Auton Robot
DOI 10.1007/s10514-012-9293-0

River mapping from a flying robot: state estimation, river detection, and obstacle mapping

Sebastian Scherer · Joern Rehder · Supreeth Achar · Hugh Cover · Andrew Chambers · Stephen Nuske · Sanjiv Singh

Received: 1 August 2011 / Accepted: 29 March 2012

Abstract Accurately mapping the course and vegetation along a river is challenging, since overhanging trees block GPS at ground level and occlude the shore line when viewed from higher altitudes. We present a multimodal perception system for the active exploration and mapping of a river from a small rotorcraft. We describe three key components that use computer vision, laser scanning, inertial sensing and intermittant GPS to estimate the motion of the rotorcraft, detect the river without a prior map, and create a 3D map of the riverine environment. Our hardware and software approach is cognizant of the need to perform multi-kilometer missions below tree level with size, weight and power constraints. We present experimental results along a 2 km loop of river using a surrogate perception payload. Overall we can build an accurate 3D obstacle map and a 2D map of the river course and width from light onboard sensing.

Keywords 3D obstacle mapping · Visual localization · Micro aerial vehicles · Self supervised learning · 3D ladar scanning

Notation

s_k	a vehicle state at time t_k
x_i	a single measurement collected with one of the sensors
m	the total number of vehicle states considered in the optimization
n	the total number of measurements acquired with a set of different sensors that are considered in the optimization
$\mathrm{p}(x_i, s_1, \ldots, s_m)$	a probability density function (PDF) describing the distribution of the measurements x_i
$\mathrm{h}_i(s_1, \ldots, s_m)$	a function modeling the sensor that provided measurement x_i based on the vehicle states $s_1, \ldots, s_m$.

1 Introduction

We are developing perception and planning algorithms to be used by a low-flying micro air vehicle (MAV) to autonomously explore rivers; mapping their width and the surrounding canopy. In some cases, the canopy can be so thick and high covering a river that it blocks GPS signals and the problem becomes one of simultaneous localization and mapping in an unstructured three-dimensional environment. The localization and mapping problem is complicated because in our mission the vehicle will fly one-way up the river and

S. Scherer (✉) · S. Achar · H. Cover · A. Chambers · S. Nuske · S. Singh
Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA
e-mail: basti@cmu.edu

S. Achar
e-mail: supreeth@cmu.edu

H. Cover
e-mail: hcover@cmu.edu

A. Chambers
e-mail: achambers@cmu.edu

S. Nuske
e-mail: nuske@cmu.edu

S. Singh
e-mail: ssingh@cmu.edu

J. Rehder
Institute of Control Systems, Hamburg University of Technology, Eissendorfer Strasse 40, 21073 Hamburg, Germany
e-mail: joern.rehder@tuhh.de

Published online: 25 April 2012

Fig. 1 A typical riverine environment that we expect to map. A small rotorcraft can fly above the shallow, fast-moving water and yet remain below the thick canopy to navigate and map the river. The foliage along the banks is dense enough to block or seriously degrade GPS signals

then return quickly at high altitude. Figure 1 shows a typical riverine environment. Exploration from a flying vehicle is attractive because it extends the sensing horizon and removes complications of navigating in shallow water and aquatic vegetation. However, a flying solution also adds constraints on the size, weight and power available for perception. This is a significant constraint given that the multi-kilometer missions will force all the sensing/computation to be conducted onboard. Given the size of rotorcraft that could reasonably fly in environments with thick canopy, it will be necessary to keep all the sensing and computation components to less than one kilogram.

These constraints on payload and the inability to rely on GPS have significant implications for our approach. First, we will need to depend on perception to produce a high resolution 6 degree of freedom (DOF) pose estimate that is much more stable than can be produced by simply integrating inertial sensors. Second, any active imaging, such as from laser scanning, will be required to be very lightweight and low power and hence will be short range. Third, predicting the river's course and following it without a prior map will require a perception system that looks significantly further than could be sensed through laser ranging.

In summary we derive the following requirements to navigate autonomously:

1. a locally consistent state estimation system,
2. the ability to sense and avoid obstacles, and,
3. a direction to follow the course of the river.

After navigating the river the returned map requires:

1. a global reference frame,
2. the course of the river,
3. the width of the river, and
4. the height of vegetation along the shore.

In this paper we describe our approach for mapping on a micro aerial vehicle in four stages: The first is a graph-based optimization for state estimation of the vehicle's motion in 6DOF. Using both relative and global constraints from visual odometry, inertial sensing, and sparse GPS, we demonstrate its ability to globally estimate the vehicle's state while maintaining an accuracy that allows for precise local mapping. The second is a long range color vision system that uses a forward pointing color camera to automatically find the river even with significant variation in the appearance of the river. We solve this problem with a self-supervised method that continually learns to segment images based only on an estimate of the horizon (from inertial sensing) and some simple heuristics that describe riverine environments. The third is an efficient method for creating obstacle maps for motion planning in 3D that are high resolution and can be scrolled over distances of multiple kilometers efficiently. The fourth contribution is a short range, laser-ranging based system tasked with obstacle detection and the creation of a metric, three-dimensional map.

Parts of our system were first shown in Chambers et al. (2011) and Achar et al. (2011). Since then our approaches have been refined to reflect our latest results. In state estimation we have improved our method, describe in detail how we achieve accuracy for loop closure and reduce computation with node merging. The river detection algorithm includes more analysis, a sun reflection detection, and a detailed description of artificial horizon line estimation. We have added a section on an improved distance transform algorithms and a comparison with other state of the art aproaches.

1.1 Related work

Previous work in autonomous river mapping has utilized small boats (Leedekerken et al. 2010) or higher altitude, fixed wing UAVs (Rathinam et al. 2007). While these platforms could be more practical in simple riverine environments, we aim to develop a platform that can perform in the most difficult situations such as rapidly flowing water, obstructed waterways, or dense forest canopies. Preliminary work in river mapping on small rotorcraft using passive vision and ultrasonic ranging (for elevation estimation) has been reported over short distances (Yang et al. 2011). Our work is similarly motivated but we explicitly consider substantially longer missions in which it is important to not only map the extent of the river but also to map the vegetation along the shore line and avoid obstacles that might appear in the middle of the river.

Obstacle avoidance is a necessary capability to operate close to the trees present at low altitude. Hrabar and Gaurav (2009) performed experiments using optical flow for obstacle avoidance, however additionally a stereo camera was used to prevent collisions from straight ahead zero-flow regions. Stereo image processing with evidence grid based filtering was also used by Andert et al. (2010), Andert and

Goormann (2007) to create a map based on stereo imagery that avoided obstacles reactively in simulation. Viquerat et al. (2007) presented a reactive approach to avoid obstacles using Doppler radar. Grzonka et al. (2009) presented a quadrotor that is capable of localization and simultaneous localization and mapping (SLAM). Mapping with a monocular camera was shown in Weiss et al. (2011) and a line laser in Shen and Kumar (2011).

The four main focus areas of this paper are position estimation, river detection, mapping, and obstacle detection for a lightweight flying vehicle. Next we discuss related work in each of these areas:

State estimation using a suite of heterogeneous sensors constitutes a well-studied problem in robotics. In the past, recursive filters have been successfully demonstrated for a variety of applications (e.g. Reid et al. 2007; Eustice et al. 2005). In our application however, we expect the state to change significantly whenever sparse GPS measurements become available, which renders approaches that linearize only once not well suited. Furthermore, recent research by Strasdat et al. (2010) suggests that recursive filtering performs inferiorly compared to optimization approaches for most problems.

We treat the state estimation as a non linear optimization problem, using a graph to represent the interdependence of vehicle states and measurements. In this sense, our approach covers a sub-class of problems that are addressed by g2o in Kuemmerle et al. (2011). In contrast to this work however, we employ an on-line graph reduction scheme similar to Folkesson and Christensen (2007) and Konolige and Agrawal (2008) which enables real-time throughput for a sliding window of vehicle states spanning multiple minutes. The estimation is performed using a sparse optimization framework by Lourakis (2010) which itself shares similarities with g2o and provides interfaces to a similar set of sparse solvers.

In the second area, our system uses a visual river segmentation algorithm to detect the extent of the river for mapping and long-range guidance. Most previous work on detecting water in images has been focused on detecting water hazards like puddles using color, texture and stereo disparity cues for autonomous ground vehicles (Rankin et al. 2004; Rankin and Matthies 2010). Our solution automatically learns models of river and shore appearance for segmentation by exploiting the structure of riverine environments in a scheme that shares some similarities to self supervised road detection (Dahlkamp et al. 2006).

