DEPTH IMAGE PROCESSING FOR OBSTACLE AVOIDANCE OF AN AUTONOMOUS VTOL UAV

Franz Andert, *Gordon Strickert* and *Frank Thielecke*
German Aerospace Center, Institute of Flight Systems
Lilienthalplatz 7, 38108 Braunschweig, Germany
{Franz.Andert, Gordon.Strickert, Frank.Thielecke}@dlr.de

OVERVIEW

We describe a new approach for stereo-based obstacle avoidance. This method analyzes the images of a stereo camera in realtime and searches for a safe target point that can be reached without collision.

The obstacle avoidance system is used by our unmanned helicopter ARTIS (Autonomous Rotorcraft Testbed for Intelligent Systems) and its simulation environment. It is optimized for this UAV, but not limited to aircraft systems.

1 INTRODUCTION

The automatic detection of obstacles and the avoidance of collisions is a useful ability for many applications like robots, cars and flight systems, used for autonomous control in unmanned systems and as an assistant in manually driven vehicles. Sensors used to detect obstacles are not just cameras, there have been a lot of good results with active sensors in the past like radar or laser scanner systems. The use of one or more cameras ends up in image processing, the research concentrates mainly on using optical flow [1, 2], stereo vision [3, 4] or both techniques [5]. Obstacle detection and avoidance is done by building a map of the environment and plan a path there or just reactive with moving away from detected obstacles like insects do.

Our proposed approach is also a reactive system, i.e. single obstacles will not be detected separately and no environmental data is stored. With ideal input data, information about free and occupied areas can be extracted out of one single depth image that has been generated from a stereo image pair. Intuitively, it can be seen which path directions will cause a collision and which will not.

Figure 1 shows a brief overview of the process how we detect flight targets to avoid collisions. A stereo image pair (a, left image only) is recorded and the depth information (b) from this point of view is calculated. This image contains information about far (darker) and near (brighter) objects, but not for every pixel (errors are white). After the filtering of false depths (c), regions in the image without near objects are detected–only they can represent flight targets (d). If they really lead to 3-D points where we can fly is checked out with the help of the three-dimensional information of the depth image. Dangerous targets will be filtered (e). Finally, one target is selected (f) and the helicopter is instructed to fly there.

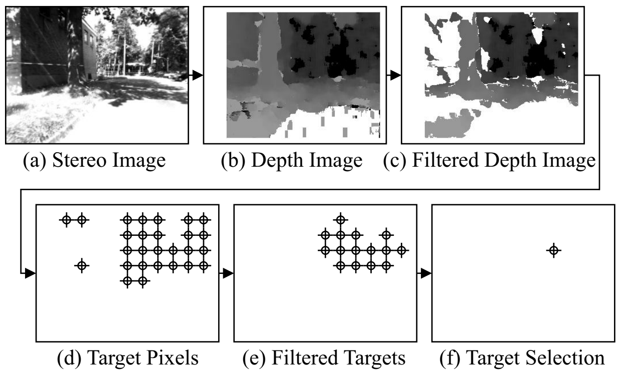

FIG. 1: The image processing chain of the obstacle avoidance system.

The result of the presented method is one of the possible path directions avoiding a collision. On the other hand it can be a warning to stop the vehicle if no free area is visible. It is compatible to a global path planning system so that the helicopter will reach its originally given goal if there is a way.

2 FLIGHT TESTBED AND IMAGE PROCESSING ARCHITECTURE

Testbed is our unmanned helicopter ARTIS [6], equipped with a stereo camera and a separate vision computer for camera control and image processing.

The task of the vision computer is to acquire images from cameras, process the images and send information to the flight control computer. The latter gives the helicopter the ability to fly to a given coordinate with the help of classical sensors like GPS and an IMU and we will send the target coordinate to it. The coordinate of the target avoiding collision is updated with every new processed image frame.

Our image source is a stereo camera rig from Videre Design. It comes with a software to calibrate the cameras and to calculate a depth image automatically, so that we do not have to concern about this. The obstacle avoidance is mainly a postprocessing of this depth image.

FIG. 2: The helicopter ARTIS with an on-board stereo camera.

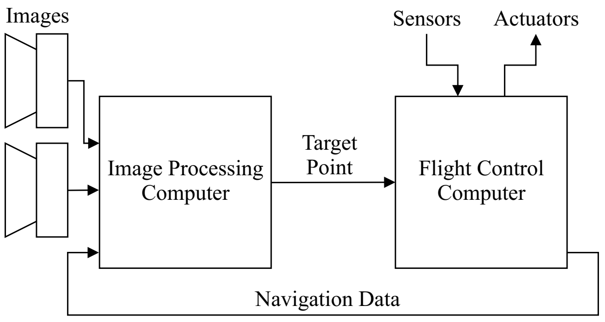

FIG. 3: Architecture of the ARTIS helicopter, simplified.

