Astro Drone: Using crowdsourcing to collect visual data for distance estimation

Master Thesis

Paul Konstantin Gerke

Student number: 0616427

Department of Artificial Intelligence, Radboud University Nijmegen

19th August 2013

Supervisors

Dr. Guido C.H.E. de Croon

Advanced Concepts Team, European Space Agency Noordwijk

Micro Air Vehicle laboratory, Faculty of Aerospace Engineering, Delft University of Technology

Dr. Ida G. Sprinkhuizen-Kuyper

Department of Artificial Intelligence, Radboud University Nijmegen

Donders Institute for Brain, Cognition, and Behaviour

Dr. Willem F.G. Haselager

Department of Artificial Intelligence, Radboud University Nijmegen

Donders Institute for Brain, Cognition, and Behaviour

Abstract

Autonomous navigation of robots requires obstacle localization techniques. Obstacles can be localized based on visual data. Large datasets are needed to test methods for visual obstacle localization quantitatively. In this thesis, I describe how such a large dataset can be obtained using crowdsourcing.

Contributors collected two types of data while playing a game with a camera-mounted model helicopter. First, Speed-Up Robust Features (SURFs) [1] were extracted from images taken while the helicopter approached an object. Second, the distance to the object (ground truth) was calculated on the basis of helicopter sensor data that is processed by a Kalman filter. I tested if the ground truth distance measurements to objects are valid and if the proposed method of collecting data using crowdsourcing is efficient.

To test the validity of distance measurements, I showed that a custom measure based on SURFs correlated with the distance to an object on manually recorded data. When the measure was calculated on Astro Drone data, the same measure correlated with the ground truth distances. This indicates that the ground truth contains valid information about object distances.

To test if crowdsourcing is an efficient way for collecting data, I compare the number of data samples that were collected by the game (718 in three months) to its development time (1 year). I conclude that crowdsourcing is not an efficient method for data collection in short-term research projects, but that long running projects can benefit from crowdsourcing because continued data collection eventually leads to large datasets.

CONTENTS

1 INTRODUCTION

Computer vision plays an important role in the field of robotics. Modern techniques in computer vision allow robots to map environments [7], perceive 3D structure through stereo vision [23], [24] or search for objects using object recognition techniques [9]. For moving robots, the recognition of obstacles is very important to prevent collisions that could damage expensive hardware. To enable a robot to successfully circumnavigate an obstacle, the robot must be able to calculate the distance between itself and an obstacle.

Measuring the distance to objects, which could be possible obstacles, is often achieved by using active sensors like laser range finders [12] or RGB-D cameras [13]. This kind of hardware, however, is relatively expensive and requires a lot of energy while active. Computer vision-based distance measuring uses sensors (for example, webcams) that cost less and require less energy than active sensors. Computer vision techniques that are typically used for distance measurements are algorithms that try to extract *depth information* from 2D images. Computing depth information means to determine distances to observations in an image. A process that extracts this kind of information allows a robot to perceive depth.

Vision-based depth perception is typically noisier than depth measurements that are obtained by active sensors. To test different techniques for depth perception quantitatively, testing data is required which consists of visual data and a *ground truth*. The ground truth consists of the true distance to an object and is used to determine the accuracy of vision-based depth estimates. Collecting such visual data with ground truth is hard and time consuming. Therefore, typical tests of computer vision techniques are qualitative or based on simulated ground truth data that is obtained using renderings of 3D models (e.g. [29], [26]). Nevertheless, there are some studies that use quantitative tests based on small sets of hand collected visual data with ground truth (e.g. [6]).

In this thesis, I will present a new way in which test datasets with ground truth can be collected for computer vision experiments. I focus on collecting data that can be used in the field of robotics to evaluate the accuracy of visual distance measurements towards obstacles. For this purpose, I collect data that consists of visual data of an object that is approached with a camera. For each recorded frame of this approach, a ground truth is saved which is the measured distance to the approached object. This collected data can be used to test the accuracy of distance predictions from different computer vision techniques by using the ground truth values as a benchmark. To collect a large amount of this type of testing data, I developed a data collection application that collects data by using *crowdsourcing*. The application was developed as a part of a research project for the European Space Agency (ESA).

1.1 Crowdsourcing

Crowdsourcing refers to the use of contributions from a crowd of people to perform some kind of work. The type of work that is usually done consists of services that are offered over the internet, for example, using online platforms like Amazon's Mechanical Turk [4]. To get people involved in a project like the proposed crowdsourcing-based visual data collection, awareness for the project needs to be raised in the public and people must be offered an incentive for their participation. To offer people an incentive to participate in the visual data collection, the data collection application was designed as a space oriented computer game, called Astro Drone. To raise

Fig. 1: The Parrot AR.Drone version 1

awareness for the project, the ESA published the game on their website and sent a press release about the game to different newspapers and press agencies.

One goal of the project is to collect *visual* data of objects from volunteers. Special measures have to be taken to protect the privacy of contributors. The game cannot, for example, send visual data that can directly be used to identify people who played the game. This makes it impossible to collect raw images as visual data.

Another criterion for developing Astro Drone is that it must run on publicly available hardware. Distances to objects cannot be measured by expensive sensors like RGB-D cameras or laser range finders. Also, no complex setup of external sensors should be required to play Astro Drone, so that the game remains easy to play.

Because the data that Astro Drone collects will be used for robotic research in the future, also data like the speed and orientation at which visual data was recorded is important. To record this kind of data, the recording device needs to be equipped with corresponding sensors.

Astro Drone is played on an Apple iPhone, iPad, or iPod Touch which is used to remote control a toy quadrotor helicopter called AR.Drone (see figure 1). The AR.Drone has been used in several scientific studies in the field of robotics [2], [25], [10]. It is equipped with sensors that allow Astro Drone to record the flight state of the drone, while also collecting images from the front camera the AR.Drone provides. The helicopter is available in numerous households, given the sales figures from its manufacturer Parrot who claims to have sold about 500000 drones [27].

To protect the privacy of players and still be able to collect useful visual data about approached objects, Astro Drone preprocesses recorded images to represent their content more abstractly as *visual features* (see section 2). Visual features cannot be used to retrieve the original image that they were computed on and, therefore, help to protect the privacy of players.

Ground truth data, which consists of the distance between the AR.Drone and an object, can be measured by using a *visual marker*. The Parrot AR.Drone is capable of detecting a visual marker (shown in figure 2) in front camera images and approximate its distance to the marker. By designing the game such that players must mark an object with a visual marker and approach it with the AR.Drone, it is possible to

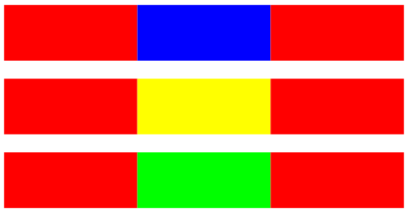


Fig. 2: Schematics of the visual markers that come with the AR.Drone.

record visual features about the approached object while also measuring its distance.

1.2 Testing data

The data that is collected by Astro Drone consists of visual features of an observed object and ground truth data which is the distance between the AR.Drone and the observed object. The purpose of collecting this data is to create a test dataset that helps to benchmark different techniques for calculating obstacle distances from visual data. The data that is collected by Astro Drone had to be carefully selected so that the collected visual data can be used for tests with a wide variety of computer vision-based distance approximation techniques.

Current, typical techniques to extract the distance of an observed object from visual data include stereo vision or exploiting parallax information. In stereo vision, distance information to an obstacle is calculated by comparing two images that are recorded from slightly different perspectives [18], [23], [24]. To exploit parallax information, a moving observer evaluates the apparent motion of otherwise static objects through his field of view and uses this motion to calculate depth information [34]. Another technique based on appearance uses the number of different textures that can be detected in an image to retrieve the distance to an obstacle [6].

Distance values that are computed using computer vision can be absolute measures like meters (used in, for example, [26]) or motion-relative measures like the time-to-contact (TTC, used in, for example, [19]). Both of these distance measures can be used for obstacle avoidance by robots.

Testing data collected by the AR.Drone cannot be used to test stereo vision-based approaches for depth perception because the AR.Drone has only one front camera. However, distance measures that are computed on the basis of parallax information or appearance information can be tested. Visual features that Astro Drone collects consist of the position and appearance of so-called interest points. Interest points are points in an image that carry a maximum amount of information about the visual content, usually describing contours or textures of objects. By using visual features, objects can be tracked in front camera images that are recorded from a moving AR.Drone. This yields the apparent motion of objects in the AR.Drone's field of view. Parallax-based distance measurement techniques use this apparent motion as input. Astro Drone collects visual features from a *series* of images to support tests of such techniques. Since visual features also include descriptions of the *appearance* of interest points, appearance-based distance measurements can also be tested with the collected data.

The ground truth data collected by Astro Drone is the distance between the AR.Drone and an observed object. As discussed earlier, the distance to an obstacle is measured by using a visual marker. The game design must therefore ensure that players

mark objects with a visual marker that they then later approach with the AR.Drone so that Astro Drone can record visual data from the object. Astro Drone tries to improve the AR.Drone's coarse distance approximation to a visual marker by also taking into account assumptions about the initial setup of the game and sensor measurements like the ground speed (see section 3.2.2). This improved distance measure has a higher resolution than the distance approximation from the visual marker. The distance is saved as ground truth for every set of recorded visual features. To be able to compare the absolute obstacle distance to motion relative distance measures like TTC, the current speed of the drone is also saved.

Other data that is saved includes the orientation of AR.Drone (Euler angles). A helicopter like the AR.Drone needs to tilt itself to start moving. The tilt influences the perspective of the AR.Drone front camera. The Euler angles are saved to allow the normalization of visual data for such perspective changes.

This is all data that Astro Drone collects to allow the evaluation of distance measurement techniques on the basis of computer vision. A more detailed look at all data is given in section 3.1.3.

1.3 Research questions

In this thesis, I will explore the quality of the collected data. The ground truth data will be obtained using a Kalman filter which uses helicopter speeds, assumptions about the initial state of the helicopter, and visual marker detection data to calculate distance values to obstacles (see section 3.2.2). It is unknown, if this method for calculating ground truth data yields valid results. Therefore, the first research question that I will answer will be:

> *"Does the data that Astro Drone collects contain valid information?"*

Furthermore, I will explore if it is efficient to use crowdsourcing to collect a test dataset for visual distance estimations. My second research question is:

> *"Can the huge time investment in developing a complex data collection application like Astro Drone be justified by the amount of data it collects?"*

In the next sections, section 2 and 3, I will explain in more detail how Astro Drone was designed and developed and what kind of techniques are used to create Astro Drone. In section 4, I will show general properties of the collected data. To show that the collected data is valid, I show in section 5 that it can be used to show a object distance related phenomenon that is similar to results from a study about the so-called *appearance variation cue* [6]. In the final section, section 6, I will come back to the research question and try to answer them on the basis of the findings from the previous sections.

2 Computer vision and visual features

To protect the privacy of players who play Astro Drone, images that are collected from the AR.Drone need to be abstracted. The used visual data abstraction must not allow to identify players but must still retain enough information of the original image so that it can be used for distance approximation techniques. Essential information that should be retained includes information about the location and appearance of objects in an image. As described earlier, Astro Drone describes images

using visual features to abstract from raw images. In this section, I will give a small overview of visual features and show which types of visual features are used by Astro Drone.

Visual features are a standard technique in the area of computer vision to discover and describe salient portions of an image. They are, for example, used in state-of-the-art object detectors to be able to distinguish between different objects [21], [9]. Visual features are abstractions of pixel data and are designed to carry all required information for recognizing objects but also to be resistant to data noise like electronic noise in digital cameras or changing lighting conditions.

The calculation of visual features is usually done in two steps. The first step involves the detection of *interest points*. Interest points are salient points in an image which, hopefully, contain lots of information about the observed scenery. In the second step, *feature descriptors* are calculated for the detected interest points. Feature descriptors are a collection of values that describe the local surroundings of an interest point. Visual features always consist of an interest point and a feature descriptor.

Using feature descriptors, visual features from one image can then be compared to other visual features in other images. If two descriptors are similar enough, one can assume that they originate from the same observed pattern. Trying to find a similar descriptor for a given descriptor, means finding a *match* between visual descriptors (and between the corresponding visual features).

By trying to match multiple visual features of a known object with feature descriptors calculated on interest points of an unknown image, it is possible to detect an object: If enough descriptors can be matched between the two images, the object is detected in the unknown image. The location of the detected object is then given by the locations of matched visual features in the unknown image.