In the third area of 3D mapping, a probabilistic map similar to Martin and Moravec (1996) is kept to filter the sensor data and to compute the likelihood of occupancy. For motion planning we require an obstacle expansion that is typically calculated by searching for the closest obstacle within a desired radius. However, aerial vehicles must stay far away from obstacles and therefore want a large obstacle expansion. The obstacle expansion is related to the distance transform and Meijster et al. (2000) presented an efficient algorithm to globally calculate the distance transform. Kalra et al. (2006) showed an algorithm to incrementally construct Vornonoi diagrams. We show an efficient algorithm similar to D* Lite (Koenig and Likhachev 2002) that updates the distance transform up to a limit incrementally. We expand on our previous work (Scherer et al. 2009) and incorporate some of the changes of Lau et al. (2010).

In the fourth area of obstacle detection and 3D mapping with an articulated 2D lidar, researchers have investigated various mounting and articulation schemes to focus the lidar scan pattern in specific regions of interest (Wulf and Wagner 2003; Holz et al. 2010). Typically, only a qualitative comparison of different scan patterns is offered. However, Desai and Huber (2009) provide a objective, quantitative method for choosing a ladar mounting and articulation pattern by measuring density and uniformity of sensor measurements in simulation. An open area of research is to compare obstacle detection probability for small obstacles for different laser scanning patterns. This is particularly important for micro aerial vehicles (MAVs) since a collision with *any* size obstacle can potentially damage and destroy the vehicle. We analyze our 3D scan pattern to find the probability of detecting small obstacles in the path of the rotorcraft and we use this information to dictate maximum safe vehicle velocities.

1.2 Contributions

In summary, the contributions of this paper are

- an online state estimation system that combines visual odometry and inertial measurements with intermittent GPS information,
- a self supervised vision based river detector that can the handle large intra- and inter-scene variations in water surface appearance commonly observed in riverine environments,
- a scrolling incremental distance transform algorithm for efficient local obstacle cost calculation used in planning, and
- a novel scanning ladar configuration and analysis for obstacle detection and mapping.

2 Approach

Our approach to solving the river mapping problem is dictated by the dominate perception challenges of the application domain. Challenges include state estimation, navigating the course of the river, obstacle detection, and measuring the river's width and the clearance and structure of the canopy above the water.

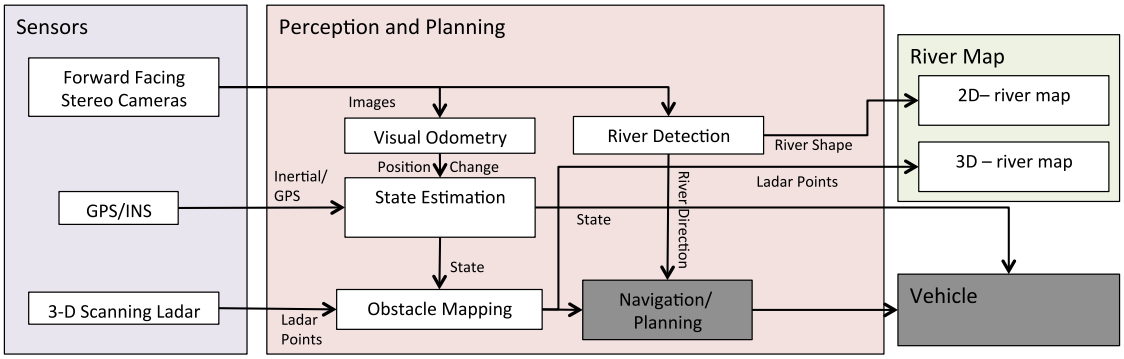


Fig. 2 The perception architecture. The sensor inputs for perception are a stereo camera pair, a 3D scanning ladar, and GPS/INS. The inputs are used to estimate the vehicle's state, detect rivers, and map obstacles

To meet the perception challenges we must combine several sensors: GPS, IMU, stereo camera pair and lidar. We must use light-weight and low fidelity sensing because the payload is limited. No single sensor is reliable on its own. We draw on the complementary characteristics of the sensor suite to provide the capabilities for our rotorcraft to execute a river mapping mission. For example, cameras are good at estimating motion, however not reliable enough to detect obstacles. Laser scanners on the other hand can detect obstacles well, but are difficult to use for localization or detecting the river. Our overall perception architecture that summarizes the algorithms developed and how they fit together is shown in Fig. 2.

For state estimation, described in Sect. 2.1, the GPS is low accuracy and has intermittent coverage, the stereo-camera can be used for precise relative motion estimates, but integrating the estimates gives a solution that drifts unboundedly, and likewise an IMU produces unbounded position drift, but can be used over a short period to correct small error and can be used to bound pitch and roll. Our approach fuses GPS, IMU and visual odometry within a robust framework that can seamlessly handle the dropouts and the various frequencies in the sensor measurements and provide a solution in dense, occluded, natural and dynamic terrain, comparable in relative accuracy to a high-end, heavy, GPS/INS unit operating in wide-open environments.

For predicting the course of the river, Sect. 2.2, we need a long range sensor, which rules out a lightweight short range laser scanner and, therefore, we use a computer vision solution based on a color camera to detect the extent of the river and to choose a suitable course to follow.

Our approach for obstacle and environment mapping is presented in Sect. 2.3. The approach uses a scrolling 3D evidence grid to efficiently update the distance to the closest obstacle. This distance or cost is important for motion planning since it allows us to stay away from obstacles if possible and get close if necessary.

The final part of the approach, described in Sect. 2.4, is the 3D laser scanner for obstacle avoidance and canopy mapping. We use a lightweight laser scanner that when stationary provides a single plane of range information, which we actuate in a novel configuration that is spinning and tilted off-axis to give a 360 degree view around the vehicle (albeit with small cone-shaped blind-spots directly above and below the vehicle). We explain key details such as analyzing the probability of detecting small obstacles at various velocities.

2.1 State estimation

Knowledge of the position and orientation of a mobile robot is essential for a variety of applications such as path planning and mapping. To autonomously follow a river and avoid obstacles, we need information about the state to back-project laser scans and river detection results into locally accurate maps, while maintaining a consistent estimate of the position and orientation in a global coordinate frame. While visual odometry and inertial measurements can be employed to accurately estimate the change in the vehicle state, they are prone to drift, resulting in unbounded errors. On the other hand, GPS provides an estimate of the position in a global coordinate frame with a bounded error. Unfortunately, riverine environments will seriously degrade any GPS signal, resulting in intermittent availability of global position information. In fact, we expect GPS to fail for periods of several minutes. During these periods, the vehicle state will drift, which will result in a potentially significant displacement in the estimated position from the global position measurement when the GPS signal is regained.

To summarize, we require our state estimation

- to fully exploit the information provided by intermittent GPS readings,
- to be able to deal with significant changes in the state in a global coordinate frame,
- to avoid discontinuities in the vehicle path and resulting inconsistencies in the map, and
- to provide state estimates with real-time throughput.

In order to meet these requirements, we employ an approach that treats state estimation as a nonlinear optimization problem over a history of states and measurements with a graphical representation. Our approach is most similar to Folkesson and Christensen (2007), but in contrast to that work which uses a Gauss-Seidel type relaxation, our optimization employs the Levenberg-Marquardt algorithm (Marquardt 1963) and changes multiple states per update step.

Given a set of measurements, we determine the optimal set of vehicle states as the one for which the given set of measurements were most likely to occur.

The probability of making a single measurement x_i for a given set of vehicle states $S = [s_1, \ldots, s_m]$ is given by the conditional probability $\mathrm{p}(x_i \mid s_1, \ldots, s_m)$.

As the measurements are assumed to be conditionally independent given the states, a set of measurements taken at different times results in the overall conditional probability

$$\mathrm{p}(x_1, \ldots, x_n \mid s_1, \ldots, s_m) \propto \prod_{i=1}^{n} \mathrm{p}(x_i \mid s_1, \ldots, s_m) \tag{1}$$

Then, the optimal set of vehicle states $[s_1, \ldots, s_m]$ is the one which maximizes the overall probability

$$[s_1, \ldots, s_m] = \arg\max_s \big(\mathrm{p}(x_1, \ldots, x_n \mid s_1, \ldots, s_m)\big) \tag{2}$$

Rather than finding the maximum of this distribution, it is more common to minimize the negative logarithm of the likelihood function.

$$\begin{aligned}[s_1, \ldots, s_m] &= \arg\min_s \big(-\log\big(\mathrm{p}(x_1, \ldots, x_n \mid s_1, \ldots, s_m)\big)\big) \\ &= \arg\min_s \left(-\sum_{i=1}^{n} \log\big(\mathrm{p}(x_i \mid s_1, \ldots, s_m)\big)\right)\end{aligned} \tag{3}$$

In the following we will assume that the measurement error is zero-mean Gaussian distributed. This is not a necessary assumption and in fact other probability distributions may be utilized in a graph-based approach. However, with this assumption, it is easier to motivate the problem as a nonlinear least square optimization, and it is a valid approximation for many real world applications.

Let ν_i be a Gaussian distributed random vector with zero mean and covariance matrix C_i, then a measurement is modeled as $x_i = \mathrm{h}_i(\hat{s}_1, \ldots, \hat{s_m}) + \nu_i$.

