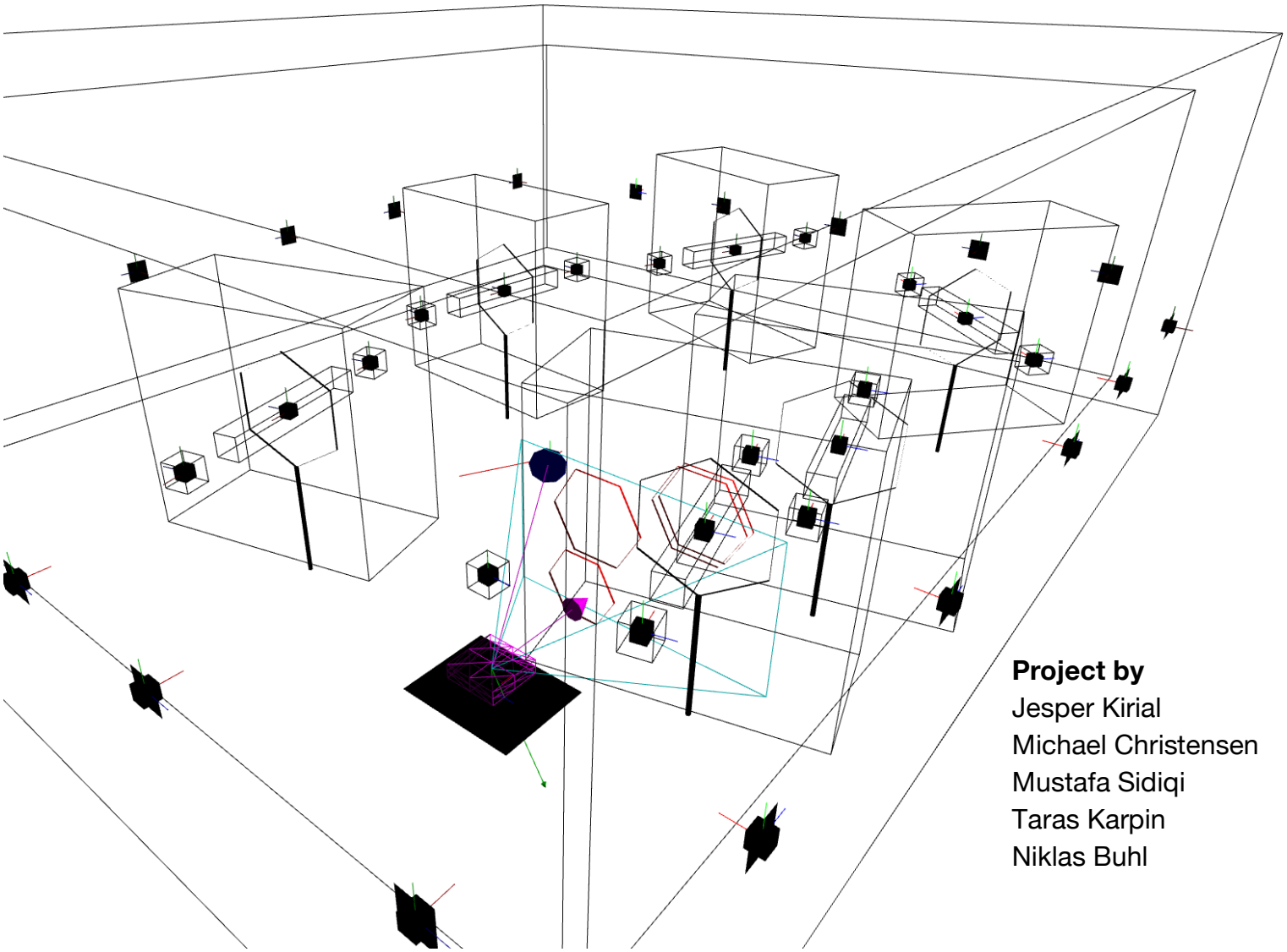
Technical University of Denmark

DTU

Drone Space

DREAM BIG. BE DIFFERENT. HAVE FUN



Project by

Jesper Kirial

Michael Christensen

Mustafa Sidiqi

Taras Karpin

Niklas Buhl

1. Preface

1.1. Team 1

Jesper Kirial	s153300@student.dtu.dk
Michael Christensen	s153712@student.dtu.dk
Mustafa Sidiqi	s153168@student.dtu.dk
Taras Karpin	s153067@student.dtu.dk
Niklas Buhl	nibuh@student.dtu.dk

1.2. Supervisors

Henrik Bechmann	hebec@dtu.dk
Paul Fischer	pafi@dtu.dk
Kurt Jeritslev	kjer@dtu.dk

1.3. Thanks To

During this period, obstacles were encountered and we had to contact and ask someone who were better at the specific area. Special thanks go to:

- George Profenza - a user from stackoverflow.com[1] for giving the idea of how to integrate CVDrone to the final solution.
- Bhupjit Singh for giving the idea on which protocol to use.

2. Introduction

This report covers a project conceived over 16 weeks by a group of five students at The Technical University of Denmark. During the duration of the course the students followed the principles of Conceive, Design, Implement, and Operate first described by MIT.

The objective of the project is to develop software that allows a *Parrot AR Drone 2.0* to guide itself through an obstacle course utilizing the onboard sensors and camera.

The course also wants the students to contemplate the ethical implications that are connected to technology especially regarding the use of drones. Furthermore, the student used previously gained knowledge from previous mandatory courses.

[1] https://stackoverflow.com/questions/44458370/connecting-openframeworks-with-exciting-project-vs2015/44471524?noredirect=1#comment75999136_44471524

3. Content

1. Preface **2**
1.1. Team 1 *2*
1.2. Supervisors *2*
1.3. Thanks To *2*

2. Introduction **2**

3. Content **3**

4. Project Scope **4**
4.1. Main objective and competition *4*
4.2. Procedure *4*
4.3. Camera Vision *4*
4.4. Drone Sensors *5*
4.5. Controlling the Drone *6*
4.6. Visual Aid *7*
4.7. Programmed Autonomy *7*

5. Design **8**
5.1. Developing Framework (Tools, API, libraries) *8*
5.2. Drone API Project Alternatives *8*
5.3. Architecture *10*

6. Implementation **15**
6.1. Challenges *15*
6.2. Computer vision *15*
6.3. Drone Control and Sensors *16*
6.4. Drone Autonomy *17*
6.5. Remote Procedure Calls *19*

7. Tests **20**
7.1. Flight *20*
7.2. Computer Vision *22*
7.3. Sensors *23*
7.4. Autonomy and Visual Aid *24*
7.5. Application *24*

8. Evaluation **25**
8.1. Results *25*
8.2. Group Work and management *26*

9. Ethics **27**

10. Conclusion **28**

11. Perspective **28**

12. Source Code & Appendix **29**
12.1. A: Visual Studio Project *29*
12.2. B: OpenFrameWorks Project *29*
12.3. C: Excel Calculations *29*
12.4. D: Time Sheet *29*
12.5. E: Video *29*

4. Project Scope

Also known as the conceive phase.

4.1. Main objective and competition

By Niklas Buhl

This project consists of four elements; conceive, design, implement and operate. In particular a *Parrot AR Drone 2.0*[2] is the center of attention of this project, the drone itself has integrated cameras, sensors, motors and an operating system making it possible to send simple commands and requests to the drone through an API. The drone opens up a lot of possibilities and applications. Conceiving: the main objective was set to develop a sophisticated generic room recognition application and 3d modelling to give visual aid both in developing and operating the drone, triangulation positioning and trajectory optimization algorithms based on given tasks to explore a wider field of autonomous flight. In this case a competition was presented, the objective is to fly through a series of hoops each giving points.

4.2. Procedure

By Jesper Kirial

When faced with a challenge, where you have no prior knowledge, the first phase is to determine what the problem is, and how to tackle it. By researching various methods of solving the problem, different approaches must be explored, to find a suitable solution that solves the problem through quick hacking and prototyping of different project frameworks and libraries. Trial and error until the testing of an idea proves to be a viable solution. This method provides a quick and not in-depth research in a particular path, because this project will mainly consist of hacking together the work of others more than building something from scratch.