2.1 Interest point detectors

To find interest points for visual features, an *interest point detector* is used. A popular type of interest point detector that was used in the past is the *Harris corner detector*. The Harris corner detector attempts to detect points in an image that are local unique areas in an image, like, for example, corners [11]. It does this by comparing a patch of an image at a given point with patches of the same image that lie in the proximity of the tested point. It then selects points that are very dissimilar from their surroundings. Compared to modern interest point detectors, the Harris corner detector lacks the ability to assign a size or orientation to detected interest points. However, size and orientation are important properties for the calculation of the feature descriptor. Knowing the size and orientation of an interest point allows to correct for rotation or scaling of an input image.

Modern methods for detecting interest points also calculate size and orientation of interest points. The size of an interest point can be calculated, for example, by using an *image pyramid*. An image pyramid can be imagined as a stack of scaled and smoothed images [20]. Interest points are detected at every level of the image pyramid and the scale is interpolated by projecting the interest point back to the base layer (see figure 3). An orientation is usually assigned by evaluating the direction of the gray scale gradient at the interest point. Often, the strength of detected interest points, which represents the salience of a point, is also saved as a so-called *response value*.

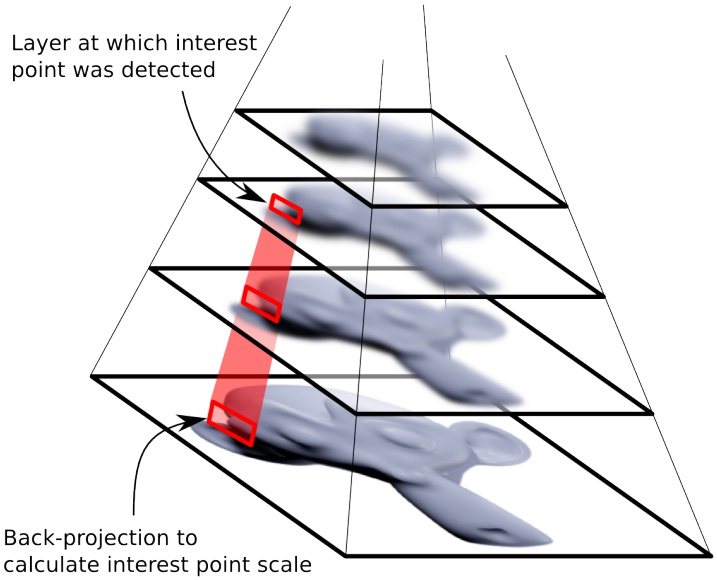
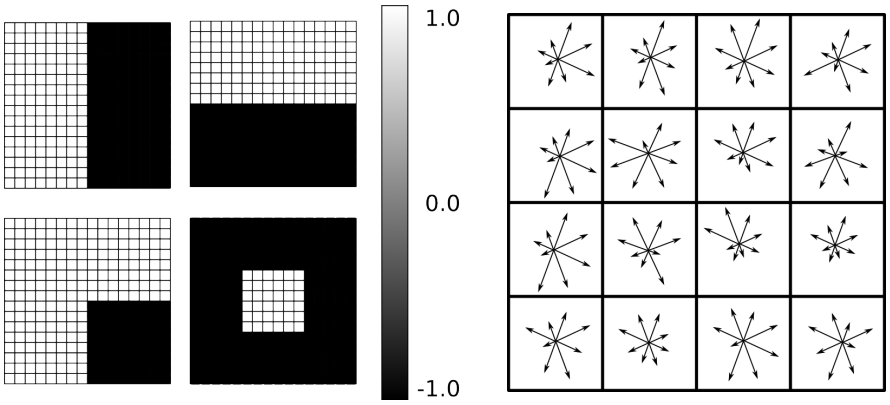


Fig. 3: Image pyramid with multiple smoothed and scaled layers of an image. The red beam illustrates how sizes of interest points can be interpolated through back-projection.



(a) Haar-like feature patterns

(b) SIFT descriptor

Fig. 4: Examples of different features descriptors

One example of an interest point detector that extracts orientation, scale and strengths for interest points is described as part of the popular *Scale Invariant Feature Transform*-feature (SIFT-feature) detector [21]. The SIFT interest point detector specifically looks for points in an image that would also be found if the image was scaled. This selection criterion creates stable interest points which are resistant to scale changes.

2.2 Feature descriptors

After the detection of interest points, feature descriptors are calculated for them. An early example of feature descriptors consists of sets of Haar-like wavelet filter responses [22]. A filter response is calculated by using a pattern of weights in the range $[-1, 1]$ like the ones illustrated in figure 4a. One can use multiple weight patterns to get a detailed description of an interest point in the form of a vector of feature values.

To compute a single value of a Haar-like feature descriptor, the lightness values $L(x)$ of pixels in an image $I(x, y)$ around a detected interest point (X, Y) are multiplied with the weight pattern. The sum of these products represents a filter response (convolution). Equation 1 shows this computation in a formal way. Here, w and h represent the width and height of a filter pattern, and $F(x, y)$ the filter weight at location (x, y). Coordinates (x, y) that fall outside the image $I(x, y)$ can be clamped to the closest valid color value inside the image.

$$\sum_{-\frac{w}{2} \leq x' \leq \frac{w}{2}} \sum_{-\frac{h}{2} \leq y' \leq \frac{h}{2}} F(x', y') L\left(I\left(X + x', Y + y'\right)\right) \tag{1}$$

An example for a newer type of descriptor are SIFT descriptors [21]. A SIFT descriptor consists of binned gray-scale gradients calculated on a 4x4 grid around an interest point (see figure 4b). 16 gradients are computed within one grid cell by sub dividing a cell using another 4x4 grid. The gradient intensities are binned in 8 bins depending on the direction of each gradient inside a cell. The sum of gradient intensities is computed in every of these bins. These 8 sums of feature gradients are used as feature values for a cell in the 4x4 grid. The SIFT feature descriptor consists of a vector containing all 16 of these 8 sums resulting in a vector with 128 elements.

To correct for rotation and scale differences, the 4x4 grid used to calculate a SIFT descriptor is scaled and rotated according to the properties of the detected interest point. Invariance to lighting differences is achieved by zeroing the value of small gradient magnitudes, to prevent fast changing gradient directions.

SIFT features are widely used in automated vision applications today [31], [32]. They offer great stability of detected features, regarding the interest points and descriptors, if images are rotated, zoomed in, or lighting conditions change. However, SIFT features are harder to compute than, for example, Haar-like wavelet features. Because of this, variants similar to SIFT descriptors have been proposed with the goal to achieve similar feature stability with less computational complexity [1], [33].

Of particular interest are *Speeded Up Robust Features* (SURFs) [1] which are used by Astro Drone. SURFs use a rotated and scaled grid of 4x4 cells around an interest point similar to SIFT, but approximate gradient values using Haar-like wavelet filters. The feature descriptor also consists of 128 elements, but they do not consist of bins of gradient magnitudes.

2.3 Application to Astro Drone

Visual features can be used to find visual correspondences between two images. They can be used to track an object in a video. To do this, visual features from two consecutive frames in a video are matched with each other. The difference in coordinates of matched visual features between the two frames represents the observed motion of appearances in the video. Such appearances usually correspond to parts of an object. This is important for the data that Astro Drone collects. It shows that visual features are sufficient to track the apparent motion of appearances in a series of images, and therefore also the apparent motion of objects. This allows collected data to be used for parallax-based distance recognition to an obstacle. Furthermore, feature descriptors hold enough information about the appearance of features, so that they can be used for appearance-based distance measures.

The feature extraction performed by Astro Drone must be computationally inexpensive because the slowest device that game was designed for is the iPhone 3GS,

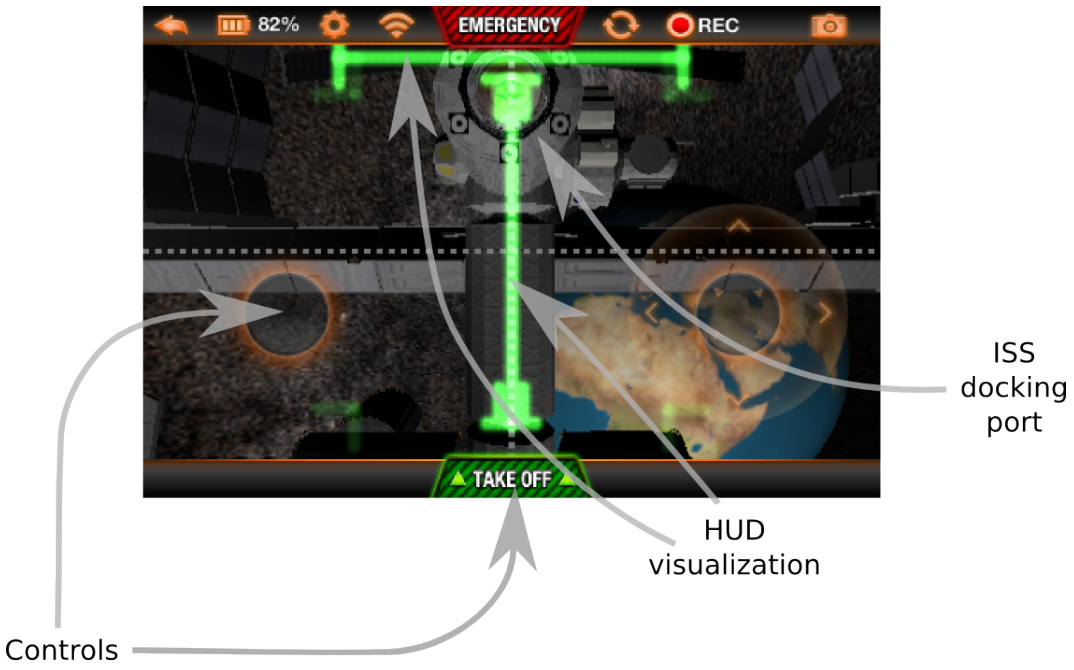


Fig. 5: In-game screenshot of Astro Drone. The ISS docking port is shown in its default position when the game is started. Heads-up display (HUD) elements are visual indicators that show players their position and orientation relative to the virtual docking port. Control elements are used to remote control the Parrot AR.Drone.

which only has a 600 MHz CPU [35]. As a computationally inexpensive but still robust type of visual feature, Astro Drone uses SURFs.

3 Astro Drone

Astro Drone is the game which I developed to collect a test dataset for evaluating computer vision-based distance approximations to obstacles. The game is a virtual reality game that is played on an Apple iPhone, iPad or iPod Touch in conjunction with a Parrot AR.Drone. The game uses visual marker detection to measure the distance to an approached obstacle. The visual marker is also used to couple virtual coordinates to the real-world coordinates. The virtual coordinates are used to map the position of a virtual representation of a docking port of the International Space Station to the real-world position of the marker. The game consists of trying to approach the visual marker in the real world with the AR.Drone. Astro Drone thereby acts as remote control for the AR.Drone and also acts as a window into a virtual world where an ISS docking port is shown at a position that corresponds to the visual marker (see figure 5). By approaching the marker in the real world, the player simulates a docking maneuver with the ISS docking port in the virtual world. The performance on the virtual docking maneuver is scored. Players can compare their scores to scores of other players in a global high score table.

The space docking theme of the game was chosen to match the profile of the European Space Agency, the institute where I developed the game. Furthermore, designing the game as a docking game allows the game to record visual data from objects that players stick the visual marker to. While approaching the marker the AR.Drone records five images from which visual features are extracted. Since a player approaches the marker while playing Astro Drone, image samples are taken at decreasing distances from the approached object. Recording image samples at

different distances from an object makes the recorded data usable for parallax-based depth recognition techniques.

The visual data that is collected by the game record consist of SURFs (see section 2). Using SURFs as image description protects the privacy of players while retaining enough information for future tests of distance approximation techniques.

To further protect the privacy of players, players must explicitly allow Astro Drone to send collected data to a data collection server. The game asks if players want to allow sending data, the first time they try to visit the high score table from the main menu. It uses the prompt shown in figure 7f which can be accepted or declined. If players decline, they are sent back to the main menu and asked again when trying to visit the high score table the next time. If they accept, they can access the high score table and while viewing the high score table, any data that was collected while playing Astro Drone is sent via the internet to a central data collection server. If players accepted to send data once, they are not asked again. Asking players explicitly to share data allows players to control if they want to contribute data. This further protects the privacy of players.

In the following sections, I will describe Astro Drone in more detail. In section 3.1, I will present more details about the design of Astro Drone. In section 3.2, I will discuss solutions to specific problems of the game design. In the final section, section 3.3, I will present an evaluation of the development process for Astro Drone.