Minimizing the negative logarithm of the corresponding probability desity function leads to the nonlinear least square problem

$$\begin{aligned}&[s_1, \ldots, s_m] \\ &= \arg\min_s \left(\sum_{i=1}^{n} \frac{1}{2} \big(\mathrm{h}_i(S) - x_i\big)^T {C_i}^{-1} \big(\mathrm{h}_i(S) - x_i\big)\right) \\ &= \arg\min_s \left(\sum_{i=1}^{n} \big\| Q_i \big(\mathrm{h}_i(s_1, \ldots, s_m) - x_i\big) \big\|^2\right)\end{aligned} \tag{4}$$

where Q_i is a symmetric, positive definite matrix for which ${C_i}^{-1} = {Q_i}^T Q_i$. Thus, each measurement provides a weighted displacement of the modeled measurement $\mathrm{h}_i(s_1, \ldots, s_m)$ to the measurement x_i that the sensor provided.

2.1.1 Graph-based representation of the problem

The problem of estimating the vehicle state in an optimal fashion can be modeled as a graph of nodes that are connected by edges. In this model, the state s_k of the vehicle at a discrete point in time t_k is represented by a node. An edge in the graph respresents a constraint between states induced by a sensor reading x_i.

In general, sensor readings can be classified based on whether they measure a relative change in the state and thus a local measurement or whether they measure the state in a global coordinate frame and hence only depend on the vehicle state at a single point in time. Visual odometry is an example of a local measurement, as it measures a relative transformation between successive states. On the other hand, a GPS measurement provides information about a single vehicle state in a global coordinate frame. In order to represent both classes of sensor measurements in a unified way, we introduced a fictitious zero state s_0^* at the origin of the coordinate frame as suggested by Konolige (2004). In our current implementation, all readings of the onboard sensors can be modeled as functions of exactly two states, either successive ones $x_i = \mathrm{h}_i(s_{k-1}, s_k)$ in case of visual odometry and integrated gyroscope readings or as dependent on one vehicle state and the zero state $x_i = \mathrm{h}_i(s_0^*, s_k)$ for global measurements. As before, we assume measurement errors to be Gaussian distributed with zero mean.

In this sense, a node is memory that stores state information, while an edge is the realization of a cost function $\mathrm{f}(s_l, s_k, x_i) = Q_i(\mathrm{h}_i(s_l, s_k) - x_i)$.

Although the approach was motivated as a Maximum Likelihood Estimation in Sect. 2.1, one may think of the edges in the graph as nonlinear springs that apply forces to the nodes (Folkesson and Christensen 2007). Solving the nonlinear least square problem may in this sense be thought of as finding a state of the system where the overall energy in the system is minimal.

2.1.2 State parametrization and sensor models

We parametrize the state as

$$s = [\Psi, t]^{\mathrm{T}} = [\phi, \theta, \psi, t_x, t_y, t_z]^{\mathrm{T}} \tag{5}$$

where $\Psi = [\phi, \theta, \psi]^{\mathrm{T}}$ denotes the orientation in Euler angles, and $t = [t_x, t_y, t_z]^{\mathrm{T}}$ denotes the position in the global coordinate frame defined by GPS. Euler angles have the advantage of constituting a minimal parametrization of the orientation. Thus, they have a natural representation of the covariance as 3×3 matrix. On the other hand, Euler angles suffer from singularities. Through an appropriate choice of Euler angle conventions and vehicle coordinate frame in our implementation, these singularities coincide with kinematically infeasible orientations of the rotorcraft in normal operation.

The state is estimated based on sensor inputs from stereo visual odometry, integrated gyroscope readings, GPS location measurements, and inclination sensing from accelerometers.

In the following, function $\mathcal{R}(\Psi)$ will denote a $\mathbb{R}^3 \mapsto \mathbb{R}^{3\times3}$ mapping of Euler angles Ψ to a rotation matrix R and function $\mathcal{R}^{-1}(R)$ the inverse mapping from a rotation matrix R to Euler angles. The weight matrices Q, employed in each sensor constraint function, depend on the estimated measurement covariances of each sensor as described in Sect. 2.1. For simplicity, we assume that the coordinate systems of all sensors coincide with the vehicle coordinate frame. In our implementation, we have taken close care to accurately calibrate for the 6 degrees of freedom (DOF) transformations between the different sensors.

We employ the stereo visual odometry approach by Geiger et al. (2011) which provides a measurement of the relative 6 DOF transformation $x = [\Psi_{vo}, t_{vo}]^{\mathrm{T}}$ that the vehicle underwent from state s_i to state s_j along with an estimate C_{vo} of the uncertainty of the transformation. With $C^{-1} = Q_{vo}^{\mathrm{T}} Q_{vo}$, the visual odometry constraint is defined as

$$\mathrm{f}_{vo}\big(s_i, s_j, [\Psi_{vo}, t_{vo}]\big) = Q_{vo} \begin{bmatrix} \mathcal{R}^{-1}(\mathcal{R}(\Psi_i)^{\mathrm{T}}\mathcal{R}(\Psi_j)) - \Psi_{vo} \\ \mathcal{R}(\Psi_i)^{\mathrm{T}}(t_j - t_i) - t_{vo} \end{bmatrix} \tag{6}$$

Note that state nodes are appended to the graph at the rate of the visual odometry subsystem and that the 6 DOF transformation is used to propagate the most recent state to obtain an initial estimate for the newly added state.

Gyroscope readings Ψ_{gyro} are integrated according to Bryson and Sukkarieh (2007) and constrain the relative orientation of successive vehicle states

$$\mathrm{f}_{gyro}(s_i, s_j, \Psi_{gyro}) = Q_{gyro}\big[\mathcal{R}^{-1}\big(\mathcal{R}(\Psi_i)^{\mathrm{T}}\mathcal{R}(\Psi_j)\big) - \Psi_{gyro}\big] \tag{7}$$

Using the fictitious zero state s_0^*, location measurements t_{gps} obtained with the GPS receiver can be expressed similarly as dependent on two states:

$$\mathrm{f}_{gps}\big(s_0^*, s_j, t_{gps}\big) = Q_{gps}\big[\mathcal{R}\big(\Psi_0^*\big)^{\mathrm{T}}\big(t_j - t_0^*\big) - t_{gps}\big] \tag{8}$$

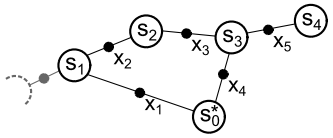


Fig. 3 A graphical representation of the vehicle state estimation problem. Nodes represent the vehicle state at different times t_k while edges corresponds to constraints on these states introduced by measurements x_i. Odometry measurements result in constraints on successive states, while intermittent global measurements anchor the state to a fictitious zero state s_0^*. The graph is sparsely interconnected, as each node is only constrained by measurements depending on a limited number of other nodes in the graph

Using accelerometers as an inclinometer as proposed by Konolige and Agrawal (2008), the orientation of the vehicle in the global coordinate frame can be constrained in 2 DOF. With the gravity vector g and accelerometer measurement a_g, the resulting constraint is expressed as

$$\mathrm{f}_g\big(s_0^*, s_j, a_g\big) = Q_g\big[\mathcal{R}(\Psi_j)^{\mathrm{T}}\mathcal{R}\big(\Psi_0^*\big)g - a_g\big] \tag{9}$$

Since Q_g has to account for measurement noise of the accelerometers and for accelerations of the aerial platform in flight as well, it was chosen to be sufficiently large in our implementation.

2.1.3 *Sparse optimization*

In our graph-based framework, the state of the system can be determined by collecting the information of all nodes, each of which holds a single vehicle state s_k. The state of the overall system is $S = [s_1, \ldots, s_m]^{\mathrm{T}}$. The energy of the system may be obtained by collecting the energy terms of all edges $E = [\mathrm{f}_1, \ldots, \mathrm{f}_n]^{\mathrm{T}}$. In order to move the system to a state with a lower overall energy, we employ the Levenberg-Marquardt algorithm (Marquardt 1963), using an optimized, publicly available implementation (Lourakis 2010). With the Jacobian $J_E = \delta E / \delta S$, the augmented normal equation is given by

$$\big(J_E{}^{\mathrm{T}} J_E + \lambda I\big)\Delta S = -J_E{}^{\mathrm{T}} S \tag{10}$$

where I is the identity matrix and λ is a damping parameter that is appropriately chosen so that the resulting matrix is always positive definite. As a result, the energy function is linearized around the current system state in every update step, thus making the solution exact and the framework able to deal with highly nonlinear constraints.

