Reactive Avoidance Using Embedded Stereo Vision for MAV Flight

Helen Oleynikova, Dominik Honegger and Marc Pollefeys[1]

Abstract— **High speed, low latency obstacle avoidance is essential for enabling Micro Aerial Vehicles (MAVs) to function in cluttered and dynamic environments. While other systems exist that do high-level mapping and 3D path planning for obstacle avoidance, most of these systems require high-powered CPUs on-board or off-board control from a ground station.**

We present a novel entirely on-board approach, leveraging a light-weight low power stereo vision system on FPGA. Our approach runs at a frame rate of 60 frames a second on VGA-sized images and minimizes latency between image acquisition and performing reactive maneuvers, allowing MAVs to fly more safely and robustly in complex environments. We also suggest our system as a light-weight safety layer for systems undertaking more complex tasks, like mapping the environment.

Finally, we show our algorithm implemented on a light-weight, very computationally constrained platform, and demonstrate obstacle avoidance in a variety of environments.

I. Introduction

Due to their mobility, Micro Aerial Vehicles (MAVs) are very well suited for a variety of robotics applications, from disaster scene surveillance to package delivery. However, in order to function in unstructured human environments, it is essential that they are able to avoid obstacles and navigate autonomously.

There exist many sophisticated systems showing this type of obstacle avoidance, in addition to building up a full map of the environment, and using advanced 3D path planning techniques. However, most of these are equipped with high-power multicore CPU systems to have sufficient power to run these algorithms, or rely on off-board processing on ground station CPUs, which in turn require communication over a high-latency wireless link.

Carrying a heavy payload, such as a high-power CPU, not only increases the power consumption and reduces the flight time, but also requires larger and more powerful motors and propellers, making the system more dangerous to be around. Smaller MAVs are safer, easier to work with, and frequently cheaper than high-powered MAV systems.

Additionally, using off-board or non-embedded CPUs instead of embedded solutions increases the latency of obstacle detection and avoidance. Minimizing the latency between seeing an obstacle and being able to respond to it is also a very desirable quality in many robotics applications: for example, being able to navigate through a moving crowd of people, or flying through an environment that has many occlusions. Reducing the time before the robot is able to react to threats makes for a safer and more robust system.

This paper shows a novel solution to these problems: using a high-speed, low-latency passive stereo vision on FPGA system [1], we are able to use simple, fast reactive avoidance algorithms to avoid a variety of obstacles. Our approach is shown on a light-weight, inexpensive, and very computationally constrained MAV platform. The resulting system is able to fly autonomously in cluttered environments indoors and outdoors while meeting low power and small size constraints. Additionally, the stereo vision system provides disparity images at 640x480 pixels at 60 Hz, which is a much higher resolution and frame rate than any comparable system. Having a higher resolution allows us to avoid smaller obstacles, while having a significantly higher frame rate (at least double of typical systems) allows us to avoid dynamic obstacles or move faster.

Since our approach is computationally light, we suggest to use it as a light-weight safety layer for more complex systems. Though currently the algorithms run on a mobile CPU, they could be re-implemented on the FPGA, creating a one-chip solution for improving the safety and robustness of MAVs by allowing them to quickly avoid obstacles. The additional requirements of the FPGA vision system and mobile CPU are only 5 Watt power draw and 50 g payload.

The contributions of this work are as follows: we first show related work in the field of obstacle avoidance on MAVs from depth or image data (Section II). Second, we provide a simple and robust obstacle detection algorithm based on disparity maps (Section III). We then present a method of planning short-term avoidance waypoints in the map space (Section IV). We then show an efficient implementation in our test hardware (Section V). Lastly, we give results of the obstacle detection in indoor and outdoor scenarios, including video of selected test flights (Section VI).

II. Related Work

There has been a substantial amount of work done in allowing MAVs to safely navigate around a variety of environments. While many of these approaches use very sophisticated techniques, one drawback they all share in common is the need for powerful onboard or off-board CPUs to do the heavy processing required.

For example, a similar result is available on a small-size quadrotor platform demonstrating high-speed reactive avoidance of trees in a dense forest environment [2], but there are a few key differences to our approach. While their approach focuses on a monocular camera and extensive

[1]The authors are with the Computer Vision and Geometry Group, Institute for Visual Computing, Computer Science Department, ETH Zürich, 8092 Zürich, Switzerland `oelena@student.ethz.ch`, {`dominik.honegger`, `marc.pollefeys`}`@inf.ethz.ch`

Dominik Honegger is partially supported by the Swiss National Science Foundation (SNF) under grant 150151.

training data from a skilled pilot, which is processed off-board the quadrotor platform at 10 Hz and sent back to the robot over a wireless link. In comparison, our approach does not rely on any training data, is entirely on-board, and runs at 60 Hz with very low latency while achieving comparable results.