3.1 Program details

Astro Drone runs on iPhones, iPads, and iPods (Touch) that run iOS version 6.1 as operating system (*iOS-devices*) and is programmed to act as a remote control for Parrot AR.Drones. The choice to develop for iOS-devices was made because at the time the game development started (beginning 2012), iOS-devices still were the prominent platform that was marketed by Parrot to be used as remote control for the helicopter. To control an AR.Drone, an iOS-device needs to be connected to the AR.Drone via WiFi . Control commands can then be sent to the drone and sensor measurements can be read back from the drone. Astro Drone processes the received sensor data which mainly consists of image data from the helicopter's front camera, flight telemetry and status information of the on-board software. A subset of this data is included in the research data that is collected by Astro Drone.

To allow the game to send data to a server, iOS devices need to be connected to the internet. iPods and some versions of the iPad do not have a mobile network connection to the internet. On these devices, the WiFi connection must first be disconnected from the drone and connected to some other WiFi network which connects the device to the internet. We offer people an incentive to try to connect their iOS-devices to the internet by giving them the option to upload their docking scores (see above) to a global high score table. The global high score table is saved on the same server on which the research data is collected.

In total, there are three types of devices involved in collecting the research data: the Parrot AR.Drone, iOS-devices, and a data collection server. A schematic illustrating the communication paths between these devices is shown in figure 6). In the next section, I will describe the three involved device types in more detail. In section 3.1.2, I show the basic architecture of Astro Drone and in section 3.1.3, I explain what data Astro Drone collects and how it is sent to the data collection server.

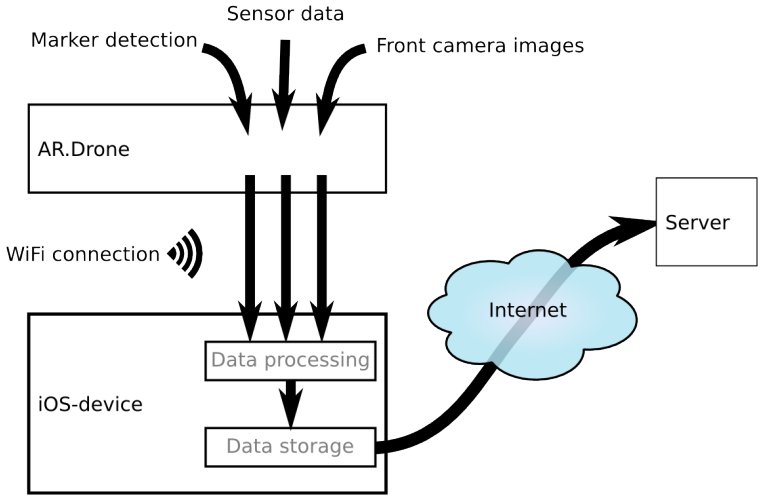


Fig. 6: Communication between the different devices involved in collecting research data.

3.1.1 Device properties

Three devices are involved in the data collection process: The Parrot AR.Drone records image data, measures its flight parameters, and measures its distances to a visual marker. The recorded data is then sent to an iOS-device where the data is processed. In a final step, the processed data from the AR.Drone is sent to a server were it is stored (see figure 6). In this section, I give an overview of the properties and capabilities of these three devices and the implication for the game Astro Drone.

3.1.1.1 Parrot AR.Drone: The Parrot AR.Drone is a toy helicopter that is manufactured by the company Parrot. Currently, there exist two versions of the AR.Drone, the AR.Drone 1 and the AR.Drone 2 (the AR.Drone version 1 is shown in figure 1). Both versions of the AR.Drone are quadrotor helicopters with a square shape with a side length of approximately $70\,\mathrm{cm}$. AR.Drones use a Linux based on-board computer for flight stabilization. The on-board flight stabilization uses measurements from a series of sensors that are installed on an AR.Drone. The sensor data can also be received by an application that controls the AR.Drone via WiFi. Data that can be received from the drone includes:

- Images from the front camera of the AR.Drone
- 3-axis gyro sensor data measuring angular momenta
- Accelerometer data measuring the orientation of the drone (Euler angles)
- Ultra sonar height measurements
- Ground speed estimates calculated from a bottom-facing camera by using optic flow
- Visual marker detection data
- (Only for the AR.Drone 2) Data from an air pressure sensor which supports height measurements, if the drone flies at high altitudes
- (Only for the AR.Drone 2) Digital compass measurements

The front camera images that are sent by the AR.Drone differ between the two versions of the drone. The AR.Drone 1 sends front camera images with a resolution of 320x240 pixels (aspect ratio of 4:3) and the AR.Drone 2 sends images with a resolution of 640x360 pixels (aspect ratio 16:9). Furthermore, the AR.Drone 2 has a better front camera than the AR.Drone 1 which produces less blurry images. The difference in camera quality could make a difference for the evaluation of data that is collected by Astro Drone. Therefore, collected visual data is marked with a version

data field that indicates which AR.Drone version was used to collect the data.

The visual marker detection on AR.Drones can detect the markers shown in figure 2. The visual marker detection data the AR.Drone computes consists of the 2D position of a detected marker in front camera images and an approximation of the distance to the detected visual marker. The resolution of the distance measurement is less than $50\,\mathrm{cm}$ at distances larger than $2\,\mathrm{m}$ from a marker. Because this distance measure is used as ground truth measure for obstacle distances in the collected research data, this measurement is not accurate enough. Astro Drone, therefore, tries to improve this distance measure by using a probabilistic model that takes other measures like ground speeds into account (see section 3.2.2).

3.1.1.2 iOS-devices: The iOS-devices that Astro Drone was developed for were iPhone, iPad and iPod Touch. These devices were were marketed as remote controls by Parrot at the time the game development was started. All targeted devices use multitouch screens as input device. The oldest device regarding its hardware specifications that had to be supported was the iPhone 3GS.

The iPhone 3GS has a 600 MHz main processor and a PowerVR graphics chip that is capable of renderning modern 3D graphics of virtual 3D scenes [35]. This allowed us to render a virtual 3D representation of an ISS docking port while flying the drone.

It was discussed earlier that different iOS-devices can connect to the internet in different ways. All targeted iOS-devices can connect to the internet using WiFi networks. In addition, iPhones and some versions of the iPad are capable to connect to the internet using mobile broadband connections. This can be an advantage because on such devices a player can directly visit the high score table while staying connected to the AR.Drone via the WiFi connection. However, many people pay their broadband data connections on the basis of data volume. Astro Drone, therefore, needs to be programmed in a way that minimizes the number of bytes that are sent to the data collection server. To achieve this, data collected by Astro Drone is compressed before it is sent to the data collection server (see section 3.2.3).

3.1.1.3 Server: The data collection server is a standard PHP-based webserver. Astro Drone only asks for sporadic communication with the server, which occurs only when someone visits the high score table. High score table data requested from the server is not large in volume (less than 2 kilobytes). Astro Drone data sent to the server is also small in volume because it is compressed before it is sent. On the server, received research data only needs to be stored. Handling high score table requests and saving research data does not require lots of processing power. Therefore, as a relatively inexpensive solution, an Apache server that runs on a virtual server was chosen as data collection server.

3.1.2 Game design

Screenshots of Astro Drone are shown in figure 7. After Astro Drone is launched, the first screen that is shown is the menu screen. While in the menu, Astro Drone tries to detect a connected AR.Drone. If an AR.Drone can be detected and Astro Drone succeed in communicating with the AR.Drone, the ”Start Game”-button becomes active and a text at the lower edge of the screen indicates that the AR.Drone has been found (figure 7a). Otherwise, the ”Start Button” stays grayed out and the text at the lower edge of the screen says that Astro Drone tries to connect to the AR.Drone (figure 7b).

Above the four buttons of the menu, a collection of billboards can be seen. This is a level selection widget. Currently only one level, the ISS-docking level is available.

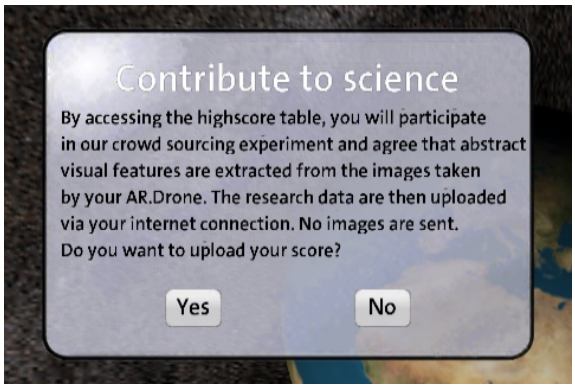
(a) Menu (connected to AR.Drone)

(b) Menu (not connected to AR.Drone)

(c) Instruction video

(d) About screen

(e) High score table



(f) Confirm participation prompt

Fig. 7: Screenshots of Astro Drone

Selecting one of the gray billboards in the background causes the ”Start Game” button to become grayed out even if Astro Drone is successfully connected to an AR.Drone.

Using the four buttons in the lower area of the menu, a player can either start the game, watch an instruction video about how the game is played, visit the high score table, or visit the about screen.

Before starting the game, players should watch the instruction video that can be reached from the menu by pressing the ”Instructions” button. The instruction video is presented in a video player where people can jump between keypoints in the video using two arrow keys (see figure 7c). The video instructs players to setup the

game by placing their AR.Drone 3.5 meters in front of a place where they put a visual marker for the AR.Drone. It also showcases how the game is played and how personal scores can be uploaded to the high score table.

When starting the game using the ”Start Game” button from the menu, players see the screen shown in figure 5. Using the displayed controls, players can start and pilot the AR.Drone. Players are supposed to try to pilot their AR.Drone in a way that they dock with the ISS docking port that is displayed on the screen. Astro Drone will thereby try to match the location of the ISS docking port with the location of the visual marker that is detected by the AR.Drone. This way, when players pilot their AR.Drone so that they approach the ISS docking port on the iOS-device, they will also approach the used visual marker with their AR.Drone in the real world and vice versa. The green lines of the heads-up display (HUD) are helpers that show docking parameters and allow players to adjust the AR.Drone position to achieve a maximum score for their docking maneuver. They also improve the graphical look of the game.

It is important to note, that matching the ISS docking port to the location of the visual marker in the real world is not trivial. To be able to render the ISS docking port in a perspectively correct way, the relative orientation and 3D coordinates of the AR.Drone in relation to the marker must be calculated. This set of data cannot be calculated by only using data from the AR.Drone marker detection. To calculate the required data, Astro Drone uses a probabilistic model that integrates visual marker detection data from the AR.Drone with assumptions about the initial setup of the game and further sensor measurements from the AR.Drone (see section 3.2.2). This method of calculating marker-relative 3D coordinates requires players to setup the game in a specific way. The information how players have to setup the game is given in the instruction video (see above and figure 7c).

When players piloted close enough to the docking port, a message tells them that they have ”docked” and asks them to land the AR.Drone. After landing, a score screen is presented, which scores the docking maneuver using criteria like the time it took a player to dock or the precision of the docking maneuver. When the AR.Drone crashes while flying or the docking maneuver fails because it was carried out too imprecisely, a fail screen is presented. If the AR.Drone still flies after the attempt is rated as failure, players are asked to land it. Afterward, players must reset the AR.Drone to its initial position and can then start another docking attempt.

Another part of the game that is important for the collection of research data, is the high score screen. The high score screen can be accessed via the start menu by pressing the ”Highscore” button. If the iOS-device which runs Astro Drone has a connection to the internet and a player has agreed to contribute research data, this will load the high score table shown in figure 7e. While visiting the high score table, data collected by Astro Drone is uploaded to the data collection server. As described earlier, the first time players try to access the high score table, they are prompted the question shown in figure 7f, which is where they can allow or forbid the application to send research data.

Finally, Astro Drone has an about screen (see figure 7d). It can be visited by pressing the ”About”-button in the start menu of Astro Drone. It shows who was involved in the project, and what public resources (like libraries and media) are used by Astro Drone.

TABLE 1: SURF extraction parameters for OpenCV

Parameter	Value
Hessian threshold	2500
#Octaves	4
#Octave layers	2
Extended descriptors	$TRUE$

3.1.3 Data collection

The data that is collected by Astro Drone contains all data that has been introduced in section 1.2. While a player flies the AR.Drone to try to dock with the virtual ISS docking port, images are recorded from the front camera of the AR.Drone, from which later SURFs are extracted. Since the player is instructed to stick a visual marker to some object and fly towards it, images that are collected while the AR.Drone detects a visual marker should always contain information about an obstacle. The information about the visibility of the marker is also recorded for every image that SURFs are extracted from.

SURFs are extracted using the computer vision library OpenCV [3]. For extracting SURFs certain parameters are required that control what sizes of keypoints should be detected (parameters ”#Octaves” and ”#Octave layers”) and how strong keypoints must be regarding their response value to be detected (parameter ”Hessian threshold”). The parameters values Astro Drone uses are shown in table 1. The parameter ”extended descriptors” refers to which version of SURF descriptors should be calculated. Extended SURF descriptors consist of vectors with 128 components, the shorter version has only 64 components [1]. Since the short version of a SURF descriptor can be computed from the extended descriptor, Astro Drone collects extended SURF descriptors to capture more information about keypoint appearances.

