Panoramic image construction and height mapping using a quadcopter

(Bachelorthesis)

Robert Musters, s1774395, a.b.musters@rug.nl,
Marco Wiering*,
Tijn van der Zant*

August 14, 2013

Abstract

This thesis describes the use of SIFT (Scale Invariant Feature Transform) and SURF (Speeded Up Robust Features) feature detection algorithms to make in flight panoramic pictures on a quadcopter. The quadcopter is an affordable UAV (Unmanned Aerial Vehicle) called AR.Drone from the company Parrot. We concentrate on indoor use, so GPS location systems can not be used. A single forward motion path is chosen for the making of the panoramic pictures. We chose the single path to decrease the problem complexity. At the same time it also constructs a height map using its onboard ultrasonic sensor. We also describe basic techniques/concepts as: sensors in- and output, obstacle avoidance and a PID controller, which can be used as a start up to real-world simultaneous localization and mapping (SLAM).

1 Introduction

Robot vehicles are often used to search dangerous or unreachable areas, like the meltdown of the Fukushima nuclear reactor or the New Orleans earthquake. Here ground vehicles equipped with cameras, means of communication and some kind of world altering actor are used to map the environment or execute some simple tasks. Unfortunately these vehicles are difficult to be moved into the third dimension. Because of new technological advancements and the increasingly cheaper and smaller becoming electronics, it is more feasible to use Unmanned Aerial Vehicles (UAVs) to do these tasks. UAVs have the advantage that they are cheap and agile, so they can move in small and narrow spaces. Another important phenomenon nowadays is the mapping of urban environments by Google and Microsoft. Capturing 3D images on large surfaces makes this task difficult and time consuming. By deploying multiple UAVs it is possible to map large surfaces in parallel. With onboard ultrasonic sensors it is also possible to map heights. Because these UAVs can be produced at low cost, they can be used to solve these problems. They can also be used for educational purposes and research. Applications could be fire monitoring, crowd-control, person following and simultaneous localization and mapping (SLAM).

In our research on image stitching we will use the Parrot AR.Drone, a four propeller based UAV. It runs on battery power and communicates through Wi-Fi. This way it becomes more environment friendly, produces less noise and would be able to map the environment without GPS.

We try to answer the following questions:

1. Can the two feature detection algorithms, SIFT and SURF, be used in a stitching algorithm with a quadcopter to construct panoramic images?

2. Can we create a height map with a quadcopter? This is meant as a pilot study to 3D position estimation and mapping.

3. Can we use external hardware, like an ultrasonic sensor, to compensate sensory drift?

*University of Groningen, Department of Artificial Intelligence

[**Outline**] In section 2 we explain further what the AR.Drone is. We do a pilot study on the use of external hardware. This hardware consists out of an Arduino Nano, an open-source electronic prototyping platform, and an ultrasonic sensor. This sensor was meant to study the sensory drift of our quadcopter. We need the quadcopter to fly a straight path to be able to stitch the images correctly. So we need to know the deviation and counter this using motor commands. The calculation of the correct motor command is done with a PID controller.

In section 3 we show how two feature detection algorithms (SIFT and SURF) can be used for image stitching, resulting in a panoramic image of the ground. These features can also be used for object detection and localization. The goal was to map the whole environment but this seemed too complex. For example we had to deal with sensory drift, movement in three dimensions, also working on a robot platform presented problems, e.g. closed source software, delay in sensor in- and output. We also explain descriptor matching techniques and filtering keypoints. At the end we show how we construct a height map, this is meant to be a pilot study into 3D position estimation and 3D mapping.

In section 4 we mention the experimental setup and results. We discuss the influence of three terrain types on the performance of the stitching algorithm: homogeneous, repetitive heterogeneously structured and heterogeneously structured. Also we discuss the parameters of the feature detection algorithms and the influence on the performance.

In section 5 we reflect on our research, answer our research questions and give future recommendations.

The purpose of this article is first to explain the computer vision problem of image stitching on a quadcopter, but our research is also intended as a step up to SLAM, to learn techniques to use and to understand the problems we may face.

2 System

In this section we describe what robot platform is used. We show a way to make this robot platform able to follow a wall with an Arduino Nano and an ultrasonic sensor. Next we show what software we use and at the end we describe techniques to cope with sensory drift.

2.1 AR.Drone

The AR.Drone from the French company Parrot is a remotely controlled quadcopter. A quadcopter is a kind of helicopter with four propellers, as can be seen in figure 1*.

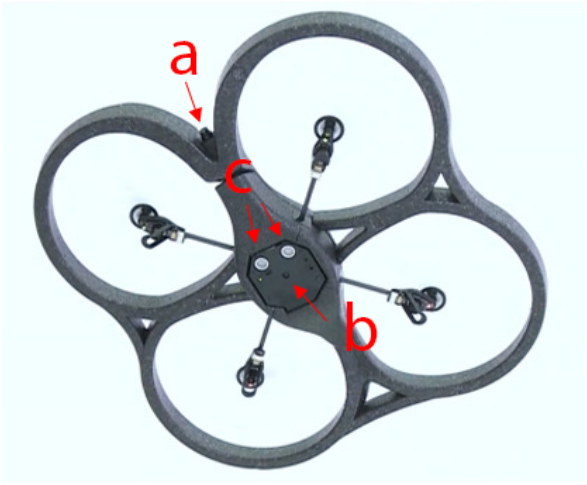

Figure 1: The AR.Drone quadcopter with a) the front camera, b) the bottom camera and c) the ultrasonic sensors.