4.3. Camera Vision

Circles

By Michael Christensen

To get the drone to fly through the course set for competition in the end of the 3-weeks project, the drone needs to be able to detect and identify the rings it needs to fly through. For this to be possible, a video feed is needed from the drone camera and use that feed to process the image. From the live feed, a frame can be fetched, processed and then fetch a new frame to repeat the process. The image is then to be manipulated through image processing to detect circles by an image processing tool that can be integrated with the rest of the project.

QR

By Michael Christensen

In the competition room where the drone needs to fly a certain course, QR codes on A3 papers is located on walls and one on each ring. These QR codes needs to be detected and then decoded. This information read from the detected QR codes can be used later in the drone flying process.

[2] Datasheet source: https://www.manualagent.com/parrot/ar-drone-2-0/data-sheet/page-2

Trigonometric and positioning

By Niklas Buhl

Based on above functionality using trigonometric will be used to determine the position of the drone. The circle and QR algorithms provide information about the relative position to the drone, with enough information this will also be used to position the hoops, i.e. if there's enough statically information about the drone position based on other factors, the hoop position can be added or altered in the application. The optical flow can both be used to determine drone movement and the initial room mapping as each point gives physical positioning cross referenced with drone movement.

Optical flow

By Jesper Kirial

Optical flow uses the process of detecting edges, contours and surfaces and tracking their position in the image over time. That is the motion relative to the observer's movement. This can be used, for instance to create a map of a room. If you follow a set of points on a barrier or obstacles to draw lines whenever the observer moves through the room. Paired with the knowledge of movement in a room. Then you have an approximated representation of what how the room looks. Of course, this works best with a stereo camera that the drone does not have.

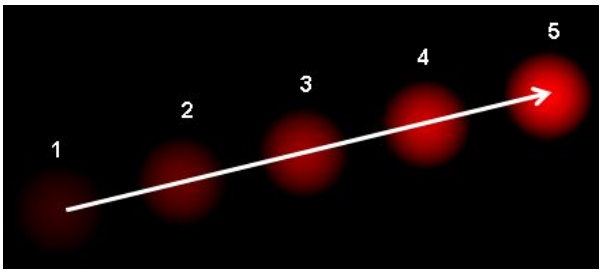


Image 1[3]

The picture above illustrates (Image 1) how a point movement is perceived across 5 images. The red dot moves across the image and the program calculates a vector based on the images.

4.4. Drone Sensors

Intro

By Mustafa Sidiqi

The core of any project about autonomous drones are the sensors, without the sensors, it would be like telling a blindfolded and numb person where to go in a bouncing castle. The drone comes with a hand full of sensors, these data can be used to stabilize, maneuvers, and command the drone where to fly. The extracted data from the sensors can be utilized to give an understanding of what "state" the drone is in.

Height

By Mustafa Sidiqi

An important data from the sensors is the height, as otherwise it would be difficult to issue commands specifying where to fly. The drone uses the ultrasound sensor located on the underside of the drone to calculate the altitude. Since the project focuses on flying

3 https://upload.wikimedia.org/wikipedia/en/1/10/Optical_flow_example_v2.png

indoors, the altitude is calculated based on data from only the ultrasound sensor (Image 2). The height can be a variable in many calculations.

Image 2[4]

Gyroscope & Magnetometer

By Jesper Kirial

The gyroscope and magnetometer are used to measure the angle of the drone, this helps it to stabilize while flying. When a movement command is sent to the drone, the gyroscope and magnetometer will act the limiter of the amount of pitch, roll and yaw it will take, or respond with information about the current orientation. This can be utilized when trying to make precision movement in the room.

Velocity (Accelerometer) Movement

By Jesper & Mustafa

The drone comes with a 3-axis accelerometer, which means it can calculate the velocity in X, Y and Z axis (Image 3). The accelerometer returns essential information, about the drone and how it is moving in a 3D space.

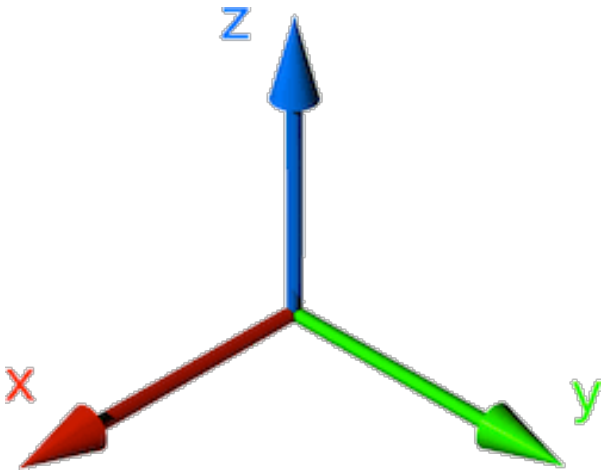


Image 3[5]

4.5. Controlling the Drone

3D Normal Vector Based Movement

By Mustafa Sidiqi

A key part of controlling the drone is the movement, otherwise it would have just been a stationary camera with sensors. Since one of the requirements of the project is for the drone to do image processing and recognize circles, the feedback systems need an agile vector based movement system that provides the ability to move the drone in any of the 3-coordinate axis.

Precision Movement

By Mustafa Sidiqi

Precision is important in any project, and it's not an exception in this case. A system component that would fly the drone with as little deviation as possible, will take the heat of other systems.

[4] http://www.shinyshiny.tv/assets_c/2010/06/869%20ar%20drone-97810.html

[5] http://gamebanana.com/threads/205156

4.6. Visual Aid

By Niklas Buhl

This is a very important part of the scope as the development of all the algorithms needs to be tested and debugged, the visual aid is also a tool for this. Furthermore, the visual modelling is used as the output of the drone room and object recognition algorithms. The room is presented as a wirebox, the hoops as open cylinder with a stand and QR codes as planes on the "walls". The drone is a simple box but is capable of reinterpretation all the physical movements of the actual drone, yaw, pitch roll and movement in the axis. A very important feature is to be able to see what the drone "thinks", "sees" and "feels" (through the sensors). A series of vectors present different information such as current destination, waypoints, straightforward, trajectory, object hitboxes, height, and distance. Furthermore, a simple system is needed to represent the computer vision output of the camera as a "display" in front of the drone. All these visual aids provided a strong tool both the for the end application but indeed very much in the developing process.

4.7. Programmed Autonomy

By Niklas Buhl

Being able to fly around in a room, recognize surroundings and objects firstly to map the room, calculate how to best complete the given objective and then finally to execute. An advanced AI core is needed with algorithms that can take unforeseen changes, and event into account. This includes reconnaissance, calculation trajectory and optimization, anti-collision systems and a very strong triangulation positioning system to keep the feedback system running.

5. Design

5.1. Developing Framework (Tools, API, libraries)

By Taras Karpin

Research is very important part of any project and the great deal of time was spent on research. There are many different projects for the *Parrot 2.0 AR drone*, written in many different languages. Because of the speed of *C/C++*, all the project research went in this direction. During the research part, many projects were found and some of them didn't meet the requirements needed, like *ARDrone*, *ROS* and *ofxARDrone*. There were more but since the amount of time used was at minimum there will be no documentation for those projects. The visual representation aspect of the application, extended the criteria to support a graphical user interface, like *Qt* and *OF*. *CVDrone* was chosen as a startup project, *OF* was chosen for visualization, *Visual Studio* was chosen as a developing IDE on *Windows* and the *GNU* compiler on *Mac* and *RT4* was chosen as a communication between the projects.

5.2. Drone API Project Alternatives

By Taras Karpin

As mentioned there are a lot of projects for *Parrot AR drone 2.0*, so to figure out which one to use, some of the opportunities has to be tested. After making a list of the potential project utilizing the drone API, first from the well documented sources and after that the less documented sources.

ARdrone

By Taras Karpin

The first one was *ARdrone Freeflight*[6], but after going through the code and trying to make it work, the project was dropped and further options got explored.

ROS

By Taras Karpin

ROS is *Robot Operating System* and the most known *Linux* library for robot control. At the time of research 3 versions of *ROS* were available, *ROS -Indigo*, *-Jade* and *-Kinetic*. *ROS* is supported on various *Linux* operating systems, like *Ubuntu*, so the first step was to install *Ubuntu* as a main operating system for application developing. Installing *ROS* gave a lot of different errors and ultimately didn't work.