There are other projects that integrate obstacle avoidance algorithms into a global mapping framework, e.g. laser SLAM, visual SLAM, and other long-term mapping algorithms. The applications of these systems are quite compelling, such as power line detection and avoidance [3] or reactive avoidance of poles and other small obstacles [4]. Others show high level planning from 3D maps from stereo [5] [6], or from Kinect [7] [8]. Other systems solve similar use-cases as ours: vision-based avoidance in GPS-denied environments [9]. Another approach uses push-broom stereo to detect and avoid obstacles, but relies on predictable forward motion of the robot [10]. However, all of these approaches use high-power off-board workstations or feature much bigger, power-hungry, and dangerous platforms with laptop CPUs onboard.

Again, our method obviates the need for high-power on-board or off-board processing, and requires only the most basic computations capable of being carried out easily on a mobile CPU. The fact that we only use a short-term map also decreases our reliance on accurate odometry or position estimates.

Other approaches focus on doing faster avoidance by simplifying the data coming from depth sensors: such as filtering depth camera data into planes [11], or converting dense stereo into digital elevation maps [12], or even single-image depth from training data [13]. But again, all of these approaches demonstrate a frame rate of 20-30 Hz and require substantial off-board processing, while our conversion of disparity images to U-maps requires much less computing power.

While some other works do demonstrate entirely on-board processing, for example using reactive avoidance on a ground robot, the latency is much higher and this paper features much more sophisticated approach to obstacle avoidance [14]. Others show flow-based navigation through corridors, though again our approach is more flexible to different types of obstacles [15].

Schmidt *et al.* also use a stereo vision system on FPGA to do high-level navigation on a quadrotor [16]. However, their system is significantly heavier (the vision stack payload requirement alone is 740 grams), processes stereo disparities at only 15 Hz, and has an overall processing latency of 250 ms - which is insufficient for high-speed, reactive avoidance. Though they demonstrate very compelling waypoint following and navigation capabilities, in order to function in changing, dynamic environments, there needs to be another extremely low-latency reactive obstacle avoidance component, which we propose in this paper.

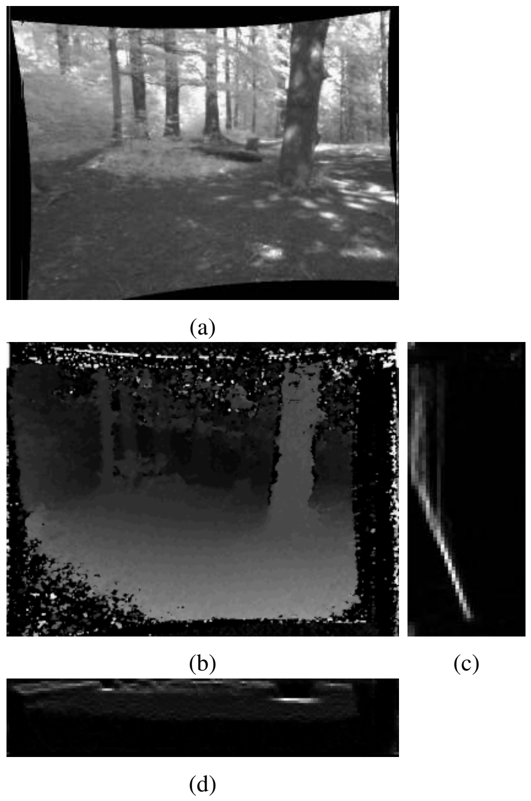


Fig. 1: A grayscale (a) and disparity image (b), collected in an outdoor forest environment. The disparity map can be split into two further maps. In the V-map (c), which is a histogram of disparity values accumulated over rows of the image, the ground plane is visible as a white diagonal line. The other representation is the U-map (d), accumulated over the columns of the image, showing obstacles as contiguous horizontal lines.

III. Obstacle Detection

In this section we describe the obstacle detection algorithm based on column accumulated dense disparity images. The overall flow of the obstacle detection algorithm is to convert the disparity image into a U-disparity map (U-map, an accumulation of disparity values along columns), exploit the special structure of this representation to detect objects in the U-map, and then cluster them into distinct obstacles.

A. U-V Disparity Maps

We base our obstacle detection method on U-V disparity maps, as described in [17]. The general idea is split the disparity image into two histograms: one accumulated along the columns of the image (the U-map) and one accumulated along the rows (the V-map). For example, an $n \times m$ disparity image with 32 distinct disparity values will produce a $n \times 32$ U-map and $32 \times m$ V-map. Note that as disparity is an inverse-depth measure, distances are not linear with position in the U- or V-maps.