This quadcopter can be controlled through Wi-Fi, making it easy to control from your personal computer. To keep it stable in flight it uses several sensors. These sensors include 3-axis accelerometer, 2-axis gyrometer, single axis precision gyrometer, 2 ultrasonic sensors and two cameras. The sensors have been validated to be accurate (Dijkshoorn, 2012). The cameras are a VGA (640x480) pixel front camera with a frame-rate of 15 frames per second and a camera mounted on the bottom of the quadcopter (see figure 1 a and b). The bottom camera is a (176x144) pixel camera (Bristeau, Callou, Vissiere, and Petit, 2012) with 60 frames per second. The 2 ultrasonic sensors together with the bottom camera are used to calculate the vertical speed and height (see figure 1 c).

2.2 Python and C++

We use Python because it is a simple programming language mostly used in the robotics field. Also a

*http://static.geekbeat.tv/wp-content/uploads/2010/10/parrot_ARDrone_helicopter1.jpg

Python library for controlling the quadcopter was already implemented by Bastian Venthur [†]. This library contains all the commands needed to move the quadcopter. It also implements the communication and video processing. We incorporated this library at certain places in the BORG architecture, which is the subsumption architecture used by the Cognitive Robotics Laboratory of the University of Groningen. A subsumption architecture (Brooks, 1986) is a layered behavior architecture where complex behaviors are built on basic behaviors. These layers can communicate and compete.
The feature detection algorithms in OpenCV version 2.4.6.0[‡] for Python are not as developed as the C++ version. For example, SIFT was not available in Python and SURF was incomplete. Python lacked int the completeness of algorithms for feature detection, so it was also a necessity. Our algorithm consists out of several steps shown in pseudocode, see algorithm 1. The flow of data is represented in figure 2.

2.3 Sensory drift

Sensory drift is the accumulation of error through time because of uncertainties and noise in sensors. We think sensory drift is present on the quadcopter, because we observed the quadcopter not flying a straight path in a controlled environment. Research shows that counteracting this effect with machine learning is effective (Cooper and Jason, 2010). In this research (Cooper and Jason, 2010) it is the case that you train the robot with a closed world assumption, meaning the environment stays the same and wear and tear of components is not taken into consideration. But in the real open-world this is not the case. This is why we introduce sensors.

2.3.1 Arduino and sensor addition on AR.Drone

To detect and counteract sensory drift when the quadcopter tries to fly a straight path, we made an easy and affordable distance calculating setup (figure 3). It is affordable since the total cost was only about 20 euros for the Arduino Nano, the ultrasonic sensor and the connection materials.

[†] www.github.com/venthur/python-ardrone

[‡] `http://docs.opencv.org/modules/nonfree/doc/feature_detection.html`

Data: Greyscale images
Result: All images stitched
initialization;
while *Images available* **do**
  read current image;
  extract keypoints and descriptors from image;
  if *Keypoints found* **then**
    Try to find matches between descriptors with brute force L1 or L2 norm distance measurement (These distance measurements are described in section 3.3 and formulas 3.4 and 3.5);
  else
    print no keypoints found;
  end
  if *Matches found* **then**
    Calculate distance between keypoints;
    Write and append distance to a file;
  else
    print no matches found;
  end
  Stitch images by using distance as overlap;
  Use resulting image to stitch the next image;
end

Algorithm 1: Pseudocode showing the basics of our stitching algorithm.

We mounted 1 ultrasonic sensor on top of the quadcopter, directed to the left. This way it will be able to, albeit crudely, detect obstacles from one side and thus the wall. The ultrasonic sensor is connected to the Arduino Nano, a programmable controller. The distance is calculated by:

$$s = \frac{34.3 * t}{2.0} = 17.2 * t \tag{2.1}$$

34.3 (cm/ms) is the speed of sound. The time t is divided by 2.0 because the sound travels from the quadcopter, bounces from the obstacle, and returns to the quadcopter.

With these sensors, and the BORG subsumption architecture used (see section 2.2), a more robust and reliable behavior can be realized. But as noise and uncertainty remain, techniques to filter noise and cope with uncertainty in sensor data need to

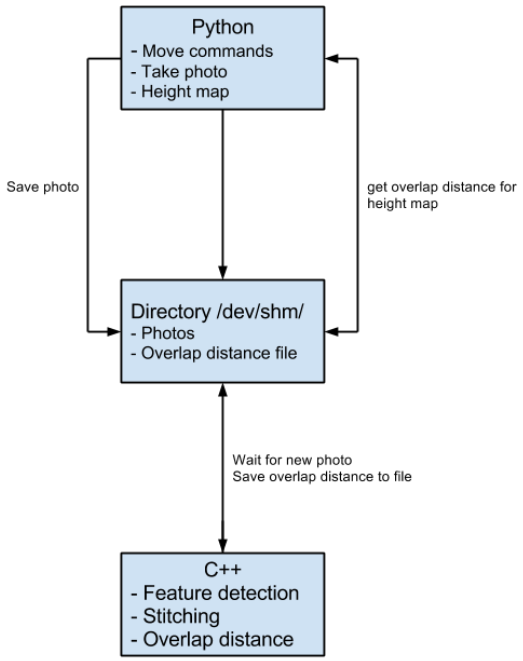

Figure 2: Flow chart of how the algorithm sends data to the python and C++ modules.

be implemented. These techniques will be discussed next.