As depicted in Fig. 3, the local nature of most state measurements results in a sparsely interconnected graph structure, which directly corresponds to a sparsity pattern in the normal equation. The structure of the normal euqatio is predominantly block diagonal with additional non-zero entries in rows and columns corresponding to the fictitious zero

state. This structure ensures that the normal equation is well conditioned in most cases and allows for the application of sparse solvers, reducing the complexity of solving the system to a fraction of the complexity as compared with a dense system. Nevertheless, there is an upper bound to the number of past states that can be incorporated into the optimization while meeting the requirement for real-time throughput of the state estimation. Real-time throughput can be guaranteed by performing the optimization over a sliding window of the most recent states and their attached constraints, rather than the entire graph. This technique essentially keeps all past states prior to the sliding window constant, anchoring the global position and orientation of the more recent states by their connecting constraints.

2.1.4 Graph reduction

Inertial measurements only constrain the global orientation of the vehicle in 2 DOF, while global heading is not observable solely based on these measurements. In our implementation, heading is inferred from multiple GPS measurements which is applicable only if multiple GPS measurements are incorporated into the sliding window of adjustable states. Hence, the frequency at which GPS readings are available determines the size of the sliding window.

In order to extend the time frame spanned by the sliding window and thus relax the requirements for GPS availability, the state estimation includes a graph reduction scheme, similar to Konolige and Agrawal (2008) and Folkesson and Christensen (2007). We apply the following steps in order to marginalize a single state node s_x from the graph ... (also see Rehder et al. 2012)

1. Solve the optimization problem for the sub-graph of nodes and edges directly connected to s_x. Determine the overall state S of the sub-system as well as its approximated Hessian $H = J_E^{\mathrm{T}} J_E$ at the state of minimal energy.
2. Using the Schur complement (Konolige and Agrawal 2008), marginalize the rows and columns corresponding to state s_x from S and H. Select a vehicle state node from the sub-graph as root node and transform state S and Hessian H into the coordinate system of this state.
3. Render node s_x and all directly attached constraints inactive and introduce a new edge that constrains the remaining states in the sub-system where the transformation of the connected nodes into the coordinate system of the root node constitutes the sensor modeling function h and the transformed optimal transformations S constitute the measurement x with $H = Q^{\mathrm{T}} Q$.

The state estimation repeatedly marginalizes every other node from the sliding window until about 60 consecutive vehicle states have been removed. The number of consecutively marginalized states was derived empirically as a trade off between accuracy and the requirement to represent a time frame of sufficient extent to incorporate multiple sparse GPS readings. For stability, it does not marginalize nodes belonging to a small set of the most recently added nodes or vehicle states for which GPS measurements are available. As all states are used to back-project laser scans and river segmentation results into a global map, marginalized nodes are not discarded but rather attached to remaining active nodes by a constant 6 DOF transformation, thus successively forming sub-maps of rigidly connected states that are transformed according to the state of the attached active node.

2.2 Visual river detection and mapping

To explore the river, the rotorcraft needs some mechanism for determing the river's course so that it can move in the correct direction. Also, river width measurements are needed to build the river map. One approach would be to use the onboard laser sensor, but it has an effective range of around 10 to 15 meters which is sufficient for obstacle avoidance but not always enough for keeping the river banks in view or estimating the course of the river. Our solution segments color images from an onboard camera to find the extent of the river from the current viewpoint of the rotorcraft. Using knowledge of the vehicle's orientation and height above the river provided by the state filter, the extent of the river in the image can be projected into a local coordinate frame. This forms a local map of the river for guiding the rotorcraft along the river's course. Pose estimates generated by the visual odometry system are used to register these local maps to each other and by fusing many such local maps over time a complete river map is built.

2.2.1 Challenges

The main challenge in detecting the extent of the river in images taken from the rotorcraft is the variability in water's appearance. From satellite imagery or the viewpoint of a high flying aircraft, waterbodies reflect the sky which makes them fairly homogeneous in appearance and easy to segment. The situation is different for a low flying air vehicle. The water has reflections from the foliage and other structures on the bank; reflections from the sky and dark regions in the shadows of foliage. In addition, ripples in the water create variations in texture. As a result, the river appears to be highly inhomogeneous from the rotorcraft's point of view. This variability in river appearance in an image is illustrated in Fig. 4.

In addition, the appearance of the water's surface can vary greatly between riverine environments and even within the same environment as conditions change. Figure 5 shows three images taken from the same location on a river at different times of day and year. This high degree of variability

makes it difficult to use a supervised learning approach to build a single model for water appearance or a single classifier that will work robustly in different environments under varying conditions.

2.2.2 *Self supervised river detection*

We exploit scene structure to build a self supervised system that learns and constantly updates a function for discriminating between river and non-river regions. The algorithm utilizes artificial horizon line information provided by the onboard IMU to automatically generate training data and update the classifier at every frame. We assume that the river lies on or below the ground plane (this is true except for places with a significant upward slope like waterfalls and rapids) which means that everything appearing above the artificial horizon line must be part of the non-river area in the image.

We then assume that areas below the horizon line that are similar in appearance to the above horizon region are unlikely to be part of the river and that areas below the horizon line that look dissimilar to the above horizon region are likely to be part of the river. By looking for areas below the horizon most dissimilar to those above, candidate river regions are found which are used for training as examples of the river class. Everything appearing above the horizon line is used to generate training examples for the non-river class.

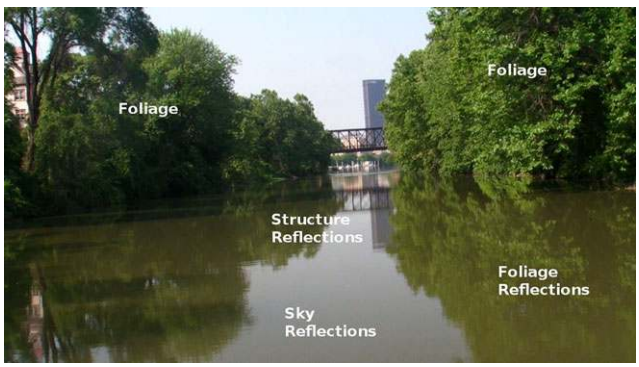


Fig. 4 An example image illustrating the variation of river appearance within a single image

Thus training examples are automatically selected at every frame to train a classifier to segment the image into river and non-river regions.

Feature extraction The input image from the camera is divided into 5×5 pixel patches with a regular grid and a feature descriptor ($X \in \mathbb{R}^n$) is calculated for each patch. This feature descriptor contains information about the patch's appearance and position in the image. The motivation behind computing the features over patches instead of individual pixels is to reduce computational requirements. By varying grid size, segmentation resolution can be traded off against speed. The most commonly used attributes for visual scene analysis tasks are color, texture, perspective information (parallel lines, vanishing points etc.) and image position. Position in an image provides a strong prior, the higher up in an image a region is the less likely it is to be part of the river and depending on the orientation of the camera with respect to the river, regions appearing away from the center towards the sides of the images could be less likely to be part of the river. Straight lines are largely absent in natural scenes so the feature descriptor we use does not include perspective cues.

The color descriptor part of the feature vector contains 6 elements, the color information encoded in the RGB and Lab colorspaces. The texture descriptor used is the response to the Laws' Masks (Laws 1980) over 4 different scales on the 3 channels of the Lab image. We used eight of the nine 3×3 Laws' Masks leaving out the mask that performs low pass filtering, so the resulting texture descriptor had length $4 \times 3 \times 8 = 96$. To compute the response to a filter over a patch we use the L^1 norm of the filter response of all the pixels in the patch. While chosing the texture descriptor we evaluated a number of different options including the SIFT-like DAISY descriptor. We found that for our application, different texture descriptors all performed roughly the same so we chose the Law's Masks because they were the computationally efficient. Details of our feature evaluation experiments can be found in Achar et al. (2011). The position of a patch in the image is described using 3 numbers, the signed perpendicular distance (in pixels) of the patch from the hori-

Fig. 5 Three images taken at the same spot on the Allegheny River at different times. Variations in lighting, water surface ripples and surrounding foliage cause large changes in the river's appearance