The final attempt, was to combine the knowledge from operating systems and running a virtual machine image with pre-installed *ROS Indigo*[7]. After some tests, the conclusion was that it didn't work. After the *ROS* was dropped one more version of *ROS* got released and it was much easier to install, but it was not used because another option was chosen.

Qt

By Taras Karpin

Qt[8] is not an API for the drone, but it is framework for *C/C++*. One of the goal of the project was to visualize drone movements, so

[6] https://github.com/yolanother/AR.FreeFlight
[7] http://nootrix.com/software/ros-indigo-virtual-machine/
[8] https://info.qt.io/download-qt-for-application-development

Qt was a framework that would make drawing all the effects much easier, and platform independency made it to an attractive framework, but still working on Linux made it little harder to explore this option. The idea was also to use *C-Lion* as a developing tool and combining three unknown factors made it harder to install all three and making them work together. After some errors, the installation was a success but while exploring other options a different framework got chosen.

ofxARDrone

By Niklas Buhl

One more option was to use *OF* library for drone *ofxARDrone*[9], and after understanding the library and writing the controlling functions that was needed, one flaw of the library was revealed. There were no camera feed library and after testing sensors from drone the conclusion was that sensor data is unstable and the missing camera feed made this option unattractive so other options got explored.

Decision

OpenFrameWorks[10]

By Niklas Buhl

The framework was chosen because of its base on *C++*, accessibility, open source background and very strong vector based visual framework and the fact it's built to easily run across all popular operating systems. Intentionally because of *ofxARDrone* and *ofxOpenCV* the framework seemed the perfect option to build the system, unfortunately the add-ons didn't meet the demands for the project and only the *C++*, visual and vector integrations were used to build the AI and visualization. OF utilized a set of preprogrammed functions to build the application architecture such as *setup()*, *update()* and *draw()*. A really important feature of *OF* is the *ofNode*, which is an entity in the 3D space which has direction, position and can be set as child of other nodes, making it possible to have multiple xyz spaces in parallel being able to communicate position relative to each other. This will be explained further in *Drone* and *DroneRoom* in the client design.

CVDrone

By Taras Karpin

CVDrone[11] was from the beginning the most desirable candidate as a starting point for the project, because from the moment it was run the first time it worked well, both controlling, camera and had functions to read sensors. Even though this option was dropped because *OF* had an add-on for the drone, the *CVDrone* was chosen again because of a missing camera feed in the *OF* add-on. The project has different usage examples and implemented *openCV* which is used for image processing so that made *CVDrone* even more suitable as a final candidate. And after nothing seemed better than *CVDrone*. The *CVDrone* project was chosen.

Microsoft Visual Studio

By Taras Karpin

CVDrone was chosen as a starting point because it worked and was easy to understand. But the main disadvantage of

[9] https://github.com/memo/ofxARDrone
[10] openframeworks.cc
[11] https://github.com/puku0x/cvdrone

CVDrone was it's complicated internal linking and in the source code there were *Linux* makefile and visual studio solution builder. The source code did not in other IDE's, like *NetBeans* which was a known IDE, *Microsoft Visual Studio* was chosen as a main IDE for developing rest of the application, and there is plugin for *OF*, this made it even more suitable candidate.

RPC

By Taras Karpin

Due to library cross-referencing and property dependencies the *CVDrone*- and *OF* project could not be compiled together. Mostly because *OF* had its own dependencies so after a few failed attempts to combine code into one project a different approach was taken to solve the problem of communication between the two projects. *RPC* was the solution. The programs had to be run on different threads so the solution was probably a good idea, and it also included Distributed systems knowledge.

5.3. Architecture

The server and client is set up with *RPC* (Remote Procedure Calls) so that the two used platforms (*OF* and *CVDrone*) can work together. *CVDrone* works as the server while *OF* works as the client calling the functions it needs via the *RPC*.

Computer vision

By Michael Christensen

The functions that contain computer vision is located on the server in *CVDrone*. As functions are only able to return one integer, string, object etc. a function is needed for every information needed in the client.

To get this information extracted from a frame fetched from drone camera, *openCV* is used as it is already part of *CVDrone*.
The information needed in the client about detected circles:

- How many circles that is detected
- Circle center coordinates [x, y]
- Circle height and width

And about the QR-codes:

- How many QR codes are detected
- QR decoded information
- QR center coordinates [x, y]
- QR size

This information is transmitted to the drone client AI in OF.

Image filtering

By Michael Christensen

Image processing can be used to find blobs in a frame and then try and see if a circle can be fitted to each individual blob. Color isolation is also being a good tool to only find what is wanted and thereby eliminating object detection error.

QR Codes

By Michael Christensen

The decoded QR codes is a string that informs where in the room the drone is looking. The data from the decoded QR code and the [x, y] coordinates can be used to help navigate the drone to proper course.

To get identify and decode these QR codes, the same frame as used for circle detection can be used. The frame needs to be searched for the unique pattern of a QR code and then extract the information by decoding

said unique pattern. For this we will use the free library Zbar[12].

Drone Sensors

By Mustafa & Jesper

The sensors are a main component of our design of the system. The accelerometer is an indicator for velocity in the 3 dimensions, ultrasonic for altitude, gyroscope for rate of rotation on the three-coordinate axis. Incorporating the sensors more deeply into the visual application completes the feedback-loop to assist the movement as well as the positioning. On the server side of *RPC* communication, an instance of *CVDrone* have been hacked and extended with the precision movement component and computer vision algorithms. Through the protocol functions can transmit instructions and request real time data.

Drone Server

Drone Control

By Jesper Kirial

The *CVDrone* project relays the commands sent by the *OF* to the drone by sending it *AT*[13] Commands over a UDP connection. These commands describe everything from maximum altitude and speed to LED lights and flipping algorithms.

Vector and Precision Movement

The precision movement component would take distance and direction in the 3-axis as input, and move correspondingly. The purpose is that the other components would perform calculations towards a target, and calculate the needed distance and direction to fly over there. The data would be sent over to the precision movement function, which in turn would send the necessary *AT* commands over to the drone for execution.

Drone Client

DroneControl

By Niklas Buhl

The *DroneControl* class is an interface between the *CV* (Computer Vision), *ARDrone RPC* classes, and keeps track of the manual control, allowing multiple instruction control to the drone compared to the *CVDrone* project, which has a poor user interface. This interface includes 3D instructions, requesting sensors, hoops and QR information. Primarily this interface was designed to help the challenge of not being able to work on the same projects, so instead of having to change a lot of function calls around the code, the *RPC* calls could easily be implemented here.

DroneRoom

By Niklas Buhl

This entity represents the room surrounding the drone, with barriers, obstacles and points of interest, in this instance it's walls, hoops, landing platform and QR codes. The room works as the first 3D space, second is the drone, projected with vectors. Each object has its own vectors making relative and absolute positioning and location possible. Furthermore, each of the known objects and barriers possess hitboxes, these are alarming the drone either leaving or entering the hitbox

[12] http://zbar.sourceforge.net/index.html

[13] https://wiki.paparazziuav.org/wiki/AR_Drone_2/AT_Commands

space, so it can make evasive maneuvers to not hit the hoops or walls.

Drone

By Niklas Buhl

This entity represents the actual drone, simple visualization and functions to present, yaw, pitch, roll and movement, actual physic mechanics could also be implemented, this project instead depends on a feedback loop rather than calculating excessive and advanced flying mechanics. The drone has its personal 3D space tangent to the *DroneRoom*, which basically gives the means to make drone instruction based on the drone point of view rather than the surrounding room. In the illustration, the green dot indicates a point in both spaces with different xyz coordinates but can be cross references (Image 4).