Fig. 1a shows a sample image of a forest scene, and its corresponding disparity map in Fig. 1b. Fig. 1d shows the U-map generated from the disparity image, where obstacles are projected onto contiguous horizontal lines. Fig. 1c shows the V-map, where the ground plane is projected into a diagonal line, and can be used for ground plane segmentation. Due to the accumulation step, both U-map and V-map are less affected by noise and incorrect estimates in the disparity map.

B. *Obstacle Detection from U-maps*

Our algorithm for segmenting obstacles from the U-map depends on obstacles mostly spanning a small number of disparity values. The tree on the right in Fig. 1a is visible in the U-map in Fig. 1d as a horizontal shape. As the distance to the camera is the same for the entire tree, all disparity values of the tree are in the same range.

Obstacle segmentation from this representation depends simply on finding connected components in this space, as shown in a red bounding box in Fig. 2a, and projecting them back into the world frame. We present an efficient implementation of this in Section V-B.

IV. Path Planning

In this section, we describe how we use the U-map obstacle detections to build a map of the environment, and then how that map is used to plan a collision-free path for the robot.

The method we present for mapping and path-planning relies on a short-term map: it deals with noise in the image and intermittent obstacle occlusions, but does not require accurate odometry and does not keep a globally consistent map over long periods of time.

Our approach converts the obstacles from Section III-B into an elliptical approximation, merges them with ellipses from previous frames to build up a short-term map, and then plans a piecewise linear shortest path through the map.

Our method has several differences and advantages over other short-term reactive avoidance methods, such as Vector Field Histogram+ (VHF+) [18]. VHF+ only plans one setpoint in advance, while we plan a path into the future, allowing for smoother trajectory control. Additionally, VHF+ and other methods require switching between an occupancy grid and a planning representation, while our planning is performed directly in map space.

A. *Short-Term Mapping*

The short-term mapping step approximates detected obstacles as ellipses and merges them into the current map. The main steps are as follows: first, convert the bounding boxes detected from the U-map into an ellipse in the world frame. Then, determine if, for each detection, the ellipse is likely to originate from the same obstacle as an ellipse from the existing map. If so, merge them together. At the end, clear out any obstacles that are older than a time horizon to keep odometry errors from accumulating into fake obstacles.

Obstacles are represented as ellipses, with an angle perpendicular to the yaw angle of the quadrotor (an example can be seen in Fig. 2c). We introduce some notation: $\mathbf{p}$ is the position of a point in 2D space, ${}_G\mathbf{p}_e^x$ is the x component of the position of the ellipse e expressed in the global frame

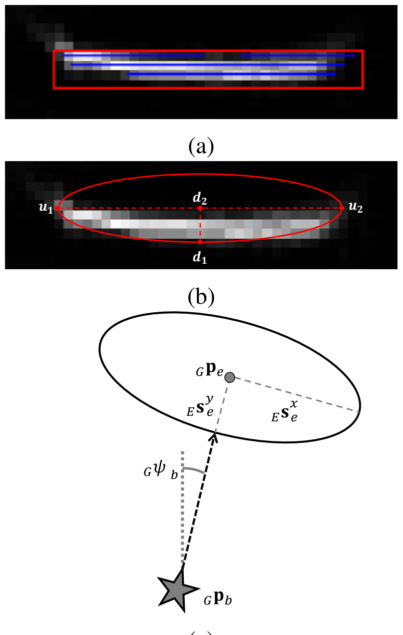


Fig. 2: We show a sample segment of a U-map (a), representing an obstacle with its detection bounding box overlaid in red. An overlay of an ellipse on the obstacle (b), showing the different quantities used for the obstacle calculation – u_1, u_2, the left- and right- most edges of the obstacles in u, and d_1 the maximum disparity and d_2 the mean disparity of the object. Part (c) shows the resulting ellipse in the world frame, and the 5 values that describe its center ${}_G\mathbf{p}_e$ and size ${}_E\mathbf{s}_e$.

G. G is the global/inertial frame, B is the body frame of the quadrotor (at the time of ellipse observation), and E is the ellipse's local frame. ${}_G\psi_b$ represents the yaw angle between the robot's body frame and the global coordinate frame.

At the first time step ($k = 0$), we initialize an ellipse for each object from the procedure described in Section III-B. Each ellipse is described by 5 parameters: 2 for the center ${}_G\mathbf{p}_e^{x,y}$ (in the global frame), 2 for the size (width and thickness) ${}_E\mathbf{s}_e^{x,y}$ (in the local frame of the ellipse), and yaw ${}_G\psi_e$ of the major axis of the ellipse relative to the world frame.

