.

ID	Parameter	Description	Data type
2	z	Altitude in meters, negative axis	double
3	vx	Velocity, in meters per second, along the x-axis	double
4	vy	Velocity, in meters per second, along the y-axis	double
5	vz	Velocity, in meters per second, along the z-axis	double
9	phi	Attitude angle in radians, roll	double
10	theta	Attitude angle in radians, pitch	double
11	psi	Attitude angle in radians, yaw	double

Table 5.2: Information sent from helicopter to Groundstation

After obtaining the flight information the Groundstation must draw this information on top of the visual information. Over the years several HUD designs have been developed. All with the intent of making it as easy as possible for the pilot to assimilate all the important flight data. As described in section 2.3 a HUD can provide a lot of mission specific information. However, there is a great deal of essential flight data which acts as a common denominator between the different HUD designs. The essential flight data includes the pitch angle and bank angle in form of an artificial horizon, height, airspeed and heading. Although different HUD designs present the data in different ways, almost every HUD includes these data. Figure 5.3 shows an example of a HUD design with the essential flight data marked.

The one item, which looks more or less the same in all of the commonly known HUD designs, is the artificial horizon, which appears as a “ladder” in the middle of the HUD. The “ladder” stays orthogonal to the horizon independent of the aircraft’s attitude. If the aircraft’s pitch increases, it will appear as if the aircraft is climbing up the “ladder” and vice versa. This makes it non-trivial to draw the artificial horizon on the HUD. Because the “ladder” needs to stay orthogonal to the horizon, the ladder’s position on the HUD will change continuously according to the aircrafts attitude. Instead of continuously calculating the position of every line that needs to be drawn on the ladder, it would be easier and “cleaner” to use affine transformations, which are described in chapter 4. In order to be able to draw the artificial horizon in the same manner regardless of the attitude of the helicopter, it is necessary to perform several transformations both before and after drawing. The first step is to translate the “window” where the “ladder” is to be drawn. The translation will put the center of the ladder into the center of the reference system. How much the “window” needs to be translated in the vertical direction depends on the pitch of the helicopter. The second step is to rotate the “window”. The window is to be rotated with the inverse of the roll angle of the helicopter. This will compensate for the roll angle of the helicopter and make the “ladder” stand orthogonal to the horizon. After the “ladder” is drawn into the “window”, the “window” needs to be rotated back to its original angle. This is done by rotating it with the roll angle of the helicopter. Finally the window has to be translated back into its original position. This is done by translating it with the inverse of what it was translated with in the first step. The steps needed to draw the artificial horizon are illustrated in figure 5.4. Transformations are used when drawing the ARCH HUD, not only for the artificial horizon but also when drawing the heading and speed indicators. On these two indicators translations are used much in the same way as on the artificial horizon.

The design of the ARCH HUD is a quite simple design and it is based on other HUD designs which have been proved to work for many years. The current design of the ARCH HUD displays only navigation information. However, the architecture of the HUD module is designed in such a way that the HUD “engine” is separated from the graphics, as illustrated in figure 5.5. In this figure the HUD class provides an interface for creating families of related or dependent objects without specifying their concrete class. This is done to make it very quick and easy to change the way the HUD appears to the pilot. This also makes it possible to have multiple HUD designs/modes, which the pilot can switch between during flight.

The implemented HUD design is called navigation-mode and is shown in figure 5.6. As shown in the figure all the primary flight data is available through the HUD. Both the speed on the left-hand side and the rate-of-climb on the right-hand side is in m/s. The altitude displayed in the bottom right corner, is given in meters and the heading, also displayed at the bottom part of the display, is given in degrees.

To make the helicopter easier and less stressful to pilot, a digital image stabilizer

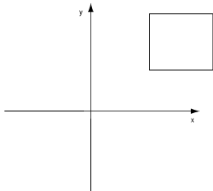
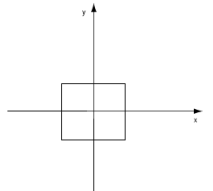
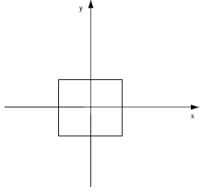
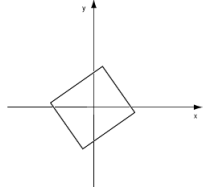
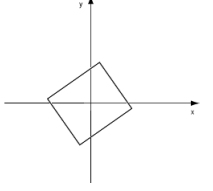
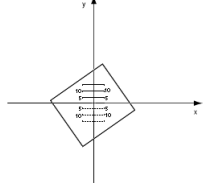
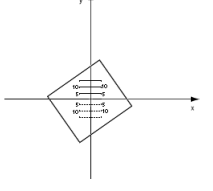
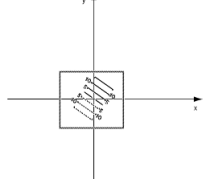
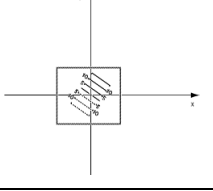
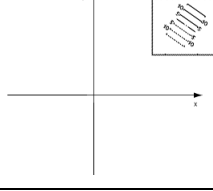
Transformation	Before	After
Translate window		
Rotate window		
Draw "ladder"		
Rotate back		
Translate back		

Figure 5.4: Drawing the artificial horizon

has been developed. This component is described in the following section.

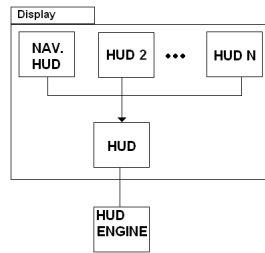


Figure 5.5: HUD Architecture

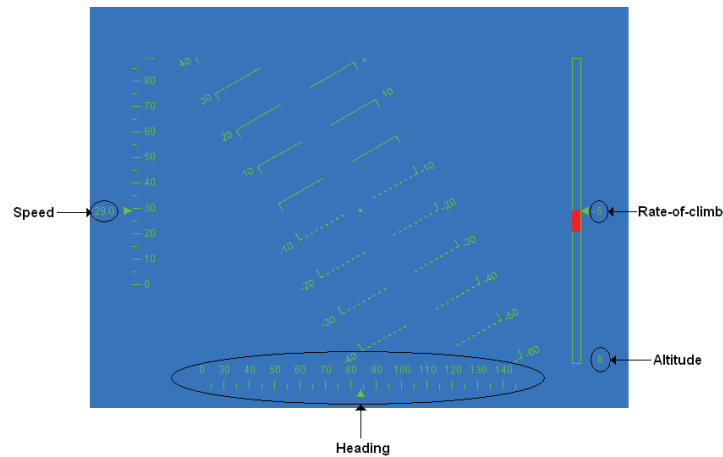


Figure 5.6: HUD (navigation-mode)

5.3 Image stabilization

One of the goals of this report is to propose an image stabilization solution for the video signal which is transmitted from the helicopter to the Groundstation. Because a small helicopter like the ARCH UAV is quite exposed to vibrations and shaking (i.e. wind), stabilizing the video signal would make the helicopter easier and less stressful to pilot on the ground. Methods introduced in chapter 4 can be used to stabilize a video signal such as this. This section describes which methods that have been applied in the ARCH project and why.

In the previous chapter four different classes of motion estimation were presented. Two of these classes stands out by appearing more suitable for real-time image stabilization than the others. Namely, motion estimation based on correlation methods and FFT methods. In order to make an analysis of the two methods as to which is the most suited for the ARCH project, two motion estimation systems have been developed. One using a correlation method and one

using a FFT method. The following sections will describe the two systems and finally compare them. Section 5.3.4 will describe the proposed motion correction system.

5.3.1 Motion estimation based on a correlation method

The motion estimation system, which uses a correlation method, developed for the ARCH project is a block matching algorithm (BMA). This algorithm performs binary motion estimation using 1-bit planes which are extracted from a video sequence. This motion estimation technique can be realized using only Boolean functions, which have significantly reduced computational complexity, while at the same time maintaining motion estimation accuracy, as shown in [Ko et al. 1998].

Before the algorithm itself is presented, the bit-plane decomposition of a gray-scale image is introduced. Let the gray-level of the pixel at location (x,y) in the t -th image frame with 2^K gray-levels be represented as:

$$f^t(x, y) = a_{K-1}2^{K-1} + a_{K-2}2^{K-2} + \dots + a_12^1 + a_02^0 \quad (5.1)$$

where $a_k, 0 \leq k \leq K-1$, is either 0 or 1. Let the k -th order bit-plane image be denoted by $b_k^t(x, y)$. This plane contains all the k -th order (a_k) bits. For example in the case of a 8-bit image, an image is composed of eight 1-bit planes $b_0^t(x, y) \sim b_7^t(x, y)$, ranging from plane 0 to plane 7. Figure 5.7 shows the eight 1-bit planes, which together make the gray-scale image. It is important to notice that only the higher order bit-plane images contain visually significant data, whereas the other bit-planes contribute to more subtle details within the image.

The algorithm estimates four local motion vectors from four sub-images (S_1, S_2, S_3, S_4) placed in appropriate positions in the bit-plane. Each motion vector of a sub-image in the current bit-plane is determined by evaluating bit-plane matching over sub-images in the previous bit-plane and selecting the sub-image which yields the closest match. This approach assumes that all pixels within the sub-image have uniform motion and the range of the motion vector is constrained by the search window. Let the size of each sub-image be $M * N$ and a search window be $(M + 2p) * (N + 2q)$. For bit-plane matching, the definition of the correlation measure is given by:

$$C_i(m, n) = \sum_{(x,y) \in S_i} b_k^t(x, y) \oplus b_k^{t-1}(x + m, y + n) \quad (5.2)$$

where $b_k^t(x, y)$ and $b_k^{t-1}(x, y)$, respectively, are the current and previous k -th order bit-planes, and \oplus is the exclusive-OR operator. At each (m, n) , $-p \leq m \leq p$ and $-q \leq n \leq q$, within the search range, the proposed matching method calculates $C_i(m, n)$ which is the number of unmatched bits between the