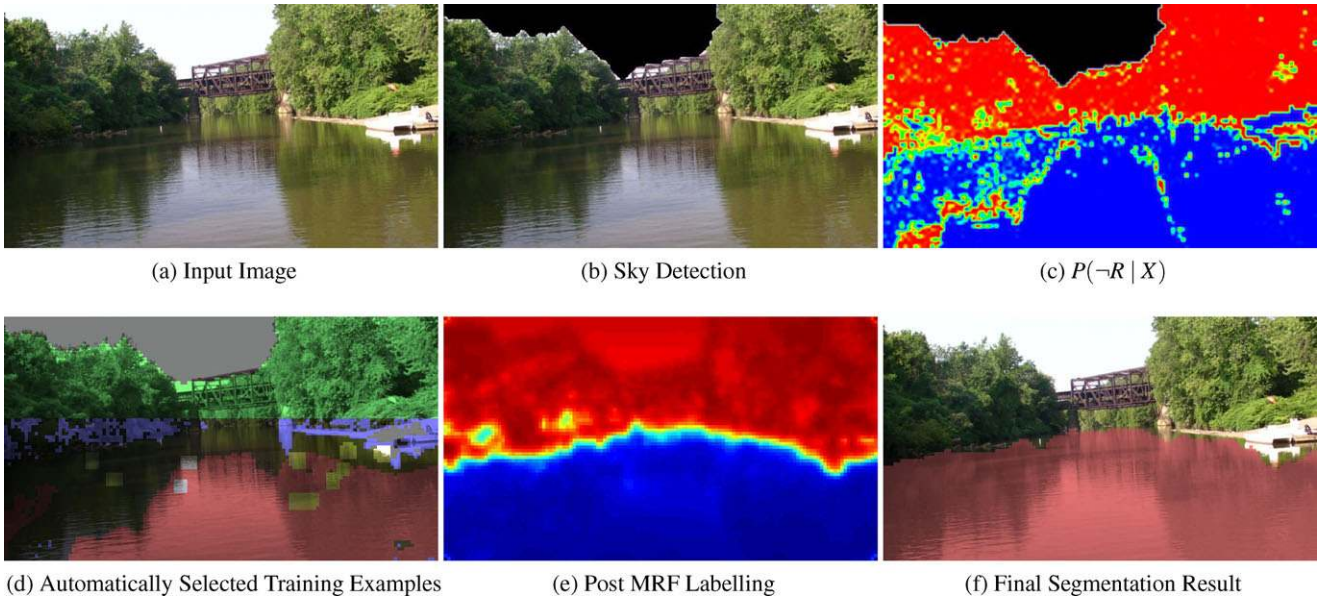


Fig. 6 Steps in the river detection algorithm (Color figure online)

zon line and the distance in pixels of the patch to the left and to the right of the image center.

Sky detection As a preprocessing step, we first segment out the sky and ignore it during all further computations. This is done because sky tends to be washed out and can thus be easily confused with shiny, mirror like parts of the water's surface. Also, removing the sky region saves computational effort in future steps.

A linear support vector machine trained over a set of labeled images is used to detect the sky region in each image. Sky regions in images have a number of characteristics; they appear high up in the image, they tend to be bright or washed out, they have little texture and are generally either blue in hue or unsaturated. The sky is relatively easy to segment out with a globally trained classifier as its appearance is not as variable as that of the river. Figure 6(b) shows an example of sky detection. The remainder of the image needs to be classified as either being part of the river or part of the shore.

The next step is to find the position of the artificial horizon line in the image. For an air vehicle flying at low altitudes, the onboard inertial sensors provide all the information needed to calculate the position of the horizon line in an image from a calibrated camera as the horizon line depends only on the camera orientation with respect to the gravity vector. The horizon line in 3D world space is the line at infinity on the ground plane in front of the camera. This line can be projected into the image using the camera projection equation to find the horizon line in the image.

Let the rotation of the camera be represented by the 3×3 matrix R where R is calculated with respect to a reference frame where the y axis of the camera is aligned with the gravity vector. Also, we use r_i to denote the ith column of R. The horizon line in 3D world space consists of all points on the ground plane (points for which $y = 0$) that are at an infinite distance along the camera's z axis (the forward direction). Thus, the horizon line in world space is the set of (homogenous) points $P_\infty = [x\ 0\ 1\ 0]^T$ where $x \in \mathbb{R}$. These points can be projected into the image as $p_\infty = K[R \mid 0_{3\times 1}]P_\infty$ where p_∞ is the homogenous image coordinate of the horizon point P_∞ and K is the camera's intrisic calibration matrix. It can be shown that the points p_∞ satisfy the line equation $l^T p_\infty = 0$ where

$$l = Kr_1 \times Kr_3 \tag{11}$$

The horizon computation assumes that the region the rotorcraft is operating in is locally flat and so the river lies on the plane perpendicular to the gravity vector. In practice, our algorithm is robust to small inaccuracies in the estimate of the horizon line position and can be calculated simply from the IMU or using the state estimator output.

Appearance modeling The goal of appearance modeling is to assign to each patch a probability of being part of the river (R) or being part of the shore ($\neg R$) on the basis of the feature vector (X) that describes it. This probability is then used to find candidate river patches (patches that have a high probability of belonging to the river).

Appearance modeling depends on being able to create a generative model of appearance from a set of observed samples. A number of methods exist for estimating a joint probability distribution $P(X)$ over a set of d variables $X = \{x_1, x_2, x_3, \ldots, x_d\}$ from a set of l samples $\{X^1, X^2, \ldots, X^l\}$

drawn independently from the underlying distribution. We chose to use a Chow Liu tree (Chow and Liu 1968). A Chow Liu tree approximates the joint probability distribution of a set of random variables (often discrete variables) by factorizing it into a product of second order distributions. It represents a significant improvement over a Naïve Bayes model as it allows for first order dependencies between variables. A Chow Liu tree is optimal in the sense that it is the distribution of its class that minimizes the KL divergence to the real joint probability distribution.

We wish to assign to each patch a probability of being part of the river (R) or being part of the shore ($\neg R$) on the basis of the feature vector (X) that describes it. By Bayes' Rule we have

$$P(\neg R \mid X) = \frac{P(X \mid \neg R)P(\neg R)}{P(X)} \tag{12}$$

Since the labels (R and $\neg R$) are the hidden quantities that are to be determined it is not possible to estimate $P(X \mid \neg R)$ directly. A new boolean variable H is defined which is true for regions below the horizon and false for regions above the horizon. Unlike R, the value of H for each position in the image is known because the equation of the horizon line is known. As mentioned earlier, it has been assumed that the appearance of the shore regions above the horizon is representative of the entire shore region in the image, this implies that $P(X \mid \neg R) \approx P(X \mid \neg H)$ which gives

$$P(\neg R \mid X) \approx \frac{P(X \mid \neg H)P(\neg R)}{P(X)} \tag{13}$$

$$= \frac{P(X \mid \neg H)P(\neg R)}{P(X \mid \neg H)P(\neg H) + P(X \mid H)P(H)} \tag{14}$$

It now remains to form generative appearance models $P(X \mid \neg H)$ and $P(X \mid H)$. These appearance distributions $P(X \mid \neg H)$ and $P(X \mid H)$ are modeled using Chow Liu trees. The feature vector X contains color, texture and image position information and has high dimensionality as it contains many texture filter responses. We create an abridged feature vector $\hat{X} \in \mathbb{R}^d$ for the Chow Liu modelling where the texture descriptor subvector is replaced by its L_2 norm because using the unabridged feature vector would slow down computation. Additionally, $\hat{X}$ does not include image position. Since we are modeling only appearance, including position information in $\hat{X}$ would introduce an undesirable bias. The complete unabridged feature vector is used in the later stage of the algorithm for the linear support vector machine where computation is less of an issue and the position information would not create a bias.

Although a Chow Liu tree can be used in a continuous domain, we discretize the feature vectors $\hat{X} \in \mathbb{R}^d$ for each image patch into a new feature vector $\tilde{X} \in \mathbb{S}^d$ where each feature in $\tilde{X}$ is assigned to 1 of 16 levels ($\mathbb{S} = \{0, 1, 2, \ldots, 15\}$) using equally sized bins that span the range of values taken by that feature. This discretization provides a simple solution to representing non Gaussian, multimodal distributions of features. Each discretized feature vector $\tilde{X}$ contains the R, G, B, L, a and b channels of the associated patch along with the patch's texture.

Two trees are built, one using the feature vectors of patches above the horizon ($\tilde{X} \notin H$) to model $P(X \mid \neg H)$ and another using the feature vectors of below horizon patches ($\tilde{X} \in H$) to model $P(X \mid H)$. The feature occurance and co occurance probabilities needed to build the trees are estimated directly from the available image data by counting, with pseudocounts added to prevent probabilities of 0 and 1. These trees are used in (13) to calculate $P(\neg R \mid X)$ for each patch in an image. An example is shown in Fig. 6(c), the regions considered to have a high probability of being part of the river are in cold colors and regions with a high probability of being part of the shore (high $P(\neg R \mid X)$ are in warm colors.

Finding reflections Reflections are a useful cue for determining the extent of the river. If an object and its reflection can be found it is fairly safe to assume that the reflection was from the water's surface. We look for reflections of salient points (Harris and Stephens 1988) that occur above the horizon line. We treat the water surface as part of a plane of known orientation and distance from the camera. With this information it is easy to use the geometry of reflections to find a small region under the horizon where each salient point's reflection (if present) should occur. We search this region for patches that are similar to a vertically mirrored version of the patch centered around the point being considered. The similarity measure used is normalized cross correlation. Pairs of patches that get similarity scores above a threshold are marked as object-reflection pairs. Patches surrounding the reflection are added to the support vector machine's training data as examples of river patches. Figure 6(d) shows the reflections that were detected in an image (the highlighted areas below the horizon).