In contrary to indoor robots that are used by many research teams, we have to detect objects at greater distances thus the helicopter can react in time. Due to the camera geometry, larger focal lengths or baselines between the cameras allow the precise measure of farther distances. We set the stereo camera to a baseline of 30 cm and use lenses with a focal length of 6.3 mm so that the field of view is approximately $64^\circ \times 50^\circ$ wide for each camera. Image sizes between 160×120 and 640×480 pixels are used. With tests we have found out that the distance estimation is sufficient from 8 up to 55 meters.

3 STEREO VISION

3.1 GEOMETRY OF TWO CAMERAS

Efficient depth estimation from stereo images requires a standard-stereoscopic view, i.e. the two images lie exactly on the same plane and do not have a vertical distance to each other. Since mounting and assembly of the cameras are usually not accurate enough to meet this requirement, we need to rectify the images by calculating a projection to the same plane. This step is feasible if the camera orientations are known and leads to a higher accuracy that is sufficient for the depth estimation procedure. The stereo camera system and its software [7] provide this step by calibrating the camera to get their internal and relative orientation and by processing the lens undistortion and image rectification of the two images to produce such an ideal image pair.

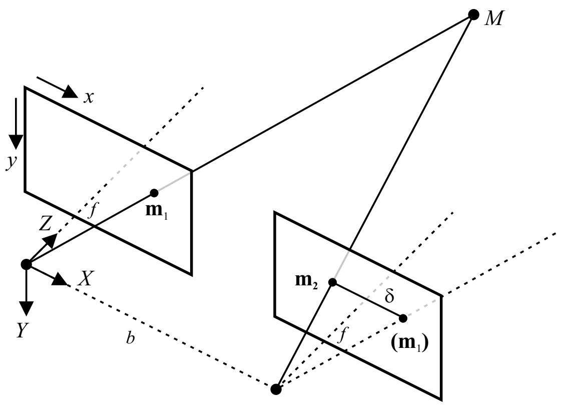

FIG. 4: Standard-stereoscopic stereo geometry. An object point $M(X,Y,Z)$ is projected to two image points $\mathbf{m}_1(x_1,y_1)$ and $\mathbf{m}_2(x_2,y_2)$. Both images are not distorted and lie on the same plane and the focal lengths are equal.

The pinhole camera model is used for the perspective projection of an object point to the image planes. In our definition, the left camera center falls together with the object coordinate center of the camera-based coordinate system, so that an image point (x,y) can be calculated from an object point (X,Y,Z) with

$$(1) \qquad z\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & x_0 & bf \\ 0 & f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

using the focal length f and the image center (x_0,y_0) of a camera. The value b set to 0 for a projection to the left image and set to the baseline between the cameras for the right image.

If an object point can be projected to both images, there will be a disparity δ between the two image point coordinates (x_1,y_1) and (x_2,y_2) due to the different camera positions. It is

$$(2) \qquad x_2 = x_1 + \delta(x_1,y_1) \quad \text{and} \quad y_2 = y_1,$$

the points do only have a horizontal and no vertical disparity to each other. The value depends on the object distance Z, so this can be reconstructed with the equation

$$(3) \qquad Z = \frac{b \cdot f}{\delta}$$

and it is obvious that this reconstruction cannot be done with only one image.

3.2 DEPTH IMAGE GENERATION

The calculation of a depth image out of a stereo image pair goes back on work like [8]. It is based on the search for corresponding image features that represent the same object in the real world and the estimation of the disparity between them. Grouping the disparity values for every position in the left image $g(x,y)$ to one function $\delta(x,y)$ and converting the disparity values to grayscale, the result is an image that provides the depth of the objects each pixel represents in the left input image.

Since our stereo camera system has an included software for disparity estimation, the resulted depth image $d(x,y)$ is regarded as sensor information in further steps. It is an array of signed 16-bit values with the disparity information

$$\delta(x,y) = \begin{cases} d(x,y)/16, & \text{if } d(x,y) \geq 0; \\ \text{n/a}, & \text{otherwise.} \end{cases} \tag{4}$$

The last case is a representation of errors where no valid disparity is available due to filtering or impossible matching.

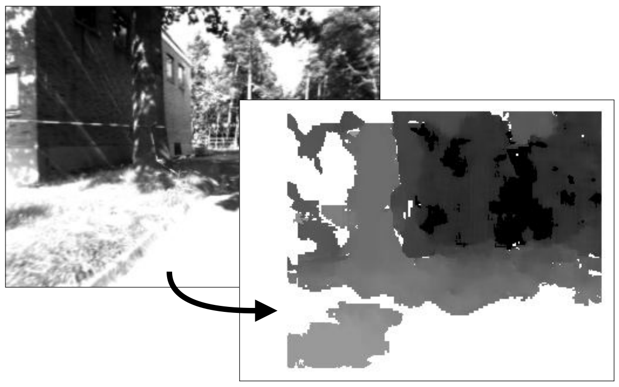

FIG. 5: Rectified left camera image $g(x,y)$ and generated depth image $d(x,y)$ of this stereo pair. Darker pixels represent greater distances and negative values are shown white here.