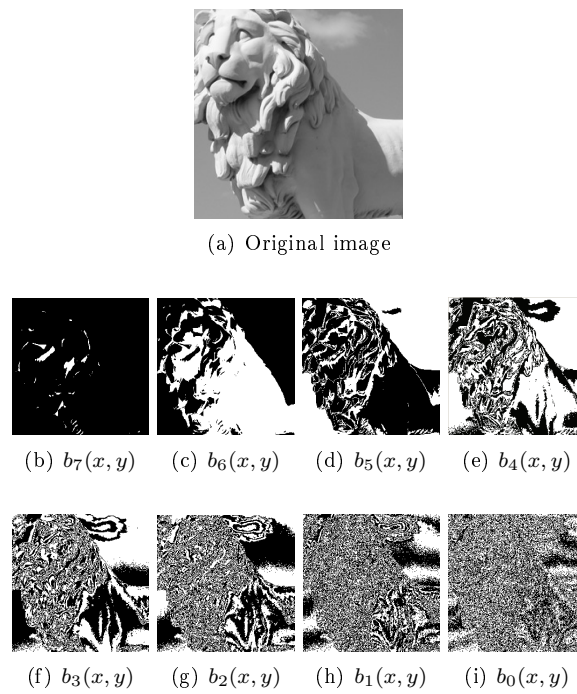


Figure 5.7: Example of generating bit-planes from gray-scale image

reference sub-image in the current bit-plane and the compared sub-image in the previous bit-plane. The smallest $C_i(m, n)$ yields the best matching for S_i , and thus the local motion vector V_i from S_i is obtained as

$$V_i^t = \arg \min \{C_i(m, n), -p \leq m \leq p, -q \leq n \leq q\} \quad (5.3)$$

Since this algorithm performs motion estimation using a single bit-plane, it is important to select an appropriate bit-plane for bit-plane matching. In this algorithm the 4-th order bit-plane, $b_4^t(x, y)$, is utilized to estimate the local motion vector since it contains both the global information and details of the original image. The entire algorithm is described in algorithm 3 which uses the auxiliary algorithm, algorithm 2. Both algorithms require access to some or all of two images, the current frame and the previous one.

Algorithm 2 Correlation

Input: C, P

1. For each pixel $p_{(x,y)}$ in C
 2. extract the bit, from the 4-th bit-plane, in $p_{(x,y)}$ into *bit1*
 3. extract the bit, from the 4-th bit-plane from the corresponding position of $p_{(x,y)}$ in P into *bit2*
 4. *sum* = *sum* + (*bit1* \oplus *bit2*)
 5. return *sum*
-

Algorithm 3 Motion estimation based on Bit-Plane Matching

1. Extract 4 sub-images ($S_1^n, S_2^n, S_3^n, S_4^n$) from the image
 2. For each sub-image S_i^n
 3. $P \leftarrow S_i^{n-1}$ // the corresponding sub-image in the previous frame
 4. $C_{min} \leftarrow \infty$
 5. For each sub-image C_j inside the search-window of S_i
 6. $C \leftarrow$ use the Correlation algorithm with C_i and P as input
 7. If $C < C_{min}$
 8. $C_{min} = C$
 9. $V_i \leftarrow$ the position of C_{min}
-

5.3.2 Motion estimation based on a FFT method

The motion estimation system, which uses a FFT method, developed for the ARCH project uses phase correlation to estimate motion. The theory behind phase correlation was used as an example of a FFT based method in chapter 4. This algorithm also estimates four local motion vectors from four sub-images (S_1, S_2, S_3, S_4) placed in appropriate positions in the frame. The local motion vectors can be found by applying the Fourier Transform to the sub-image of both the current and the previous frame:

$$\begin{aligned} FS_i^t &= \text{FourierTransform}(S_i^t) \\ FS_i^{t-1} &= \text{FourierTransform}(S_i^{t-1}) \end{aligned} \quad (5.4)$$

where S_i^t and S_i^{t-1} , respectively, are corresponding sub-images from the current and previous frame. And then calculating the cross-power spectrum by:

$$FCP = \frac{FS_i^t * FS_i^{*t-1}}{|FS_i^t * FS_i^{*t-1}|} \quad (5.5)$$

where FS^* is the complex conjugate of FS . And finally take the inverse Fourier Transform of FCP :

$$CP = \text{FourierTransform}^{-1}(FCP) \quad (5.6)$$

This will result in a image, CP , which has a distinct peak at the location where the center of the previous frame was located. As shown in figure 5.8. The entire algorithm for motion estimation based on phase correlation is described in algorithm 4. Algorithm 4 uses a function called `fft`, which stands for fast Fourier transform (FFT). The FFT is a discrete Fourier transform algorithm which reduces the number of computations needed for N points from $2N^2$ to $2N \lg N$, where \lg is the base-2 logarithm. Fast Fourier transform algorithms generally fall into two classes: decimation in time, and decimation in frequency. The Cooley-Tukey FFT algorithm first rearranges the input elements in bit-reversed order, then builds the output transform (decimation in time). The basic idea is to break up a transform of length N into two transforms of length $\frac{N}{2}$ using the identity:

$$\begin{aligned} \sum_{n=0}^{N-1} a_n \exp \frac{-2\pi i n k}{N} &= \sum_{n=0}^{\frac{N}{2}-1} a_{2n} \exp \frac{-2\pi i (2n)k}{N} + \sum_{n=0}^{\frac{N}{2}-1} a_{2n+1} \exp \frac{-2\pi i (2n+1)k}{N} \quad (5.7) \\ &= \sum_{n=0}^{\frac{N}{2}-1} a_n^{\text{even}} \exp \frac{-2\pi i n k}{\frac{N}{2}} + \exp \frac{-2\pi i k}{N} \sum_{n=0}^{\frac{N}{2}-1} a_n^{\text{odd}} \exp \frac{-2\pi i n k}{\frac{N}{2}} \end{aligned}$$

This is called the Danielson-Lancos lemma. For more information about the Fast Fourier transform see [FFT Web]. Algorithm 4 also uses a function called `conj`, which stands for complex conjugate. The complex conjugate of a complex number $z \equiv a + bi$ is defined to be $z^* \equiv a - bi$. The conjugate of a matrix $A = (a_{ij})$ is the matrix obtained by replacing each element a_{ij} with its complex conjugate, $A^* = (a_{ij}^*)$. For more information concerning the complex conjugate see [Conj Web].

5.3.3 Bit-plane matching or phase correlation

There are three main criteria when deciding which of the two algorithms is the more suitable for the ARCH project. Most important may be how well it

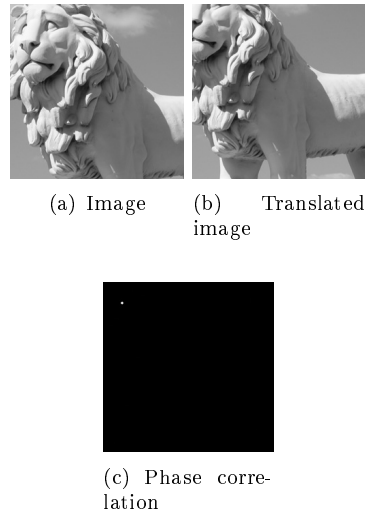


Figure 5.8: Inputs and output of phase correlation

Algorithm 4 Phase Correlation

-
1. Extract 4 sub-images $(S_1^n, S_2^n, S_3^n, S_4^n)$ from the image
 2. For each sub-image C , $C \in S_i^n$
 3. $P \leftarrow S_i^{n-1}$ // the corresponding sub-image in the previous frame
 4. $FC \leftarrow \text{fft}(C)$ // Fast Fourier Transform of current sub-image
 5. $FP \leftarrow \text{fft}(P)$ // Fast Fourier Transform of previous sub-image
 6. $FM \leftarrow FC * \text{conj}(FP)$ // multiply the current frame with the complex
// conjugate of the previous frame
 7. $AFM \leftarrow |FM|$
 8. $FCP \leftarrow \frac{FM}{AFM}$ // finish calculating the cross-product between the current
// and previous frame
 9. $CP \leftarrow \text{fft}^{-1}(FCP)$ // apply the inverse Fourier transform on the
// cross-product
 10. $point \leftarrow \max(CP)$ // locate location of the peak
-

performs under less than perfect conditions, since the video signal that will be transmitted from the helicopter to the Groundstation may be subject to a great deal of noise. The second most important criterion is the performance of the algorithm, especially since this will have to be performed in real-time. The final criterion is how much memory the algorithms use. This is not really a crucial factor, because the Groundstation is run on a laptop which have more than enough memory needed to perform any kind of motion estimation. But in any case the less memory an algorithm needs the better, and is therefore a part of this assessment.

Real world signals usually contain departures from the ideal signal that would be produced by the camera mounted on the helicopter. Such departures are referred to as noise. Noise arises as a result of unmodelled or unmodellable processes

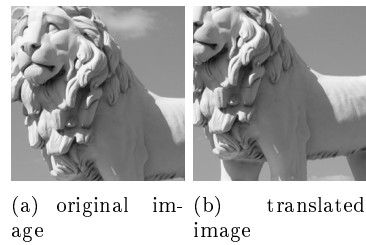


Figure 5.9: test-images

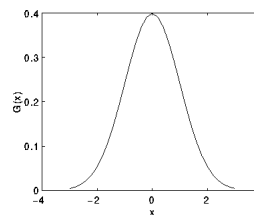


Figure 5.10: 1-D Gaussian distribution with mean 0 and standard deviation 1

going on in the production and capture of the real signal. It is not part of the ideal signal and may be caused by a wide range of sources, e.g. variations in the detector sensitivity, environmental variations, the discrete nature of radiation, transmission or quantization errors, etc. It was mentioned in chapter 4 that since the Fourier methods used the frequency domain, the methods achieved excellent robustness against correlated and frequency-dependent noise. This implies that the phase correlation algorithm is more robust against noise than the bit-plane matching algorithm. The reason why both of this algorithms were implemented was to run different tests on them and compare them with each other. To test how robust the two algorithms are against noise, they have both been used to estimate the motion between two images (figure 5.9) with increasing noise.

The two images have been corrupted with additive noise with a Gaussian distribution. The 1-D Gaussian distribution has the form shown in Figure 5.10. This means that each pixel in the noisy image is the sum of the true pixel value and a random Gaussian distributed noise value. To increase the noise in the images a coefficient has been added, which can be set to an increasingly larger number to obtain an increasingly noisier image:

$$N_{(x,y)} = I_{(x,y)} + G(p) * \sigma \quad (5.8)$$

Where N is the noisy image, I is the original image, $G(p)$ is a random Gaussian distributed noise value and σ is the coefficient which can be set to obtain the desired degree of noise. Figure 5.11 shows some samples of images with different

degree of noise where the two different motion estimators have been applied. On each image the estimated motion is illustrated with a thin black line from the center of the image to where the algorithm believes the center of the next frame is.