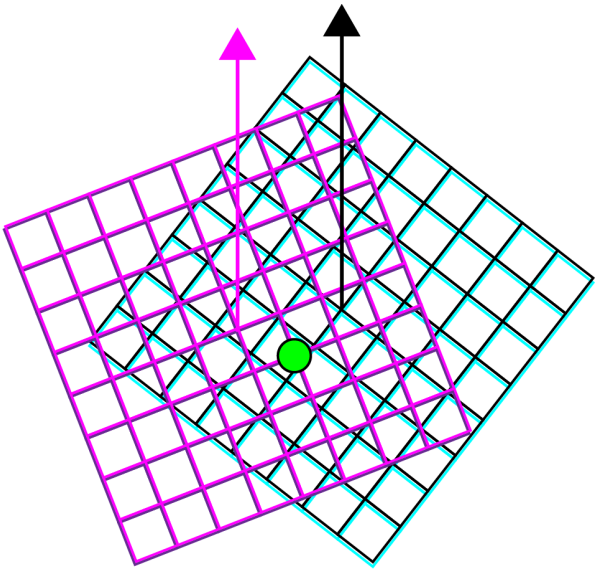


Image 4

A set of functions are developed to set destination, get destination and relative vectors and angle both to objects and global space to help the *DroneAI* to calculate all the trajectories, both present and future. Mainly the AI can set a given destination to the drone, the drone can then by itself calculate absolute vector movement and angle offset. A set of vectors and projections are also designed to show what the drone is "thinking" and "seeing", these functionalities are very strong tools to develop and debug the AI algorithms. This is also utilized in the drone view functions, the hoop and QR code information is projected from the drone and into the room making calculations easier as they both consist relative positions to the drone and to the global room space. Furthermore, the AI can set the drone into different flight modes and drone modes to keep track of progress and current task.

Drone Viewport True XY Coordinates

By Niklas Buhl

Because of the lens in the camera everything the drone sees can't be directly interpreted but has to be translated into true xy-coordinates. Based on the precision of the computer vision, geometric identification and optical flow all this information can be used together with known object sizes and drone movement, to some extend locate object, barriers, points and their relative position from the drone, in this project a vector and angle algorithms are used (Image 5).

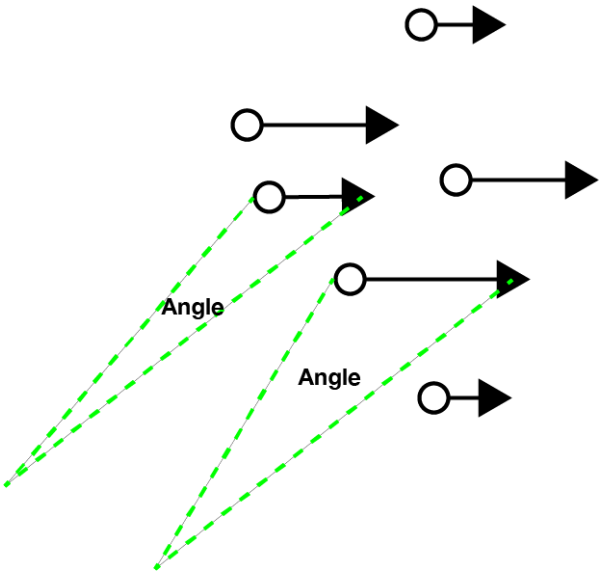


Image 5

DroneAI (Autonomy)

By Niklas Buhl

The *DroneAI* is based on a feedback loop, a given mission to complete, then it calculates the trajectories, optimizes them and starts to complete the mission. The drone needs all the feedback it can get from the computer vision and sensors to be able to recalculate position and thereby updated instructions to get to the designated destination.

Trajectory Calculation

By Niklas Buhl

Take-off, looping, landing are the basic mission states of the drone in this project. Take off means to go from landed to being aerial and ready to enter the looping state (Image 6). The looping is the most advanced trajectory, as the drone is moving around the room the task or objective for the drone to complete, such as search and rescue, book delivery or flying through a number of hoops. As the drones aren't expendable the drone must be able to find its way back to its origin or another suitable place to be picked up and recharged for next adventure, this is the landing trajectory.

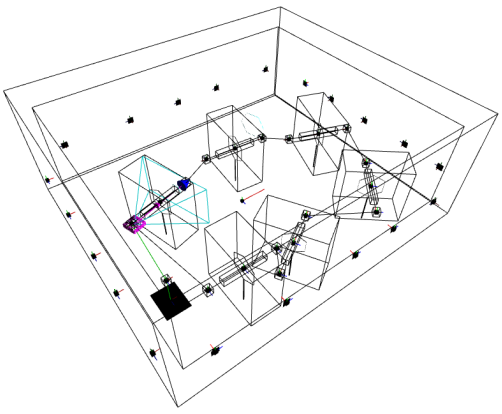


Image 6

Trajectory Optimization

By Niklas Buhl

After a trajectory is calculated the AI must make sure all known object is avoided in the best possible way and the movement of the drone is effective and precise, this is done through trajectory optimization.

Collision avoidance system

By Niklas Buhl

Of course, everything can't be planned to perfection and as the drone is very unstable, it must be able to both avoid unknown or unpredicted events and obstacles, this is where the collision avoidance system does its job (Image 7) red vector is the back inside.

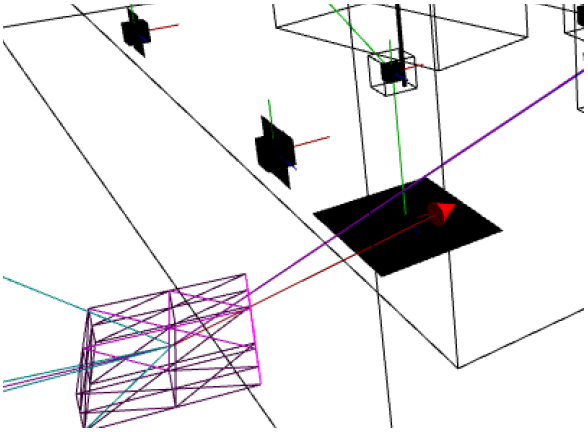


Image 7

Vector Based Triangulation

By Niklas Buhl

A set of different strategies based on what information is known such as drone orientation, known obstacle position, relocate the drone position based on computer vision. Sensor input is recalculated into vectors and furthermore used to reposition the drone in the room, updating the position and complete the feedback loop.

UserInterface

Graphical User Interface

By Niklas Buhl

A user interface is needed to translate all the above information into a 3D world to show all the information. The room is presented as a wirebox, the hoops are drawn as open cylinders, the landing and QR codes are basic planes. The trajectories are a set of lines around the space and the hitboxes are either internal or external wirebox frames. The drone is presented with a basic box and a set of arrows pointing different ways to give the user different indication of drone activity.

Keyboard and mouse input

By Niklas Buhl

The application is controlled with keyboard input such as takeoff (T), land (L), restart (R), manual mode(M), and the drone control in manual mode: forward (W), backward (S), strafe left (A), strafe right (D), turn left (Q), turn right (E), up (SPACE), down (SHIFT). The OF camera is controlled with the mouse, scroll to zoom, click and move to pan around and if the edge of the display is clicked and hold while moving the mouse around the presence of view is rotated.

ofxNBxyz Wirebox

By Niklas Buhl

The wirebox is a 3D object designed to meet certain needs the original *OF* didn't possess. The *ofBoxPrimitive*[14] is designed to be calculated as a set of triangles in order to easily be processed by a graphics card, it didn't meet the demands of this project, instead a very simple wireframe box with functions to rotate get both get relative and global side and corner positions is designed, a highlight the functions to tell if a given position is either inside or outside and how to fastest get to the opposite.

RPC Protocol

By Taras Karpin

Since actual commands to drone were written in the *CVDrone* project and the logic was written in *OF* project, the *RPC* was implemented for communication between the projects. The reading of the sensors and controlling the drone was implemented with the foundation from the *CVDrone*, with the exception that *RPC RT* (Run Time) didn't like floating point numbers, since it expected integers, to solve that problem the numbers got multiplied by 1000 on *OF* side and divided by 1000 on the *CVDrone* side.
Image processing functions had to be implemented differently. Since the image processing could return different amount of data depending on how many ellipses or QR codes it could detect. Since there were 26 QR codes, each QR code got their own bit. When the predefined QR code was detected the bit would be set to 1 and if not it would remain 0. If the QR code was not predefined it would get one of the not defined bit.
The same process applies for ellipses. The correct circle was found, if the ellipse was paired with a QR code. On the other hand, if the circle was not paired with a QR code it could mean error reading or the drone is in the wrong position. There were six circles and first six bits were given to each circle, and rest of the bits were given to unknown circles.