3.3 IMAGE IMPROVEMENT

Although the camera software provides filtering methods to detect false values, e.g. caused by occlusions or ambiguous feature matching, the depth images must be filtered again for good results.

3.3.1 Speck Removal

At first, several little, mostly bright areas can be found in the depth images calculated by the used software. With the assumption that there are no such “spikes” in the real world, these areas are caused by false depth estimation and should be treated as blotches that have to be removed. Specks in the image are usually smaller than projected real objects and their distance or gray value differs significantly from the pixels surrounding the speck.

A way to eliminate this kind of errors is to segment the image into regions with similar values and to remove the image segments that are smaller than a specified threshold.

Here, two neighboring pixels (x_1,y_1) and (x_2,y_2) belong to the same segment, if

$$|d(x_1,y_1) - d(x_2,y_2)| < \tau_{\text{difference}} \tag{5}$$

applies. The neighborhood of one pixel is defined by the eight surrounding ones. The value $\tau_{\text{difference}}$ indicates when the two pixels are similar to each other. In a region, every pixel is similar to at least one of its neighbors.

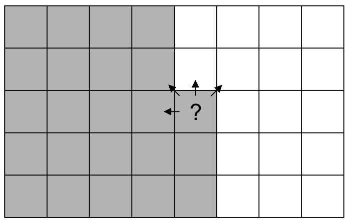

FIG. 6: Segmentation. The marked pixel will be added to the left, dark segment because of the similar gray value. If a pixel is similar to neighbors of different segments, these segments are merged. The size of each segment is stored.

For each pixel, the region with similar pixels and its size is calculated comparing the pixel with those pixels that have been assigned to a region before. This is done line by line, beginning in the upper left corner of the depth image. Figure 6 illustrates the segmentation process. If the segment size falls below a threshold, all pixels of the depth image in this region are marked as invalid.

This method will remove small regions and keep larger elements without loss of sharpness. It will also keep thin structures that are a part of larger regions. If the value of $\tau_{\text{difference}}$ is greater than zero, regions with increasing or decreasing depth are recognized as one and will not be removed if they are large enough.

3.3.2 Sky Detection

Another improvement is the sky detection. In recorded images, the sky is usually an untextured region, that means no edges are inside except clouds. No significant features can be matched in sky regions and no confident depth information can be extracted there. But sky regions represent possible flight targets so that the collision avoidance system needs the information that the sky is far away and not an obstacle.

Here, a simple and very fast method of sky detection adapted by [9] has been turned out as useful. Regions of the rectified left original image that are bright and not structured are interpreted as sky regions if no valid depth information is available. A pixel $d(x,y)$ of the depth image is set to zero, meaning infinite distance, if the equations

$$d(x,y) < 0, \tag{6}$$

$$|g(x,y) - g(x-1,y)| < \tau_{\text{texture}} \tag{7}$$

and

$$g(x,y) > \tau_{\text{brightness}} \tag{8}$$

are fulfilled. The first column of the image where $x = 0$ is ignored. The disparity of the image point must be invalid and the thresholds τ_{texture} and $\tau_{\text{brightness}}$ which define textureless and bright areas are to specify by the user. Suitable values depend mainly on the image brightness and noise.

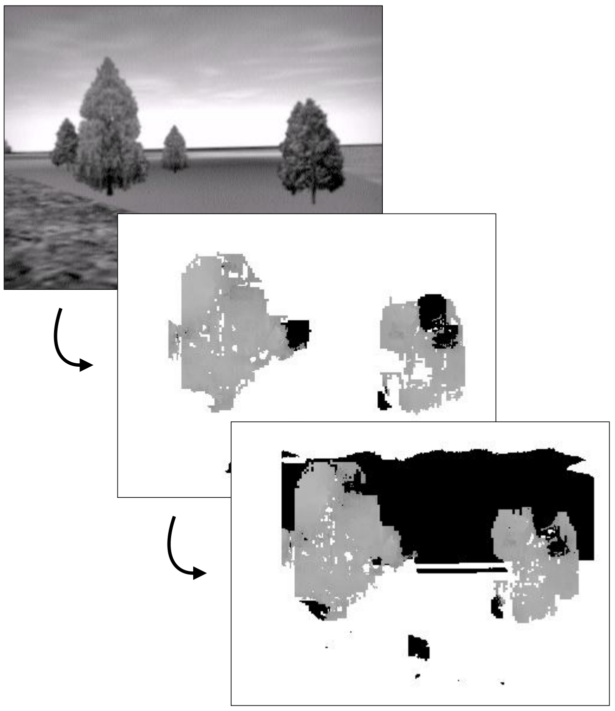

FIG. 7: Example of the sky detection. Left camera image (top) and generated depth image (center), where no depth information is available in sky regions. In the optimized depth image (bottom), the sky has infinite distance and is represented by black pixels.