The other information that is required according to section 1.2 is a ground truth measure for the distance to the approached object. This distance is obtained from the AR.Drone marker detection data. However, as discussed in section 3.1.1, the distance approximations to markers that are computed on the AR.Drone have a very low resolution for distances larger than $2\,\mathrm{m}$. Therefore, the values that are saved as ground truth measure are a *smoothed* set of (x, y, z) coordinate offsets that represent the position of the AR.Drone relative to the visual marker. These (x, y, z) coordinate offsets are the same coordinates that are computed for visualizing the ISS docking port in Astro Drone. How the (x, y, z) coordinate offsets are computed is explained in more detail in section 3.2.2. The (x, y, z) coordinate offsets to the marker are measured in meters and are saved for every front image that is captured from the AR.Drone.

The full set of data that is recorded for every image that is captured from the AR.Drone consists of:

- Visual SURFs computed on an image recorded by the front camera of the AR.Drone
- The current 3D coordinate offsets (x, y, z) to the visual marker
- The current ground speed as measured by the AR.Drone
- The current Euler angles as measured by the AR.Drone
- If a visual marker was detected in an image or not
- Time since AR.Drone takeoff

During one flight of the AR.Drone in which a player tries to dock with the virtual ISS docking port, five of these image data samples are recorded. The samples are recorded in $250\,\mathrm{ms}$ intervals when the drone is detected to fly forwards at a speed of at least $0.15\frac{\mathrm{m}}{\mathrm{s}}$. The recorded image data sequence is called a *flight sample*. Only

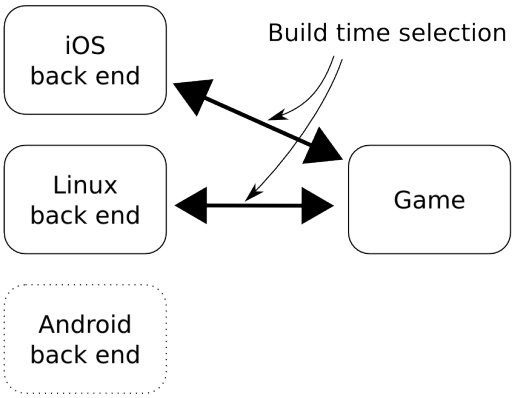


Fig. 8: Cross platform design with exchangeable system back ends that are chosen at build time

one flight sample is recorded during per simulated docking maneuver.

Flight samples are saved as plain-text JSON structure [5] on iOS-devices so that they can be sent to the data collection server when a player visits the high score table. I chose JSON structures to save flight samples because the format does not obfuscate data and is openly accessible to people who are worried about what kind of data we collect. However, before flight samples are sent to the collection server, JSON structures are compressed because the uncompressed JSON structures are too big for mobile internet connections (see section 3.2.3).

The server simply saves the compressed data that is sent by Astro Drone. To get uncompressed data for experiments, one must download a set of compressed flight samples from the server using a web interface. The flight samples can then be decompressed by using a specialized program that is also developed as part of the Astro Drone project.

3.2 Implementation

In this section I will focus on implementation specific issues regarding Astro Drone. I will describe how the architecture of Astro Drone is designed to facilitate future ports of the game to other devices and operating systems, describe in detail how the 3D reconstruction of AR.Drone coordinates works, and describe how recorded SURFs are compressed before they are sent to the data collection server.

3.2.1 Cross platform design

Astro Drone was carefully engineered so that it can easily be ported to other devices and operating systems in the future. Astro Drone was targeted at iOS-devices, but to support its development process and remove the dependence on iOS-devices for debugging, it was also programmed to run under Linux on a personal computer. Future plans of the ESA involve extending the game to also run on Android-based smart phones and tablets. To write code that can be run on such different hardware with different operating systems, Astro Drone needs to be developed using a *cross platform design*.

TABLE 2: Library choices for the cross platform design

Function	Realization
3D Rendering	OpenGL 3.3 for PCs and OpenGL ES 2.0 for iOS and Android devices. OpenGL 3.3 is for the most part a superset of the functions of OpenGL ES 2.0, therefore, designing the game for OpenGL ES 2.0 makes it (largely) compatible to OpenGL 3.3, too [1].
File read/write access	C++ standard libraries (iostream). Opening files is done via the platform specific system back end, since configuration files and resource files are stored in different directories for different target platforms.
Thread management	C++ Boost library.
HTTP Protocol	Standardized interface through CURL library.
Video playback	Cross compiled library libogg and libtheora. This implies that no hardware video acceleration on ARM-devices can be used for video decoding.
Compression library	lzma from libxz, cross compiled for target platform.

For an effective cross platform design, common capabilities of the targeted hardware and operating systems had to be found. The platforms that had to be considered for this are iOS-devices (iPhone, iPad, and iPod Touch), personal computer (with Linux as operating system), and Android-based mobile devices. The capabilities that had to be analyzed consist of the common hardware of these devices and the programming language that can be used on the devices. Finally, it needs to be addressed how function calls to the operating system can be unified so that Astro Drone can use a standardized interface to these functions on all targeted platforms.

The hardware of iOS-devices and Android-based mobile devices are very similar. Most Android devices have a multi-touch touchscreen and a WiFi network adapter which are the integral elements that Astro Drone requires. Personal computers do not usually have a touchscreen, but they can simulate single-touch input by using mouse inputs. A WiFi network adapter for controlling the AR.Drone can easily be added to the hardware of a computer.

Programs for iOS-devices are written in a special C-dialect that is called *Objective C*. Programs that are written for Android are written in a variant of Java. However, programs on both systems are able to link to or make calls to native machine code that has been programmed in C++. On personal computers with Linux as operating system, C++ is a standard programming language. Therefore, Astro Drone is written in C++, a programming language that can be used for all targeted systems.

Finally, a way must be found to offer a unified interface for calling operating system functions on the different devices. To give access to those functions, iOS uses Objective C-based libraries and Android uses Java-based libraries. Both cannot directly be used by program code written in C++. To unify calls to operating system functions and allow functions to be called from C++, Astro Drone uses an adapter software design pattern [30] that I called *system back end*. The appropriate system back end for a given operating system is selected at build time (see figure 8).

Even though, most calls to operating system functions can be unified using the described system back end, some operating system functions did not need to be unified because they were based on common standards or could be unified through the use of third party libraries. A compilation of general functions that were unified

1. In theory, OpenGL 4.0 is the superset of OpenGL ES 2.0, because OpenGL adds tessellation to the function set of OpenGL 3.3, which is also supported by OpenGL ES 2.0. However, since the game does not make use of tessellation, OpenGL 3.3 still suffices to be a superset of OpenGL ES 2.0 functionality. A small intermediary shader library is used to insure shader function names that exist on iPhone and PC platforms are equal between systems.

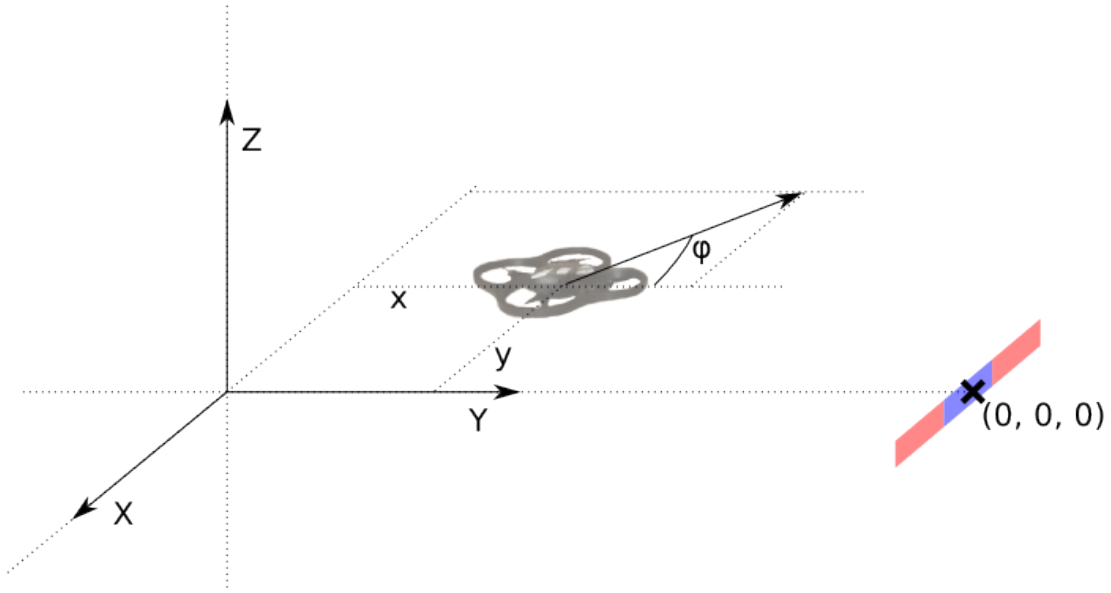


Fig. 9: Coordinate system that is used for coordinate and angle reconstruction of the drone in relation to the visual marker. All reconstructed values (x, y, z, ϕ) are shown.

by standards or third-party libraries is shown in table 2.

3.2.2 3D Coordinate Reconstruction through Kalman Filtering

In order to render the docking port of the AR.Drone in a perspectively correct way, the 3D position (x, y, z) and orientation ϕ of the AR.Drone in relation to the visual marker must be known (see figure 9). The measurements of the 3D position and orientation must be of high spatial and temporal resolution to avoid that the rendered ISS docking port makes large visual jumps while playing the game.

To calculate the 3D position and orientation of the AR.Drone, Astro Drone uses a probabilistic model called *extended Kalman filter* [17], [15]. A Kalman filter uses an iterative model to describe how the state $\mathbf{s}_t$ of a system changes based on a previous state of $\mathbf{s}_{t-1}$ and an input vector $\vec{i}_t$. The state $\mathbf{s}_t$ of the system cannot be observed directly. However, $\mathbf{s}_t$ can be observed indirectly using measurements $\mathbf{m}_t$. The Kalman filter uses a measurement model that describes what measurements $\mathbf{m}_t$ can be observed given the system state $\mathbf{s}_t$. By reversing the inference from $\mathbf{m}_t$ to $\mathbf{s}_t$, the Kalman filter can infer the state of the system.

In a Kalman filter, system states $\mathbf{s}_t$ and sensor measurements $\mathbf{m}_t$ are not modeled as simple value vectors but as multivariate Gaussian probability distributions. This allows the filter to account for noisy sensor measurements. The modeled noise of measurements is used to infer the uncertainty of the system state $\mathbf{s}_t$. The uncertainties of the system state and sensor measurements are represented by the corresponding covariance matrices $\mathbf{s}_t^{\Sigma}$ and $\mathbf{m}_t^{\Sigma}$.

For Astro Drone, the Kalman filter is used to compute the position and orientation of the AR.Drone in relation to the visual marker, as well as the ground speed $(v'_x, v'_y)_t$ and height h'_t of the AR.Drone: $\mathbf{s}_t = \left((x, y, z, \phi, v'_x, v'_y, h')_t, \mathbf{s}_t^{\Sigma}\right)$. The measurements $\mathbf{m}_t$ that are used to infer $\mathbf{s}_t$ consist of marker detection data, ground speed $(v_x, v_y)_t$, height h_t, and angular speed around the z-axis a_t^{ϕ} that are measured by sensors on the AR.Drone. The marker data consists of the position of the marker in the front camera image $(m_x, m_y)_t$ and the distance approximation towards the marker $(m_d)_t$. This leads

to measurements being modeled as follows: $\mathbf{m}_t = \left((v_x, v_y, h, a^\phi, m_x, m_y, m_d)_t, \mathbf{m}_t^\Sigma\right)$. The input vector $\vec{i}_t$ consists of a set of user inputs that control the AR.Drone's velocity as well as the rotation around the z-axis.

Astro Drone uses a Newtonian physics-based, nonlinear model to describe how state $\mathbf{s}_t$ can be derived from a previous state $\mathbf{s}_{t-1}$ and an input vector $\vec{i}_t$. The measurement model that predicts marker detection data $(m_x, m_y, m_d)_t$ from a state $\mathbf{s}_t$ uses the perspective projection of point $(0, 0, 0)$ onto a virtual camera that is placed at position and orientation $(x, y, z, \phi)_t$. The predictions for the measured ground speeds $(v_x, v_y)_{t+1}$ and drone height h_{t+1} are made by simply copying the corresponding values from the state $\mathbf{s}_t$.