These values are derived from the bounding boxes from Section III-B, where f is the focal length in pixels, b is the baseline of the stereo setup, u_1 is the minimum u value of the bounding box (left edge), u_2 is the right edge, d_1 is the maximum disparity, and d_2 is the mean disparity. Fig. 2b shows the ellipse fitted to a U-map obstacle, and Fig. 2c, shows the dimensions of the resulting ellipse in the world frame.

$${}_B\mathbf{p}_e = \begin{bmatrix} \frac{b(u_1+u_2)}{2d_2} \\ \frac{fb}{d_2} \end{bmatrix} \quad (1)$$

$${}_B\mathbf{s}_e = \begin{bmatrix} \frac{b(u_2-u_1)}{2d_2} \\ \frac{fb}{d_2} - \frac{fb}{d_1} \end{bmatrix} \quad (2)$$

$${}_G\psi_e = {}_G\psi_b \quad (3)$$

Then, while the width ${}_E\mathbf{s}_e^x$ and thickness ${}_E\mathbf{s}_e^y$ of the ellipse remain in the ellipse's local frame (which is defined by the yaw ${}_G\psi_e$, and is the same as the robot's yaw at the time of observation ${}_G\psi_b$), the center coordinates need to be converted into the robot's world frame. Below, we describe the state by the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\sigma}$, which makes it simpler to perform operations on multiple ellipses.

$$\begin{aligned} {}_G\mathbf{R}_b &= \begin{bmatrix} \cos({}_G\psi_b) & -\sin({}_G\psi_b) \\ \sin({}_G\psi_b) & \cos({}_G\psi_b) \end{bmatrix} && (4) \\ {}_G\mathbf{p}_e &= {}_G\mathbf{R}_b \, {}_B\mathbf{p}_e + {}_G\mathbf{p}_b && (5) \\ \boldsymbol{\mu} &= \begin{bmatrix} {}_G\mathbf{p}_e^x & {}_B\mathbf{p}_e^y & {}_E\mathbf{s}_e^x & {}_E\mathbf{s}_e^y \end{bmatrix}^\top && (6) \end{aligned}$$

To get the covariance, we find the Jacobian of the equations derived in Eqs. (1 – 2) above.

$$\begin{aligned} \mathbf{J} &= \frac{\partial(x_b, y_b, ...)}{\partial(u_1, u_2, ...)} && (7) \\ &= \begin{bmatrix} \frac{b}{2d_2} & \frac{b}{2d_2} & 0 & -\frac{b(u_1+u_2)}{2d_2^2} \\ 0 & 0 & 0 & -\frac{fb}{d_2^2} \\ -\frac{b}{2d_2} & \frac{b}{2d_2} & 0 & -\frac{b(u_2-u_1)}{2d_2^2} \\ 0 & 0 & -\frac{fb}{d_1^2} & \frac{fb}{d_2^2} \end{bmatrix} && (8) \\ {}_B\boldsymbol{\sigma} &= \mathbf{J}^\top \mathbf{J} + \mathbf{Q} && (9) \end{aligned}$$

Where $\mathbf{Q}$ is empirically determined measurement noise, usually diagonal. Then the covariance is rotated into the world frame:

$$\begin{aligned} {}_G\mathbf{R}_{4b} &= \begin{bmatrix} {}_G\mathbf{R}_b & 0 \\ 0 & {}_G\mathbf{R}_b \end{bmatrix} && (10) \\ {}_G\boldsymbol{\sigma} &= {}_G\mathbf{R}_{4b}^\top \, {}_B\boldsymbol{\sigma} \, {}_G\mathbf{R}_{4b} && (11) \end{aligned}$$

Over subsequent timesteps ($k > 0$), we again follow a similar procedure. Each of the detections in the current frame is transformed into the world frame. Then, for each object in the current frame, it is compared with objects found in the previous frames, and a confidence value is calculated for the detected objects belonging to the same physical entity.