4 REACTIVE OBSTACLE AVOIDANCE

4.1 FINDING A SAFE FLIGHT PATH

The task of our reactive system is to analyze the depth images in order to find a point where the helicopter can fly to without collision. The flight controller does not need any information about single obstacles, only a safe flight path will be the result of the procedure that is described here. This path is sent to the flight control computer in form of a single target point.

We make the following assumptions:

- The camera is immovable installed and looks straight on. There will be little misalignment, i.e. translation and rotation between camera- and helicopter-based coordinates. This deviation is considered later as a transformation of the returned target point.
- Instead of a path between the helicopter-based coordinate center and a world coordinate, a path starting at the camera center is calculated. This simplification leads to higher performance and has no large effects because the coordinate centers have rather a short distance to each other.
- The obstacle avoidance system will only work correctly if the helicopter flies to the direction in which it looks since the camera does not move.
- Invalid depth image values do not contain information and will be ignored. It is assumed that an obstacle in this direction can be recognized through other pixels around. Anyhow, paths without any valid information in this direction cannot become targets.

4.2 THE FLIGHT CORRIDOR

As seen in figure 4, every pixel (x,y) projects an object position (X,Y,Z) in the world that lies on a ray going through the camera center and the point on the image plane. The main goal here is to find out whether this ray is a good flight path or not.

With the help of the depth image, the distance Z of the point is known if the disparity is valid at this position and the original point can be reconstructed by equation 3 and

$$(9) \qquad X = Z \cdot \frac{(x - x_0)}{f} \quad \text{and} \quad Y = Z \cdot \frac{(y - y_0)}{f}.$$

For pixels whose disparity value refer to a near object point in front of a specified threshold distance τ_{target}, it is trivial that the ray given by this pixel is not a suitable flight path.

For other pixels, we define a flight corridor. It is a cylinder around the known ray with a radius r that is large enough that a helicopter of given size can fly through, considering a safety distance to obstacles. The pixel represents a flight path, if the distance of every object point inside the corridor is larger than the value of τ_{target}. These points are given by other pixels of the depth image.

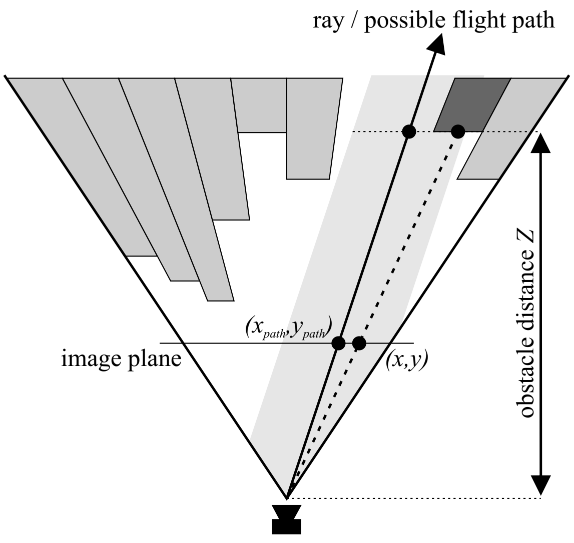

FIG. 8: Flight path and corridor given by a pixel coordinate and obstacles given by the depth image values.

Our Algorithm to check out if a pixel (x_{path}, y_{path}) represents a valid flight path is a fast calculation if other object points are inside the flight corridor. It is tested for any other pixel (x,y) in the depth image if this applies.

It is an easy way to measure distances between points inside a plane with the depth Z of the image point (x,y). This plane intersects with the ray at the point (X_{path}, Y_{path}, Z) calculated by equation 9. The intersection between the cylindrical corridor and this plane is an ellipse around (X_{path}, Y_{path}, Z), the lengths of its semi-axes are

$$(10) \qquad \begin{aligned} a_x &= \sqrt{f^2 + (x_{path} - x_0)^2} \cdot \tfrac{r}{f} \text{ and} \\ a_y &= \sqrt{f^2 + (y_{path} - y_0)^2} \cdot \tfrac{r}{f} \,, \end{aligned}$$

dependent on the ray pixel (x_{path}, y_{path}). Because the obstacle point (X, Y, Z) lies also on that plane, it will be inside the corridor exactly if it is inside the ellipse. For that,

$$(11) \qquad \frac{(X - X_{path})^2}{a_x^2} + \frac{(Y - Y_{path})^2}{a_y^2} \leq 1$$

must be fulfilled. Otherwise, the pixel (x, y) does not represent a dangerous obstacle for the flight corridor given by (x_{path}, y_{path}). The output of this algorithm is the smallest distance of an object inside the corridor. If no pixel leads to a point inside the corridor due to invalid ones, this will not become a flight path since we have no obstacle information at all.

The algorithm is extended with three improvements making it much faster. At first, we use the fact that the output value of the algorithm can only decrease with each new pixel that is tested. If obstacles with the distance Z have been found inside the corridor, pixels (x, y) whose disparity lead to a distance greater or equal to Z can be omitted.