Finally, the Kalman filter requires an estimate of the initial state $\mathbf{s}_0$ and the covariance matrix representing the noise of the measurements $\mathbf{m}_0^\Sigma$. To set $\vec{s}_0$ the Kalman filter assumes that the Astro Drone is started in the position that is presented in the Astro Drone instruction video (see section 3.1.2). The initial state $\mathbf{s}_0$ is modeled after this setup. $\mathbf{m}_0^\Sigma$ was manually set to a diagonal matrix that lead to good results when testing the Kalman filter.

The x, y, z coordinates that are computed by Kalman filtering are measured in meters. Extensive qualitative testing during the game development showed that the reconstructed AR.Drone coordinates and orientation work relatively well for matching the ISS docking port to the position of a visual marker. The filter successfully smooths low-resolution data from the visual marker detection on the AR.Drone and is able to reconstruct continuous coordinates that match the flight trajectories of the AR.Drone. Because the reconstructed x, y, z coordinates show such a better resolution compared to raw data from the marker detection, the coordinates are collected as ground truth values by Astro Drone for measuring the distance between the AR.Drone and an object.

3.2.3 Flight sample compression

Flight samples that are collected by Astro Drone are saved in JSON structures [5]. The raw size of these structures which must be sent to the data collection server can reach $1.5\,\mathrm{MB}$. This is too large, considering the development goal not to stress mobile internet connections by sending large volumes of data. Therefore, Astro Drone uses lossy compression to compress the size of the sent JSON structures to a size of about $200\,\mathrm{kB}$.

The biggest size related issue with JSON structures is that they use a very wasteful representation for floating point numbers. Floating point values are represented as plain-text in JSON. This matches the goal to use an open format for sending data, but the plain-text format usually requires more than 12 bytes to represent one value:

```
1.1123181581497192
1.2316429372823e-16
```

Floating point values from SURF descriptors form the largest payload of the sent data. Navigation data, like speeds, height, or Euler angles are measured also using floating point values, but do not occur as frequent in a flight sample. SURF descriptors are responsible for on average of $5\,\mathrm{images} \times 100\frac{\mathrm{features}}{\mathrm{images}} \times 128\frac{\mathrm{floats}}{\mathrm{features}} \times 12\frac{\mathrm{bytes}}{\mathrm{floats}} \approx 750000\,\mathrm{bytes}$ in the standard JSON structure. This size can be even larger if more than 100 features [2] are extracted per recorded image.

2. 100 features are an example for a typical large number of SURFs that are detected in images from an AR.Drone 1.

To limit the amount of data that needs to be sent to the data collection server, Astro Drone limits the maximum number of SURFs that are extracted per image. For every recorded image, only the 125 *strongest* SURF features are saved. The strength is given by the response strength of the SURF interest point detector.

In a next step, all values of all feature descriptors from one flight sample are collected in one large value vector. The values of the original feature descriptors are replaced by pointers into the newly created large descriptor vector. Then, the floating point values (4 bytes) in the large value vector are converted to half-floats (2 bytes) [14]. During this process information is lost. However, most values of a SURF descriptor seem to fall in the interval $[-1; 1]$. A half-float represents values in this interval with a resolution of at least 0.000488. This is still a relatively high resolution for encoding appearance differences in a SURF descriptor (see appendix A for details).

In a last processing step, the binary vector of half-floats is further compressed using the *Lempel-Ziv-Markov chain algorithm* (LZMA) [28]. The compressed, binary data then is embedded in the JSON structure as a base64 [16] encoded string, adding a 33% percent overhead to the compressed data.

Floating point values that are not part of a SURF descriptor, are also compressed by truncating them to 3 positions after the decimal point. This ensures enough precision for all affected values: Angles are measured in degrees, distances in meters, times in seconds. No sensor on the AR.Drone has enough precision to require more than three positions after the decimal point for an accurate representation of its measurement. This lossy compression by truncation, reduces the size of affected floating point values by approximately 8 bytes.

Since the compression algorithm used to compress flight data samples uses LZMA, the achieved compression ratio is not constant. However, the size of sent data for feature-rich flight data samples is reduced from about $1.5\,\mathrm{MB}$ to $200\,\mathrm{kB}$. A lot of the remaining size of the JSON file is used up by names for fields. These could be further compressed to achieve an even higher compression ratio by choosing shorter field names for repeating fields. However, I did not implement this kind of compression since the goal was to use an open and easy to use data structure. Shortening field names would obfuscate the sent JSON structure. Furthermore, sending $200\,\mathrm{kB}$ of data volume causes not much more data traffic than, for example, accessing a medium sized web site.

3.3 Project evaluation

The development of Astro Drone took me about one year. Compared to the initial estimates which projected the implementation to take three months, this is very long. The false estimate of three months was mainly caused by inexperience with projects of this magnitude.

However, the amount of extra time that was required to implement Astro Drone is also related to unexpected problems that came up during the development of the game. For example, the development of debugging tools, which were able to show what images were captured from the AR.Drone on iOS-devices, took about three weeks. Another massive delay was caused by trying to create Astro Drone for an old version of OpenGL (OpenGL ES 1.1). Late in the development stages it showed that the AR.Drone libraries from Parrot required to use a moderner version of OpenGL (OpenGL ES 2.0) which required me to rewrite large portions of Astro Drone's code.

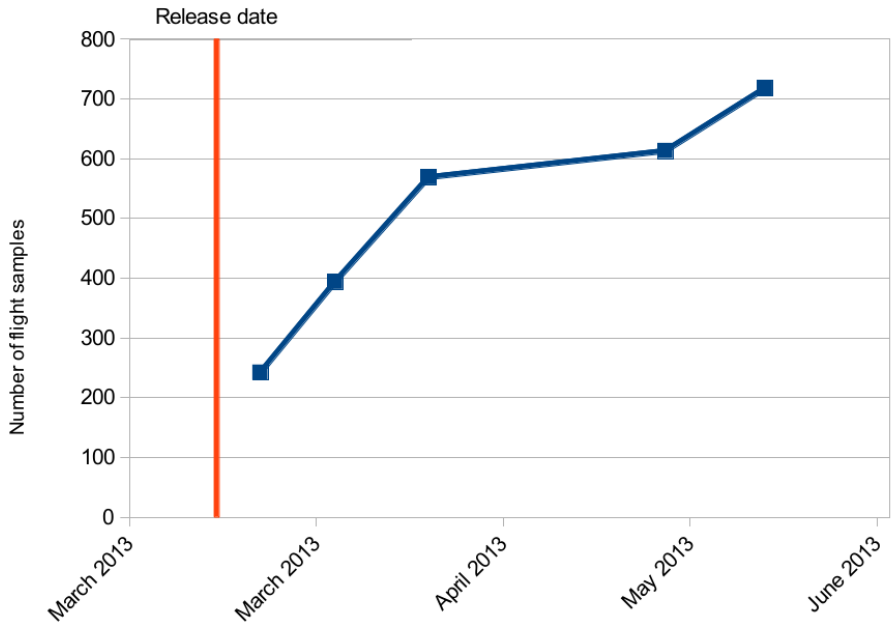


Fig. 10: Development of the number of features recorded in the database over time.

Astro Drone was published by the European Space Agency on 15/03/2013 in the Apple App Store. In the three months since it was published, a small number of minor bugs appeared. The bugs were mainly caused by bad user interface design or wrong assumptions about the use of the software. Most of these bugs were fixed with patches. Despite the minor bugs, the game received good ratings on the App Store.

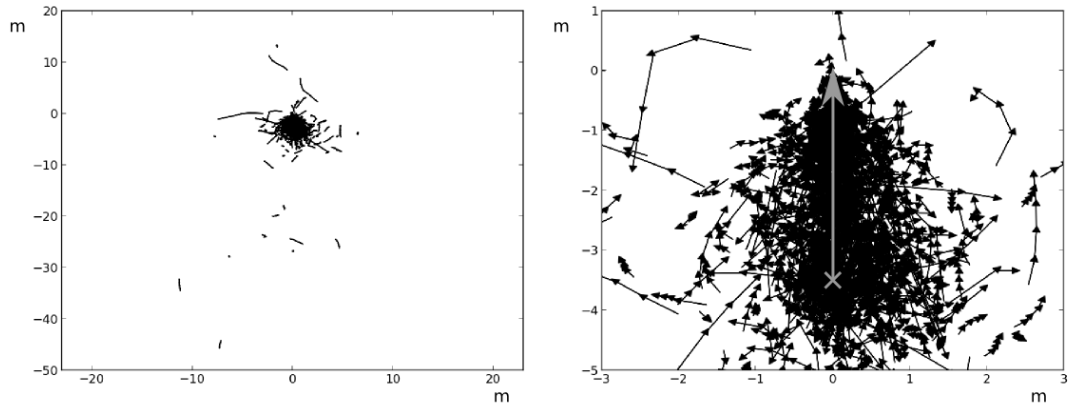
I succeeded in implementing all program features that are required to allow Astro Drone to collect testing data for evaluating computer vision techniques for approximating obstacle distances. The game runs stable and is available as a free download from the App Store for iOS-devices.

4 Properties of the collected data

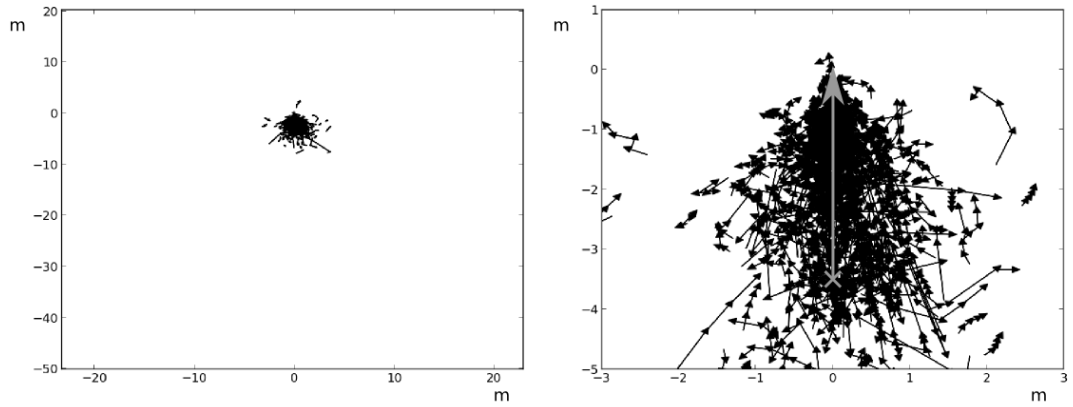
In this section, I will show general characteristics of the collected visual features and ground truth data. In section 4.1, I show how many data samples were collected since the release of Astro Drone. In section 4.2, I show flight paths that are reconstructed from the collected ground truth data. In section 4.3, the 2D distribution of all visual features from all recorded flight samples is shown. In the final section, section 4.4, I sum up all interesting findings from the previous sections.

4.1 Collected data

Astro Drone was launched on 15/03/2013. Until 11/06/2013, 718 flight samples were collected which is the set of samples that I will use for all future analysis in this thesis. Figure 10 shows how the number of collected data samples developed over time. Note that the number does not only consist of ”third” people contributing, but also of data that was collected during tests and demonstrations of the game. The data that was *not* collected by ”thirds” amounts to approximately 200 flight samples. The true number of samples that were contributed through tests and demonstrations cannot be reconstructed because of the privacy insuring techniques we employed in the software design.



(a) Flight paths from all flight samples



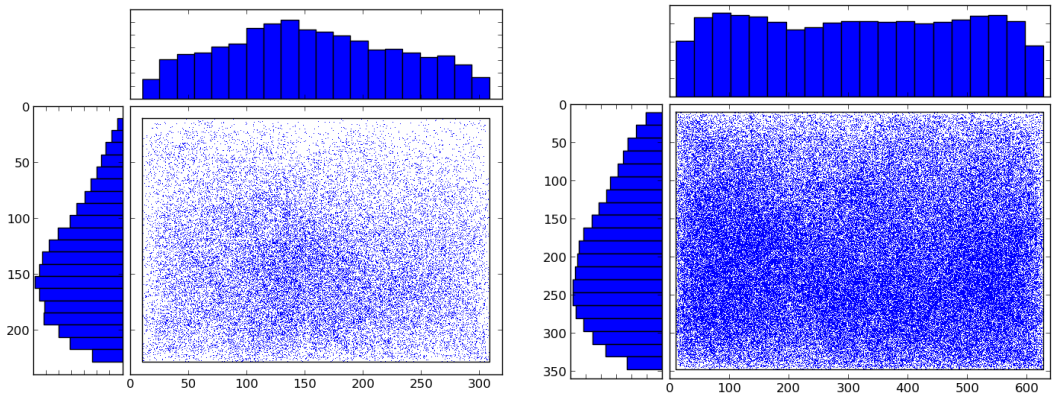
(b) Flight paths from flight samples with detected visual marker

Fig. 11: Flight paths created by using the (x, y) coordinates from collected ground truths. The gray arrow indicates the expected start position of the AR.Drone $(0, -3.5)$ and points to the expected position of the visual marker $(0, 0)$. The right figures are a zoomed-in version of the left figures showing the area the start and end-point of a flight in more detail.