In order to do this check on comparable ellipses, the ellipses in the existing map are rotated to the quadrotor's current yaw. We now introduce a new coordinate frame: N, which is the global frame rotated to match the quadrotor's current yaw.

$$\begin{aligned} {}_N\boldsymbol{\mu} &= |{}_G\mathbf{R}_b|_G\boldsymbol{\mu} && (12) \\ {}_N\boldsymbol{\sigma} &= |{}_G\mathbf{R}_{4b}|_G^\top\boldsymbol{\sigma}|{}_G\mathbf{R}_{4b}| + \mathbf{Q} && (13) \end{aligned}$$

We estimate the probability of the two ellipses belonging to the same object using the Gaussian probability density function, computing the probability of the means and covariances originating from the same distribution:

$$p = \mathrm{pdf}({}_N\boldsymbol{\mu}_2 -_N \boldsymbol{\mu}_1,_N \boldsymbol{\sigma}_2) \qquad (14)$$

If the two objects are determined to belong to the same physical entity, then their ellipses, described by the old mean and covariance ${}_N\boldsymbol{\mu}_1$ and ${}_N\boldsymbol{\sigma}_1$ and new ${}_N\boldsymbol{\mu}_2$, ${}_N\boldsymbol{\sigma}_2$ are merged based on minimizing their combined covariance [19]. The output is the merged ${}_N\boldsymbol{\mu}_m$ and ${}_N\boldsymbol{\sigma}_m$, and we accept the latest yaw estimate ${}_G\psi_2$ as the yaw of the resulting obstacle.

$$\begin{aligned} {}_N\boldsymbol{\sigma}_m &= ({}_N\boldsymbol{\sigma}_1^{-1} +_N \boldsymbol{\sigma}_2^{-1})^{-1} && (15) \\ {}_N\boldsymbol{\mu}_m &= {}_N\boldsymbol{\sigma}_m \, ({}_N\boldsymbol{\sigma}_1^{-1} \, {}_N\boldsymbol{\mu}_1 +_N \boldsymbol{\sigma}_2^{-1} \, {}_N\boldsymbol{\mu}_2) && (16) \\ {}_G\psi_m &= {}_G\psi_2 && (17) \end{aligned}$$

Note that we add to the diagonal of the covariance an extra error to reflect the additional uncertainty of the merged estimate, especially since we allow for dynamic obstacles. It is possible to neglect this term, but then dynamic obstacles have a very significant time delay between changed position and update in the map.

B. Waypoint Planning

We plan a path forward directly in front of the quadrotor, then take the shortest path around any obstacles that are in the way. The general idea behind the algorithm we use is that the shortest path is along the edge of an obstacle (assuming we inflate all obstacles by a safety margin), as long as it does not overlap with another obstacle.

The general procedure for this waypoint planning is described in Algorithm 1. We use the version of the ellipses rotated into the current yaw angle of the robot to simplify calculations, as these are already available from Eq. (12, 13).

For determining whether there is a collision (function *hasCollision* in the algorithm) between the waypoint and an obstacle, we use the following formulas, where a is the width of the obstacle at the intersection of the waypoint with the ellipse (${}_G\mathbf{p}_w^y$), and "collision" checks whether the x coordinate of the waypoint, ${}_G\mathbf{p}_w^x$, is within the width of the ellipse or not.

$$\begin{aligned} a &= {}_E\mathbf{s}_e^x \sqrt{\frac{1 - ({}_G\mathbf{p}_w^y -_G \mathbf{p}_e^y)^2}{{}_G\mathbf{s}_e^{y\,2}}} && (18) \\ \text{collision} &= \left| \frac{{}_G\mathbf{p}_w^x -_G \mathbf{p}_e^x}{a} \right| > 1.0 && (19) \end{aligned}$$

The algorithm in described in Algorithm 1 continues to test waypoint candidates until an allowable collision-free path is found. In the case that there is no admissible path, the quadrotor plans a path to the side until an admissible path is found. However, this algorithm is designed for handling environments with sparse obstacles, rather than complex environments.

V. Implementation

In the following section, we introduce the physical test system and its constituent parts, and then describe an efficient implementation of the developed obstacle avoidance algorithms within the different hardware components.

A. Test System

The system consists of three main hardware components: the stereo vision system on FPGA, mobile CPU for higher-level vision processing (housed on the same board as the FPGA), and a PX4 FMU for low-level flight control. An overview of the components and their respective tasks is

Algorithm 1: Waypoint Checking

```
Input : waypoints, a list of waypoints in robot's body frame, $(x, y)$
Output: wps_out, waypoints representing path with no obstacle collisions, $(x, y)$

foreach wp in waypoints do
    wps_to_check.append(wp)
    while allowed_wps is empty do
        foreach wpc in to_check_wps do
            collision ← obstacles.hasCollision(wpc)
            if collision then
                leftpath ← obstacle.avoidLeft(wpc)
                rightpath ← obstacle.avoidRight(wpc)
                to_check_wps.append(leftpath)
                to_check_wps.append(rightpath)
            else
                allowed_wps.append(wpc)
        end
    end
    wps_out($k$) ← allowed_wps.closest(wps_out($k-1$))
    k++
end
```

shown in Fig. 3. We use a combination of FPGA and mobile CPU as presented in [1] to process image data. The mobile CPU receives disparity values estimated within the FPGA and calculates the U-map. In conjunction with an FMU for attitude estimation the detected obstacles are converted into ellipses in the world frame. Based on a short term map waypoints are generated and sent to the position controller on the FMU. An optical flow sensor as shown in [20] is used to measure local position and support the short term mapping. Though integrating optical flow leads to some drift, the advantage of our approach is that it is immune to slow drift in the position estimate.