[14]http://openframeworks.cc/documentation/3d/ofBoxPrimitive/

6. Implementation

6.1. Challenges

MacOS, Windows, Linux development platform, developing on same project. Cross platform development.

By Michael & Taras

When combining a project where one part is built on *MacOS* and the other on *Windows* it can create challenges related to *Windows* and *UNIX* differences, even though both projects are written in the same language. For instance, a *Mac OS GNU C++* compiler compiles the line: int a, b, c = 0; As all the variables a, b and c are initialized and set to 0. If the same code is compiled in *Windows*, the compiler argues that the variables a and b isn't defined. This creates errors, that will have to be fixed before the program can run. Another issue between the operative systems, was that *MacOS* didn't complain the specifies return type wasn't returned, meaning a int function could return nothing without errors, while on *Windows* it will generate pre-compiling errors.

Library referencing

By Michael Christensen

When using *VS*, introducing a new external library is quite challenging, especially if you haven't done it before. If all the steps in including an external library hasn't been done properly, a linking error while pop up when compiling; etc. "*LK2001*". This was especially a problem when including the *Zbar* library for QR detection and decoding.
A thing to remember is to set path for #include file(s) (.h) and library file(s) (.lib) to a general path instead of local machine folder path. Otherwise the library included only works for that local PC, but will throw linking errors when compiled on different machines or if any change in folder path location is made.

Vision from an Unstable drone

By Michael Christensen

A lot of factors is included when using drone vision to navigate the drone. A simple factor that can completely ruin vision navigation is the camera focus. If the focus is off, it will be harder even useless to process and might get false camera data readings.
Even though the camera is adjusted, the detection of circles is almost impossible to get perfected. When the video feed fetches a frame to analyze and detect circles/ellipses, it first finds all contours (connecting points that forms a shape). Then tries to fit an ellipse to each contour where a certain minimum of points is needed before a circle is fitted. That means, if anything resembles a circle and has the number of defined amount of points, a circle will be fitted. Even though red color is isolated (for the hoops), this remains a problem because of light conditions and reflections. If false circles are detected or a real circle is detected but connected with smaller/bigger actual width and/or height, the drone navigation is affected.

6.2. Computer vision

Circles

by Michael Christensen

To detect circles, a frame is fetched from drone video feed and processed. First of all the frame is converted into a *CV_BGR2HSV*

format by the *openCV* function *cv::cvtColor*. This new image format has red/warm colors isolated, by giving the image a range of colors to keep and all others blackened. This is done by the *openCV* function *cv::inRange* which takes the input image, a range of *Scalar* color codes and an output picture as parameters. The outputted image is then analyzed for all contours by the openCV function *cv::findContours*. These contours are processed and analyzed to see if an ellipse can be fitted; if so, an ellipse is saved. Analysis is done for all contours found in each processed frame.

QR code

by Michael

For detection of QR codes, a frame is fetched as with circles detection and undergoes processing and analyzing. First the image is converted to a grayscale image as no color is needed, and only serves as useless information. The library *Zbar* is used to scan the image for QR codes. An object of *zbar::ImageScanner* is made which contains the function scan which takes a grayscale *zbar::Image* as a parameter and gives this zbar::image information of QR codes detection. If all four points of the square QR code are found, the QR code is read. The Zbar code can detect multiple QR codes per image, a for-loop retract all information about each QR code detected in a single frame. Both detected circles and QR codes, data are sent to the client if requested.

6.3. Drone Control and Sensors

3D Movement

by Mustafa & Jesper

The 3D Movement function depends a lot on the feedback received from the drone. If the drone is told to rise 50 cm, but it actually rises 75 cm. That is a big margin of error that makes it unsuitable for precision navigation. On the *AR Drone 2.0* this is the story with most of the sensors. The magnetometer would even have great deviations on its imprecisions.

Precision Movement

By Mustafa & Jesper

Getting a precise measure of movement would have been beneficial in all parts of the system, as a main part of the project is to maneuver the drone in a confined space and making decisions based on the position of the drone.

A function that, over time added the axial data from the gyroscope in m/s and multiplied that with the time it had taken to run the function. Adding these numbers made it possible to estimate the distance covered. Implementing this functionality into the *OF* project allowed the movement of the drone on all axis down to +-5%.

Sensors

by Mustafa & Jesper

Implementing the feedback from the sensors are vital to other parts of the dynamic system, as other functions and components are relying and making calculations based on sensor data. The sensor data is extracted by using the *CVDrone* library standard functions, and

extended with running average algorithms, which have given more stable and accurate data.

6.4. Drone Autonomy

By Niklas

The drone autonomy includes three phases and manual override. Takeoff phase, looping phase and landing phase. The end of looping and start of landing is based on a timer set to 4,5 minutes. After 5 minutes, it will immediately tell the drone to land, instead of finding its way back the to the original landing platform.

Drone and Drone Room,

By Niklas

These 3D visual aids are implemented using *OF* 3D standard functions and a wirebox class specifically developed for this project. The yellow vector is the relative destination point for the drone while the purple is a forward vector indicator (see image below), these are aligned to a given max angle offset, so if the drone is turning the wrong way it will hover while turning until it continues to pursuit the destination. This vector translation from global to relative makes it easy to give the actual drone true instructions.

Collision Avoidance System

By Niklas

The wirebox is implemented using nine *ofNodes*[15], one being center and the rest are the corners of the perceived box. At beginning, all of the eight corner nodes are projected one unit away from the center in able to make set size-, position- and rotate functions. Furthermore, two important function are implemented to tell if a given relative position is either inside or outside of the box. The functions take a global position, in this case the drone can translate that into a relative position in order to take rotation of the box into account. From there it's just a matter of higher/lower than the sizes, furthermore it returns the shortest vector from the input position to either get out or get into the hitbox, this is called the escape vector.

Image True XY

By Niklas

After a few image tests, it was obvious - the round shape of the barreled lens distortion had to be corrected in order to get true xy coordinates before translating the information into xyz (Image 8).

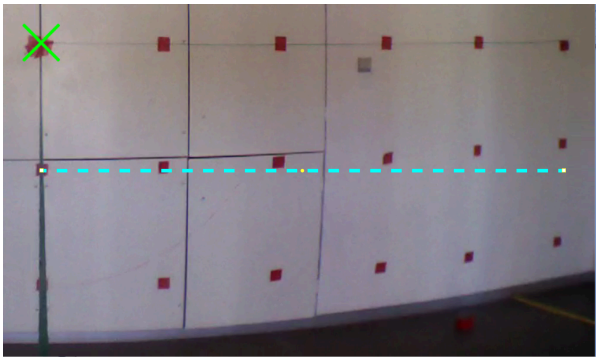


Image 8

Performing a simple experiment with a lot of red squares for the camera to recognize showed clear lens distortions. Translating the point to vectors and simple excel functions to create equations, proved useful results. One very important discovery was that both x and y offset had impact on the true x or y.

15 http://openframeworks.cc/documentation/3d/ofNode/

Image 9

The drone was aligned with the center and an image was used for the calculations (Image 9).

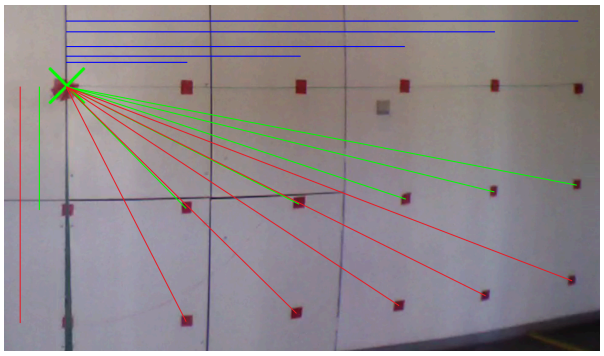


Image 10

By hand all the indicators were translated into vectors (Image 10).