4.2 Flight paths

The flight paths for all flight samples that were collected by Astro Drone are shown in figure 11a. During 429 of the 718 collected flight samples, a visual marker was detected. Only flight paths of flights during which a visual marker was detected are shown in figure 11b. Flight paths are calculated by using the x and y components of the coordinates reconstructed by the Astro Drone's Kalman filter (see section 3.2.2).

Given that the game instructs players to start at (x, y)-coordinates $(0, -3.5)$ and fly to coordinates $(0, 0)$, the majority of all detected flights should be located in the area around these two points. As can be seen from figures 11a and 11b this assumption holds true. However, flight paths during which no visual marker was detected by the AR.Drone have a lot more outliers that lie far away from points $(0, -3.5)$ and $(0, 0)$ than flight paths during which a visual marker was detected. This indicates that the Kalman filter successfully integrates information about the marker: When players fly close to a visual marker, it is detected by the AR.Drone which is used by Astro Drone as a fixed point of reference. Therefore, (x, y) of the 3D coordinates (x, y, z) that are computed by Kalman filtering must lie closer to $(0, 0)$ which are the



(a) AR.Drone 1 feature distribution (b) AR.Drone 2 feature distribution

Fig. 12: Distribution of features on the x-y image plane for the AR.Drone 1 and the AR.Drone 2. Histograms on both axes for each plot illustrate the relative feature density along the two image dimensions. Both plots show the corresponding resolution for AR.Drone 1 and AR.Drone 2 images.

virtual coordinates of the visual marker. If no marker is detected, players can fly anywhere with no fixed point of reference which leads to larger variation in flight trajectories. This corresponds to the observations that can be made based on the data shown in figures 11a and 11b.

4.3 Feature distribution

Figure 12 shows the image locations of all collected visual features in all collected flight samples as a scatter plot, split for the two drone types. For both drone types, a lower density of features can be found further in the upper part of an image than in the lower part of an image. The highest feature density is found at the lower third of an image. The density of features over the x-domain declines towards the edges of an image for the AR.Drone 1, while for the AR.Drone 2 the density seems to be more uniformly distributed. The bounding boxes in figures 12a and 12b show a boundary of about 10 pixels to all edges in which no features are found for images from both drones.

The decline in the number of features at higher y values is an interesting observation that I cannot explain. This ”horizon effect” [3] seems to be an intrinsic property of the recorded images. Though, what causes this effect cannot be investigated, since the original images are lost.

The decline in the density of found visual features towards the left and right edges for AR.Drone 1 images can be explained by the bad quality of the front camera of the AR.Drone 1. Images that come from the AR.Drone 1 are generally blurred in the corners whereas the corners are sharp on the AR.Drone 2. This blurriness of the corners would probably also be visible in the visual feature distributions along the y-axis, but is overshadowed by the described ”horizon effect”.

The boundary of 10 pixels around images in which no visual features are detected can be explained by the way SURFs are extracted. SURFs have a minimal size so that SURF descriptors are always computed over a large enough portion of an image that contains enough pixels for the SURF descriptor to carry useful information. The

3. In sceneries where the horizon can be observed, one can usually find more visual details in the lower area of the field of view where one will find features of closer objects. The upper portion of the field of view will be mainly filled with sky where one cannot find lots of different visual features.

minimal size for a SURF seems to be 20 pixels. This creates a boundary of 10 pixels around an image in which no SURFs can be found because otherwise portions of a SURF descriptor would need to be calculated on data outside the image.

4.4 Discussion

Astro Drone was successfully used to collect 718 flight data samples within three months after its release. More than half of these samples stem from AR.Drone flights during which a visual marker was used. This is good because only flight samples during which a visual marker was detected can provide a reliable ground truth estimate towards an approached obstacle.

Flight trajectories lay within a reasonable area around the expected start and docking points. This indicates that the reconstruction of the drone coordinates using Kalman filtering seems to work and that people mainly flew in the area that was relevant for the game.

Visual features were detected over the whole area of AR.Drone 1 and AR.Drone 2 images but the density of detected features varies. In particular, a horizon effect can be observed in the data such that there were generally less features detected in the upper portion of a drone's field of view than in the lower portion for both drone versions (see figure 12).

5 Replication of the appearance variation cue experiment

In 2010, de Croon et al. [6] presented a computationally inexpensive way to let a robot visually detect its proximity to an obstacle, using the so-called *appearance variation cue*. The process to compute the appearance variation cue from an image consists of two steps:

1) The number of different texture types that are visible in an image determined using a texture-type dictionary. This texture type detection uses *textons* [36] to describe the appearance of textures.
2) The Shannon entropy $H(p)$ is computed over the histogram of detected texture types p using equation 2. Thereby, p_i denotes the relative frequency of occurrence of a given feature type i.

$$H(p) = -\sum_{i=0}^{n} p_i \log_2(p_i) \tag{2}$$

De Croon et al. showed that $H(p)$ decreases when a robot approaches an obstacle with a forward-mounted camera. Their explanation for this effect is that when a robot approaches an obstacle, objects that are visible in the peripheral area of the robot's field of view become increasingly occluded by the approached obstacle. Surfaces of objects and obstacles typically consist of only a few textures. Therefore, $H(p)$ decreases because textures of objects in the peripheral area of the field of view drop out of sight. In the end, only the texture of the approached object remains.

However, de Croon et al. also showed that there are exceptions to this relation and that for some texture-rich obstacles $H(p)$ increases when they are approached. For obstacles that show the inverse relation, increasingly more textures come into sight if the obstacle is approached.

To illustrate that the dataset that is collected with Astro Drone can be used for testing obstacle distance measurements that are based on computer vision techniques,

I will try to replicate the results of de Croon et al. using the collected data. For this I develop a custom visual distance measure on the basis of SURFs. This measure is designed using the same principles that de Croon et al. used for creating their appearance variation cue.

5.1 SURF-based distance measure

The SURF-based appearance variation cue measure that I use in this experiment will omit the texture type classification step of the appearance variation cue. Texture classification requires a dictionary that needs to be derived from training data. Deriving the dictionary is a complicated process in itself, adding complexity to the appearance variation cue by including clustering techniques and the selection of proper training data. Instead, I will test if the *number of SURF interest points* that are detected in a given image can already give sufficient information about the proximity to an object.

The underlying idea of this modified appearance measure is similar to the principles behind the original appearance variation cue: For the appearance variation cue, de Croon et al. assumed that the texture of an approached object becomes dominant in a robot's field of view as the robot approaches the object. Therefore, the entropy over the distribution of detected different textures declines the closer the robot gets to an object. The modified measure that I propose is based on the same idea. It assumes that visual details like contours of objects fall out of sight when the approached object occludes other objects. Therefore, the number of detected SURFs should decline with the proximity to an object.

5.2 Feature compression problems

Trying to test if the number of extracted SURFs is a good measure for the object distance with data collected by Astro Drone poses one problem: The compression method described in section 3.2.3 limits the number of saved features to a maximum of 125 per image. This might introduce a ceiling effect in the collected data for images where SURF counts were very high. To mitigate this ceiling effect an alternative measure will be tested, too, which consists of the *summed SURF keypoint strengths* for an image.

This measure should partially suppress the ceiling effect, because if more than 125 features are found only the 125 strongest features are selected. This introduces an explicit selection bias. If more than 125 features are detected in an image, their summed feature strength will on average be higher than the summed feature strength for images where only 125 or less features are detected. I test this measure as an alternative indicator for object distances.

In the next sections, I will present experiments to investigate if the modified SURF-based distance cues (the number of detected SURFs or the sum of feature strengths) can be used to calculate the distance to an object. In a first experiment, I will try to show that values for both proposed variants of the SURF-based distance cue decline when closing in on an object. If it can be shown that at least one of both measures can be used for distance approximations to an object, I will try to replicate the results from the first experiment using data collected by Astro Drone. Both experiments are carried out in separate for the AR.Drone 1 and AR.Drone 2 data because different resolutions of the recorded images could lead to different results.

(a) Wall-video (b) Cabinet-video

(c) Bookcase-video (fast) (d) Bookcase-video (slow)

Fig. 13: The four videos recorded for the qualitative analysis of the new proposed distance measure based on SURFs. The small rectangles show the end position for each video.

5.3 Qualitative analysis

In this qualitative experiment I investigate if the number of SURFs and sum of feature strength decline when approaching an obstacle. I use four hand-recorded videos as testing data. The videos were recorded using an iPad 2. I started approximately 3.5 meters away from an obstacle and then I walked at a constant speed towards it while recording a video with the iPad pointing towards the obstacle. In total, I created videos of three different sceneries that are shown in figure 13:

1) The wall-video shows a feature-rich scene. The images in the surrounding area fall out of sight while I approach the concrete wall. The wall itself features a porose texture which comes into sight at closer distances and could lead to an increase in extracted SURFs.
2) The cabinet-video shows a scene with few features, and I approached an area that also only has few visual features. A decline in the number of SURFs is expected in this case.
3) The bookcase-video features another feature-rich scene. In this case visual features have a low contrast in the beginning of the video because of poor lighting conditions. As I approach a light spot on the bookcase, the lighting conditions improve and visual features become more pronounced regarding their contrast. This last video should be the most challenging one because it is designed as a counterexample for which both proposed SURF-based distance cues should not work.

In addition to the three different scenes, I recorded two versions of the bookcase-video. I did this because while recording the different videos, I noticed motion blur effects in the fast version of the bookcase-video. To further investigate the motion blur effects on contour-rich sceneries, I recorded two similar videos of the bookcase at different walking speeds. The slower video is recorded at approximately a tenth

of the walking speed of the faster video to reduce the observed motion blur.

The videos were recorded with a resolution of 1280x720 at frame rate of 30 frames per second. I scaled each video to resolutions of 320x240 and 640x360 to meet the respective resolutions of the AR.Drone 1 and AR.Drone 2. After scaling, I computed the SURFs for each frame of the videos using OpenCV [3] with the same settings Astro Drone uses to extract features (see table 1). From the collection of extracted SURFs, I calculated the number of extracted SURFs and the summed feature strength from the response values given by the SURF keypoint detector.

5.3.1 Results

Figure 14 shows the development of the number of detected SURFs and the development of the summed feature strengths in relation to the frame number for each recorded video. Results of videos resampled to resolutions of the AR.Drone 1 and AR.Drone 2 are shown in separate. Video frames with higher frame numbers represent images closer to the approached obstacle.

The graphs show that there is no clear difference in the behavior of the proposed cues for the two different video resolutions. The two proposed SURF appearance cue values seem to be scaled by a factor of 2 for frames with AR.Drone 2 resolution, if compared to values for frames with AR.Drone 1 resolution.

A clear decline for both proposed SURF appearance cue values can be observed for the wall- and cabinet-videos. For the bookcase-video that was recorded at a fast walking speed, a decline can only be observed for the last 10 frames. For the bookcase-video that was recorded at a slow walking speed, no clear decline of the cues can be observed. Instead, a steady increase in the number of detected SURFs and summed feature strengths can be observed over the whole video.

5.3.2 Interpretation

For most of the recorded videos a decline in the number of detected SURFs can be observed. In the bookcase-video that was recorded at fast walking speed, the decline occurs very late during the last 10 frames. The shapes of the two curves of the number of extracted SURFs and the summed feature strengths are similar for all image resolutions in all videos. Therefore, the number of extracted SURFs and the summed feature strength seem to be both appropriate measures for proximity to an obstacle.

The bookcase-video where I walked slowly does not show a decline in the number of detected SURFs or summed feature strengths, but instead an increase. This corresponds to the expectations regarding the video because of how the video was designed.