The graph in figure 5.12 shows the results from running these algorithms on increasingly noisier image pairs. The y-axis in the graph is the euclidean distance³ from the actual center of the next frame to where the algorithm believes the center of the next frame to be. From the graph it is easy to see that the phase correlation algorithm is considerably more robust against noise than the bit-plane matching algorithm. While the bit-plane matching algorithm starts to “wander off” when $\sigma > 1300$, the phase correlation algorithm stays “on target” while $\sigma < 140000$. It is clear to see from the images in figure 5.11 that it is impossible to pilot the helicopter based on the video signal if the noise on that signal is as dominant as the ones corrupted with $\sigma > 10000$. Therefore, the phase correlation algorithm is more than enough robust against noise. Whether or not the bit-plane matching algorithm is robust enough is more uncertain. It is quite accurate while $\sigma < 1300$, and as seen from the images in figure 5.11 there is quite a bit of noise when $\sigma = 1000$. However, when comparing the two, the phase correlation algorithm is clearly preferable when it comes to performing under imperfect conditions.

Performance (complexity and computation time) is another important criterion because motion estimation has to be done in real-time. It is found that the size of the sub-images (block size) greatly affects the performance of the two motion estimators. It is easy to see that the smaller the block size, the smaller the computation time will be. However, the blocks should be large enough to group pixels with similar motion but should also be small enough to separate pixels with different motion and multiple-object movement. In bit-plane matching the size of the search window (neighbourhood) also greatly affects the performance. Therefore, it is important to adjust the neighbourhood to be as small as possible while making sure that the movement between two frames will not reach outside the neighbourhood. One of the weaknesses of the bit-plane matching algorithm is that in order to obtain good accuracy the complexity and computation time might be high, because it searches every single “frame” inside the neighbourhood to find the matching block. The proposed phase correlation algorithm, in contrast, has much lower complexity by measuring the motion directly from phase correlation. Table 5.4 shows the time both of the motion estimators used to estimate the motion from the example image pair showed in figure 5.13. This image pair differs from the one used when testing with noise. This was done to give the bit-plane matching algorithm a better starting point, because it makes it possible to use a smaller search-window. It will also give more realistic results since the movement between each frame from the video-signal will not be very large. All the computations are performed on a computer with the specifications given in table 5.3. The block size used is 32x32 and the search window

³The Euclidean distance can be considered to be the shortest distance between two points, and is basically the same as Pythagoras’ equation when considered in 2 dimensions





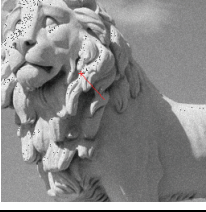
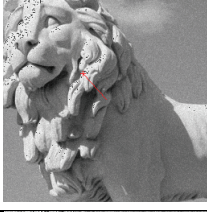
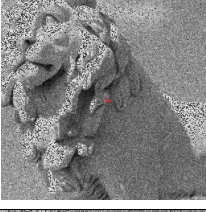
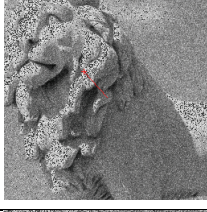
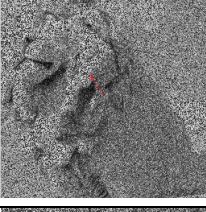
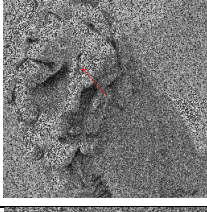
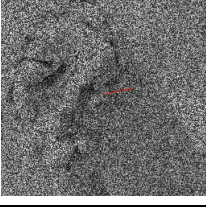
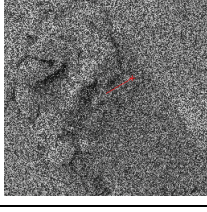
σ	Bit-plane matching	Phase correlation
0		
500		
1000		
5000		
10000		
15000		

Figure 5.11: Motion estimation on noisy images

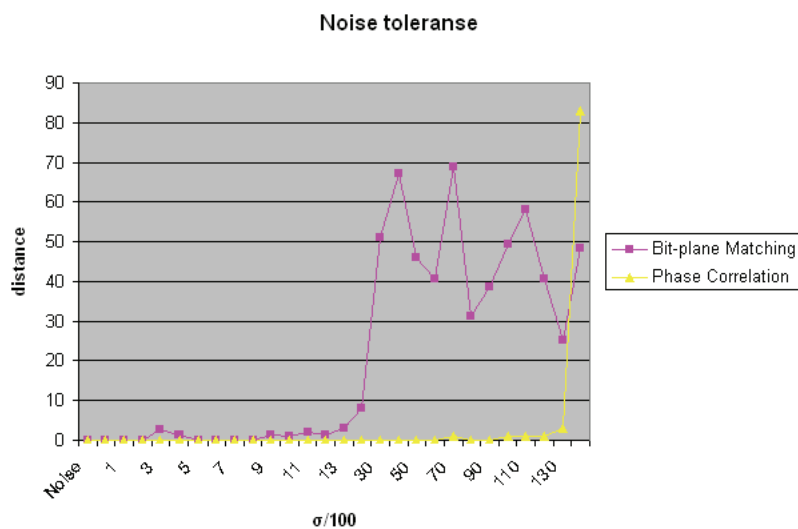


Figure 5.12: Test results from motion estimation

used in the bit-plane matching is 5×5 . From the results it is easy to see that compared to the bit-plane matching algorithm, the phase correlation algorithm is very computationally efficient.

Item	Version
Processor	i686 Intel(R) Pentium(R) 4 CPU 3.00GHz
Memory	512 DDR
Kernel version	2.6.8.1-12mdkmp
Operativ system	Linux

Table 5.3: Technical specifications on the computer



(a) original image (b) translated image

Figure 5.13: Image pair used in performance test

Memory usage is the final criterion. Because the bit-plane matching algorithm

Motion Estimator	Time
Bit-plane matching	45 ms
Phase correlation	18 ms

Table 5.4: Performance of motion estimators

only checks for correlation between different “images” it only needs to hold the integer with the number of mismatched bits between the two “images” which currently has the lowest number of mismatched bits. Therefore, the bit-plane matching algorithm hardly uses any memory at all, unlike the phase correlation algorithm which needs to hold on to different “versions” of the two images during the process. Therefore, when it comes to memory usage, the bit-plane matching algorithm is clearly favourable.

Based on these test criteria the phase correlation algorithm seems to be the better alternative. Even though it uses more memory than the bit-plane matching algorithm, its memory usage is relative moderate. Because memory usage is the least important criterion and the phase correlation algorithm proved to be considerably more robust against noise, and even a bit more effective, the phase correlation algorithm is used as a motion estimator in the ARCH project.

5.3.4 Motion correction

After the motion has been estimated between two frames the new frame needs to be corrected in respect to the previous one in order to remove shaking, vibrations, etc. The motion correction system implemented for the ARCH project is based on the theory on motion correction systems presented in chapter 4. This motion estimation system performs global motion decision using local motion vectors $(V_1^t, V_2^t, V_3^t, V_4^t)$, which are estimated by the motion estimator, and the previous global motion vector V_g^{t-1} . The global motion vector V_g^t is obtained by taking the median of these five motion vectors. The median of vectors is determined by separately selecting medians of each vector elements. The median filter is effective in eliminating impulses. Therefore, it can exclude such abrupt local motion vectors, caused by moving objects, and produce a global motion vector similar to the previous one. A motion correction system must also decide whether or not the movement from one frame to another is caused by shaking or intentional panning. For this decision, the global motion vector of a frame is integrated with a dampening coefficient, and the integrated motion vector designates the final motion vector of a frame. The final motion vector V_α^t is given by:

$$V_\alpha^t = D_1 V_\alpha^{t-1} + V_g^t \quad (5.9)$$

where V_g^t is a global motion vector and $D_1 (0 < D_1 < 1)$ is a damping coefficient

for smooth panning. The structure of the motion correction system is shown in figure 5.14. Figure 5.15 illustrates the effect of the image correction system.

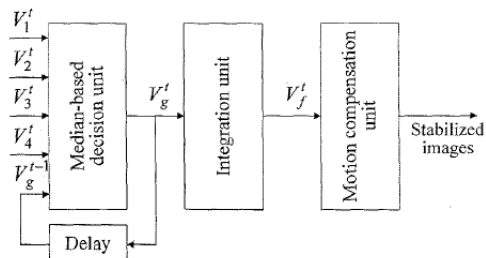


Figure 5.14: Basic structure of the motion correction system



(a) Original image (b) Translated image



(c) Cropped original image (d) cropped and corrected translated image

Figure 5.15: Illustration of image correction

5.4 Integrating it all into the Groundstation

The final task was to integrate all of the components into the existing Groundstation. This is done to provide a complete system, which enables the ARCH UAV to operate in semi-autonomous mode. Details about the Groundstation can be found in appendix C. To add the new functionality a new tab was added to the GUI of the Groundstation called 'Remote Control'. On this tab a user

can gain access to all the functionality described in the previous sections. Figure 5.16 shows all the components integrated into the ARCH Groundstation. Notice the right side where it is possible to turn the remote control system on and off. It is also possible to choose whether or not to use the image stabilizer. At the lower part of the pane it is possible to switch between the different flight-modes. To disengage the pursuit-mode the pilot can choose one of the other flight-modes or simply move the joystick, this will automatically put the helicopter in absolute altitude mode.

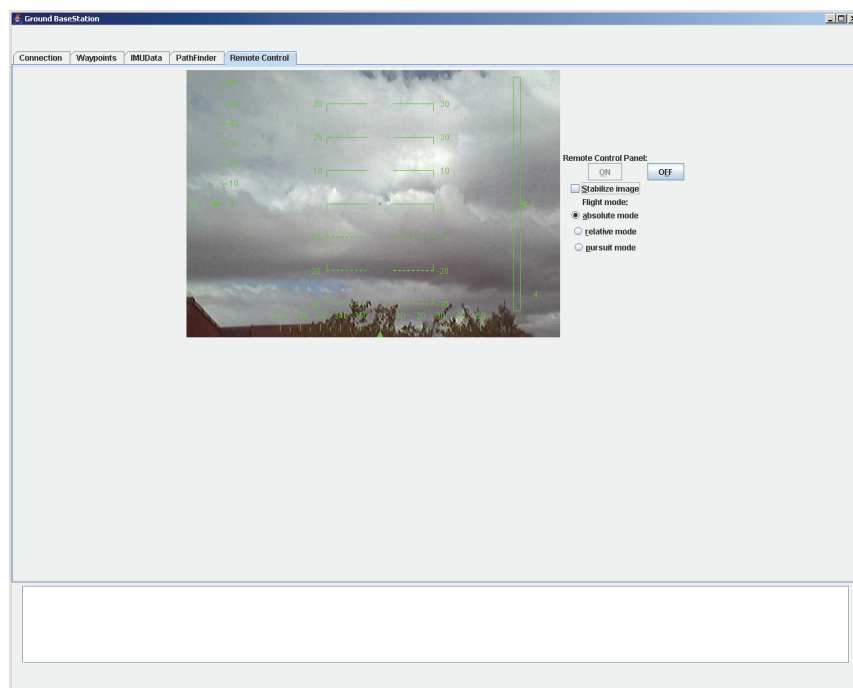


Figure 5.16: The Groundstation GUI

Chapter 6

Discussion

The solutions presented in the previous chapters are not the only solutions which could have been chosen. This chapter discusses the choices that have been made, and points out issues, which have emerged during the work.