Finding novel regions When new appearance models for the river and non-river regions are built from scratch for every new frame, the algorithm is unable to handle novel objects like piers or boats that appear below the horizon because of the assumption that the above horizon part of the image is representative of the entire non-river region. To solve this problem we maintain two appearance models for the river, one built as before from just the current frame, P_{cur} and another built from patches sampled from the river region segmented out in previously images P_{old}. Patches are sampled selectively from previous frames to build P_{old}, with patches from older images being less likely to be used than patches from more recently seen images. The probability of

a patch from the kth frame before the current frame being sampled for P_{old} is λ^k where $\lambda < 1$ is a rate of decay parameter. We define a heuristic measure of novelty $\eta(X)$ for each patch whose value is given by

$$\eta(X) = \log \frac{P_{cur}(X)}{P_{old}(X)} \tag{15}$$

$\eta(X)$ is a measure of how well a patch fits with the appearance model of the river built over previous images. Patches with a high value of $\eta(X)$ are different in appearance from the river and are likely to be part of the bank or an object in the water. We threshold $\eta(X)$ to select high novelty patches below the horizon which are added to the set of negative training examples in the SVM classifier. Figure 6(d) shows the novel patches (marked in purple) that the algorithm found in an image.

Classifier training For each patch, the probability of being part of the shore region $P(\neg R \mid X)$ is calculated using (13). The patches that are least likely to be on the shore ($P(\neg R \mid X) < \theta$) are used as candidate river patches. Using a low value for θ reduces the chance that shore patches are accidentally used as candidate river patches, but if θ is set too low then the selected region will be too small to provide a good representation of the river region. In our experiments, θ was set to 0.01. Patches where reflections were found are also added to the set of river training examples. The training examples used for the shore class are the patches that are considered to be novel and all the non-sky patches that appear above the horizon line. These training examples are used to train a two class (river/shore) linear support vector machine.

The SVM uses the unabridged, continuous valued feature vectors X, including image position and the complete texture descriptor. The support vector machine is trained using an online stochastic gradient descent method. Since we are learning a new classifier for each new frame, while the appearance of the river is likely to remain fairly constant over short periods of time, the SVM training updates the SVM model learnt on the previous frame. This reduces the number of gradient descent iterations needed to train the classifier for each new frame. Figure 6(d) shows the river and shore training examples picked by the algorithm in an image. Candidate river patches (patches for which $P(\neg R \mid X)$ was below the threshold θ) are in red, the highlighted regions are places where reflections were detected. The green patches are the patches selected as shore training examples because they lie above the horizon and the patches in purple were selected as shore training examples because they are sufficiently novel in appearance.

Using the output of the Chow Liu tree directly to classify the scene (say by thresholding $P(\neg R \mid X)$ at 0.5) does not work well. There are many image regions that would be mislabelled by the Chow Liu tree, the SVM introduces a height prior and has access to a more discriminative (but longer) feature descriptor which enables it to classify ambiguous regions better.

Detecting water The learnt SVM is a vector of weights $w \in \mathbb{R}$. This SVM is used to assign a label y^i that could either be 'river' or 'shore' to each patch x^i. One way to classify patches would be to consider each patch in isolation and assign it to the river class if its dot product with the weight vector is positive and assign it to the non river class otherwise.

$$y^i = \begin{cases} +1 & \text{if } w \cdot x^i > 0 \\ -1 & \text{if } w \cdot x^i \leq 0 \end{cases} \tag{16}$$

This approach disregards the spatial distribution of labels. Neighboring patches are likely to share the same label, this fact can not be exploited if a hard assignment of labels is made to patches individually. Instead, a soft assignment of labels is made ($y^i \in (0, +1)$)

$$y^i = 0.5 + \frac{1}{\pi} \arctan(kw \cdot x) \tag{17}$$

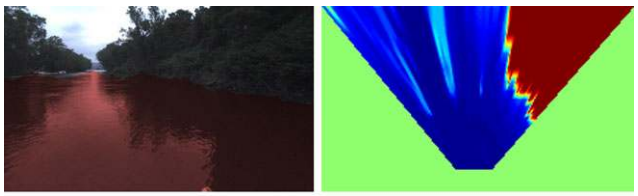
The effect of the soft assignment is that patches closer to the SVM decision boundary are labeled with less confidence than points with larger margins. These label assignments are used to construct a Markov Random Field (Li 1994). The MRF structure chosen is a regular lattice with each patch forming a node connected to its 4 neighbors. The soft assignments y^i made by the support vector machine are used as measurements and a Gauss-Markov energy model is applied to find the most likely continuous label l^i for each patch. This goal is equivalent to finding the labeling L that minimizes the energy $U(L)$

$$U(L) = \alpha \sum_{(i,j)\in\mathcal{E}} \left(l^i - l^j\right)^2 + \sum_{i\in\mathcal{V}} \left(y^i - l^i\right)^2 \tag{18}$$

where a node pair (i, j) is in $\mathcal{E}$ if they are neighbors, $\mathcal{V}$ is the set of all nodes and α is a parameter that controls the amount of spatial smoothing performed by the MRF. This energy function $U(L)$ can be minimized efficiently in one step as it is equivalent to a system of linear equations. An example of the values assigned by the MRF is in Fig. 6(e). Values of l closer to one (river areas) are in shades of blue and patches with low l values (shore region) are in red. Figure 6(f) shows the final output of the segmentation algorithm formed by thresholding Fig. 6(e) at zero.

2.2.3 2D river mapping

We build a 2D map of the river by registering the outputs of the self-supervised river detector into a global frame using vehicle pose estimates from the state filter. The output



(a) (b)

Fig. 7 Local 2D River Mapping: (**a**) Image with the detected extent of the river (thresholded at $P_{river} = 0.5$) overlaid in *red*. (**b**) The same detected extent of the river projected into a top down view. *Blue* regions are part of the river and *red* regions are the shore. The surrounding *green area* is unobserved in this image as it lies outside the viewing frustum (Color figure online)

of the river detector is used as an estimate of the probability of each point in the image being part of the river. Since the river surface is assumed to lie on the ground plane, the probabilistic labeling from the vehicle's viewpoint can be projected into a top down view using knowledge of the vehicle orientation and height provided by the state filter. Each of these top down views is a local, vehicle centered map of a small section of the river (Fig 7(b)). These local maps are registered globally with position information from the state filter and fused into a global map of the river using occupancy grid mapping, see Fig. 19(b).

2.3 Obstacle mapping

We keep a three dimensional representation of the world to allow for obstacle avoidance and motion planing. The 3D evidence grid expresses the belief that a volume of space is occupied and represents relevant obstacle occupancy (Martin and Moravec 1996). An evidence grid can be updated in realtime and can incorporate sensor uncertainty. It is arranged as a regular grid of voxels that store the log-likelihood ratio of the voxel being occupied. The raw range sensor data from the ladar is transformed to an occupancy probability and the result is mapped into an evidence grid using position and orientation information from the state filter. The method used to update the evidence grid can be found in Scherer et al. (2008).

2.3.1 Distance transform

In addition to the evidence grid, a key element required for motion planning and obstacle avoidance on the rotorcraft is a cost map based on the 3D evidence grid information. The dominant factor for the cost of occupying a location is the distance to the closest obstacle. For this reason we use a distance transform based on the evidence grid that stores the distance to the closest obstacle at each grid cell. An efficient non-incremental linear time algorithm to calculate the distance transform was proposed by Meijster et al. (2000). While efficient if the environment does not change, it has been shown in Scherer et al. (2009) that repeatedly recomputing the result for changing obstacles takes too long to be useful for navigation on a large grid.

A simple incremental approach is to update the grid with a mask of the distances to the obstacles. Every time an obstacle is added, a convolution of the surrounding area is performed to check if any of the distances are larger than the distance in the mask. In the case of obstacle removal, cells in the mask are set to infinity and a region of two times the size of the mask is scanned for existing obstacles that are re-added. While better than recomputing the entire distance transform when new information is received, this algorithm has to check many cells that are already correct especially if multiple obstacles close to each other are modified.

An efficient incremental algorithm was proposed in Scherer et al. (2009) based on the propagation of wavefronts outwards from added or removed obstacles to modify only cells that need updating. This algorithm was improved in Lau et al. (2010) to reduce duplicated work at the wavefronts and reduce the storage required at each cell. Our algorithm is a modified version of the Lau algorithm, which fixes a bug and further improves the algorithm by removing duplicated cell processing at the wavefronts and reducing the storage required at each cell. The bug (lines 17–20 of Algorithm 1 in Lau et al. 2010) causes some cells to be given the wrong priority in the queue because they have their distance reset to maximum before being added to the queue with the maximum distance, rather than the existing distance, used as the priority.