2.3.2 Queue

We implemented a queue for incoming sensor data. A queue is an abstract data type which takes a certain amount of data. We've chosen the queue to be of size five. This size was chosen because when the queue is divided by five, a temporal effect is represented. The temporal effect is achieved because every value in the queue is measured with a couple of milliseconds in between. The value of five gave the best temporal effect. When the queue of this amount is full, a data item is dequeued and the new data item is enqueued according to the first-in first-out principle. The quadcopter wouldn't take off if the queue wasn't full. We put the quadcopter at a fixed distance (40 cm from the center of the quadcopter) of the wall and tried to make it fly in a straight line, counteracting the sensor drift. The

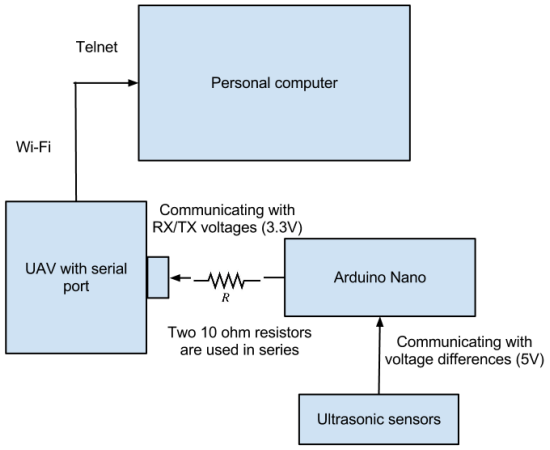

Figure 3: Scheme of the components and how they are connected

queue didn't exactly do what we wanted, because a straight forward flight path wasn't maintained. We already discarded invalid values, but a better solution had to be found.

2.3.3 Proportional Integral Derivative (PID) controller

A better solution to the problem of getting the quadcopter to fly a straight path and get sensible data, is to take into account the present, past and future data (Bennett, 1993). This is exactly what a PID feedback controller does, it depends on the present error, on the accumulation of past errors and a prediction of the future errors based on the current rate of change. The formula's output is the error over time, which is the ideal value minus the measured value. The ideal value is, in this case, the distance from a wall which the quadcopter wants to maintain. The error is how much centimeters the quadcopter is away from the ideal distance value.

$$output(t) = error(t) = (ideal\ value - measured\ value)(t)$$

The formula can be split up into its three components: the proportional, integral and derivative terms:

$$output(t) = K_p error(t) + K_i \int_0^t error(\tau)\delta\tau +$$

$K_d \frac{\delta}{\delta \tau} error(t)$

For now we only implemented the proportional part, K_p, since the tuning and implementing of the rest of the components in three dimensions does not have the main focus in this article. The PID controller approach gave a better result for the wall-following behavior as we show in figure 4, because the ultrasonic sensor data to measure its distance from the wall is better represented. Without the PID controller it would simply fly into the wall. Now the horizontal displacement stays about the same, but we need a robust feature detection and matching algorithm to make panoramic image construction possible.

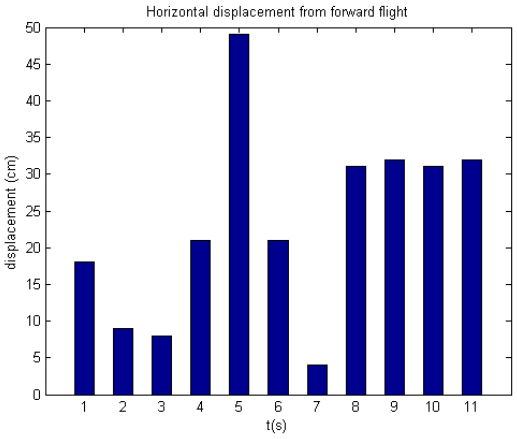

Figure 4: Graph showing the displacement distance when the quadcopter is moving forward with proportional error correction. The greater the value, the greater the displacement from the starting distance from the wall (40 cm from the center of the quadcopter). The last four seconds show a stable horizontal displacement.

3 SIFT, SURF and height maps

In this section we explain step by step how the feature detection and extraction algorithms SIFT and SURF work. We explain the difference between SIFT and SURF and discuss feature matching techniques. At the end, the construction of a height map is described.

3.1 SIFT (Scale Invariant Feature Transform)

SIFT can be used to extract features and match them between images (Lowe, 1999; Suzuki, Amano, Hashizume, and Suzuki, 2011). These features are abstract representations of meaningful pieces of data from the image. Keypoints are selected by comparing all the pixels with surrounding pixels and choosing the ones with the most variation.
SIFT has six interesting properties:

1. Scale invariance, not like the Harris corner detector for example, (Harris and Stephen, 1988).
2. Rotation invariance, it allows up to a 20 degree rotation in 3D
3. Illumination invariance
4. Viewpoint independency
5. Biological plausibility (Lowe, 1999)
6. Real-time feature detection and extraction

These properties are important in the field of computer vision and robotics. We also choose SIFT because it has a biological background. In primate vision (neurons in the inferior temporal cortex), the neuron responses share properties with SIFT features. Also certain features are detected, like edges and corners (Lowe, 1999).

3.1.1 Steps in the SIFT algorithm and parameters

Constructing scale space

To acquire scale, rotation and illumination invariance, a search space of several scales and blurred images is created. The scale space is dependent on the number of octave layers. The larger the number of octave layers, the larger the scale space. An octave represents a number of blurred images. The amount of octave layers represent the number of images with a smaller becoming scale. So the more octaves and octave layers, the larger the search space. With the parameter σ, the amount of blurring can be set. We expect more layers resulting in better matching in scale, because we have more scales to compare. We expect to find a

certain amount of blurring to be optimal for rotation. The amount of images taken in each octave is calculated automatically with respect to the photo resolution. The more octaves, the more computational time it costs [§].

Difference of Gaussians (DoG)
The Difference of Gaussians is the result of subtracting two images which have been blurred by convolution with a Gaussian function (see equation 3.1).

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (3.1)$$

In computer vision, convolution is the operation of a kernel on the image pixels. The resulting image shows edges and corners (figure 5). With increasing the parameter contrastThreshold, weak features can be discarded. With increasing the parameter edgeThreshold, less features are filtered out.

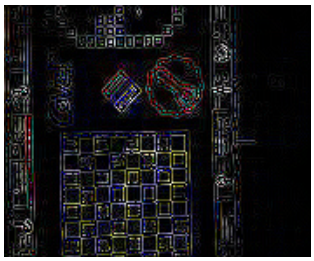

Figure 5: Difference of Gaussians (DoG) on the heterogeneous structured surface type which represents edges and corners shown as white pixels.