Using excel to create the function for a single axis. Then it became obvious the same effect must be true for the other axis, giving this equation:

$$X_{True} = x \cdot (0{,}0003 \cdot x + 1) * (0{,}0003 \cdot y + 1)$$

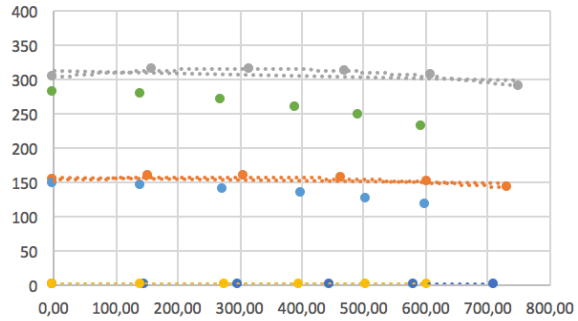


Image 11

As seen in plot (Image 11) it is still not completely corrected, but for the purpose of this project it was adequately, because the drone should only use information in and around the center of the screen to make sure the data is true. Showing green, blue and yellow points as original point compared to the grey, red and blue. See appendix C for details.

Trajectory

By Niklas

Based on how many objective points or tasks, in this case hoops it has to go through the AI will calculate each trajectory waypoint (implemented in as a structure) in a circular linked FIFO list. For each trajectory there's a *head*, *tail* and *current* pointer. Furthermore, functions are implemented to add, insert, remove, complete or end trajectory points (Image Image 12). The function *complete* is used for the looping trajectory as it takes the current head, use the 'prev' pointer and takes that point and connects it with the tail making an infinite loop. The *end* function takes the head and sets an end flag, during in landing or takeoff the AI knows it's about to either go to looping or send the drone landing instructions. The trajectories are visualized with lines, green is takeoff and red in landing while black is the looping trajectory. There's also implemented functions to reset the linked

lists and to reset the drone, making it go back to origin and restart the process.

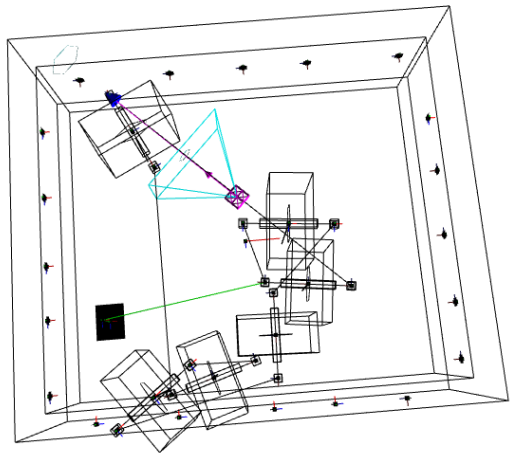


Image 12

Trajectory Optimization

By Niklas

Once the AI has calculated a new trajectory it will make triangular calculation for all objective to see if they're being obstacles in the path. This is simply done with angle offset, distance offset and rectangular distance to the path. If those three tests are not passed, a reroute will be calculated based on the circular safety distance to the obstacle (Image 13).

```
Loop Iteration: 9.
Flight distance AB: 266.878540.
Hoop 0 distance: 103.844017, angle: 30.099295.
Hoop 1 is is too far away.
Hoop 2 angle is too large.
Hoop 3 is is too far away.
Hoop 4 angle is too large.
Hoop 5 distance: 228.624496, angle: 24.352642.
Check Hoops Count: 2
Hoop 0 with 103 int dist, has distance 103.844
Hoop 5 with 228 int dist, has distance 228.624
```

Image 13

6.5. Remote Procedure Calls

By Taras Karpin

There are many different methods implementing *RPC*, like *REST* or *SOAP*. Since Windows was the operative system on which the application was developed, the standard library *RT4* was used to make the communication between OF and *CVDrone* possible. The first step for implementing the communication between the two projects, was to make an *IDL* (interface definition language). The *IDL* file had a unique identifier, name and a version, besides these differences the *IDL* file was similar to a standard header file. In order to use this in *VS* you would have to include the library *rpcrt4*.lib and add an extra command for *IDL* file (*/app_config*). *VS* would then generate files that you would need, so that the programmer would not have to worry about network programming. *VS* generates 3 files; client, server and header file. If there's a reason to add extra functions you would have to regenerate the three files. Delete old ones and include new generated files. One thing to remember is, that the changes that are made in these files, will be deleted as well.

7. Tests

7.1. Flight

Precision Movement & speeds

by Mustafa & Jesper

The system has a precision movement function, which allows us to specify a distance and direction, and the drone will fulfill that command. The precision unit had to go through a series of tests, and be fine-tuned before it worked as intended.

Many factors that affected the results when the function was called. Factors like wind flow in the room, stability of the drone or unstable data from the sensors, however velocity was the deciding affecting factor. Three recorded tests (Image 14) for each time the speed was changed. The recorded measurements are an average of all tests performed for a desired speed. The precision has been rounded up or down to the closest number.

The drone was extremely unstable and off target with a speed of one m/s and distance of one meter. On average the drone was around 10-20 centimeter of target. Velocity was too high, and that the data from the sensors were not stable enough to perform correct calculations.

Lowering the velocity improved the precision, as now the sensors returned more stable data, and calculations were more accurate than the first test. The precision was +/- 15 cm per 100 cm, this was still not accurate enough.

Since the last test was not accurate enough, the speed was lowered again to 0.5 m/s, and flying a distance of one meter, returned satisfying results. The precision was +/- 5 cm per one centimeter consistently. This was within an acceptable margin of error as the drone was flying short distances only using the precision function.

Speed [m/s]	Distance [m]
1	1
Test	**Precision [+/- cm]**
1	(+20)
2	(+10)
3	(+20)
Speed [m/s]	Distance [m]
0.8	1
Test	**Precision [+/- cm]**
1	(+10)
2	(+15)
3	(-10)
Speed [m/s]	Distance [m]
0.5	1
Test	**Precision [+/- cm]**
1	(+5)
2	(-5)
3	0

Image 14

Magnetometer

by Mustafa & Jesper

Sitting stable on a desk, the reading of the magnetometer would return an ever-increasing angle over time (Image 15). Read degrees would increase at the speed of 0.010

degrees per second, and in the next run in the same conditions, the increase would be 1 degree per second.

```
ardrone.roll  = -11.533 [deg]
ardrone.pitch = -11.577 [deg]
ardrone.yaw   = -142.086 [deg]
ardrone.altitude = 1.5 [m]
ardrone.vx = 0.8 [m/s]
ardrone.vy = 0.2 [m/s]
ardrone.vz = 1.0 [m/s]
ardrone.battery = 47 [%]
ardrone.roll  = -12.234 [deg]
ardrone.pitch = -13.347 [deg]
ardrone.yaw   = -142.559 [deg]
ardrone.altitude = 1.5 [m]
ardrone.vx = 0.8 [m/s]
ardrone.vy = 0.2 [m/s]
ardrone.vz = 1.0 [m/s]
ardrone.battery = 47 [%]
```

Image 15

We tried to combat this by making a function that, prior to takeoff, would make 200 measurements of the magnetometer over time. The program should take this as an extra variable when determining the yaw needed.

3D Move Manual Control

by Mustafa & Jesper

There were issues with the stability just after take-off in the beginning of the project. The drone would drift off to one side as soon as it took off the ground. It was quickly discovered, after some research, that the drone used the camera on the bottom to stabilize (Image 16, Image 17). This meant that during take-off, the drone tries to analyze the image from the bottom camera, and looks after patterns and contrasts on the floor, most likely using optical flow analysis. Since the drone was flown indoors, the floor had the same color and no patterns.

Image 16[16]

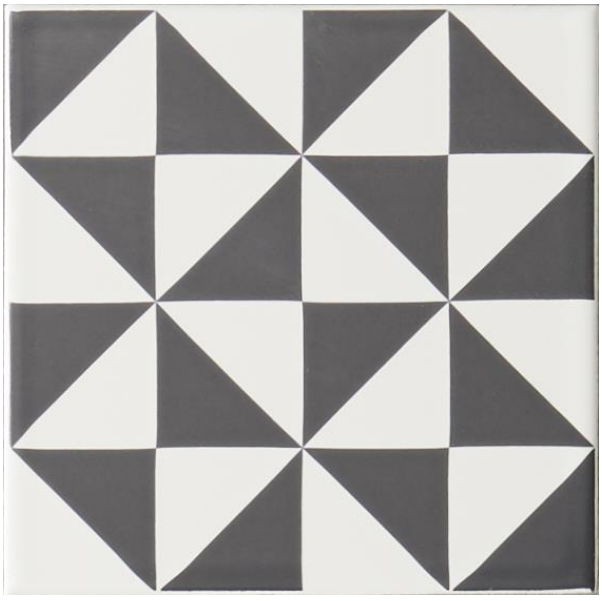


Image 17[17]

The new information was used to add pattern to the floor, and by simply adding a print out of a helipad or placing paper-packages on the floor, the drone would take off and stabilize much quicker (Image 18).