6.1 Remote control system

One of the most fundamental questions that should be asked when developing new technology is whether or not the new technology will meet genuine demands. In chapter 2 several scenarios are described, which show the need for UAVs with both autonomous and manual capabilities. It was decided that an operator should not be able to gain direct control of the UAV. An operator may rather control the helicopter in semi-autonomous mode, which is a lot like a fly-by-wire system, which is also described in chapter 2. In semi-autonomous mode the onboard control system keeps the UAV stable in the air. The result is a UAV that anyone can operate. The UAV will naturally have a more limited flight envelope in semi-autonomous mode than if the operator were to have direct control of it. However, having direct control of the UAV, would require great skills from the operator.

In the current version of the remote control system, there are two different flight-modes in which an operator can control the UAV. They differ in the way an operator controls the altitude of the UAV. It is possible to extend the remote control system to support many different flight-modes, which allows an operator to have various degrees of control of the UAV. If for instance there is a scenario, where an operator wishes to maintain a specific altitude to the ground, a flight mode could be developed where the onboard control system maintains the correct altitude as the pilot navigates the UAV around.

The last feature in the current version of the remote control system is the pursuit-mode. In pursuit-mode the remote control system makes the UAV fol-

low an object autonomously. The pursuit-mode is developed under the assumption that the tracking system, which is currently under development within the ARCH project, is already present in the Groundstation. Because the ARCH UAV is a helicopter, which will pitch and roll quite a bit when flying, the camera is used for the tracking system needs to be mounted on a gimball. Without mounting the camera on a gimball, it would be difficult to determine if a registered motion comes from the actual object's movement or the UAV's. For instance, if the tracking system was to detect that the object was moving upwards on the video image. Is this detection based on the actual object gaining altitude, or is it based on the helicopter pitching forward. Certainly, it is possible to use the onboard IMU to compensate for the movements of the UAV. However, the movements of a helicopter are quite significant, for instance if a helicopter flies at full speed forward, the nose points more or less straight to the ground. Therefore, there is a chance that the objects would disappear completely from the video image when the helicopter moves, if the camera was not mounted to a gimball. There is a weakness in the current version of the pursuit-mode. As described in section 5.1, it follows the bounding box received from the tracking system. If the bounding box gets smaller, it moves the UAV closer to the object. However, if it were following an oblong object, for instance another helicopter, and this object turned around its own axis, the bounding box would either become bigger or smaller. This would result in the UAV moving respectively away or towards the object, without the object moving away or towards the UAV. A possible future solution would be to have the tracking system recognize the object it's tracking and determine whether it is the long or short part of it which makes out the bounding box. This would make it possible to avoid this unwanted behaviour. There is still room for improving the pursuit-mode feature. Instead of following the bounding boxes directly, one could for instance use several bounding boxes over a given time frame to calculate the objects trajectory, and follow this trajectory instead. This would perhaps give the UAV smoother movements when following an object. However, since the tracking system is not yet available in the Groundstation, these ideas have been difficult to test and have therefore not been a priority in this thesis.

6.2 Virtual cockpit

As described in section 5.2, the virtual cockpit is necessary to achieve the full benefits of the remote control system. The work on the virtual cockpit has focused on creating a framework, where different HUD designs can be used. The retrieval of the flight information from the UAV and converting this to the proper format has been separated from the actual appearance of the HUD. Therefore, changing the appearance of the HUD is very quick and easy. The main functions to the current HUD design have been to test that the virtual cockpit functions properly and to show the functionality of a HUD. However, designing the best looking or most usable HUD for a pilot has not been a priority

in this work. Separating the HUD's appearance from the "engine", that runs the HUD, has made it possible to easily add several HUD "look and feels", which an operator can interchange between in flight. It is thought that in the future different HUDs can show different mission specific information, and an operator can switch between these HUDs, depending on the task she or he is currently performing.

6.3 Image stabilization

A choice was made to stabilize the video signal using digital image processing techniques, instead of mechanical motion detection techniques - using gyro sensors or fluid prism. With the computational power of today's computers, it is possible to perform real-time image stabilization, using only digital image processing techniques, which is preferred because, among other things, there is no need for additional sensors. However, there are already gyro sensors present at the UAV, which could have been used to perform motion detection. The mechanical techniques require of course less computational power, which would make more resources on the Groundstation available for other tasks. If image stabilization, based on mechanical motion detection techniques were to be used, the stabilization would most likely have to be done onboard the UAV, because it would be very difficult to synchronize the video signal transmitted from the UAV, and the output from the gyros transmitted from the UAV to the ground. The fully digital approach was chosen in the end, because a lot more resources are available on the ground than on the UAV. So, everything that does not need to be on the UAV, should not be placed there. And the gyros, which are crucial to the onboard control system, were not utilized.

In chapter 4 different digital image processing techniques that could be used for motion estimation were described. Based on this, two different motion estimators were developed and described in chapter 5. These two motion estimators were tested to determine which one was the most suitable for this task. Finally, the one using the phase correlation technique was chosen. It was chosen mainly because it showed excellent robustness against noise and it was somewhat more effective than the motion estimator using a bit-plane matching technique. When performing time-critical operations such as real-time operations, there is always a compromise between the amount of work and the time it takes to perform it. The motion estimator, developed in this thesis, estimates motion from four sub-images each with a block-size of 32×32 . It is important that the blocks are large enough to group pixels with similar motion, but they should also be small enough to separate pixels with different motion and multiple-object movement. With block-sizes smaller than 32×32 the motion estimator were no longer able to estimate the correct motion. Therefore, 32×32 was a lower bound for the block-size. It would perhaps have been preferable with a larger block-size than 32×32 , or more than four sub-images in each frame. However, this was not possible to perform in real-time with the motion estimator developed in this thesis.

Like the rest of the Groundstation, the image stabilization system is written in the programming language Java. This is done from the desire to keep the entire Groundstation as platform-neutral as possible. However, Java is not famous for its performance, therefore larger block-sizes or more sub-images could be possible by implementing the image stabilization system in a language with higher performance capabilities, such as C/C++. But this would result that the Groundstation no longer would be able to run on an arbitrary platform. In theory it is possible to write C/C++ programs which can run on arbitrary platforms. But is necessary to recompile it on every platform, and only use programming libraries which are present at every possible platform. This would make changing the platform a cumbersome process and the Groundstation would no longer be truly platform-neutral.

Other options to obtain higher performance are also possible, for instance: the Fourier Transform could be implemented in hardware or utilizing the fact that estimating the local motion vectors of each frame are ideal for parallel processing. However, optimizing the image stabilization system outside the boundaries of Java was considered to be outside the scope of this thesis.

After the local motion vectors from a frame have been estimated, a global motion vector must be calculated. Several algorithms already exist, which are described in [Liu et al. 1993, Musmann et al. 1985, Okada. 1996, Kinugase et al. 1990]. However, most of these algorithms are quite complex. In this thesis a simple and robust decision algorithm for determining the global motion vector is proposed. The median-based correction scheme proposed in this thesis has proved to be quite robust to irregular conditions such as moving objects.

6.4 Test results

A couple of test runs were made on the ARCH UAV itself. However, it seems that the inner container is still too exposed to the vibrations of the helicopter. The result of this is that the onboard computer is not able to write to the hard disk in flight. It would seem that more work needs to be done in softening the vibrations on the inner container. Placing it inside a layer of bubble-wrap was apparently not enough. Another option which is currently considered is using something that is less vulnerable to mechanical influence than a hard disk to store the data on.

Because the ARCH project is at an early stage of development and the ARCH UAV is not yet fully operative, the possibilities to test the different components developed in this thesis have been limited. The result of this is that both the RCS and the HUD in the virtual cockpit have only been tested on simulated data. Even though the tests in the simulator have been very promising, further testing on the actual UAV is necessary. This is especially true with the pursuit-mode feature in the RCS, which needs to be combined with the tracking system

- currently under development within the ARCH project - before it can be properly tested.

The digital image stabilization system was mostly tested with images that were translated to one another with a known distance and direction. This was done to test that the system estimated the correct motion, and extracted the correct sub-block from the images. The DIS was of course also tested on video streams, but here it was difficult to give an exact measure of the quality of the stabilization. However, the stabilized video streams did appear significantly smoother and more pleasant to look at.

Chapter 7

Conclusion and future work

The work presented in this thesis consists of three main parts:

- A remote control system which enables operators to control the ARCH UAV in semi-autonomous mode has been proposed and developed. The RCS includes a set of different flight-modes, with the possibility to expand. These different flight-modes gives an operator various ways of controlling the UAV, and the operator may choose the flight-mode most suitable for the current task. One of the flight-modes separates itself from the others, i.e. that the operator no longer controls the UAV. Instead the RCS makes the UAV follow an object autonomously. This flight-mode is called pursuit-mode and it makes use of a tracking system currently under development within the ARCH project to enable it to autonomously follow objects.
- A virtual cockpit which keeps the pilots “eyes in the sky” has been proposed and developed. The virtual cockpit displays the video image, sent from the camera mounted on the UAV’s nose, to the ground. On top of this visual information a HUD is drawn, containing the primary flight information. The look of the HUD itself has been based on a best-of-breed (BOB) design. The appearance of the HUD has been separated from the rest of the HUD’s generator, which makes it easy to change the look of the HUD if this should be desired in the future. It is even possible to have several HUD designs which the operator can switch between in flight.
- A digital image stabilization system has been proposed and developed. The task of the DIS is to stabilize the video image transmitted from the UAV to the Groundstation. The DIS uses phase correlation to estimate the motion between two subsequent frames. It then uses this and prior information to determine whether or not this motion is caused by shaking or intentional panning. Finally it extracts and displays the correct sub-block from the current frame.