Detecting keypoints
The keypoints are detected by finding maxima and minima in the DoG search space. By convolution with a 3x3 matrix around a pixel through the scale space, we find these maxima and minima. With non-maximum suppression we find the edges that are strong enough (magnitude, see formula 3.2)

$$M_{ij} = \sqrt{((A_{i,j} - A_{i+1,j})^2 + (A_{i,j} - A_{i,j+1})^2)} \quad (3.2)$$

[§] `http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/`

at different orientations (see formula 3.3).

$$R_{ij} = atan2(A_{i,j} - A_{i+1,j}, A_{i,j+1} - A_{i,j}) \quad (3.3)$$

Where A is the pixel matrix from the image. In order to make the keypoints robust to lighting or contrast changes, we determine the dominant orientation. This is determined by constructing a histogram of local image gradient orientations. After smoothing, the orientation which has the peak in the histogram is selected as the dominant one.

Extracting features Features are extracted by taking 16x16 pixel windows around the keypoint pixel. These windows are subdivided in 4x4 windows where magnitudes and orientations are calculated. By dividing in 4x4 windows, a robust local representation of the pixel's neighborhood is achieved. The orientations are continuous, but we make them discrete by fitting them in eight bins. Eight bins are chosen as a trade off between preciseness and. The more bins, the more accurate the matching, but at the cost of the ability to match to a bit similar features. The less bins, the matching becomes less accurate, but more able to match to a bit similar features. So now we have 128 (4x4x8) values for the feature vector.

Matching The matching of keypoints is done by comparing descriptors between images and selecting the ones with the smallest distance.

Filtering keypoints
We filter low responses and too large distances between matched features.

3.2 SURF (Speeded Up Robust Features)

SURF is another widely used faster and more robust feature detection and extraction algorithm. In our literature study we found that there are many differences with respect to SIFT (Bay, Ess, Tuytelaars, and Van Gool, 2008). Integral images and Hessian boxes are used to speed up the detection of keypoints. Integral images can help finding intensities in a pixel group within a box. The integral images are made by summing the pixels from the

top left corner and working your way to the right bottom. Hessian boxes are a less complex way to detect corners and edges. They serve as a replacement for the computational expensive second order Gaussian filters. Hessian boxes are more discrete, this makes detection less robust, but requires less computation at the detection stage.
Also constructing a scale space is computationally expensive in SIFT. By upscaling the filter size, different scales can be created quickly in SURF. So no scale space has to be created, only filter sizes have to be upscaled. Interest point are localized by non-maximum suppression, this is a technique to determine if the pixel lies on an edge. If the gradient magnitude assumes a local maximum in the gradient direction, this is the case. The orientation is calculated by Haar wavelets. A Haar wavelet is a discrete kernel which can be convolved over an image to find sudden transitions in pixels. Again the integral images can be used to speed up the calculation of the dominant orientation. Here SURF doesn't use a histogram but a sliding window to select the dominant orientation.
Concluding the big difference lies in the use of integral images and Hessian boxes.

3.3 Matching

The matching translates to comparing each feature p from picture A, to all the features from picture B. This is called brute force matching. In 128 dimensions a descriptor is built from surrounding pixels around the keypoints. Values like the mean and orientation are stored. A measure to compare how well they matched can be measured with the L1-norm (Manhattan distance) or L2-norm (Euclidean distance).
The formula for the L1-norm is:

$$d_1(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{3.4}$$

.
The formula for the L2-norm is:

$$d_2(p,q) = \sum_{i=1}^{n} (p_i - q_i)^2 \tag{3.5}$$

We compare both formulas and we can already say that formula 3.5 gives more weight to large distances between features, because they contribute quadratically.

3.3.1 Filtering of bad keypoints

We filter bad keypoints by thresholding the responses, distances between features and crosschecking. Responses are the strength of the detected feature. We disregard features which are below a given threshold because they are not distinctive enough. Also by thresholding the distance between features, a better result can be achieved. When the distance is too large, we expect the certainty of this match to be correct to drop. We also filter by crosschecking. In figure 6 we show cross checking in 2 dimensions. In this application we use keypoints with descriptors of 128 dimensions.

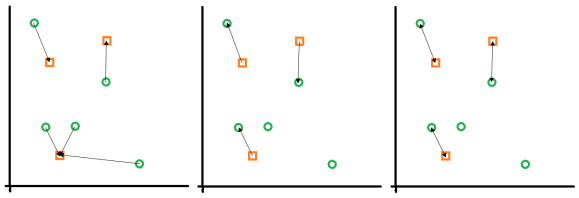

Figure 6: The left picture shows matches from the circles to the squares. The middle vice versa and in the right the matches are cross checked.

3.4 Height map

The height-map is constructed dynamically using stitching results and the on-board hardware. We use the distance between matched features of the stitching algorithm to determine the overlap in height data. The on-board ultrasonic sensor is used to determine the height. The height-map is constructed by calculating the overlap between images (relative distance) (see figure 7). The height-map only shows the measured height in combination with the overlap from the stitching algorithm. It is not meant to show performance on spatial resolution or accuracy, but only to show a height-map can be constructed and edges are visualized where the color transition is large.

4 Experiment setup and results

In this section we show the experiment set up to make the quadcopter able to stitch images. We

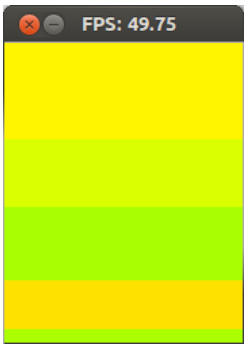

Figure 7: Height map showing the height transition: red is high, green is intermediate and yellow is low. Here the quadcopter starts at the top and ends at the bottom. This height map corresponds with the stitched image of figure 17

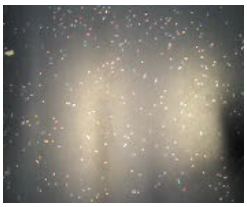

Figure 8: A Homogeneous structured surface type on which we evaluate SIFT and SURF.