However, a question that should be discussed about the bookcase-video regards the difference between the slow and fast walking speed versions of the videos. The video recorded at slow movement speed shows the effect that was expected from the beginning, displaying a slight increase in the number of detected features when closer to the bookcase. In the video that was recorded at a faster movement speed, the number of detected SURFs seems to stay constant until the last 10 frames where it starts to decline. This difference can be explained by motion blur effects. In the video recorded at slow movement speed, more details of the bookcase are visible when it is approached. In the video recorded at fast movement speed, visual details of the bookcase are affected by motion blur. The closer the camera comes to the bookcase, the faster objects move through the camera's field of view. This increases

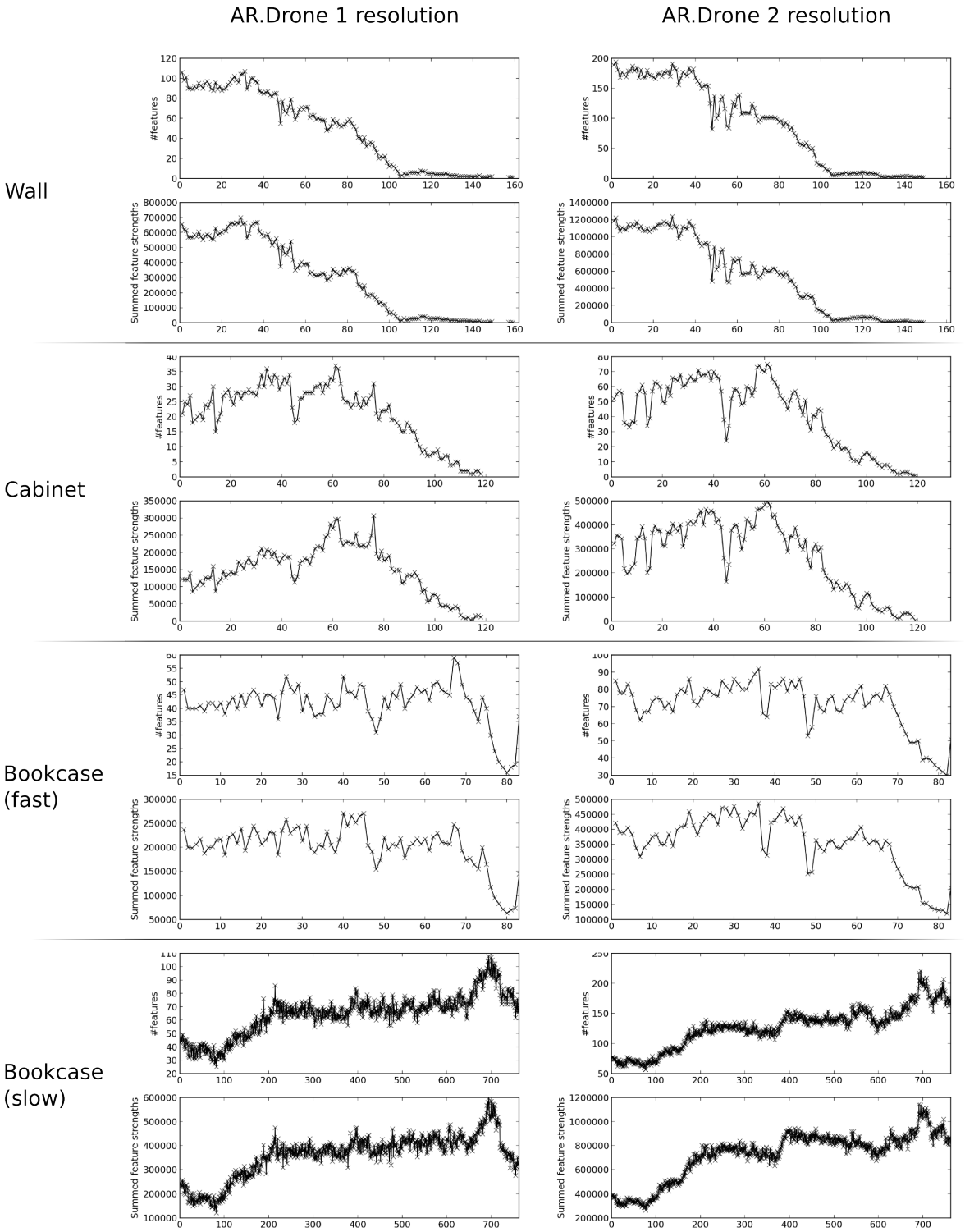


Fig. 14: Results of the qualitative analysis of the two proposed distance cues: The numbers of extracted features and the summed feature strengths. The x-axis in every plot shows the frame number of the corresponding video.

visual motion blurring of such objects and is responsible for the decrease in the number of detected SURFs in the last 10 frames.

5.4 Visual appearance cue in the Astro Drone dataset

In the first experiment that I did, I was able to show that both proposed distance cues, the number of detected SURFs and the sum of feature strengths, can be used to determine the proximity to an obstacle. In this second part of the experiment, I try to show that the same effect can be replicated with data collected by Astro Drone.

Before the experiment can be replicated with Astro Drone data, a subset of Astro Drone flight samples must be selected that contains visual information about an approached object. This selection is done using four criteria:

To select a flight sample from the Astro Drone dataset, the drone must

- have detected the visual marker during its flight to make sure that people were flying close to the object (that the marker has been stuck to),
- have been closer than 5m to the marker for all five data points of a flight sample so that samples fall in an area in which the decline in appearance variation can be observed,
- have flown at least 30cm in total according to the Kalman-filter reconstructed coordinates to make sure that the visual data covers a sufficient large range of different distance values from the marker, and
- have flown towards an area around $0.5\,\mathrm{m}$ left or right from the visual marker to make sure that people were pointing towards the marker while flying.

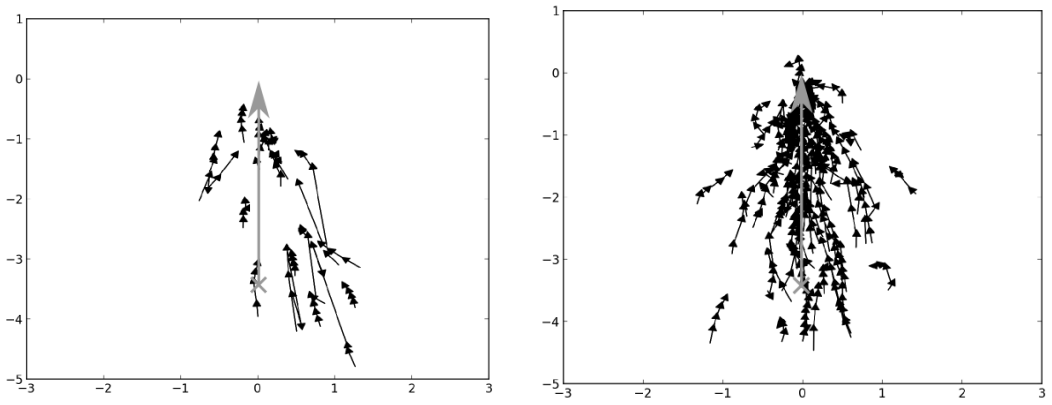
The filtered flight samples are split in two groups depending on the drone version (AR.Drone version 1 and 2). For each flight i, which consists of 5 samples of SURFs taken at 5 different positions, I compute two types of regression lines $f(d) = ad + b$, where d denotes the Euclidean distance to the visual marker: One type of regression line is computed between d and the number of extracted SURFs ($\#SURFs$) and the other between d and the summed feature strengths ($\sum strengths$). The Euclidean distance to the marker is computed using $d = \sqrt{x^2 + y^2}$ where x and y are the (x, y) coordinates from the Kalman filter (see figure 9). Both types of regression lines computed for each flight sample i yield a slope: $a(\#SURFs_i)$ and $a(\sum strengths_i)$.

The effect that I want to observe says that there are less detected SURFs to find in an image the closer to an object it is taken. Rephrasing this relation by using the *distance* to an obstacle means that I look for an *increase* in the number of features with increasing distance to an obstacle. Therefore, the slopes of regression lines computed for each flight sample for both proposed distance measures ($a(\#SURFs_i)$ and $a(\sum strengths_i)$) are expected to be positive on average. Given that $\mathrm{M}(v)$ denotes the median of a distribution of values v_i, this leads to the following two formal hypotheses:

$$\mathrm{M}\left(a\left(\#SURFs\right)\right) > 0 \tag{3}$$

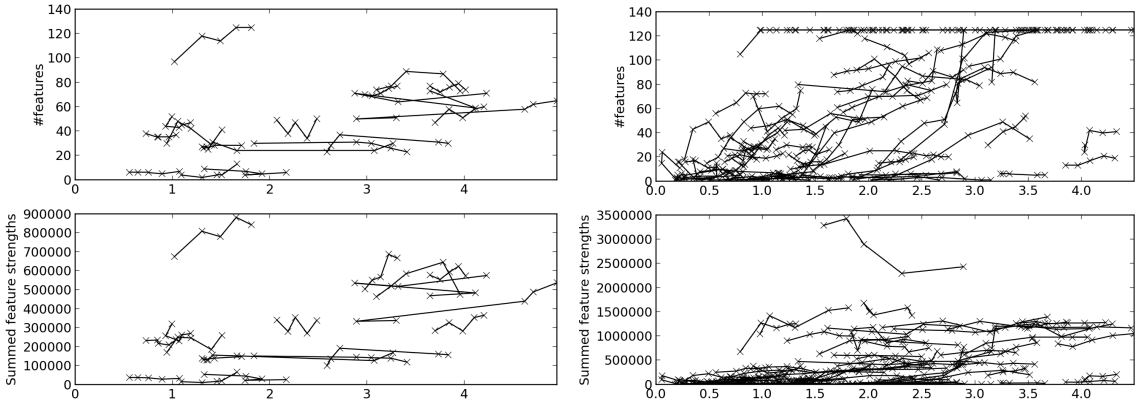
$$\mathrm{M}\left(a\left(\sum strengths\right)\right) > 0 \tag{4}$$

Since the distributions for the two types of computed slopes $a(\#SURFs_i)$ and $a(\sum strengths_i)$ are unknown, I will use *bootstrapping* [8] to calculate confidence intervals over the distributions of slopes for each distance measure. For this, I resample the calculated slopes 100000 times to derive the bootstrap distributions for both types of slopes. For calculating the confidence intervals, I will use a 5% probability threshold to test my hypotheses.



(a) AR.Drone 1 flight paths (b) AR.Drone 2 flight paths

Fig. 15: Flight paths of a subset of Astro Drone data. The subset was created by selecting flight samples for which the AR.Drone flew towards a visual marker.



(a) AR.Drone 1 data (b) AR.Drone 2 data

Fig. 16: The number of detected SURFs and summed feature strengths plotted against the distance to an object in meters (x-axis). Each line represents data from one flight sample collected by Astro Drone. The data is split for AR.Drone 1 and AR.Drone 2 data samples.

5.4.1 Results

Astro Drone collected 160 flight samples from AR.Drones version 1 and 556 form AR.Drones version 2. After filtering the flight samples for samples that approached an obstacle, there were 18 flight samples left from the AR.Drone 1 samples and 92 flight samples were left from the AR.Drone 2 samples.

Figure 16 shows the number of extracted SURFs and the sum of extracted feature strengths in relation to the measured obstacle distance for both drones. For the AR.Drone 2 data, the expected ceiling effect can be seen in the number of extracted features at distances larger than $1\,\mathrm{m}$ from an obstacle. The ceiling effect seems weaker

TABLE 3: Statistical measures for slopes $a\,(\#SURFs_i)$ and $a\,(\sum strengths_i)$

Slope for	Drone version	Lower bound ($p = 0.025$)	Lower bound ($p = 0.05$)	Median	N
$\#SURFs$	AR.Drone 1	0.535	1.122	4.824	18
	AR.Drone 2	7.527	8.371	23.825	92
$\sum strengths$	AR.Drone 1	13729.816	19264.262	62310.921	18
	AR.Drone 2	40739.241	48933.334	91248.406	92

for the sum of feature strengths measure.

The statistical measures calculated by bootstrapping for testing the hypotheses shown in equations 3 and 4 are shown in table 3. The median of the bootstrap distribution and the lower confidence bounds for the median computed with $p = 0.025$ and $p = 0.05$ are shown. The lower confidence bound with a given p says that a median that is smaller than the bound can only be observed with a probability of less than p when sampling from the population.

This means that only with a probability of less than 2.5% the median of any of the two slopes, $\mathrm{M}\left(a\left(\#SURFs\right)\right)$ and $\mathrm{M}\left(a\left(\sum strengths_i\right)\right)$, is smaller than zero. This holds true for data from both AR.Drone versions. This result supports the hypotheses 3 and 4.

5.4.2 Interpretation

As can be seen from the presented confidence boundaries, it is unlikely ($p < 0.025$) that $\mathrm{M}\left(a\left(\#SURFs\right)\right) <= 0$ and that $\mathrm{M}\left(a\left(\sum strengths_i\right)\right) <= 0$ for features recorded by both drones.

Both hypotheses 3 and 4 are supported by the statistical analysis, which shows that both proposed measures $\#SURFs_i$ and $\sum strengths_i$ work for estimating the proximity to an obstacle: The values of both measures, the number of detected SURFs and the summed feature strengths, increase with increasing distance to the marker. The effect is similar to the effect found by de Croon et al. in [6]. The effect was found for the number of detected SURFs despite a ceiling effect for AR.Drone 2 images that can be seen in figure 16.