Secondly, the calculation can be stopped when the minimal object distance reaches or falls below the decision threshold τ_{target} within the search. The coordinate (x_{path}, y_{path}) does not represent a suitable flight path in this case.

The last improvement is to build an image pyramid with depth images of lower resolution. A new level is created by summarizing each 2×2 pixel region of the level below using their maximal disparity value. To calculate the image pyramid up to level n, the edge lengths must be multiples of 2^n. If needed, the image is increased and filled up with invalid depth values at the edges. The search for obstacles inside the corridor starts in the highest pyramid level from now on. Any pixel there stands for the minimal distance represented by all pixels in a $2^n \times 2^n$ square of the original depth image.

For a check if the represented object is inside the corridor or not, it is sufficient to check the distances of the nearest and the farthest point of that square relative to (x_{path}, y_{path}). If both are outside, all object points represented by pixels of this region must be outside the corridor. If both are inside, at least one is inside and an obstacle is found. If the ellipse curve intersects with the square, the four squares of the previous level are examined as illustrated in figure 9. This is done recursively down to level zero, i.e. the original depth image if needed.

4.3 TARGET SELECTION

Each pixel of the depth image can represent a flight path and it can be determined if this path is free of obstacles up to a specified distance with the help of the flight corridor.

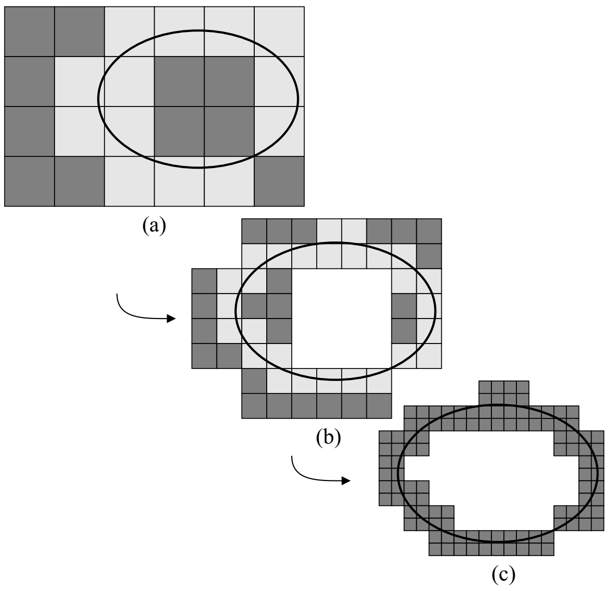

FIG. 9: Pyramid-based Search. The dark marked pixels or square regions are fully inside or outside of the ellipse. As seen in (a) and (b), some regions have to be checked in lower levels and only in the bottom level (c) it becomes clear if they represent obstacle points or not.

Additionally, a search space of this algorithm is needed to specify which pixels of the image could represent flight targets. Then, the corridor search algorithm is applied to them. If more than one suitable flight corridor is returned, they must be compared to choose one as the final target.

4.3.1 Quadtree-based Search

An easy way would be trying for each pixel if there is a free corridor around the ray the pixel represents. This is very slow and not usable in a real-time application. Since we have the information that bright pixels represent obstacles, omitting them will make the search faster.

The next improvement is a kind of forecast which pixels could represent targets at all, so it is sufficient to pick out some dark pixels and not to check everyone. Since usual depth images will be rather low-textured, free points are found near other free points and obstacle points near others that represent the same object. When some pixels are skipped in the corridor test, we may lose some good target points but should find alternatives next to them without loss of generality, though. Of course, skipping too many pixels will increase the risk that a free area will not be found, especially if it is very small.

It is quite efficient to divide the depth image into dark and bright regions and to check the dark only. The threshold for partitioning the depth image into dark and bright parts is given by τ_{target} that divides object points into near and far ones.

To determine whether an image region is dark or not, the same image pyramid as described in section 4.2 is used.

The search for dark regions is done between a maximal and a minimal level of the image pyramid, making this fast and let the user control the maximal number of possible "free" regions. If a dark pixel in a high level is found, the lower levels of this pixel will be ignored. For each found pixel, representing a square-sized image region of the input depth image, a corridor is searched around the ray represented by the center of this region.

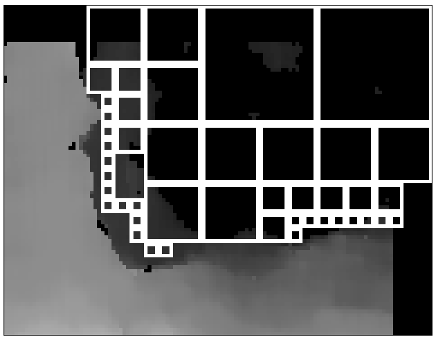

FIG. 10: Depth image example and the dark regions found, marked with squares. Flight Corridors around the center of every region are searched.