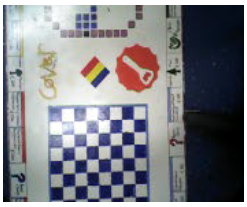

Figure 9: Repetitive heterogeneous structured surface type on which we evaluate SIFT and SURF. Mark the squares of the chess area.

show what types of image surfaces can be classified and do a performance analysis to find optimal parameters for these surface types. Also the performance of the Arduino Nano and the ultrasonic sensor on wall-following and obstacle avoidance is evaluated.

4.1 Experiment setup

The experiment consists out of an in-flight quadcopter and a control base. The control base sends commands to the quadcopter. The quadcopter has to fly a straight path, since we established this to be difficult (see figure 4), we correct the quadcopter into this straight path. At manually chosen intervals the quadcopter is stopped and at button press, a picture is created. At the same time, the height data is read and a height map is constructed.

The SIFT and SURF algorithms are tested on three types of terrain: homogeneous (figure 8), repetitive heterogeneous structured (figure 9) and heterogeneous (figure 10).

To check performance and establish a baseline of parameters for the feature detection algorithms in-flight, we take 3 pictures by hand as if the quadcopter was flying. These pictures follow the same straight path behavior and have a certain overlap calculated by the stitching algorithm. This overlap is measured and compared to an at sight perfect stitched image (see figure 11), which was manually created by comparing pixels from both images. With at sight we mean that at pixel level, manually chosen reference points are aligned. For each feature detection algorithm and each terrain type, a set of optimal parameters is found. Optimal parameters are found by manually comparing the length of the chosen reference points and measuring the distance between them. So a reference point from a to be stitched image and a reference point from the next to be stitched image are chosen and the overlap length calculated (figure 11).

The performance of the stitching algorithm is how much the ratio of the lengths of the manually chosen reference points of the stitched image by the algorithm (figure 11) compares to the lengths of the same reference points in the manually stitched image.

As a final result, the aforementioned parameter baseline is used to check performance on a real world environment, namely the Cognitive Robotics Laboratory.

For both SIFT and SURF the descriptor size is kept the same (vector of 128 elements), as well as the number of octave layers (namely three). The number of octaves in SIFT is calculated automatically with respect to the resolution of the image. For SURF we keep this fixed at four, because then

Figure 10: Heterogeneous surface type on which we evaluate SIFT and SURF. Also the first picture to be stitched.

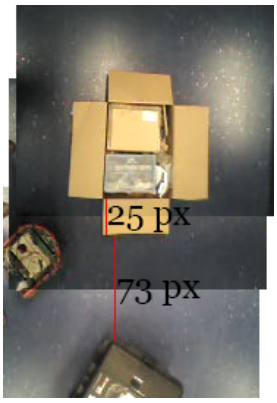

Figure 11: The at sight perfect stitched image with overlap length (in pixels) between manually chosen reference points.

it has the ability to match in four different scales, which is good because we only need one since we fly at the same height. To check performance in a real world environment we use the Cognitive Robotics Laboratory. Upon inspection we can say it looks like a heterogeneous environment. SIFT and SURF work best in heterogeneous environments because here, the SIFT or SURF features have a greater distinctiveness.

The parameters which result in a ratio closest to the one shown in figure 11, are considered optimal.

The quadcopter's control and image processing libraries are successfully implemented in the BORG architecture.
The quadcopter is stable enough when not carrying the equipment used for obstacle avoidance, but when it is carrying the extra equipment, there is a displacement to the left, as can be seen in figure 18. You can also see the motor resonance since the

Figure 12: Second picture to be stitched.

Figure 13: Third picture to be stitched.

integral and derivative part of the PID controller are not implemented yet. Tthe wall-following behavior is counter-acting the displacement error to keep the quadcopter at a fixed distance (40 cm) from the wall.

4.2 Results

4.2.1 Obstacle avoidance and Arduino results

We have shown a implementation of extra sensors on a quadcopter. Not only did it function as expected, it is also easy to implement (see figure 3). Unfortunately the quadcopter turned on its yaw-axis when commanded to fly a straight path. We think this is partially due to the presence of the wall. When we fly in an open environment, it flies in a more straight manner. We think the quadcopter gets drawn to the wall due to aerodynamics and the built-in position estimation algorithms in combination with error compensation algorithms (Bristeau et al., 2012) are not working perfectly. We also think the turning on the yaw-axis is due to sensory drift (section 2.3) and weight imbalance of the extra hardware.

4.2.2 SIFT and SURF stitching results

For every surface type, three images were taken by the quadcopter in a forward flying path. These

Figure 14: Result of figures 10, 12 and 13 stitched with SIFT.

images are stitched by the SIFT and SURF algorithms. Performance is then tested by comparing and measuring the images with a perfect result made by hand (see figure 11). We check per parameter the influence on the performance, these parameters will be discussed in the next section.
The problem with feature based stitching algorithms is that they require surfaces with a lot of variety in pixel values (Szeliski, 2006). Only the heterogeneous structured surface type had this variety and showed good results upon inspection (see figure 10), the homogeneous and heterogeneous structured surface showed very poor performance upon inspection. The homogeneous structured surface type has pixels which look too much like noise (see figure 8 and 15). This performance includes stitching in the wrong direction and overlap distances which were too large or small. Therefore we only examine the heterogeneous structured surface type.