We use a Samsung Exynos 4412 System on Chip Module with a built in Cortex-A9 Quad Core mobile CPU. The FPGA system is based on a Xilinx Artix7 XC4A100T module and is connected to the dedicated camera interface of the mobile CPU. The two cameras are equipped with MT9V034 CMOS image sensors from Aptina with global shutter. Images with 640x480 pixels resolution and corresponding disparity map are provided to the mobile CPU with 60 frames per second update rate.

We use a custom quadrotor based on an ARDrone frame and motors. A PX4 FMU is used for attitude and position estimation. This system is powered by a 2200 mAh battery, running at 11 Volts. This is sufficient for about 15 minutes of flight time, with all the systems running and doing vision processing at frame rate. Fig. 4 shows the test system quadrotor, where the FPGA/mobile CPU vision system is on top, stereo head mounted in front, and a downward facing PX4 FLOW optical flow sensor is placed in the back.

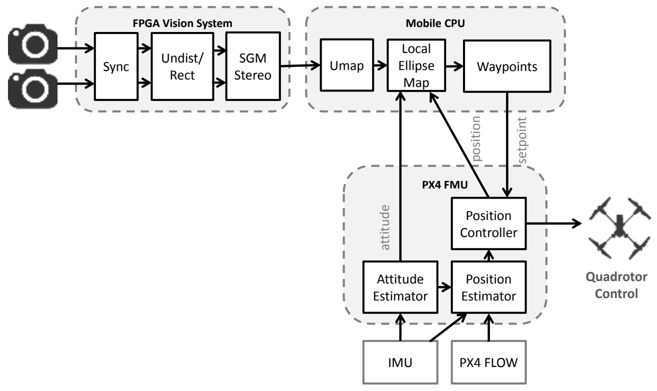


Fig. 3: System overview, the FPGA vision system estimates disparity values and transmits them to the mobile CPU. Obstacles are detected in the mobile CPU based on a U-map and transformed to ellipses. Generated waypoints are sent to a position controller running on the FMU. Finally thrust commands are sent to the motor controllers.

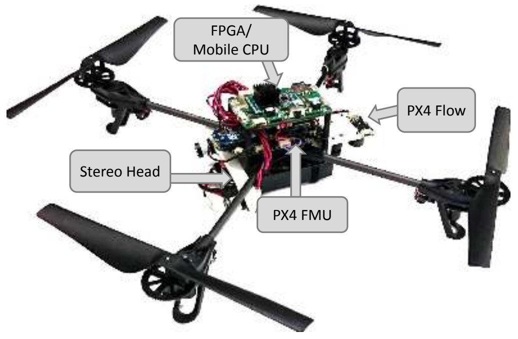


Fig. 4: Test system quadrotor, with relevant parts labeled. At the center of the quadrotor, we have the PX4 FMU which is responsible for the low-level flight control and state estimation, and above is the stereo vision on FPGA system, including a mobile CPU where these algorithms run. Additionally, there is a stereo head in the front, and a downward-facing optical flow sensor with sonar in the back.

B. Obstacle Detection

In order to be able to run this on a computationally constrained platform, we need to construct U-maps and segment obstacles from U-maps very efficiently. Our goal is to process the images at least at frame rate, so the total time budget from acquisition to outputting a new waypoint estimate is only 16 ms.

When we construct a U-map from the disparity image, we subsample the U-map by a factor of 4 in the horizontal dimension. We still iterate over every pixel in the original image, so we do not lose any data, but simply filter out some of the noise by using larger bins.

Then, we iterate over each horizontal line in the U-map, and collect all contiguous lines whose individual pixels are above a certain threshold (10% of the maximum value), and the sum of whose pixels is also above a different threshold

(200% of the possible value of each individual histogram bin).

The next objective is to group the contiguous lines into cohesive objects that may span several disparity ranges. This is accomplished by iterating over every line, in order of biggest disparity (closest obstacle) first, and finding any lines that overlap horizontally in the next farther disparity. If so, these lines are now considered part of the same object, shown as red bounding boxes in Fig. 5c.