Valid targets are only these points, where the flight corridor is free of obstacles up to a specified distance. If more than one target can be achieved from this image, the target minimizing the cost function $K(Z,\Delta\alpha_x,\Delta\alpha_y)$ is chosen, which is given in equation 12. It is

(12) $$K(Z,\Delta\alpha_x,\Delta\alpha_y) = c_z \cdot Z + c_x \cdot \Delta\alpha_x + c_y \cdot \Delta\alpha_y$$

with the distance Z and the horizontal and vertical change of the flight direction, given by

(13) $$\Delta\alpha_x = |\alpha_{x_{path}} - \alpha_{x_{act}}| \quad \text{and} \quad \Delta\alpha_y = |\alpha_{y_{path}} - \alpha_{y_{act}}|,$$

based on the spherical angles of the proposed target $(X_{path}, Y_{path}, Z_{path})$ and the actual velocity vector $(X_{act}, Y_{act}, Z_{act})$ in camera coordinates. The angles of a point are

(14) $$\begin{gathered}\alpha_x = \operatorname{atan2}(X,Z) \\ \text{and} \\ \alpha_y = \operatorname{atan2}(Y,\sqrt{X^2+Z^2}),\end{gathered}$$

whereby α_x is the azimuth and α_y the elevation angle. The actual velocity vector is a part of the navigation data that can be acquired from the flight control computer.

To prefer corridors with large distances i.e. no near obstacles, the value for c_z is negative. Since near obstacles are not existing in the corridors, we set this value to 0, so that the corridor with the lowest change of the actual flight direction is chosen. Different values for c_x and c_y make it possible to set the priority to the change of heading or the change of flight altitude.

4.3.2 Target Tracking

The method just described is a search for a target point in the whole image, even in every new frame when analyzing image sequences. In our testing videos the "best" target point can jump from one image side to the other using the quadtree-based method only. The result is an ambiguous recommendation for the flight target between succeeding frames. However, changing the direction smoothly can be handled much better by any kind of vehicle system.

So what we do is a kind of adaptation what a human would do for obstacle avoidance intuitively: first look around and search for a target, but stick to this in future moments with little corrections when driving. Only change the target if it becomes useless, e.g. too near, or a previously unseen obstacle appears in this direction.

Looking around corresponds to the search in the whole image. Pointing and holding one special target is described in this and the next subsection.

One way is to stare on the object at the actual target direction and recalculate with every frame, whether a safe flight is possible. Since the helicopter moves, this object may not be located on the same pixel coordinate in the next image. By calculating the optical flow, the new position can be estimated. If a previous target is described by the image coordinates (x_{t-1}, y_{t-1}), the new point (x_t, y_t) where a corridor is searched is given by

(15) $$\begin{aligned} x_t &= x_{t-1} + v_x \\ y_t &= y_{t-1} + v_y \end{aligned}$$

with the optical flow vector (v_x, v_y) at this point between the times $t-1$ and t. This vector is estimated with feature tracking, using the Lucas Kanade algorithm [10]. Here, the original images are used instead of the depth images because of the bad performance of tracking points in low-textured images without significant features like edges and corners.

Since a previous coordinate is required, this method is not applicable with the first image. The target is initialized by the pixel representing the way straight on or a path searched by the quadtree method.

4.3.3 Searching near a previous target

Even if the target tracking is good enough to get a smooth change of the flight direction, it happens often that a safe flight path cannot be found at the new position and the target point is lost. Possible reasons are the false estimation of the optical flow vector, errors in the depth image resulting in a wrong corridor calculation or simply the real object that is coming too close.

In this case, searching for a flight target nearly around the previous one will result in a more or less gentle transition between two image frames, contrary to the quadtree-based search.

As a compromise between performance and accuracy due to the amount of pixels that are tested or skipped, the search is done in two steps. At first, a target is searched in a small circle around the old target point with high precision. If no suitable way could be found there, the search is extended by testing pixels inside a larger circle, but with less precision. If there is again no flight corridor found, the rest of the image is searched as defined in 4.3.1.

If the previous pixel representing a flight target was (x_{t-1}, y_{t-1}), candidates for the new target are the pixels (x_t, y_t) fulfilling the equation

$$\begin{aligned} & x_t - x_{t-1} \equiv 0 \mod d_1 \\ \wedge \quad & y_t - y_{t-1} \equiv 0 \mod d_1 \\ \wedge \quad & (x_t - x_{t-1})^2 + (y_t - y_{t-1})^2 \leq r_1^2 \end{aligned} \tag{16}$$

in step 1 or

$$\begin{aligned} & x_t - x_{t-1} \equiv 0 \mod d_2 \\ \wedge \quad & y_t - y_{t-1} \equiv 0 \mod d_2 \\ \wedge \quad & r_1^2 < (x_t - x_{t-1})^2 + (y_t - y_{t-1})^2 \leq r_2^2 \end{aligned} \tag{17}$$

in step 2. The parameters d_1, d_2, r_1 and r_2 are to specify.