The input to the distance transform update algorithm is a list of grid cells that have been changed by either adding or removing an obstacle. Distances are stored as square Euclidean distances to allow for the use of compact fixed point numbers and to remove an expensive square root operation when calculating the distance. The expansion is limited to a maximum distance of d^2_{max} to limit the number of cells that must be modified when obstacles are added or removed. Cells further than the maximum distance from an obstacle are not modified, however all cells are initialized with a distance of d^2_{max}. The method is based on the propagation of wavefronts with the cells in the wavefront kept in a priority queue. Each cell s is a structure that contains the distance to the closest obstacle, $dist_s$, an in-queue flag indicating whether it is on the queue, inQ_s, and a reference to the position of the closest obstacle, $obst_s$. The in-queue flag is always set immediately prior to adding the cell to the queue and cleared when the cell is dequeued. The obstacle reference can be a pointer to the cell which is the closest obstacle, however a more compact method is described below. The cells are initialized to have the maximum distance, d^2_{max}, and a null obstacle reference. Figure 8 contains all the functions used by the Euclidean distance transform algorithm, with the priority queue or Open list being referred to as O in the algorithm.

```
INITIALIZE()
1  O ← ∅
2  foreach cell s
3   dist_s ← d²_max
4   obst_s ← ∅
5   inQ_s ← false

UPDATEDISTANCES(O)
1  while O ≠ ∅
2   s ← POP(O)
3   if dist_s = d²_max
4    RAISE(s)
5   else
6    LOWER(S)

SETOBSTACLE(s)
1  if dist_s ≠ 0
2   dist_s ← 0
3   obst_s ← s
4   inQ_s ← true
5   INSERT(O, s, 0)

REMOVEOBSTACLE(s)
1  if dist_s = 0
2   dist_s ← d²_max
3   obst_s ← ∅
4   inQ_s ← true
5   INSERT(O, s, 0)

ISVALID(s)
1  if dist_s = 0 or dist_obst_s = 0
2   return true
3  else
4   return false

LOWER(s)
1  foreach n ∈ Adj(s)
2   d ← DISTANCE(n, s)
3   if dist_n > d
4    dist_n ← d
5    obst_n ← obst_s
6    if ¬inQ_n
7     inQ_n ← true
8     INSERT(O, n, d²_max + d)
9   inQ_s ← false

DISTANCE(n, s)
1  v ← pos_n − pos_obst_s
2  return v · v

RAISE(s)
1  foreach n ∈ Adj(s)
2   if ¬inQ_n
3    if ISVALID(n)
4     inQ_n ← true
5     INSERT(O, n, d²_max + dist_n)
6    if dist_n ≠ d²_max
7     obst_n ← ∅
8     inQ_n ← true
9     INSERT(O, n, dist_n)
10    dist_n ← d²_max
11  inQ_s ← false
```

Fig. 8 Distance transform algorithm

There are two types of processes that can be called when a cell is dequeued, LOWER or RAISE. The LOWER process propagates out new lower distances from a new obstacle or valid cell. A valid cell is one that references an obstacle that has not been removed. The RAISE process propagates out cleared distances when an obstacle is removed. The differences between our algorithm and the Lau algorithm are the use of the in-queue flag to prevent duplication of the wavefront during raise and lower propagations and the use of wave sequencing. When a valid cell is added to the queue, it is given a key of $d^2 + d^2_{max}$, where d^2 is the cell's square distance at the time of adding. When a non-valid cell is added to the queue, it is given a key equal to the square distance it had before it was cleared. This means that keys of cells added to the queue are always proportional to the distance from the origin of the current wave ensuring a uniformly expanding wavefront. The extra d^2_{max} term for valid cells ensures they always have a larger key than non-valid cells resulting in wave sequencing, i.e. a full raise wave is completed before continuing the propagation of a lower wave. This has no impact when obstacles are being added, however when obstacles are removed, a raise wave is propagated outwards until all the nearby invalid cells have been cleared and all nearby valid cells have been found. Only then is a lower wave propagated from the valid cells to give the cleared cells new distances. The Lau algorithm adds both cleared and valid cells with a key equal to the distance from the relevant wave's origin such that the two waves, raise and lower, run simultaneously. In many situations this leads to cells being added to the queue multiple times creating extra work. Figure 9 shows a raise wavefront completing before a subsequent lower wave is propagated.

As a consequence of the sequenced waves, the order in which obstacles are added or removed makes a difference. If all the updates, adds and removes, are processed together the removed obstacles will be processed first because of the higher priority given to raise waves. In many situations this results in some cells being needlessly cleared and then later given a new distance by a new obstacle in the same update. By adding all the new obstacles and running a full update before removing any obstacles, many of those cells can be preemptively given a new distance without being modified twice. When the obstacles are removed, the raise wave will

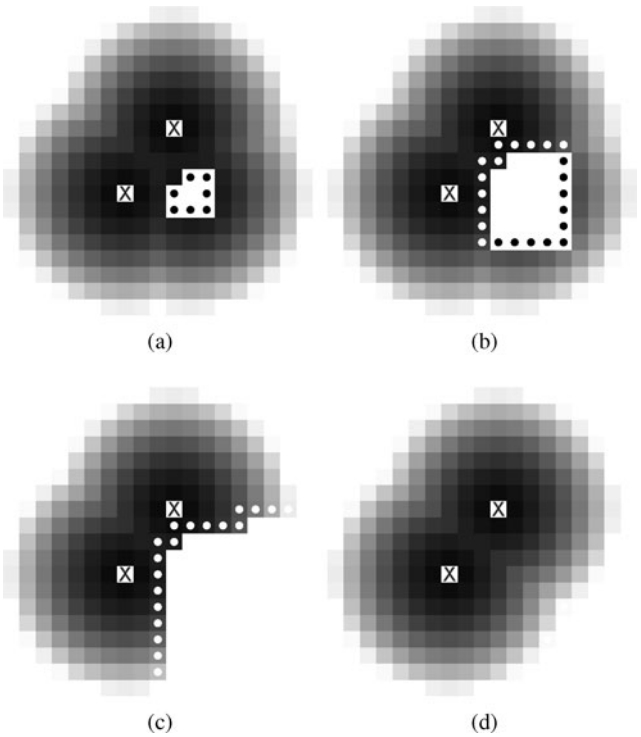


Fig. 9 Obstacle Removal: A raise wavefront propagates outwards clearing invalid cells and finding a surface of valid cells due to nearby obstacles. When all the invalid cells have been cleared a lower wavefront propagates out from the surface of valid cells and terminates when the maximum distance is reached

not have to propagate as far because it will terminate at the now valid cells created by the new obstacles.

Each cell must store a reference to its closest obstacle; the simplest way to do this is to store a pointer to the obstacle cell. Pointers can be large especially on 64-bit systems which when multiplied by the number of cells in a large grid can greatly increase memory requirements. The storage required can be reduced by replacing both the pointer and distance stored in each cell with an index into a pre-computed array of possible obstacle offsets and distances. This method leverages the fact that the distance to the closest obstacle will be at most d_{max} and is a trade off between the reduced memory used by each cell and the memory used to store the pre-computed look-up table. In most cases the savings at each cell, which are multiplied across the entire grid, will far outweigh the space required to store the look-up table. For example a $1000 \times 1000 \times 200$ grid would require 2000 MB of memory if a pointer is used, and only 800 MB if an index is used. The look-up table for a fairly large, 50 cell, distance expansion only requires an additional 5 MB.

2.3.2 Map scrolling

The area traversed by an aerial vehicle can be very large, in particular the rivers we explore with the rotorcraft can be many kilometers long. Covering the entire area with a single grid would require a prohibitively large number of cells. The map scrolling algorithm moves the distance transform as the vehicle moves such that obstacle information is always defined in the region around the vehicle but the active memory required remains constant. The terminology used during a scroll event can be seen in Fig. 10.

When a scroll of the distance map is triggered, a new volume in the direction of travel must be represented. This is done by clearing the vacated volume behind the new grid position and reallocating its memory. To minimize the work required, cells in the existing volume that remain unchanged are not copied or moved. As a result the physical position of cells does not map directly to their position in memory. A memory mapping function takes the position in the grid of a cell (x, y, z), and how much the grid has scrolled along each dimension (sx, sy, sz). The output is an index indicating where the cell is stored in memory. The scroll parameters keep track of the current wrap around of the grid rather than store the absolute offset from the starting position. This ensures that the scroll parameters stay small, taking values from 0 to the grid's size in each dimension, without limiting how far the grid can scroll. The memory mapping function first undoes the wrap around of the grid in each dimension giving the cell's memory space coordinates, i.e. the 3D position the cell would have in an unscrolled grid. Equation (19) shows how a cell's memory space x-coordinate is found using the x-component of its position in the grid, x, the current