After iterating through all detected lines and merging into detection boxes, each detection box now represents a distinct obstacle.

C. *Waypoint Planning*

Waypoints are laid out evenly spaced directly in front of the quadrotor. They are then fixed in one axis, and are only allowed to move to the left or right to avoid obstacle collisions.

During our experiments, we used a small number of waypoints with large spacing. As long as obstacles were inflated by at least half the waypoint spacing, this is safe and no obstacles will be missed due to being between waypoints.

An updated version of the closest waypoint, in the world frame, is sent from the mobile CPU to the FMU with each new image frame (60 Hz), so the quadrotor is always following the most current estimate of the world state.

VI. Results

In this section, we first show results of the obstacle detection and path planning algorithm in indoor and outdoor environments, including the expected latency. We then show autonomous flight and obstacle avoidance in both environments on our physical test platform.

A. *Obstacle Detection and Path Planning*

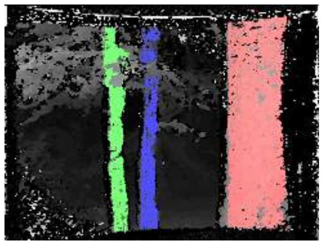
Fig. 5a shows a forest scene with trees detected as obstacles. The segmentation of the three closest trees is overlaid as colors in Fig. 5b. Detected obstacles are visible in the U-map in Fig. 5c and the corresponding ellipses in a short term map are presented in Fig. 5d. Obstacle detection and ellipse representation transformation runs at frame rate of the disparity maps.

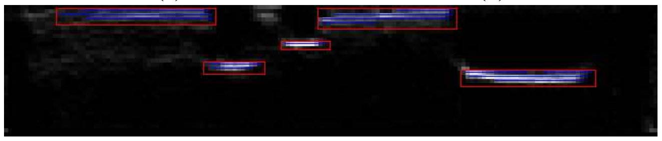
The timing of the individual steps of the pipeline, including the cumulative latency is shown in Table I. The timings are taken from the beginning of the image exposure to the updated waypoint being sent to the flight controller. The disparity estimation runs at frame rate of the image sensors. The radial distortion correction module buffers 40 lines of the image and therefore causes most of the latency within the FPGA system. On the mobile CPU, the U-map generation is the most time-consuming task.

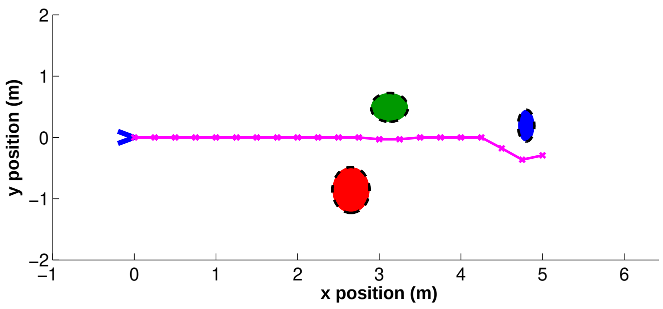
B. *Image Segmentation*

Additionally, as can be seen in the results in Fig. 5b, this algorithm can also be used for image segmentation by back-projecting obstacles detected in the U-map back into image space. This approach gives very clean disparity-based segmentation of obstacles. If further accuracy is needed, then the ground plane pixels can also be removed from the segmentation using the V-map and ground plane estimation [17].

(a)


(b)


(c)



(d)

Fig. 5: An example frame from an outdoor forest scenario including corresponding image segmentations and planned waypoints. The grayscale image is shown in (a). Part (b) shows the disparity map, with the segmentation of the closest three obstacles shown in colors, the corresponding U-map is shown in (c), where individual obstacles are highlighted with red bounding boxes. The current position of the quadrotor in the world frame (d) is shown as a blue arrow, the suggested waypoint path around the three elliptical obstacles is shown in pink. Note that some of the background obstacles are out of range of the map representation.

Device	**Operation**	**Time** (us)	**Cumulative Latency** (us)
Camera	Image Capture	3000 us	3000 us
FPGA	Undistort./Rectification	1200 us	4200 us
	SGM Stereo	300 us	4500 us
Mobile CPU	Camera Driver	4100 us	8600 us
	Umap Extraction	5300 us	13900 us
	Obstacle Segmentation	80 us	13980 us
	Ellipse Map	110 us	14090 us
	Waypoint Planning	20 us	14110 us

TABLE I: Table of timings of individual parts of the algorithm and overall latency. The timings begin at the beginning of image exposure and end at waypoint generation.