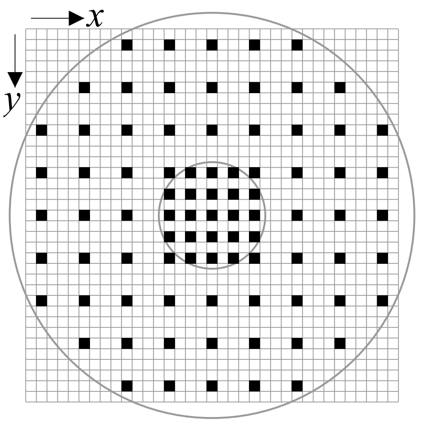

FIG. 11: Example for pixels (black) around the center (x_{t-1}, y_{t-1}) that are tested for a flight corridor and circles with the radii r_1 and r_2. Here, the parameters are $d_1 = 2$, $d_2 = 4$, $r_1 = 5$ and $r_2 = 19$.

If more than one suitable target is behind these pixels, minimizing a cost function similar to the equations 12 and 13 finds out the best target point. Instead of the actual velocity the old target is used as reference in order to stick to this.

4.3.4 Joining the different methods

As a conclusion, the resulting obstacle avoidance method is the following:

1. Check out the corridor at the position of the pixel representing the actual flight velocity vector. If the distance to an obstacle is far enough, do not change the direction. This check is done parallel and independent to an alternative point that is tracked.
2. If 1. fails, search whether the path of a tracked point is free and change the direction to there if it applies.
3. If 2. fails, search around the previous point for a new target and change direction to the best suitable path.
4. If 2. and 3. are skipped because no previous target is available or no path can be found in 3., check the whole image using the quadtree method. Fly to the best point that is sufficient.
5. If no point is found at all, stop at the current position. Instruct the helicopter to hover in place and change heading until a free area can be detected.

Finally, a target point will be transformed to the helicopter coordinate system using the equation

$$\begin{pmatrix} X_f \\ Y_f \\ Z_f \end{pmatrix} = \mathbf{R} \begin{pmatrix} X_{path} \\ Y_{path} \\ Z_{path} \end{pmatrix} + \mathbf{t} \tag{18}$$

with a 3×3 rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$. It is an Euclidean motion between the two coordinate systems given through the camera misalignment. This coordinate (X_f, Y_f, Z_f) is sent to the flight control computer that lets the helicopter move there.

5 EXPERIMENTAL RESULTS

We have finished testing the collision avoidance software in our "hardware-in-the-loop" simulation where images are processed in real but the visible scene is generated and the helicopter flight done by a simulation environment that considers the behavior of the vehicle.

In this simulation, a generated scene is projected on two screens and captured by our stereo camera as seen in figure 12. The cameras are connected to the vision computer that is used on our helicopter and it is unknown for the vision software that it is a simulation. We also use the original flight control computer, only the sensors are emulated and the visible scene is generated from a 3-D model of our real test area.

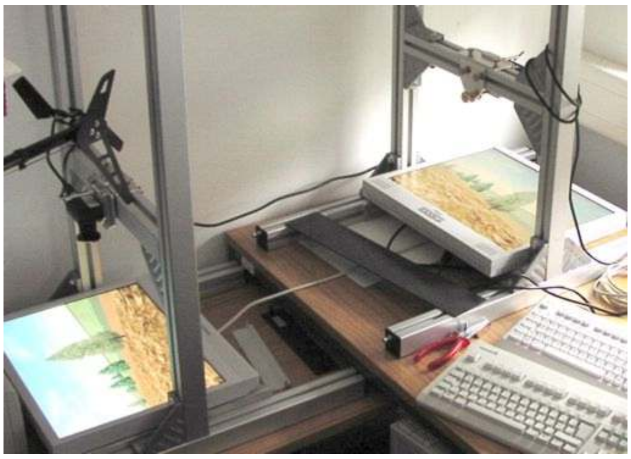

FIG. 12: Hardware-in-the-loop simulation by capturing images from displays showing generated views.

In this test, the helicopter is instructed to change the direction if an obstacle comes nearer than $\tau_{\text{target}} = 30$ meters. The maximal speed is set to $5\,\text{m/s}$, the acceleration rate restricted to $2\,\text{m/s}^2$. A tree with a diameter of 10 meters is placed into the scene and the helicopter is set to a flight path that intersects with this tree.

Figure 13 shows the results of several test flights in comparison to a flight without obstacle avoidance. Here, the helicopter is instructed to keep the original heading if possible. Using images with a size of 320×240 pixels, a 3.0 GHz Pentium IV processes between 5 and 20 frames per second including depth estimation, image improvement, obstacle avoidance and sending the safe target to the flight controller. Due to noise, the distance measurement is rather inaccurate but a distinction between obstacles and free areas is possible, even in smaller images with a size of 160×120 pixels.