These three parts are all combined and integrated into the Groundstation to provide a fully functional platform for an operator to control the UAV. Because the ARCH UAV itself is not yet operational, this platform has not been tested to the desired degree.

The following issues have been identified for further investigation;

- Practical tests with the actual UAV will have to be performed. This also includes more testing with the RCS's pursuit-mode after the tracking system is integrated into the Groundstation.
- Designing the optimal look for the HUD. Either by studying other HUD designs or designing one from scratch with the guidance from future users of this system. Because the needs and views from future users of this system might differ from that of actual pilots.
- Improving the performance of the DIS. This involves both improving the current implementation and perhaps looking outside the boundaries of Java.
- More functionality can be added, both with different HUD designs and more flight-modes, to accommodate more flight mission requirements.

Appendix A

Glossary

ABS	Anti-lock Braking System. A computer system in cars that prevents the driver from locking the brakes when braking hard
API	Application Program Interface. It's the specific methods prescribed by a computer operating system or an application program by which a programmer writing an application program can make requests of the operating system from another application
ARCH	Autonomous Remote Controlled Helicopter. Ongoing research project at IDI, NTNU.
ASC	Anti-Spin-Control. A computer system in cars that prevents the car-wheels from spinning
BMA	Block Matching Algorithm. Algorithm used to estimate the motion between two images
C++	C++ is an object-oriented programming language. C++ is a superset of the language C.
CCD	Charge Coupled Device. CCD is a light-sensitive integrated circuit that stores and displays the data for an image in such a way that each pixel (picture element) in the image is converted into an electrical charge, the intensity of which is related to a color in the color spectrum.
DIS	Digital Stabilization System. A system for stabilizing a video-image based purely on digital image processing techniques
DOF	Degree Of Freedom. The number of directions in which an object is able to move

EFIS	Electronic Flight Instrument System. A system that provides the instrumentation for attitude, engine and position
ESP	Electronic Stabilisation Programme. An active safety feature in cars which can help to improve the occupants safety by registering and helping to correct oversteer, understeer and loss of stability in the vehicle.
FLIR	Forward Looking Infra-Red sensor which captures the visual information with an infra-red sensor.
fly-by-wire	A term used, originally in aircrafts, when the operator no longer has direct control of the aircraft. Instead, the operators' inputs are converted into the correct settings for the engine and controls
GA	Genetic Algorithms. GAs are simple and robust methods for optimization and search
gimbals	An appliance for permitting a body to incline freely in all directions
GPL	General Public License is intended to guarantee your freedom to share and change free software
GPS	Global Positioning System. System used to calculate the position
GUI	Graphical User Interface. Denote a graphical rather than a purely textual user interface to a computer
HMD	Helmet Mounted Display. Displays information as a graphical overlay inside the helmet of a pilot
HUD	Head Up Display. Displays information as a graphical overlay in the pilot's line of sight
IMU	Inertial Measurement Unit. A unit that measures rotation and acceleration about/along the x,y and z axis
INS	Inertial Navigation System. System that calculates orientation and position from gyro and accelerometer readings
Java	Java is an object oriented programming language, which is platform-neutral
LAN	Local Area Network is a group of computers and associated devices that share a common communications line or wireless link and typically share the resources of a single processor or server within a small geographical area
LM	The Levenberg-Marquardt algorithm is a non-linear optimization algorithm

MAD	Mean Absolute Difference is a correlation criterion
MMI	Man-Machine Interaction. The subject of how man interacts with machine
MP	Morphological Pyramid is a data structure which represents images with decreasing resolution
MPIR	Morphological Pyramid Image Registration is an algorithm that employs feature-based methods in order to perform image registration
MSE	Mean Square Error is a correlation criterion
OCU	Operator Control Unit is the collective term for all the components, software and hardware, that combined enables an operator to control the UAV
RCS	Remote Control System, enables a pilot to control the UAV from the Groundstation
RPV	Remote Piloted Vehicles. Vehicles guided without an onboard crew
STL	Standard Template Library. Library often used when programming in C/C++
UAV	Unmanned Aerial Vehicle. Powered aerial vehicles sustained in flight and guided without an onboard crew
USB	Universal Serial Bus is a plug-and-play interface between a computer and add-on devices
VTOL	Vertical Take Off and Landing. Capability of specific aerial vehicles, such as helicopters.

Appendix B

ARCH API and communication protocol specifications

The ARCH API and the ARCH server (which parses messages using the ARCH communication protocol) has been written in the C++ programming language as a layer above the existing `autopilot` software system.

The `autopilot` software system is available under GPL at [Autopilot Web].

This appendix provides an overview of the interface functions used in the ARCH API, the messages used in the communication protocol and the unit parameters which are currently supported.

B.1 ARCH API interface functions

The detailed description of the API interface functions in this section is provided as a list of C++ interface function signatures. Each interface function is accompanied with a description of its functionality and intended usage.

`ARCH_interface(ARCH_client* client)` - This is the constructor interface function, used for creating a new interface objects upon which the other interface functions may be called. Implicitly assumes that the ARCH server is running on the local machine. Takes a pointer to an `ARCH_client` object as argument, because the `ARCH_client` class defines an interface for callback functions used to push data from the ARCH server to the client.

`ARCH_interface(ARCH_client* client, char* remote_address)` - Has the same effect as the constructor described above, except that this one is used in cases where the interface object is used to connect to an ARCH server on a remote machine, specified by the `remote_address` parameter.

`void ARCH_unit_data(const int unit_id, vector<ARCH_arg>& data_list)` - This function can be used in cases where it is desirable to pull data from the server instead of waiting until they are pushed upon the client object. The argument `unit_id` specifies the helicopter unit from which data is to be fetched. The argument `data_list` is a reference to an STL¹ `vector`. After the function call has finished, this vector will have been filled with data from the specified helicopter unit.

`void ARCH_unit_data(const int unit_id, const int arg_id, double& arg_value)` - This function has the same functionality as the one described above, except that it is used for fetching data from only one of the parameters of the specified helicopter unit. For example, this function could be used in cases where only, say, the altitude was needed, and not all of the parameters from the INS/GPS unit.

`bool ARCH_client_subscribe(const int unit_id, vector<int> param_ids)` - This function is used to subscribe to parameters of helicopter units. The argument `unit_id` specifies the helicopter unit which is to be subscribed to. The argument `param_ids` is a `vector` containing the ids of the specific parameters of the subscription. For example, a subscription to the GPS coordinates of the helicopter would specify the id of the INS/GPS unit as the first argument, and the next argument would be a `vector` containing the ids of the parameters northing, easting and down, respectively. Returns `true` if the operation succeeds.

`bool ARCH_client_unsubscribe(const int unit_id, vector<int> param_id)` - This function has the exact opposite functionality of the one above - it is used to unsubscribe to helicopter units. The arguments of the function has the same meaning as the arguments of the function described above. Returns `true` if the operation succeeds.

`bool ARCH_client_begin_task(const int unit_id)` - This function is used to specify the beginning of a new task on a helicopter unit, specified by the argument `unit_id`. Note that if a previous task was defined by the client on this unit, but not yet requested, this previous task will be deleted as a result of a call to this function. Returns `true` if the operation succeeds.

`bool ARCH_client_end_task(const int unit_id)` - This function is used to specify the end of a task on a helicopter unit, specified by the argument `unit_id`. Returns `true` if the operation succeeds.

¹Standard Template Library

- `bool ARCH_client_request_task(const int unit_id)` - This function is used to request a registered task to be performed on the helicopter. The task will be performed immediately if possible. If not, the task will be queued up and performed when possible. The value *true* is returned if the operation succeeds, but not until the entire task has been performed. Note that since the call is blocking, clients might want to call this function in a separate thread to enable execution to continue while waiting for the response. For example, a task such as moving to a new waypoint may take a considerable amount of time. If *false* is returned, either the helicopter was not able to complete the specified task, the task has been aborted by a client with higher privileges, or no task to be performed has been registered.
- `bool ARCH_abort_task(const int unit_id)` - This function is used to abort the task currently being performed by the helicopter. Clients may abort their own tasks and the tasks of clients with lower privileges. Returns *true* if the operation succeeds. Note that *false* will also be returned as a result of the corresponding request-call to the client that requested the task.
- `bool ARCH_suspend_task(const int unit_id)` - This function is used to suspend the task currently being performed by the helicopter. The task will not be deleted, and the request-call corresponding to the task that is being suspended will still be blocked (no value will be returned). Clients may only suspend their own tasks or tasks of clients with lower privileges. Returns *true* (to the calling client) if the operation succeeds.
- `bool ARCH_resume_task(const int unit_id)` - This function causes the helicopter to resume the last task that was suspended. If no task has been suspended, or if the helicopter for some reason is not able to resume the task, *false* will be returned. Otherwise, *true* will be returned.
- `bool ARCH_client_set_parameters(const int unit_id, const vector<ARCH_arg> params)` - This function is used to set the parameters of a task to be performed on the helicopter. Typically, a sequence of calls to this functions will be used to specify a task. For example, if the task to be performed is to move along a flight path specified by a sequence of waypoints, this function should be called as many times as there are waypoints in the flight path, with each function call corresponding to one waypoint. Returns *true* if the helicopter is able to accept the parameters.
- `bool ARCH_current_task_parameters_count(const int unit_id, int& count)` - This function is used to get the number of parameter updates in the task currently being performed on the helicopter. The argument `unit_id` specifies the unit on which the task is being performed, and the argument `count` is a reference to an integer. If the result of the function call is *true* - that is, if a task is currently being performed on the

specified unit - the argument `count` will have been set to the number of parameter updates in the current task.

`bool ARCH_current_task_parameters(const int unit_id, const int index, vector<ARCH_arg>& arguments)` - This function is used to get one of the parameter updates of the task currently being performed on the unit specified by the argument `unit_id`. The argument `index` specifies the index of the parameter in the sequence of parameter updates which constitutes the task. The function call returns `true` if there is a task currently being performed on the specified unit and index is in the range $[0, n)$, where n is the number of parameter updates. A typical usage scenario for this function would be to first use the function described above to get the number of updates in the current task, and then use this number as a limit in a loop that continuously calls this function to investigate each of the individual updates.

`bool ARCH_current_task_parameters_invalidate(const int unit_id, const int index)` - This function can be used to invalidate (delete) a parameter update from the task currently being performed on the unit specified by the argument `unit_id`. The function will return `true` if the invalidation of the parameter update was successful.