C. Dataset Testing

In order to validate both our image segmentation and planning algorithms, we recorded 17 diverse datasets by carrying our system around hand-held, and avoiding running into obstacles. Of the datasets, 7 were in an indoor/lab setting, 5 were taken in a dense forest, 3 in a more sparsely occupied park, and 2 datasets had quickly moving dynamic obstacles. In each of the situations except the dynamic obstacle scenarios, the quadrotor was moved at a speed between 1 and 5 m/s. We confirmed correct obstacle segmentation and reasonable avoidance behavior by inspection of the planned path.

D. Autonomous Flight

We verified the system in indoor and outdoor test flights. We set a start and an end waypoint with obstacles in direct line of sight. With the obstacle detection and path planning algorithm enabled the quadrotor sucessfully avoided any obstacles on the way to the end waypoint. This paper is accompanied by a video showing the flight tests.

VII. Conclusion

In this work, we have presented a low latency obstacle avoidance system that is suitable for running on low-power embedded devices. We demonstrate successful obstacle avoidance on an MAV platform in a variety of outdoor and indoor environments.

Our algorithm has two major contributions over existing work: first, it uses a novel method of segmenting obstacles directly from the U-map representation of a disparity map, which is much faster than plane fitting or more complex obstacle detection schemes. Second, it demonstrates the very low computational power requirements of our system, by processing 640x480 pixel images at a frame rate of 60 Hz, which is a larger resolution and twice the frame rate of other systems, on-board a low power mobile CPU and FPGA combination.

Due to the simplicity of the algorithm, this system is suitable for conversion into a single safety layer chip, which would output obstacle positions and object segmentations at a high frame rate with minimum latency. The cumulative latency from image exposure to waypoint planning is 14.1 milliseconds. At a flight speed of 5 m/s the system is therefore able to react to an obstacle within 0.07 m flight distance. This could then be used as an additional level of robustness for more complex systems, or a pre-processing step for object detection or mapping. Such a fall-back safety layer is essential in any applications in which robots are placed in the same environment as humans, who tend to be the most unpredictable dynamic obstacles.

References

[1] D. Honegger, H. Oleynikova, and M. Pollefeys, “Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu,” in *Intelligent Robots and Systems (IROS), IEEE International Conference on*, IEEE, 2014.

[2] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive uav control in cluttered natural environments,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1765–1772, IEEE, 2013.

[3] S. Hrabar, “3d path planning and stereo-based obstacle avoidance for rotorcraft uavs,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pp. 807–814, IEEE, 2008.

[4] S. Scherer, S. Singh, L. Chamberlain, and S. Saripalli, “Flying fast and low among obstacles,” in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2023–2029, IEEE, 2007.

[5] F. Andert, F. Adolf, L. Goormann, and J. Dittrich, “Mapping and path planning in complex environments: An obstacle avoidance approach for an unmanned helicopter,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 745–750, IEEE, 2011.

[6] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, “Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing,” in *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 2472–2477, IEEE, 2011.

[7] 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 *International Symposium on Robotics Research (ISRR)*, pp. 1–16, 2011.

[8] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[9] S. Ahrens, D. Levine, G. Andrews, and J. P. How, “Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments,” in *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on*, pp. 2643–2648, IEEE, 2009.

[10] A. J. Barry and R. Tedrake, “Pushbroom stereo for high-speed navigation in cluttered environments,” in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[11] J. Biswas and M. Veloso, “Depth camera based indoor mobile robot localization and navigation,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, IEEE, 2012.

[12] F. Oniga and S. Nedevschi, “Processing dense stereo data using elevation maps: Road surface, traffic isle, and obstacle detection,” *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 3, pp. 1172–1182, 2010.

[13] I. Lenz, M. Gemici, and A. Saxena, “Low-power parallel algorithms for single image based obstacle avoidance in aerial robots,” in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 772–779, IEEE, 2012.

[14] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf, and F. S. Osório, “Mobile robots navigation in indoor environments using kinect sensor,” in *Critical Embedded Systems (CBSEC), 2012 Second Brazilian Conference on*, pp. 36–41, IEEE, 2012.

[15] A. Beyeler, J.-C. Zufferey, and D. Floreano, “3d vision-based navigation for indoor microflyers,” in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1336–1341, IEEE, 2007.

[16] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, “Stereo vision based indoor/outdoor navigation for flying robots,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3955–3962, IEEE, 2013.

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

[18] I. Ulrich and J. Borenstein, “Vfh+: Reliable obstacle avoidance for fast mobile robots,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 2, pp. 1572–1577, IEEE, 1998.

[19] J. E. Davis, “Combining error ellipses,” *CXC memo*, 2007.

[20] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys, “An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1736–1741, IEEE, 2013.