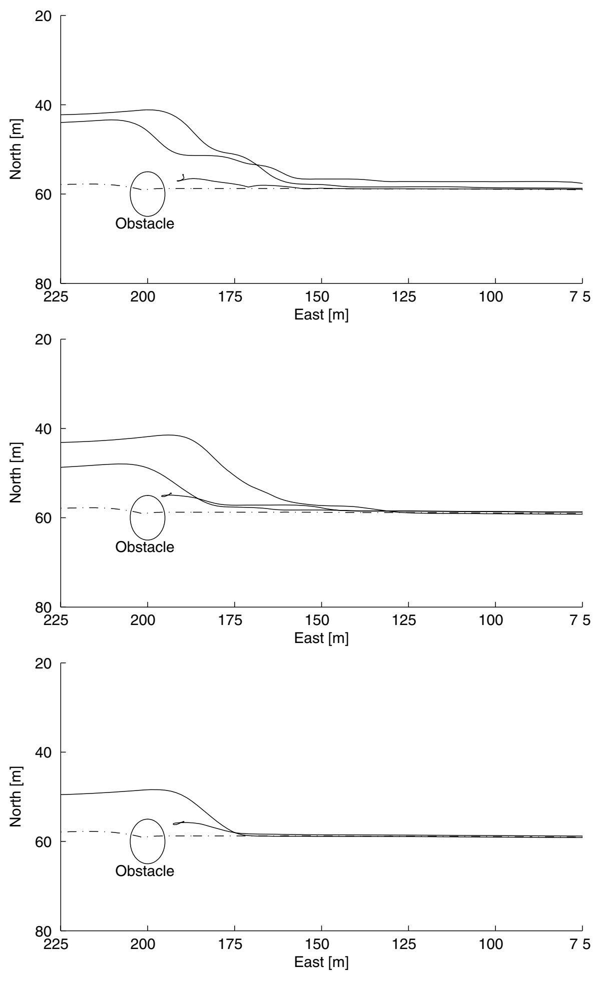

FIG. 13: Results of simulated flight tests. Images with resolutions of 640×480 (top), 320×240 (center) and 160×120 pixels (bottom) are used. The graphs show flown paths from east to west with (solid lines) and without obstacle avoidance (dotted line) as reference.

If the distances are measured too high, the helicopter changes the direction earlier. Otherwise, if the tree is detected late, no alternative way is visible and the vehicle stops. But the obstacle could be detected in every case and the helicopter did not hit it at all.

6 CONCLUSIONS

A way to process depth images for collision avoidance is described. It searches for free areas and, if possible, returns the best free area of the given image. This leads to a path without obstacles and an instruction for the helicopter to move on this. Our tests have shown that this method works and has a processing speed that is sufficient for this realtime application.

Even though it produces successful results, there are several challenges for improvement. First of all, the algorithm must become more robust to noise and false information. Another focus of research is to add obstacle information into known environment maps and to integrate the collision avoidance system into a global path planner. The goal is to navigate autonomously in uncertain areas, finding safe and optimal paths to waypoints.

REFERENCES

[1] W. E. Green, P. Y. Oh, and G. Barrows: "Flying Insect Inspired Vision for Autonomous Aerial Robot Maneuvers in Near-Earth Environments." Proceedings of the IEEE International Conference on Robotics & Automation, pp. 2347-2352, 2004.

[2] J.-C. Zufferey and D. Floreano: "Optic-flow-based steering and altitude control for ultra-light indoor aircraft." Technical Report, Swiss Federal Institute of Technology in Lausanne (EPFL), 2004.

[3] D. Murray and J. Little: "Using real-time stereo vision for mobile robot navigation." University of British Columbia, 1998.

[4] K. Sabe et al.: "Obstacle Avoidance and Path Planning for Humanoid Robots using Stereo Vision." Proceedings of the IEEE International Conference on Robotics & Automation, pp. 592-597, 2004.

[5] S. Hrabar et al.: "Combined Optic-Flow and Stereo-Based Navigation of Urban Canyons for a UAV." Proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2005.

[6] J. Dittrich, A. Bernatz and F. Thielecke: "Intelligent Systems Research using a Small Autonomous Rotorcraft Testbed". 2nd AIAA Unmanned Unlimited–Systems, Technologies and Operations–Aerospace, 2003.

[7] K. Konolige, D. Beymer: "Small Vision System – User's Manual" and "Calibration Addendum to the User's Manual." Software version 4.1e. SRI International, Menlo Park, 2005.

[8] B. D. Lucas, T. Kanade: "An Iterative Image Registration Technique with an Application to Stereo Vision." International Joint Conference on Artificial Intelligence, pp. 674-679, 1981.

[9] T. D. Cornall and G. K. Egan: "Measuring Horizon Angle from Video on a Small Unmanned Aerial Vehicle." 2nd International Conference on Autonomous Robots and Agents, 2004.

[10] J.-Y. Bouguet: "Pyramidal Implementation of the Lucas Kanade Feature Tracker." Description of the algorithm, Intel Corporation, 2000.