[16] http://puntabejuco.org/wp-content/uploads/2015/01/helipad-300x296.jpg

[17] https://www.originalstyle.com/tiles/product/8506AGR

Image 18

Autonomous Control

By Niklas

Due to lack of time this wasn't tested to the extent of actually being able to say anything about the results, but it was tried and the AI could successfully control the drone based on the decision of the AI. When started, the drone took off, hovered until turning the right direction and then ascent into the trajectory, turning and moving based on the AI calculations. Of course, these were nowhere near to reality because of no timing and movement testing or the fact the application didn't run fast enough, probably because of all the *RPC* wait calls.

7.2. Computer Vision

Hoops

By Michael

As seen this image is non-processed, but clearly a ring is here (Image 19). First the circle needs to be found. The detection of circles started off with perfect circles only. The drone also needed to see the hoops from different angles. A more advanced approach was needed.

Image 19[18]

The same image is processed in the way described in section 3.3, where the picture got converted to grayscale and all the ellipses were found.

A whole lot of ellipses is found even though clearly only one is in the picture (Image 20). The red/warm colors are isolated and this is the following results.

Image 20

18 https://cdn.shopify.com/s/files/1/1513/9688/products/IMG_20170329_143427_799_large.jpg?v=1490851432

Now it just finds the one circle, but occasionally another false circle is detected out of the same points of the correct detected circle.

The number points needed to fit an ellipse is adjusted for better precision by increasing the said number from the standard 5. It was settled at 250 (Image 21).

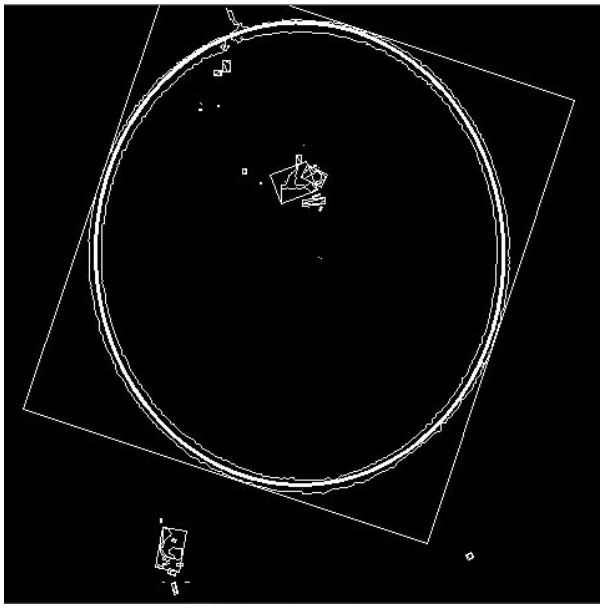


Image 21

This test shows what information that can be gathered from detected circles (Image 22).

```
Ellipse angle:          178.738
Ellipse height: 371.522
Ellipse width:          327.439
Ellipse center x:       237.586
Ellipse center y:       199.186
```

Image 22

QR-codes

By Michael

Several libraries were tried to find QR codes, before settling with *Zbar* library. Detection of a QR-code:

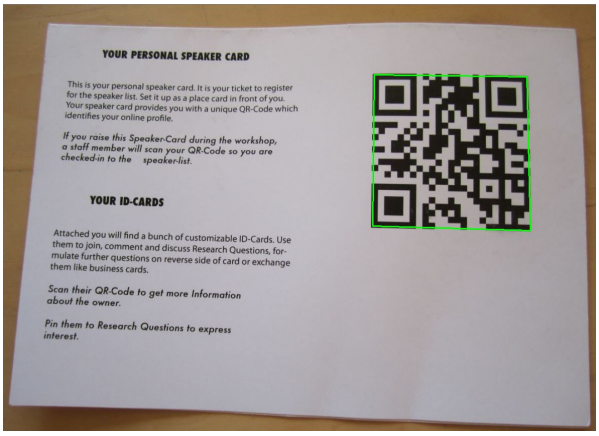


Image 23[19]

As the picture above shows, a QR code was detected and decoded (Image 23). A max of three QR codes can be detected in one frame analyzed.
Here are the results of what information can be gathered from QR codes (Image 24).

```
[10:39:42] 0 decoded QR-Code symbol "https://sQRtalk.com/ig/T292"
QR code location data:
line 1 [x, y] = [739, 139] to [736, 439]
line 2 [x, y] = [736, 439] to [1050, 448]
line 3 [x, y] = [1050, 448] to [1043, 143]
line 4 [x, y] = [1043, 143] to [739, 139]
centerX: 892
centerY: 292
```

Image 24

As shown all the information that is needed from detected QR codes can be gathered.

7.3. Sensors

Yaw - Magnetometer

by Mustafa & Jesper

The data from magnetometer proved itself to be unreliable. When visualized in the *OF* project the “compass needle” was all over the place.

Height - Ultrasonic Sensor

by Mustafa & Jesper

[19] http://37.59.139.1/astrid_2015/wp-content/uploads/2011/11/IMG_2393.jpg

During tests, we found that the ultrasonic sensor was off one meter, when the drone was about 2.5 meters in the air. Again, the faulty data meant that it could not be used.

7.4. Autonomy and Visual Aid

by Niklas

The virtual autonomy is working, this is tested using the 3D visual aids. Meaning that the virtual drone takes off, loop for the given period of time and then finds its way back to the original landing platform. The visual aid has also proven to be very useful and works really well in developing the AI and up till being able to tell what the drone "thinks" and "sees".

7.5. Application

by Taras

The first test of the projects functionality was made exclusively with manual control of the drone. Analyzing the drones' behavior, the error was found. The bug was miscommunication between the projects. In *OF* the definition of axis was not the same as *CVDrone*. This resulted in that the Y and Z axis had switch places, which resulted with the drone flying up, rather than flying to the side.
After fixing the error the drone flew even better than when it was controlled by the *CVDrone* manual control function. The reason for that was that the OF accepted multiple inputs from keyboard while the *CVDrone* accepted only one input at a time.
The second try was the implementation with full automatic control.
The program crashed every time it was started due to linking problem of *Zbar*. Even though the linking was corrected in the main project. Compiling the two projects in separate Solutions, would output a functional program, however if the two projects was compiled in the same Solution, the outputted executable would not work. The solution was to run programs separately.
The whole program worked, but because of the missing feedback from the drone, the simulation and the real drone did not move as one. There was unfortunately no time to fix this problem and the bug is still there, but the solution should be to give a better feedback from the drone to *OF* application.
The image processing and the drawing of the simulation takes quite some time and the drone executes last instruction given to it, because of the delay from drawing simulation, the drone doesn't get the new command in time. There's no solution implemented but one solution would be to multithread the program for logic and drawing parts.

The current state of the solution is: *Drone* has 2 modes, automatic and manual. Manual works as it should and flies drone smoothly. The automatic part of the drone does not have a feedback from the drone and simulation runs blindly while drone sometimes crashes into the walls. The QR code reader finds QR codes and shows on the simulation, and the same with the ellipse detection, but however the functionality is not implemented in AI yet.

8. Evaluation

8.1. Results

Main Objective

By Niklas Buhl

The project needs more work before it meets the requirements set for the main objective, which was to be able to map a room, use the gathered information combined with the camera and the drone sensors to be able to complete the task, which in this case was the hula hoop competition. That being said, a solid foundation has been built, from learning and building algorithms in computer vision, sensors and the 3D world that can support and execute the objective. The project scope was a moonshot, but it's definitely headed in the right direction! Being able to virtually navigate a world, recognize the objects of interest and communicate in the distributed system is working.

The latest version of the project is about 75% complete. The drone has a working AI, the missing part is the not yet implemented communication between AI and the drone control.
which means that the AI doesn't yet know the exact position of the drone and doesn't map room that it is in. The competition is not going to be discussed in detail, but the result is -2 points.