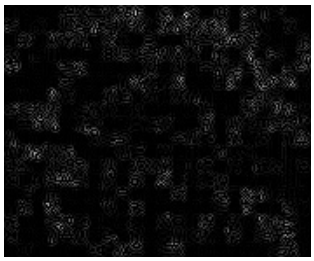

Figure 15: Difference of Gaussians (DoG) on the homogeneous structured surface type (figure 8). The image shows edges and corners in white pixels. These look random and not distinctive when comparing with figure 8

To get more keypoints you need to lower thresholds, but this means the keypoints become less meaningful. These keypoints are likely to mismatch. When the threshold is too high, the keypoints will be less likely to match to a bit less similar features. When the environment changes a bit, the keypoints will have a bit less similar features and due to the high threshold, matching is not possible. The repetitive heterogeneous structured surface type has a lot of mismatches because certain areas of pixels which exhibit the same contrast are selected as keypoints (see figure 16). When there are too many mismatches or too little matches, the image can't align. (Szeliski, 2006).

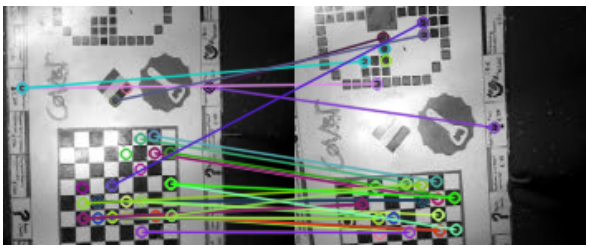

Figure 16: Groups of pixels exhibiting the same contrast are difficult to distinguish by both SIFT and SURF due to their descriptor extraction techniques.

The pixel ratio is used to check performance. This pixel ratio is a measurement taken from the overlap between two images and the two manually chosen feature points (see figure 11). The overlap distance is also checked, because it has to be the same more or less. This is the case because for every surface type the same photos are used. Also the number of keypoints is checked, because the same result with less keypoints is desirable for reducing computation time. Below are the standard parameter settings for SIFT and SURF. These settings will be optimized and discussed in the next section.

The standard parameter settings for SIFT are: OctaveLayers = 3
contrastThreshold = 0.04 (the larger, the less features
edgeThreshold = 10 (the larger the more features
sigma = 1.6
filtering = 1 (yes) or 0 (no)

The standard parameter settings for SURF are: hessianThreshold=100

nOctaves=4
nOctaveLayers=3
filter_flag= 1 (yes) or 0 (no)
sigma_threshold = 1

4.2.3 Influence of parameters on SIFT and SURF performance

In this section the influence of every parameter of the SIFT and SURF algorithm on the stitching performance is explained. The tables in this section show this influence for different parameter settings. The homogeneous and repetitive heterogeneous surface type are discarded section as useful (see section(4.2.2), therefore we only do our parameter analysis on the heterogeneous structure type (figure 10).
The tables are made easier to read by the following abbreviations:

1. The pixel ratio is a measurement taken from the overlap between two images and the two manually chosen feature points (see figure 11, the best value is 25/73). The value before the "/" is the amount of pixels between the manually chosen features from image 1 (figure 10) and image 2 (figure 12). The value after the "/" is the amount of pixels between the manually chosen features from image 2 (figure 12) and image 3 (figure 13).

2. "e" means that upon inspection the picture is not stitched correctly.

3. "n" means the overlap distance is negative. The pictures should be stitched in a forward motion, but apparently the features are matched in a backward motion. This can't be correct, since we fly in a forward motion.

4. $n_{keypoints}$ means the amount of keypoints in the first and second image. The value before the "/" is the amount of keypoints in image 1. The value after the "/" is the amount of keypoints in image 2.

5. The "overlap distance" shows the calculated mean overlap between matched keypoints. The value before the "/" is the overlap distance in pixels from image 1 (figure 10) and image 2 (figure 12). The value after the "/" is the overlap distance in pixels between image 2 (figure 12) and image 3 (figure 13).

OctaveLayers	Pixel ratio	$n_{keypoints}$
2	e	76/106
3	e	86/132
4	e	99/148
5	e	110/160

Table 1: The influence of the parameter OctaveLayers on the stitching performance of the SIFT algorithm. The pixel ratio with 3 OctaveLayers was recommended (Lowe, 1999). The pixel ratio with 5 OctaveLayers showed the best stitching results.

SIFT: octaveLayers The octave layers represent the scale space (see table 1). Since we almost fly at the same height, we get the same scale. A lot of variety in scale is not needed, because the quadcopter flies at the same height, therefore we keep it at 3.

contrastThreshold	Pixel ratio	$n_{keypoints}$
0.02	e	118/153
0.03	e	98/137
0.04	18/45	86/132
0.05	e	78/114
0.06	e	74/101
0.1	e	48/66
0.2	n	7/1

Table 2: The influence of the parameter contrastThreshold on the stitching performance of the SIFT algorithm

SIFT: ContrastThreshold Since the only value for this threshold which gives a stitched picture is 0.04 (see table 2), this value is chosen. When the contrast value is chosen to be high, the neighboring pixels of the interest point must exhibit a really high contrast to be selected as a keypoint.

SIFT: EdgeThreshold The edgeThreshold selects keypoints which are more "edgy" than others

EdgeThreshold	Pixel ratio	$n_{keypoints}$
8	e	86/122
10	e	86/132
12	e	87/139
16	e	90/151
25	e	94/163

Table 3: The influence of the parameter edgeThreshold on the stitching performance of the SIFT algorithm

and thus a better feature. This has to do with the magnitude and dominant orientation of the pixels surrounding the interest point. A larger magnitude means a stronger edge. We couldn't measure the performance since the stitched image didn't show the reference points to get the pixel ratio from (see figure 11 and table 3).