5.5 Discussion

Using the number of detected SURFs and sum of feature strengths for distance approximations to an obstacle is based on the same principles as the appearance variation cue that is described in [6]. In my qualitative experiment, I showed that both proposed SURF-based distance cues do not work reliably for distance approximations in special cases. However, I was also able to show qualitatively that both cues can work in common cases.

In my second, quantitative experiment I tested both SURF-based distance cues with data collected by Astro Drone. I showed again that values for both cues decline with the proximity to an obstacle. Testing the summed feature strengths as alternative distance measure proved to be unnecessary. Despite a found ceiling effect for the number of detected SURFs in AR.Drone 2 images, the number of detected SURFs still showed a significant positive correlation with the distance to an object.

In the first experiment, I showed that the number of detected SURFs can be used as a visual measurement for the distance to an object. In the second experiment, I showed that in the Astro Drone dataset a positive correlation can be found between the number of detected SURFs and the distance computed on the basis of coordinates reconstructed by the Kalman filter. This means that the distances that can be computed from Kalman coordinates also encode information about the distance to an object or obstacle. However, the quantitative accuracy of distances that can be calculated from Kalman reconstructed coordinates need to be investigated in a future experiment.

6 Conclusion

In this thesis, I describe the development of Astro Drone, a game that uses crowdsourcing to collect testing data for visual distance estimation techniques. The game uses a Parrot AR.Drone to collect two types of data. First, visual features (SURFs) of objects are detected from front camera images of the AR.Drone. Second, the game calculates a ground truth which is the actual distance to objects that SURFs are collected from. The data is collected with the intention to determine the accuracy of visual distance estimation techniques by comparing distance estimates of a technique to the ground truth data. As part of the work for this thesis, it was investigated if the data collected by Astro Drone is valid and can be used for such accuracy testing. It was also investigated if the time investment in creating Astro Drone can be justified by the amount of data it has collected.

The first research question is if the data that Astro Drone collects is valid. More precisely, the question is if the collected ground truth distance measurements are correct. To show that ground truth data depends on distances to an object, a custom visual estimate for the proximity to an object was developed that uses SURFs as input. The estimate was developed on the basis of prior work on the so-called appearance variation cue [6]. I demonstrated that the measure correlates to the distance to an object based on a self-collected set of data. I then showed that the measure when calculated on data collected by Astro Drone also correlates with collected ground truth distances. This indicates that the ground truth estimate that is collected by Astro Drone contains valid information about the distance to an object. Moreover, the data collected with Astro Drone show a phenomenon that is similar to a phenomenon found in previous research [6]: The number of SURFs that are detected in images decreases with the proximity to an object. To summarize, the data that is collected by Astro Drone appears to contain valid information. However, more experiments must be done to test the accuracy of distance values quantitatively. Only if the accuracy of the ground truth measure is known, it is viable to use it as a benchmark for vision-based distance estimates.

The second research question is if data collection via crowdsourcing is an efficient approach considering the complexity of developing an application like Astro Drone. To answer it, the development time of Astro Drone needs to be compared to the number of currently collected flight samples. The development of Astro Drone took one year and since its release three months ago, Astro Drone collected 718 flight samples. An equal number of flight samples could have been hand collected in less time. Therefore, collecting data using crowdsourcing seems to be time inefficient for short-term research projects. However, for collecting a test dataset, it is an effective approach: The dataset collected by Astro Drone is already larger that test datasets used in many studies (e.g. [6], [26]). Since Astro Drone has only been released a few months ago, more data can be expected in the future. Also, Astro Drone data is collected from different people in different locations. This makes the dataset unique: People in different places can easily contribute visual data from different visual environments. This likely increases the variation of collected data in comparison to data that is collected in a laboratory. The variation in the collected visual data was not tested in this thesis but should be tested in a future experiment. Furthermore, Astro Drone offers an easily extensible platform which can collect visual data with distance measurements. The game can be extended for future experiments in robotics to record additional data that might prove to be useful. The planned porting of the game to Android devices will eventually cause more people to play the game. New

interest in playing the game can be raised by adding new content like new levels. These capabilities can facilitate the data collection process of future experiments.

To conclude, in this thesis, I tested a new method for collecting data using crowdsourcing. I showed that valid data for an experiment can be collected using crowdsourcing and that using crowdsourcing can be efficient depending on the time frame of an experiment. Future experiments may reveal extra utility in collecting visual data from people at different locations. Also, future research projects may benefit from the developed game by modifying it, which allows them to collect research data efficiently using crowdsoucing.

REFERENCES

[1] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Computer Vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[2] C. Bills, J. Chen, and A. Saxena, "Autonomous MAV flight in indoor environments using single image perspective cues," in *Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE, 2011, pp. 5776–5783.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] M. Buhrmester, T. Kwang, and S. D. Gosling, "Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data?" *Perspectives on Psychological Science*, vol. 6, no. 1, pp. 3–5, 2011.

[5] D. Crockford, "RFC 4627 - the application/json media type for javascript object notation (JSON)," Internet Engineering Task Force, Tech. Rep., 2006. [Online]. Available: http://tools.ietf.org/html/rfc4627

[6] G. C. H. E. De Croon, E. De Weerdt, C. De Wagter, and B. D. W. Remes, "The appearance variation cue for obstacle avoidance," in *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, 2010, pp. 1606–1611.

[7] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, 2001.

[8] B. Efron, "Bootstrap methods: another look at the jackknife," *The annals of Statistics*, pp. 1–26, 1979.

[9] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2. IEEE, 2003, pp. II–264.

[10] P. K. Gerke, J. Langevoort, S. Lagarde, L. Bax, T. Grootswagers, R. J. Drenth, V. Slieker, L. Vuurpijl, P. Haselager, I. Sprinkhuizen-Kuyper, M. van Otterlo, and G. de Croon, "BioMAV: bio-inspired intelligence for autonomous flight," in *Proceedings of the International Micro Air Vehicle conference and competitions*, 2011, pp. 12–15.

[11] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50.

[12] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1814–1820.

[13] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in *Proc. IEEE International Symposium of Robotics Research (ISRR)*, 2011.

[14] *754-2008 - IEEE Standard for Floating-Point Arithmetic*, IEEE, 2008.

[15] A. H. Jazwinski, *Stochastic processes and filtering theory*. Academic Press (New York), 1970.

[16] S. Josefsson, "RFC 4648 - the base16, base32, and base64 data encodings," Internet Engineering Task Force, Tech. Rep., 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4648.txt

[17] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[18] R. Labayrade, D. Aubert, and J.-P. Tarel, "Real time obstacle detection in stereovision on non flat road geometry through," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2. IEEE, 2002, pp. 646–651.

[19] D. N. Lee and H. Kalmus, "The optic flow field: The foundation of vision [and discussion]," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 290, no. 1038, pp. 169–179, 1980.

[20] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.

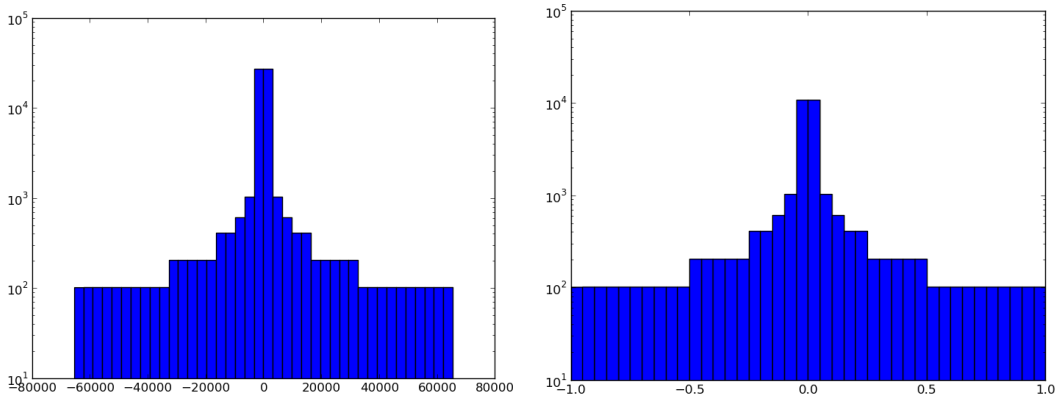
[21] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. IEEE, 1999, pp. 1150–1157.

[22] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 7, pp. 674–693, 1989.

[23] D. Marr, T. Poggio, E. C. Hildreth, and W. E. L. Grimson, *A computational theory of human stereo vision*. Springer, 1991.

[24] D. Murray and J. J. Little, "Using real-time stereo vision for mobile robot navigation," *Autonomous Robots*, vol. 8, no. 2, pp. 161–171, 2000.
[25] W. S. Ng and E. Sharlin, "Collocated interaction with flying robots," in *RO-MAN, 2011 IEEE*. IEEE, 2011, pp. 143–149.
[26] D. Nister, "An efficient solution to the five-point relative pose problem," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 2, 2003, pp. II–195–202 vol.2.
[27] Parrot, "Parrot establishes itself on the civil drones market," 2013. [Online]. Available: http://www.parrot.com/paris-air-show-2013/usa/bg-press-release.pdf
[28] I. Pavlov. (2010) LZMA SDK (software development kit). [Online]. Available: http://www.7-zip.org/sdk.html
[29] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, 2006, pp. 519–528.
[30] A. Shalloway and J. Trott, *Design Patterns Explained: A New Perspective on Object-Oriented Design (2nd Edition) (Software Patterns Series)*. Addison-Wesley Professional, 2004.
[31] R. Sim, P. Elinas, M. Griffin, J. J. Little *et al.*, "Vision-based SLAM using the Rao-Blackwellised particle filter," in *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, vol. 14, no. 1, 2005, pp. 9–16.
[32] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 1470–1477.
[33] G. Takacs, V. Chandrasekhar, S. Tsai, D. Chen, R. Grzeszczuk, and B. Girod, "Unified real-time tracking and recognition with rotation-invariant fast features," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 934–941.
[34] J. T. Todd and P. Bressan, "The perception of 3-dimensional affine structure from minimal apparent motion sequences," *Perception & Psychophysics*, vol. 48, no. 5, pp. 419–430, 1990.
[35] Unknown. (2013) GSM arena technical specifications for the iPhone 3GS. [Online]. Available: http://www.gsmarena.com/apple_iphone_3gs-2826.php
[36] M. Varma and A. Zisserman, "Texture classification: Are filter banks necessary?" in *Computer vision and pattern recognition, 2003. Proceedings. 2003 IEEE computer society conference on*, vol. 2. IEEE, 2003, pp. II–691.

APPENDIX A
VALUES OF FEATURE DESCRIPTORS



(a) All half-floats (b) Interest interval $[-1; 1]$

Fig. 17: Histograms of values that can be represented by half-floats. Figure 17a shows the histogram of all values half-floats can represent. Figure 17b shows only values in the interval of $[-1; 1]$.

The compression of flight sample data that is presented in section 3.2.3 uses lossy compression of SURF descriptors. The compression is based on representing 32 bit floating point values as 16 bit floating point values, also called half-floats. In this section, I will show that half-floats still offer enough precision to retain most of the discriminative power of SURF descriptors.

The lossy compression that is used for feature descriptors has impact on its values. Half-floats use 16 bits to encode a floating point value, which can, theoretically, encode 65536 different values. SURF descriptors consist of unit vectors, so the length of the vector consisting of all 128 descriptor values has a length of one. As a result, all descriptor values lie in the interval $[-1; 1]$. Half-floats encode 30722 different values in this interval. The resolution of half-floats increases for values closer to zero. This effect is illustrated in figure 17 which shows histograms of the number of values that a half-float can represent in a given interval, excluding *Nan* (not-a-number) and *Inf* (positive or negative infinity) values.

On the interval $[-1; 1]$, the average distance from one half-float to the next is 0.000065102 with a standard deviation of 0.00013 [4]. The maximum difference between a pair successive of half-floats is 0.000488.

The order of magnitude of the component-wise distances between two arbitrary SURF feature descriptors thereby often reaches 10^{-1} (tested on a subset of the data collected by Astro Drone). This means half-floats have enough resolution to encode differences between SURF descriptors. During the half-float based compression of SURF descriptors only little information is lost. Differences between SURF descriptors still are visible because of the relatively high precision of half-floats in the interval $[-1; 1]$. Therefore, half-floats suffice to encode data from SURF descriptors without influencing their discriminative power too much.

4. Please note that the distance values are not distributed according to a normal distribution. The values are provided to give the reader an idea of the effective distances between successive half-float values.