Typically, the three functions described last will be used by a client to traverse all of the parameter updates of a task to be performed on a helicopter unit. The *invalidate* function can be used to remove a parameter update from the sequence. An example scenario could be an onboard client with micro-navigation capabilities. Such a client should not only have the ability to suspend and resume the task of a lower privileged client, but also remove waypoints from the suspended task that are not valid any more after the micro-navigation task has been completed (steering around an unknown obstacle, for example).

The interface also includes the class `ARCH_client`. This class contains a pure virtual callback function. Clients using the API should subclass this class and provide an overloaded implementation of the callback function. The functionality of the callback function is now described.

`virtual void ARCH_unit_data(int unit_id, vector<ARCH_arg> data)` - This function is called on the client when new data is available from the unit specified by the argument `unit_id`. The function will only be called if the client is subscribing to parameters of the unit. The argument `data` is a `vector` containing values for unit parameters on which the client is subscribing. For example, if the client is subscribing to GPS coordinates from the INS/GPS unit, this function will be called each time the INS/GPS unit provides updated coordinate estimates. The first argument will be the id of the GPS/INS unit, while the second argument will be a vector containing three `ARCH_arg` elements, each of which contains an id and a value.

The data type `ARCH_arg`, used in many of the above described interface functions, is simply a container that holds a parameter id and a parameter value. For a more detailed insight into this data type, readers are encouraged to inspect the source code of the ARCH server and API themselves.

B.2 ARCH protocol messages

This appendix gives a description of the ARCH communication protocol. The ARCH protocol must be used by remote clients, which cannot use the ARCH API presented in the last section (though remote clients written in C++ may use the ARCH API to communicate with the server directly instead of implementing code for parsing and generating protocol messages).

B.2.1 Description

The ARCH API presented in appendix B.1 is designed in such a way that the ARCH communication protocol can be deduced nearly directly from the interface function signatures. This way, understanding and implementing the ARCH communication protocol is greatly eased through the similarity with the API. The main ideas behind the protocol are:

- Communication between the ARCH server and the remote client is achieved through textual *messages* sent over the communication link (TCP/IP). Messages are sent as sequences of bytes/characters, terminated by the special character `\n`.
- Each message is composed of a number of *words*. Each word in a message is terminated with the token `%`. In the current implementation, all of the helicopter units convert words into numbers - integers or doubles. However, future helicopter units may choose to treat the words as raw byte data.
- The first word of a message is a special code identifying the operation which is to be performed. Each interface function in the API has such a code associated with it. There are also codes for the interface functions' return values. A specification of these codes can be found in the next section.
- All of the word succeeding the first one corresponds to parameters of the interface function. Interface functions which include a vector of either integers or `ARCH_arg` objects take this vector as their last argument. This enables the protocol messages which correspond to these interface functions to treat the n last words of a message as elements of a vector, where n is the total number of words in the message minus the number of arguments preceding the vector. For vectors which consist of `ARCH_arg` objects,

n is assumed to be an even number. For a pair of words, the first one will be treated as an integer corresponding to the id of the parameter. The interpretation of the last one will depend on the helicopter unit. The IMU and INS/GPS units treat all of these values as doubles.

- In cases where arguments of an interface function are references to variables, i.e. variables that are updated by the function if the function call succeeds, words with values corresponding to the values of these variables will be sent as a message from the server to the client. Note that in all cases where references to variables are taken as a function argument, these reference variables appear as the last arguments of the function. If a function has n arguments, where p of them are parameters acting directly upon the server and q of them are reference variables to be filled with values from the server, the corresponding protocol message from the client to the server will contain p words in addition to the first identification word. If the operation on the server succeeds, the client will receive a message from the server consisting of q words corresponding to the reference variables of the interface function.

Some examples will make the ideas described above clearer. The first example shows a message where the client is requesting a task on the IMU unit:

$$302\%1\%$$

The first word, 302 , is the code corresponding to the interface function `ARCH_client_request_task` of the API. Message identification codes are always interpreted as integers. The second word, 0 , is also interpreted as an integer in this case, because the first argument of the corresponding interface function is an integer. As one can see, this is the id of the helicopter unit upon which a task is requested. Since this function call returns a boolean value, a message should be sent from the server to the client indicating if the request was successful or not. Thus, the following message will be sent as a response from the server to the remote client:

$$1000\%302\%1\%$$

The identification word is in this case 1000 , a special id reserved from boolean responses. The first argument, 302 , indicates that this is a response to a message type of id 302 . The last argument, 1 , indicates that the boolean result of the operation on the server was *true*. If the result was to be *false*, this would have been indicated by 0 being included as the last argument instead of 1 . The next example shows a message corresponding to an interface function where a vector is one of the arguments:

$$200\%0\%0\%476.32\%1\%300.11\%2\%-40.0\%$$

The identification word, *200*, corresponds to the interface function `ARCH_client_set_parameters`. The first argument is an integer specifying the id of the helicopter unit that the client is specifying parameters for. In this case it is *0*, which is the id of the INS/GPS unit. The succeeding sequence of words is treated as the contents of the vector. Since this vector contains `ARCH_arg` objects, an even number of words is expected. As one can see, the number of words is six in this case, so the message is a valid one. Pairs of words are, in this case, treated as pairs of integers and doubles, since the INS/GPS unit is expecting parameter ids as integers and parameter values as doubles. The parameter ids, *0*, *1* and *2*, corresponds to the desired coordinates of the helicopter. The values, *476.32*, *300.11* and *-40.0*, corresponds to the values of the desired coordinates. The last example of this section illustrates a sequence of messages where the server responds to the client's request by sending parameter values back to the client:

500%0% - client to server

500%17% - server to client

1000%500%1 - server to client

The first message is sent from the client to the server. The identification word, *500*, corresponds to the interface function `ARCH_current_task_parameters_count`. As one can see from the presentation of the API in section B.1, the first argument of this function is the id of the helicopter unit from which the client is trying to fetch the number of task parameters. This parameter is included as a word in the message from the client to the server. The second parameter of the interface function, however, is a reference variable intended to be updated with a value from the server. Therefore, a message with the same identification word as the first one is sent as a response from the server to the client. The first word of this message corresponds to the first argument of the interface function which is a reference parameter. In addition, the client also received a message indicating that the operation on the server succeeded. If the operation on the server did not succeed, the client would only have received a message indicating that the operation had failed (that is, a message with identification word *1000*).

B.2.2 Protocol messages

This section specifies the structure of the messages used in the API. Table B.1 specifies the messages used between clients and the ARCH server. Table B.2 specifies the messages used between the ARCH server and clients. For both of these tables, references to the corresponding interface functions in the API are given. The code for the protocol message is also given, together with a representation of the syntax of the succeeding arguments. In this representation, the operator `+` is used to indicate one or more repetitions of its operand. The

term *bytes* indicates an array of bytes. In the current version of the system, all of the units treat all of their parameter values as doubles. The meaning of the arguments of the different protocol messages can be inferred directly from the description of the corresponding interface function in section B.1. Therefore, they are not repeated here.

Table B.1: Messages from client to server

Interface function	Code	Arguments
ARCH_unit_data	2	<i>int%</i>
ARCH_client_subscribe	100	<i>int%(int%)+</i>
ARCH_client_unsubscribe	101	<i>int%(int%)+</i>
ARCH_client_set_parameters	200	<i>int%(int%bytes%)+</i>
ARCH_client_begin_task	300	<i>int%</i>
ARCH_client_end_task	301	<i>int%</i>
ARCH_client_request_task	302	<i>int%</i>
ARCH_abort_task	400	<i>int%</i>
ARCH_suspend_task	401	<i>int%</i>
ARCH_resume_task	402	<i>int%</i>
ARCH_current_task_parameters_count	500	<i>int%</i>
ARCH_current_task_parameters	501	<i>int%</i>
ARCH_current_task_parameters_invalidate	502	<i>int%int%</i>

Table B.2: Messages from server to client

Interface function	Code	Arguments	Comment
ARCH_unit_data	1	<i>int%(int%bytes%)+</i>	Sent to subscribed clients
ARCH_unit_data	2	<i>(int%bytes%)+</i>	Response to client
ARCH_current_task_parameters_count	500	<i>int%</i>	Response to client
ARCH_current_task_parameters	501	<i>(int%bytes%)+</i>	Response to client

Table B.3 presents protocol messages which does not have any corresponding interface function in the API.

B.3 Unit parameters

This section presents the unit parameters defined for the helicopter unit interface which has currently been implemented. For each parameter, flags are given to

Table B.3: Special messages

Description	Code	Arguments	Comment
Boolean return value	1000	<i>int%(0/1)%</i>	First argument is code of request-message

indicate if it is *settable* and *readable*. A settable parameter is a parameter that may be set by a client through a task on the unit. A readable parameter is a parameter that may be read and subscribed to by clients. Flags surrounded by parentheses indicate that the feature is planned but has not been implemented in the current version of the system. Table B.4 shows the parameters defined for the INS/GPS unit. The INS/GPS unit has the unit id 0.

Table B.4: Parameters for the INS/GPS unit

ID	Parameter	Description	Data type	Settable	Readable
0	y	Position	double	Y	Y
1	x	Position	double	Y	Y
2	z	Position	double	Y	Y
3	vy	Velocity	double		Y
4	vx	Velocity	double		Y
5	vz	Velocity	double		Y
6	p	Rotational velocity	double		Y
7	q	Rotational velocity	double		Y
8	r	Rotational velocity	double		Y
9	phi	Attitude angle	double	(Y)	Y
10	theta	Attitude angle	double	Y	Y
11	psi	Attitude angle	double	(Y)	Y

Appendix C

ARCH GroundStation

The purpose of the GroundStation is to enable people to monitor and control the helicopter's actions. It is meant to provide an interface to UAVs, where a user can plot flight paths and designate areas for UAVs to cover. The GroundStation is connected to the helicopter through a wireless LAN. Although the helicopter could function well on its own, it is assumed that the helicopter will always be connected to the GroundStation. This means that the reach of the wireless LAN, or the lack thereof, implies a serious restriction on the reach of the helicopter. This is something that will have to be changed or fixed before the ARCH helicopter can perform tasks of significance beyond research and development. In the next section a description of what can be done from the Groundstation and how to do it is given. Thereafter follows a more technical description of how the Groundstation is constructed and why it's constructed the way it is.