Sigma	Pixel ratio	$n_{keypoints}$
0.5	18/53	339/415
0.8	29/47	300/376
1.0	23/47	207/262
1.2	e	138/185
1.6	e	86/132
2.0	e	75/99

Table 4: The influence of the parameter sigma on the stitching performance of the SIFT algorithm

SIFT: Sigma The sigma (see table 4) sets the sigma of the Gaussian blur (see formula 3.1)

SURF: HessianThreshold So we see the overlap distances remains the same (see table 5), and the lowest hessianThreshold is the best, but does compute a lot of keypoints. If the response (determinant) of the hessianBox detector is filtered at a high threshold, less keypoints will remain. We choose 200 as the hessianThreshold because it is recommended (Bay et al., 2008). We will also check for a hessianThreshold of 10 (but this is almost too low to consider). Other values were also checked and upon inspection it showed that the range of 10 to 500 gave the best results.

hessianThreshold	Pixel ratio	Overlap distance	$n_{keypoints}$
10	26/65	40/46	104/139
50	32/61	43/43	58/116
100	31/63	41/44	50/106
200	32/54	44/43	27/49
500	e	e	e

Table 5: The influence of the parameter hessianThreshold on the stitching performance of the SURF algorithm

nOctaves	Pixel ratio	Overlap distance	$n_{keypoints}$
1	32/58	44/43	72/73
2	33/55	46/42	39/89
3	32/57	45/49	40/90
4	32/55	45/43	40/90
5	32/55	45/43	40/90

Table 6: The influence of the parameter nOctaves on the stitching performance of the SURF algorithm

nOctaves So we can see the number of octaves or scales doesn't matter (see table 6). This is correct because nOctaces represent the number of octaves, which represents the number of different scales to which an image is compared. When the quadcopter stays at the same height, the scale stays the same. Since it doesn't matter what number of octaves is chosen, a value of four is chosen because it is recommended (Bay et al., 2008).

SURF: nOctaveLayers Again the overlap distance remains the same (see table 7). This is due to our flying behavior. It is possibly the case that we flew the quadcopter the same amount of time and then stopped to take a picture.

nOctaveLayers of 5 is chosen here because it corresponds the most with the best at sight determined pixel ratio (see figure 11) and takes less computation time than 6 nOctaveLayers. The blurring does allow many keypoints, but does also keep performance.

nOctaveLayers	Pixel ratio	Overlap distance	$n_{keypoints}$
1	31/56	44/43	26/50
2	32/56	45/43	40/90
3	32/63	42/43	52/107
4	29/64	41/44	57/117
5	27/64	41/45	60/130
6	27/63	41/45	62/134

Table 7: The influence of the parameter nOctaveLayers on the stitching performance of the SURF algorithm

SIFT/SURF: Filtering Upon inspection we could see, by looking at stitched image, filtering doesn't matter. This is probably due to the optimal setting of parameters.

SURF: Final hessianThreshold check We only have to look at the setting for an hessianThreshold of ten, see table 8, as we mentioned in paragraph 4.2.3. This hessianThreshold is with the best settings found in the previous parameter sweep.

hessianThreshold	Pixel ratio	Overlap distance	$n_{keypoints}$
10	21/74	37/48	32/200

Table 8: Checking if the hessianThreshold could be better than the one found in paragraph 4.2.3.

With a different overlap distance but a better overlap ratio, it looks like a hessianThreshold of 10 is better (see table 8). We can explain this because Bay used pictures of 512x512 and we 176x261. So using a smaller hessianThreshold to select keypoints in a smaller image makes sense.

For SIFT, the optimal parameters are:

OctaveLayers = 5
contrastThreshold = 0.04
edgeThreshold = 10
sigma = 1.0
filtering =0 (no)

The optimal parameters for SURF are:
hessianThreshold=10
nOctaves=4
nOctaveLayers=5
filter_flag= 0 (no)

The filtering of keypoints was not necessary, because we chose our parameters settings well. Upon inspection, filtering did not gave a better result. We thought our parameter settings for the heterogeneous surface type would give a good stitching result on the Cognitive Robotics Laboratory surface. However this was not the case (see figure 17), probably because the parameter settings found on the heterogeneous surface could not find (distinctive) enough keypoints in the Cognitive Robotics Laboratory surface and therefore matching performance was poor.

Figure 17: We show our final stitched image using SIFT of a random area of the Cognitive Robotics Laboratory. It should have shown the ground, a small piece of a chair, a table with case and cables in it and again a chair. This is hard to distinguish, so our stitching performance is poor.

Since the quadcopter experiences a lot of aerodynamic problems from the environment, mass inertia, inability to respond quickly and poor self stabi-

lizing properties, we had to fly it manually. In figure 17, the stitching performance is shown, which is quite poor. It should have shown the ground, a small piece of chair, a table with case and cables in it and again a chair. Instead it shows fragments, e.g. the bundle of cables should be fully visible. This is due to poor features and matching. Due to too low thresholds, less meaningful features relative to the environment get selected. When the environment changes, the threshold also has to change to get a good matching result. For every new picture, new parameters have to be set to get good features. We could only take four pictures, because the fifth picture resulted in such bad matching it wanted to overlap in the different direction. Our program doesn't permit this.
Concluding, SIFT and SURF alone are not good enough for panoramic construction with a quadcopter.

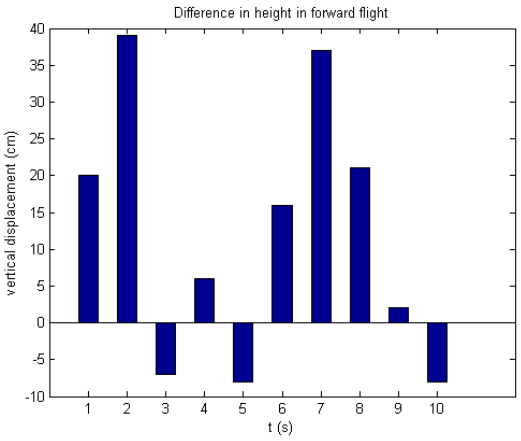

Figure 18: Graph showing the error in height during forward flight. When the quadcopter has taken off and is hovering, the height is set at zero centimeters.