Computer vision

By Michael & Taras

As shown in the test section - sec 5.2, the detection of circle was adjusted to the best of efforts with the time on hand and found circles/ellipses with good precision. The color isolation is however constant, which means it does not adjust for light in the room used for flight or if a different color needs to be isolated. For the needs of the competition, this constant range of color isolation fits well enough as a wider spectrum of color is isolated (warm colors) instead of just a specific version of red.
The QR detection and decoding was very successful with the use of *Zbar* library. A lot of information could be retrieved.
Almost all the testing was done with the drone on the ground. When the drone is in flight, a fetched frame can be blurry if the drone is moving too much when frame is being fetched. This would of course result in very limited to no detection of QR codes and/or ellipses.

Drone Control and Sensors

By Mustafa & Jesper

Since the sensor data is an important dataset used by the other components of the project. It is critical that the sensors return accurate data. For the most part, this was not the case. The only sensor that consistently gave precise information was the accelerometer, which allowed development of the precision flight module.

As for the ultrasound, gyroscope and magnetometer, the data was unstable and not near enough reliable to be used in calculations. The data would vary and oscillate, which basically made it useless. To sum up, the sensors were mostly unreliable and returning unstable data.
For future reference, more focus should have been put on relying on the camera feedback rather than other sensors.

Autonomy

By Niklas Buhl

When connected to the drone the AI immediately gives an impression of an entity thinking by itself, trying to solve the task it's been programmed too. The current state of autonomy opens up a lot of opportunities to use the visual aids as a framework to develop advanced algorithms solving different given tasks.

8.2. Group Work and management

The group work can be divided into two forms hierarchical and synergistic. where the hierarchical group would have a leader and workers to follow the orders of the leader, while synergistic is more cooperative group work. The form we chose was synergistic group work. The scope of the project was discussed and in the end the goal was set. our group consists of five students so there had to be made some agreements before the group project could begin.
The agreements were:

- Show up on time
- Clear the plans for the 3-week period
- To work from 9 to 16 every day

The day begun with every student filling in on what they are working on and what challenges do they face, so the other could help if the help was needed, and make sure that all were working towards the same goal. There were a lot of discussions on how to implement the solution that was chosen and who would implement the solution based on the skill and background. Niklas was studying design and that made him a candidate for structuring the program and designing the simulation also it came natural for him to act as a leader of the project. Mustafa is working as an IT-supporter and he spend quite a while for testing the projects found. Michael had no background in IT and electronics before he started at DTU so, since no one knew anything about image processing he volunteered to take this task on himself. Jesper doesn't have an IT background, but he is a very good at research and his skills to understand the problem made him the main researcher and he was switching between tasks and helping others. Taras has a background in programing and electronics and that made him a candidate for working with the code and a little with drone control.

Project Leading and Management

In this project Niklas has acted as working leader and project manager, having the overview of different task and objectives being pursued, giving the rest of the group more time to dive into their individual research areas. So, they can develop the best solutions and obtain in-depth knowledge of the technologies in order to take the right decisions. Leading is not about making the decision, actually opposite: but to have the team as a unit make decision based on well informed reflections. Though sometimes when everyone is running opposite ways, setting a common direction is needed.

Project Time Consumption

In total, we used 1375 hours on this project. An average engineers' cost per hour is 1.000, the total cost of this project is 1.375.000 DKr. See appendix D for details.

9. Ethics

The definition of ethics has been challenged a lot since the word was created. Ethics was initially defined by what you *do*, but today it is defined by what you *should do* instead. This makes ethics a relatively subjective definition, which can vary a lot from person to person, as each individual has his/her own perspective of how you should do things. Generally, there is a basic understanding of what is right and wrong, for instance killing an innocent man for no reason is observed as ethically wrong by most people, however many subjects have "grey zones" that can't really be defined as right or wrong. Drones is one of these grey zones that has a lot of arguments for and against it. Most known discussions of drones in terms of ethics, is surveillance.

Using drones as surveillance sounds very good from the viewer's point, as the observer can watch whatever without being there himself/herself. However, for the persons being watched, it's a different feeling, for most at least. If the surveillance is for security reasons, some people will feel a sense of security while others feel as if their privacy has been stripped away. Both perspectives are well argued and this is why this problem probably never will be resolved. Measures has been made many places to give the general population a somewhat respectable solution. For instance, flying a drone around wherever you wanted was legal until recently as many people used the drones for surveillance on other people. Now it's only legal to fly on private grounds.

A new challenge arises when it's not humans but rather robots, AIs or programs taking the decisions. Especially when it is civilian lives on stake. Just by looking at the number of civilian death caused by U.S drone strikes, makes one question where to draw the line when it comes to the definition of ethics. The number casualties just in the month of March 2017 hit a record high 1000 deaths[20] of civilians. With number as high as 1000, it challenges us as human to ask whether it is ethically correct to accept civilian casualties. What is an acceptable margin of error when it comes to AI decision making.

The new technological advances make our lives easier and thereby changes our everyday life. But there will be situations where It sets us humans in a tough position. Where we would have to weigh humans lives against each other, where both options would lead to civilian deaths. These question needs to be answered, preferably before it is set in action. But the rate of advancement in our technology is far greater than our ability to solve the ethical questions.

Development of applications that has a component of artificial intelligence, creates new responsibilities, where the developers' ethics is put to the test,
As the author and spiritual master, Amit Ray said:

"As more and more artificial intelligence is entering into the world, more and more emotional intelligence must enter into leadership."

- Amit Ray

20 http://www.independent.co.uk/news/world/americas/us-politics/donald-trump-civilian-deaths-syria-iraq-middle-east-a7649486.html

10. Conclusion

The time frame for the project was too short for us to optimally implement all the things we intended in our project scope. Our vision was maybe a little bit too extensive, but as a great technology pioneer once said, “The people who are crazy enough to think they can change the world are the ones who do.” We are proud of the product we made, even though our drone did not finish the obstacle course.

As a learning experience, working with an unknown device, operating system, and technologies, where none of us had any prior knowledge or experience. Helped us extend our knowledge horizon by getting familiar with some new areas in computer science, like computer vision, working in different frameworks, using new network protocols, 3D visualization, and of course interfacing with a quadcopter.

The group has been an amalgamation of two earlier groups. Mustafa and Michael have previously formed a group, while Jesper, Taras and Niklas have previously worked together. When we formed the group, we had to match our expectations, to make sure that everybody wanted to work toward the same goal. Conclusively the entire group feels that the dynamics has had its up's and downs but an overall great feeling.

11. Perspective

We’re are in the brink of an age, where drones will be much more prevalent in our daily life. from delivering packages and groceries to us, to autonomously hovering above us to deliver wireless connectivity and personal transport. Giving us a peek at what is possible to do has certainly fueled our interest for exploring this field further. We would all like developed a similar project using a drone with better capabilities.

12. Source Code & Appendix

12.1.A: Visual Studio Project

Main.h *rewritten by Taras*
imageProccesing *by Michael*
RPC_drone_Interface_c *by Taras*
RPC_drone_s *by Taras, Mustafa and Jesper*

12.2.B: OpenFrameWorks Project

By Niklas Buhl

Download OpenFrameWorks to compile the project without *RPC*, set *#define HOOP_RANDOM true* in *setttings.h* to either have the hoops randomized or hardcoded to the competition setup.

Drone.cpp *By Niklas Buhl*
Drone.h *By Niklas Buhl*
DroneAI.h *By Niklas Buhl*
DroneAI.h *By Niklas Buhl*
DroneControl.cpp *By Niklas Buhl*
DroneControl.h *By Niklas Buhl*
DroneRoom.cpp *By Niklas Buhl*
DroneRoom.h *By Niklas Buhl*
Main.cpp *By Niklas Buhl*
ofApp.cpp *By Niklas Buhl*
ofApp.h *By Niklas Buhl*
settings.h *By Niklas Buhl*
UserInterface.cpp *By Niklas Buhl*
UserInterface.h *By Niklas Buhl*
RPC.h *By Taras*
ARDrone.h *By Jesper & Mustafa*
CV.h *By Michael*

12.3.C: Excel Calculations

12.4.D: Time Sheet

12.5.E: Video

A short video showing visual feedback from the drone seeing QR and hoops.