C.1 Functional description

The GroundStation is the gateway to communicating with one or more UAVs. Its purpose is to allow users to easily and intuitively monitor and control the helicopter. In the Graphical User Interface (GUI) the functionality is divided into several tabs. All information received from the helicopter will be printed in the status window, on tab 1 shown in figure C.1. After connecting to a helicopter, a user is able to navigate through the different tabs to perform different tasks. The following subsections will describe the different tabs, and the functionality provided in them.

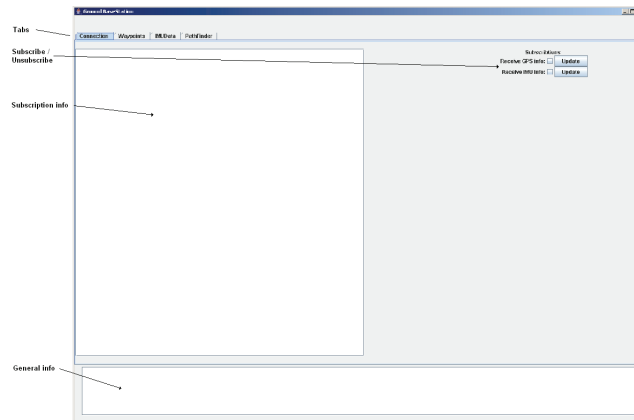


Figure C.1: Connection tab

C.1.1 Connection tab

The connection tab is the first thing that meets the user when the Groundstation is started. In this tab the users are able to subscribe or unsubscribe on different information available on the UAV. If, for example, the user wishes to receive information on the whereabouts of the UAV, he/she can subscribe to the GPS coordinates of the UAV. This subscription will make the UAV send its GPS coordinates to the GroundStation regularly. The GroundStation will, in turn, print this in the subscription info panel shown in figure C.1. In later versions this could easily be converted to plotting the UAV's position on a map, instead of only printing it in the subscription panel. This is, however, not yet implemented in the GroundStation. The general info panel shown in figure C.1 prints general info from the UAV and information about what the GroundStation is doing. This panel is available independently of which tab the user is operating.

C.1.2 Waypoints tab

In the waypoints tab, the user can either type in the waypoints by hand in the “plot waypoints” panel, or open and upload a text file containing the desired waypoints in the “Read / Upload Waypoints” panel. This is shown in figure C.2. The text-file containing the waypoints must be in the following format:

```

North East Down
North East Down
. . .
North East Down

```



Figure C.2: Waypoints tab

The user should also be able to designate waypoints by clicking on the map in figure C.2, but this is not implemented yet. The waypoints tab is a low-level flight control panel, where a user can control the UAV's movement by telling it where to fly. The UAV will move in a straight line between the waypoints given and in the altitude given, independent of the terrain. This functionality is for testing purposes only, and does not serve any purpose outside research and development. For a novice user, it would be more natural to use the functionality provided in the Pathfinder tab described in section C.1.4.

C.1.3 IMUData tab

The IMUData tab is, at present, meant as an example tab showing how one can create a new tab for each new sensor or functionality one puts on the UAV. In the IMUData tab, the users are able to set specific values on the Inertial Measurement Unit (IMU).

C.1.4 PathFinder tab

In the PathFinder tab, the user is able to make the GroundStation calculate the optimal path between two positions on the map by pressing the "calculate path" button shown in figure C.3. The input-box in figure C.4 is then presented to the user. In the current version, the only single path-planner available in the GroundStation is the shortest path planner. In later versions, the users also will be able to choose other single UAV path planners.

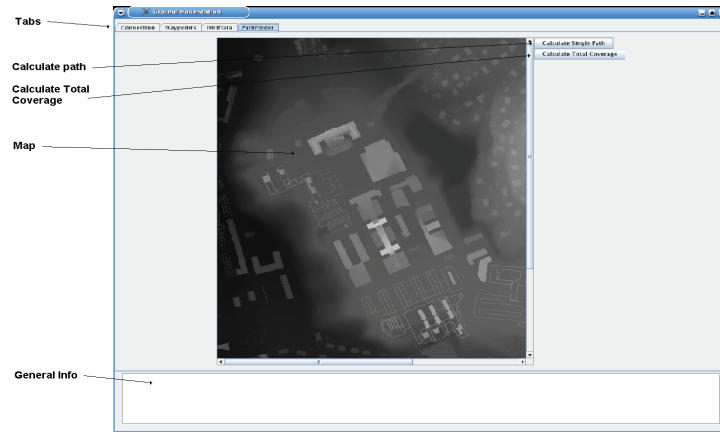


Figure C.3: PathFinder tab

The user is also able to make the GroundStation calculate a total coverage path for one or more UAVs that will enable the UAVs to cover the entire area with some sort of remote sensing devices, e.g cameras. This can be achieved by pressing the “Calculate total coverage” button. The input-box in figure C.4 is then presented to the user. Again, many of the options on the input-box are for testing purposes, allowing the ones creating the “total coverage” methods to experiment with different variables. The average user would not have to be concerned with many of these variables.

It is also thought that the user will be able to designate waypoints by clicking on the map in figure C.3 or drag a rectangle across the map to designate an area to be covered. However, this is not implemented in the current version.

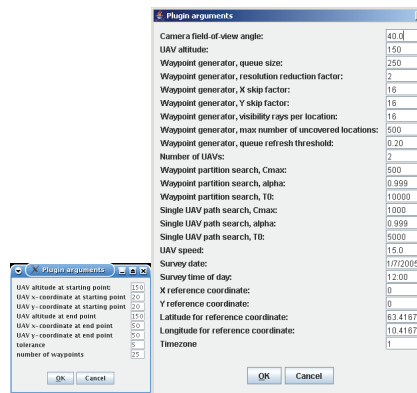


Figure C.4: Input-boxes for 'calculate path' and 'calculate total coverage'

C.2 Architectural description

The architecture of the Groundstation is divided into three layers: communication-, logic- and GUI layer, as shown in figure C.5. The communication layer uses the ARCH protocol, described in, appendix B, and enables the layer above to communicate with the helicopter without worrying about loss of packages or other underlying problems. Because the ARCH protocol is designed to support more advanced onboard clients, only selected features of the ARCH protocol are supported by the GroundStation. The features supported in the Groundstation are mainly the ones that read parameters from a UAV and update parameters to make it perform tasks.

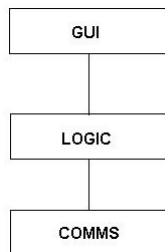


Figure C.5: The GroundStations' architecture (high level).

The GroundStation is used in research and development of the ARCH UAV. This requires the GroundStation to have a high degree of modifiability because of the constant need to add new features and options. This makes it important to be able to control the time and cost to implement, test, and deploy changes. One way to do this is to reduce the number of modules that are directly affected by a change. Although there is not necessarily a precise relationship between the number of modules affected by a set of changes and the cost of implementing these changes, restricting modification to a small set of modules will generally reduce the cost. Another thing is to prevent ripple effects. A ripple effect from a modification is the necessity of making changes to modules not directly affected by it. For instance, if module A is changed to accomplish a particular modification, then module B is changed only because of the change to module A. B has to be modified because it depends, in some sense, on A. To avoid this, one can deploy tactics like *hiding information* where the goal is to isolate changes within one module, and prevent changes from propagating to others, *maintaining existing interfaces* where one creates abstract interfaces that mask variations, and *restricting communication paths* where one reduces the modules with which a given module shares data. To accommodate this need, the architecture is highly modular with few couplings between the modules, which makes it fast and easy to add or remove modules. As can be seen from figure C.6, there is only one link between each module, none of the other modules know of each other. This

makes it quick and easy to change, add or replace modules.

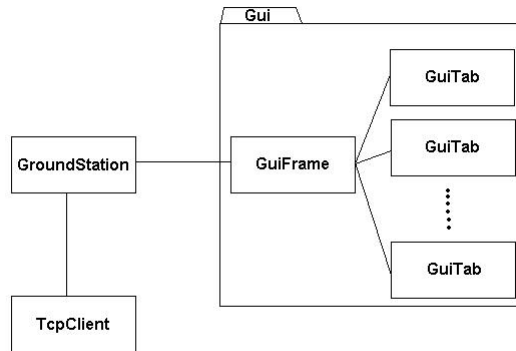


Figure C.6: The GroundStations' architecture (low level)

When creating any application that is to be used by users, one must always be concerned with usability. Usability is concerned with how easy it is for users to accomplish a desired task and the kind of support the system provides to the user. Usability is enhanced by, among other things, giving the user feedback as to what the system is doing, and by providing the user with the ability to issue usability-based commands such as cancel and undo, to support the user in either error-correction or more efficient operations. Researchers in human-computer interaction have used the terms “user-initiative”, “system-initiative” and “mixed-initiative” to describe which of the human-computer pairs takes the initiative in performing certain actions and how the interaction proceeds. For example, when issuing a “calculate path” command, the user issues the command “user-initiative”, and the system responds. During the “calculate path”, however, the system may put up a progress indicator, “system-initiative”. Thus “calculate path” demonstrates “mixed initiative”.

To use the “calculate path” example again: When a user issues a “calculate path” command, the system must be listening for it, in this way there is the responsibility to have a constant listener that is not blocked by the actions of whatever else is going on. When the system takes the initiative, it must rely on some information, a model, about the user, the task being undertaken by the user or the system state itself. In the GroundStation, progress bars are used on actions that takes a perceptible amount of time to make the system appear more responsive. We have also made it possible to abort any action that deals with the UAV at any time. Since the GroundStation still is used mainly for research and development, much work has not been put down in the current GUI. However, the functionality has been separated into natural groupings to

make it intuitive and effective for a user to operate.

Bibliography

- [AHP Web] Autonomous Helicopter Project. Available on www at <http://roboticflight.org/>. Last visited: 14.06.05.
- [Airforce Web] Airforce-technology. Available on www at <http://www.airforce-technology.com/contractors/cockpit/vsi/vsi2.html>. Last visited: 14.06.05.
- [Alliney et al. 1986] Alliney, S. and Morandi, C. A. "Digital image registration using projections". *IEEE Trans. PAMI-8*, 2, pp 222-233. 1986.
- [Angel 2000] Angel, Edvard. Interactive computer graphics : a top-down approach with OpenGL. Addison Wesley Longman, Inc, 2 edition, 2000.
- [Autopilot Web] Autopilot: Do it yourself UAV. Available on www at <http://autopilot.sourceforge.net/>. Last visited 04.05.05.
- [Barnea et al. 1972] Barnea, D.I. and Silverman, H.F. "A class of algorithms for fast digital registration", *IEEE Trans. Comput.*, vol. C-21, pp. 179-186, 1972.
- [Brown et al. 1992] Brown, L.G. "A survey of image registration techniques", *ACM Computing Surveys*, vol. 24, no. 4, pp. 325-376, Dec. 1992.
- [Collinson 2002] Collinson, R.P.. *Introduction to Avionics Systems*, chapter 2. Second edition. Springer. December, 2002.
- [Condensation Web] The Condensation Algorithm. Available on www at http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/ISARD1/condensation.html. Last visited 04.06.05.
- [Conj Web] Mathworld - Complex Conjugate. Available on www at <http://mathworld.wolfram.com/ComplexConjugate.html>. Last visited 16.05.05.