4.2.4 Height map results

We have shown, albeit crudely, the use of ultrasonic sensors to construct height maps (figure 7). The height map lacks spatial data. It does not show the height of small objects, lacking a good resolution. To improve the resolution, vision algorithms (like optical flow) can help. With optical flow, the height of smaller objects, can be estimated.
The height of smaller objects can't be measured because the ultrasonic sensor listens to echoes of its pulse which arrive first. Sound bounces of the highest objects first, because they are closer to the sensor. Also the field of perception is small and cone shaped. To measure smaller objects, smaller distances can be flown and more measurements can be taken and combined. The main problem was the altitude control loop which tried to keep the quadcopter at a fixed height. When the quadcopter flew over an object, it compared its last distance to the current distance and tried to counteract this height distance by flying up or down. This could not be disabled, because it was in the firmware, resulting in a height map with mostly the same height measured. The altitude control loop is not perfect, because the same height is not always maintained (figure 18).

Figure 7 shows our height map constructed using the overlap calculated by our feature matching algorithms. It shows the height differences encountered whilst flying over the ground, chair, table and chair as shown in figure 17.

Concluding, making a height map on a quadcopter is possible using ultrasonic sensors, if there weren't the software limitations of altitude control.

5 Discussion

The Arduino is a good platform to use when extending sensory capabilities. The communication of the hardware with the computer could be better, because data is (partly) lost or delayed.
We also experienced several difficulties when working with the quadcopter. It became clear that the altitude loop control algorithm was a big obstacle, with open-source software this obstacle can be overcome.
The small spatial resolution of the ultrasonic sensor could also be overcome with a Kinect like technique. This technique uses a grid which is projected on a surface, this grid gets deformed and then an algorithm uses this deformed grid to calculate the distance. Also a radar could be used but this is too heavy, a laser would be a better option. Fortunately there is an AR.Drone version 2 which contains more (accurate) sensors.

Also the tuning of frequency parameters was

difficult. The Arduino now has a frequency of 2 Hz per sensor. The Telnet connection (mentioned in figure 3) is not very reliable and the time between the measured values is also not constant. It would be a good idea to tag the measured values with the quadcopter's internal time. This requires altering the quadcopter's firmware, which we took a look at, but falls outside the scope of the thesis. Also the quadcopter's internal timer is not working. When it is in flight, it should start counting the milliseconds it is in flight, but the timer can't be started.
The BORG system is also working on a frequency, which is set at 5 Hz. There is a lot of mismatching between measurements and corresponding actions over time due to different frequency settings.

The panoramic images are crudely stitched, but can be optimized with sensor fusion (to get more accurate sensor data) (Nagai, Chen, Shibasaki, Kumagai, and Ahmed, 2009) and an Extended Kalman Filter (better position estimation by handling sensor uncertainty and modeling of motions and actions)(Caballero, Merino, Ferruz, and Ollero, 2009). Just flying a single path, stopping and then making the picture with the press of a button is not a realistic way of making a map automatically.

Closed source software makes it difficult to disable control loops on the quadcopter, like the altitude control loop.

To answer our first research question: Can the two feature detection algorithms, SIFT and SURF, be used in a stitching algorithm with a quadcopter to construct panoramic images? We have to conclude it is possible when the parameters are optimized in a certain environment and matching is done in the same environment. When the environment is changed, matching becomes unreliable. The stitching algorithm highly relies on the variation and contrast in the images. When this is lacking, the matching becomes highly unreliable.
Our second research question states: Can we create a height map with a quadcopter? This question is answered positively with a side note. The ultrasonic sensor can be used to register height differences (figure 18). Unfortunately the altitude loop control algorithm prevented us to measure in flight.
The third research question: Can we use external hardware, like an ultrasonic sensor, to compensate sensory drift? The Arduino extended with the ultrasonic sensor is able to counteract sensory drift to some degree. The quadcopter is able to follow a wall with the new hardware. Without it would just crash into the wall. With the new hardware, it would also exhibit this behavior in the end because the quadcopter turns on the yaw axis. Another ultrasonic sensor can be used in combination with the added one to counteract this problem.
To sum up the conclusions of our research questions, for reliable environment mapping, both scenery and height, we need more sensors and better stitching algorithms.

References

H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 110:346–359, 2008.

S. Bennett. A history of control engineering. *IEEE Control engineering series*, 47:46–60, 1993.

P-J. Bristeau, F. Callou, D. Vissiere, and N. Petit. The navigation and control technology inside the AR.Drone Micro UAV. *Preprints of the 18th IFAC World Congress Milano*, pages 1477–1484, 2012.

R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

F. Caballero, L. Merino, J. Ferruz, and A. Ollero. Vision-based odometry and slam for medium and high altitude flying uavs. *Journal of Intelligent and Robotic Systems*, 54:137–161, 2009.

B. Cooper and Y. Jason. MAV stabilization using machine learning and onboard sensors., dec 2010.

N. Dijkshoorn. Simultaneous localization and mapping with the AR.Drone. Master's thesis, University of Amsterdam, Amsterdam, 2012.

C. Harris and M. Stephen. A combined corner and edge detector. *Proceedings of the Alvey Vision Conference*, pages 147–151, 1988.

D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision*, 47, 1999.

M. Nagai, T. Chen, R. Shibasaki, H. Kumagai, and A. Ahmed. UAV-Borne 3-D Mapping System by Multisensor Integration. *IEEE transactions on geoscience and remote sensing*, 47(3):701–708, 2009.

T. Suzuki, Y. Amano, T. Hashizume, and S. Suzuki. 3d terrain reconstruction by small unmanned aerial vehicle using SIFT-Based Monocular SLAM. *Journal of Robotics and Mechatronics*, 23:292–301, 2011.

B. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):42–51, 2006.