- [De Castro et al. 1987] De Castro, E. and Morandi, C. "Registration of translated and rotated images using finite Fourier transforms", *IEEE Trans. Pattern Anal. Machine Intell.* vol. PAMI-95, pp. 700-703. Sept. 1987.
- [Doherty 2004] Doherty, Patrick. *Advanced Research with Autonomous Unmanned Vehicles*. Proceedings on the 9th International Conference on Principles of Knowledge Representation and Reasoning. 2004.
- [FAS Web] FAS Military Analysis Network - Smart Weapons - BGM - 109 Tomahawk. Available on www at <http://www.fas.org/man/dod-101/sys/smart/bgm-109.htm>. Last visited: 08.05.05.
- [FAS3 Web] FAS Military Analysis Network - B-2 Spirit. Available on www at http://www.fas.org/nuke/guide/usa/bomber/b-2_gallery.htm. Last visited: 14.06.05.
- [FAS2 Web] FAS Military Analysis Network - F/A-18 Hornet. Available on www at <http://www.fas.org/man/dod-101/sys/ac/f-18.htm>. Last visited: 14.06.05.
- [F-16 Web] F-16 - The ultimate F-16 reference. Available on www at www.f-16.net. Last visited: 14.06.05.
- [FFT Web] Mathworld - Fast Fourier Transform. Available on www at <http://mathworld.wolfram.com/FastFourierTransform.html>. Last visited: 16.05.05.
- [Gharavi et al. 1990] Gharavi, H. and Mills, M. "Blockmatching motion estimation algorithms-new results", *IEEE Trans. on Circuits and Systems*, vol 37, no 5, pp. 649-651, May 1990.
- [Gimbals Web] Definision of *Gimbals* from the Free Dictionary.com. Available on www at <http://www.thefreedictionary.com/gimbals>. Last visited: 01.06.05.
- [Goldburg. 1989] Goldburg, D. E. "Genetic Algorithms in Search: optimization and machine learning", *Reading, Mass.* Addison-Wesley,1989.
- [Holland. 1975] Holland, J. H. "Adaptation in Natural and Artificial System", *University of Michigan Press*, Ann Arbor, 1975.

- [Jinput Web] Jinput: Java Input API Project. Available on www at <https://jinput.dev.java.net/>. Last visited 05.05.05.
- [JMF Web] JMF: Java Media Framework API. Available on www at <http://java.sun.com/products/java-media/jmf/>. Last visited 05.05.05.
- [Kalman 60] Kalman, Rudolph, Emil. *A New Approach to Linear Filtering and Prediction Problems*. Transactions of the ASME-Journal of Basic Engineering, vol. 82D, p.35-45, 1960.
- [Keller et al. 2002] Keller, Y. and Averbuch, A. "FFT Based Image Registration", *IEEE international conference ICASSP*, Orlando. 2002.
- [Kinugase et al. 1990] Kinugase, T., Yamamoto, N., Komatsu, H. and Imaside, T. "Electric image stabilizer for video camera use", *IEEE Trans. on Consumer Electronics*, vol. 36, no. 3, pp 520-525, Aug. 1990.
- [Ko et al. 1998] Ko, S.J., Lee, S.H. and Lee, K.H. "Digital Image Stabilizing Algorithms based on Bit-Plane Matching". *IEEE Transactions on Consumer Electronics*, Vol. 44, No. 3, August, 1998.
- [Liu et al. 1993] Liu, B. and Zaccarin, A. "New fast algorithms for the estimation of block motion vectors", *IEEE Trans. on Circuits and Systems for video Technology*, vol 3, no. 2, pp. 148-157, Apr. 1993.
- [Mars Web] Mars Exploration Rover Mission Home. Available on www at <http://marsrovers.jpl.nasa.gov/home/>. Last visited 08.05.05.
- [Morales et al. 1995] Morales, A., Acharya, T. and Ko, S. "Morphological pyramids with alternating sequential filters", *IEEE Trans. Image Processing*, vol.4, pp. 965-977, 1995.
- [Musmann et al. 1985] Musmann, H. G., Pirch, P. and Gralleer, H. J. "Advances in picture coding", *Proc. IEEE*, vol 73, no 4, pp. 523-548, Apr. 1985.
- [NASA Web] NASA - Digital Fly By Wire. Available on www at http://www.nasa.gov/vision/earth/improvingflight/fly_by_wire.html. Last visited 08.05.05.
- [Okada. 1996] Okada, M. "Video camera and video signal reproducing apparatus with shake-detection and correlation operation", U.S. Patent, no. 5,502,484, Mar. 1996.

- [Paik et al. 1992] Paik, J.K., Park, Y.C. and Kim, D.W. "An adaptive motion decision system for digital image stabilizer based on edge pattern matching", *IEEE Trans. on Consumer Electronics*, vol. 38, no. 3, pp. 607-615, Aug. 1992.
- [Rao et al. 2004] Rao, C.V., Dr. Rao, K.M.M., Manjunath, A.S., Srinivas, R.V.N. "Morphological Pyramid Image Registration". *XXth ISPRS Congress*, p 698. July, 2004.
- [RC Web] RC Discounts - Radio Controlled Cars, Boats & Planes. Available on www at <http://www.rcdiscounts.com/>. Last visited 14.06.05.
- [Rosenfeld et al. 1982] Rosenfeld, A. and Kak, A. C. "Digital Picture Processing". *Vol I and II*. Academic Press, Orlando, Fla. 1982.
- [Svedlow et al. 1987] Svedlow, M., McGillem, C.D. and Anuta, P.E. "Experimental examination of similarity measures and pre-processing methods used for image registration". *The Symposium on Machine Processing of Remotely Sensed Data*, pp. 4A-9. June 1987.
- [Thevenaz et al. 1998] Thevenaz, P., Ruttimann, U.E. and Unser, M. "A pyramid approach to sub pixel registration based on intensity", *IEEE Trans. Image Processing*, vol.7, pp. 27-41, 1998.
- [UAV Web] Unmanned Aerial Vehicles - NASA. Available on www at <http://uav.wff.nasa.gov/>. Last visited 30.05.05.
- [Uomori et al. 1990] Uomori, K., Morimura, A., Ishii, H., Sakaguchi, T. and Kitamura, Y. "Automatic image stabilising system by full-digital signal processing", *IEEE Trans. on Consumer Electronics*, vol. 36, no. 3, pp. 510 - 519, Aug. 1990.
- [Xie et al. 2003] Xie, H., Hicks, N., Keller, G. R., Huang, H. and Kreinovich, V. "An IDL/ENVI implementation of the FFT-based algorithm for automatic image registration". *Int.Jl of Computers and Geosciences*, vol .29 ,pp 1045-1055, 2003.
- [Zheng et al. 1993] Zheng, Q. and Chellappa, R. A. "Computational vision approach to image registraion", *IEEE Transactions on Image Processing*, pp 311-326, July, 1993.
- [Zhongxiu et al. 2000] Zhongxiu, H. and Scott, T.A. "Morphological Pyramid Image Registration". *4th IEEE south west symposium*, pp 227. 2000.

Index

- ABS, 7
- accelerometers, 18
- affine transformations, 36
- API interface function, v
- ARCH, 17
 - API, v
 - clients, 42
 - communication protocol, v
 - helicopter, 17
 - server, 42
- artificial horizon, 47
- ASC, 7
- Autopilot, 19

- best-of-breed, 69
- bit-plane, 51
- block size, 57
- BMA, 28
- Boolean functions, 51

- C++, v
- Cartesian space, 36
- CCD, 19
- complex conjugate, 30, 54
- complexity, 57
- computation time, 57
- control signals, 42
- correlation coefficient, 28
- correlation methods, 26
- Cross-correlation, 27
- cross-power spectrum, 30

- dampening coefficient, 60
- DIS, 25
- DOF, 17

- EFIS, 19
- ESP, 7

- feature-based methods, 26
- FFT-based methods, 26
- flight envelope, 44
- flight paths, xv
- flight-mode, 43
 - absolute altitude, 44
 - relative altitude, 44
- FLIR, 13
- fly-by-wire, 6
- Fourier domain, 28
- Fourier Transform, 28
- frequency spectrum, 29

- GA, 32
- Gaussian distribution, 56
- gimbal, 14
- GPL, 19
- GPS, 18
- graph-theoretic methods, 27
- GUI, xv
- gyroscopes, 18

- hicam, 19
- HMD, 14
 - BHMD, 15
- Homogeneous coordinates, 36
- HUD, 12

- IMU, 17
- inner container, 20
- INS, 19
- inverse Fourier transform, 30

- Java, 23
 - JInput, 23
 - JMF, 23
- Kalman filter, 19
- LAN, 23
- layer, xix
- Linux, 17
- LM, 32
- MAD, 28
- magnetometer, 18
- Mathematical morphology, 31
- median filter, 60
- median-based method, 35
- messages, 42
- micro-navigation, viii
- Mini-ITX, 17
- MMI, 12
- morphological filters, 31
- motion correction, 25
- motion estimation, 25
- MP, 31
- MPIR, 30
- MSE, 28
- navigation-mode, 48
- neighbourhood, 57
- noise, 55
- OCU, 21
- outer container, 20
- phase correlation, 53
- Pro X2, 19
- pursuit-mode, 44
- Raptor 90, 17
- rate of climb, 44
- RCS, 41
- registration, 26
- RPV, 15
- search-window, 57
- semi-autonomous mode, 41
- Shift Theorem, 30
- simulator, 45
- smooth panning, 35, 61
- spatial mapping, 25
- STL, vi
- sub-images, 25
- tabs, xv
- task, vii
- total coverage path, xviii
- transformation, 36
- UAV, 5
- unit parameters, v
- USB, 22
- virtual cockpit, 46
- visual information, 46
- VTOL, 2
- waypoint, 42
- words, ix