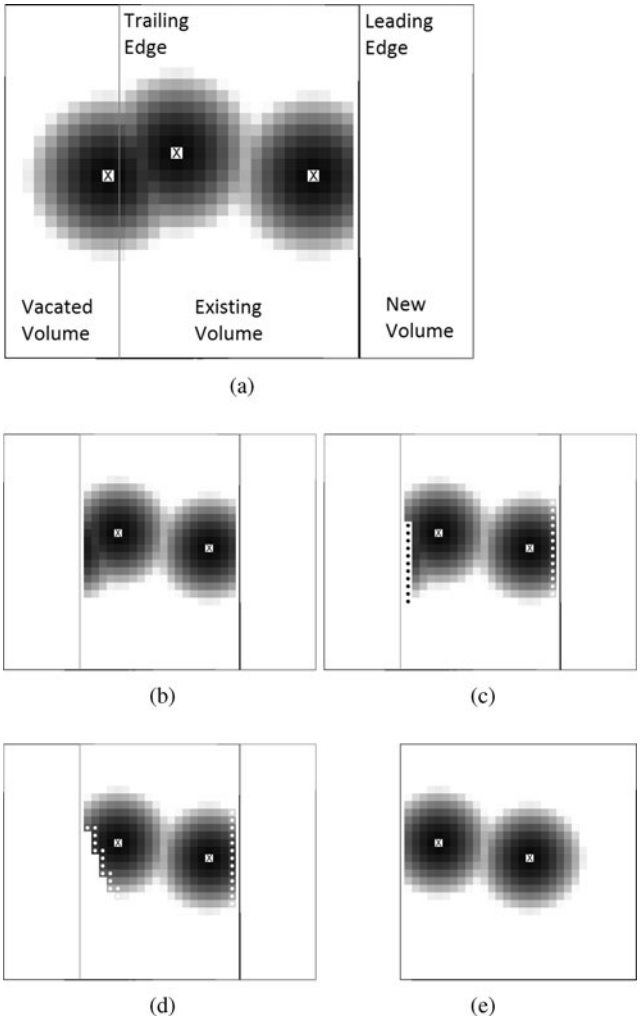


Fig. 10 Scroll Event: The grid has 3 obstacles. It is scrolled to the right creating errors at the trailing and leading edges. A raise wave is triggered at invalid cells on the trailing edge and a lower wave is triggered at valid cells on the leading edge. The wavefronts propagate out fixing the errors

wraparound in the x-direction, s_x and the size of the grid along the x-direction, d_x. An equivalent equation is used for the y and z components. Equation (20) shows how the memory index is found from the memory space coordinates of a cell and the dimensions of the grid.

$$m_x = x + s_x + \left\lfloor \frac{x + s_x}{d_x} \right\rfloor d_x \tag{19}$$

$$i = m_x d_y d_z + m_y d_z + m_z \tag{20}$$

As a result of scrolling, cells can become inconsistent at the trailing and leading edges. Errors occur at the trailing edge if there is an obstacle in the vacated volume and cells continue to point to it. When the vacated volume is cleared and the memory reallocated these cells will end up pointing to an unrelated cell on the other side of the map causing serious problems. These inconsistent cells need to be cleared

or changed to point to a different obstacle that is inside the new grid boundaries. Errors occur at the leading edge when an obstacle inside the existing volume is near the leading edge and there are newly added cells on the other side of the edge that should have a lower distance. Both these errors are present in Fig. 10. After clearing the cells in the vacated volume and reassigning their memory, the errors are corrected by clearing and triggering a raise wave at any cells on the trailing edge that are not valid and a lower wave at any cells on the leading edge that are valid. The waves are triggered by adding the cells to the priority queue and running an update as described in Sect. 2.3.1. These waves will proceed as normal, clearing any non-valid cells then propagating out new lower distances to cells that need them.

As the vehicle moves, scroll events need to be triggered so that the distance map can keep up. When a scroll is triggered the grid is moved so that the vehicle is in the center of the grid. Deciding when to scroll the map is dependent on a number of considerations. The first is that all new sensor information should always be captured. To fulfill this requirement the grid must scroll before the vehicle gets closer than its maximum sensor range to an edge of the grid. The second consideration is that the amortized work required (the total time spent performing scrolling operations) should be minimized while the latency should be maximized. Latency is dependent on the per event work required, and is important because the distance transform can not be updated with new sensor information while it is being scrolled. The work performed during a scroll event primarily involves clearing the cells in the vacated volume and correcting the inconsistencies at the trailing and leading edges. Frequent scrolling means that the edges have to be corrected more often but there are less cells to be cleared at each event. The last consideration is which dimensions the grid should scroll along. The size of the trailing and leading edges, surfaces in 3D, is greatly increased if the grid is scrolled in multiple directions. This creates extra work to ensure the distances are consistent along these edges. For this reason the scroll trigger is checked separately along each dimension and the grid is only centered around the vehicle in the dimension of the trigger. This reduces the per scroll work load because each dimension is usually scrolled separately, although they could occur simultaneously, and only when a scroll is actually required along that dimension. For example our aerial vehicle does not change its altitude enough to require the grid to scroll in the vertical direction. Checking the dimensions separately stops small unnecessary grid movements in the vertical direction because of scrolls triggered at the horizontal edges.

$$b_x < \alpha_h r_h \tag{21}$$

The trigger for the x-direction is shown in Eq. (21) where b_x is the distance to the closest boundary along the x-axis, r_h is the maximum horizontal sensor range and α_h is a tunable parameter. The horizontal parameters are shared by the triggers along the x and y-axis, the z-axis trigger uses separate parameters. Changing the value of α affects how often the grid is scrolled. The value of α can vary from 1, which means the grid will not scroll until the maximum sensor range is at the boundary, to half the grid size along the relevant axis, which means the grid will scroll every time the vehicle moves. Setting α is a trade off between total scrolling work done over time and work required per scroll event.

2.4 3D Ladar scanner

The rotorcraft must be able to operate in the space between the water's surface and the tree canopy. In this cluttered space, a reliable short range perception system is necessary for obstacle avoidance and mapping. To measure 3D information about the environment, we use an off-axis rotating 2D laser line scanner. As seen in Fig. 11(a), a Hokuyo UTM-30LX is mounted with the scan plane tilted at 45° with respect to a sweep motor axis.

Other ladar mounting and actuation configurations such as nodding, spinning on-axis, or roundly swinging (Matsumoto and Yuta 2010), did not provide the same scan density or sensing field of view. Our configuration has the advantage of equal detection of horizontal and vertical obstacles with a full 360° field of view, which matches the omnidirectional mobility of the rotorcraft. Figure 11 shows the hatched scan pattern sensed by the spinning scanner. This scan pattern detects thin horizontal and vertical obstacles equally well as opposed to a nodding laser, which has difficulty detecting thin horizontal obstacles or a vertically-mounted on-axis spinning laser, which has difficulty detecting thin vertical obstacles. In a natural river environment, thin horizontal and vertical tree branches should be expected and can be reliably sensed with our configuration.

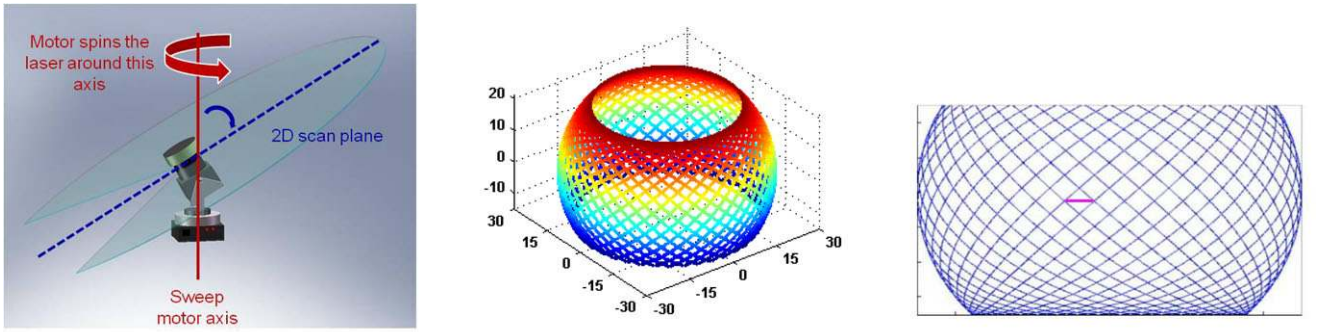
2.4.1 Registration

To register ranged data points P_r from the laser into a non-spinning base frame P_b, we apply a series of 4×4 homogeneous coordinate transformations to each ranged point.

$$P_b = T_w T_m T_s P_r \tag{22}$$

P_r: Ranged point in the coordinate frame of the ladar optical receiver.

T_s: Time dependent transform around a single axis of rotation, which expresses the orientation of the rotating mirror within the laser scanner.



(a) Off-axis spinning scanner (b) Isometric View (c) Side View

Fig. 11 Laser scan pattern when the scan plane is tilted at 45° to the sweep motor axis. The *horizontal magenta line* segment in (**c**) shows the widest unseen visual angle at the hatched diagonals (Color figure online)

T_m: Full 6 DOF transformation, which remains constant and represents the mounting configuration of the scanner on the sweep axis mount. This transform takes into account the tilt of the laser scan plane, the translation between sweep axis of rotation and the laser receiver, and any mechanical misalignment and can be found using a nonlinear optimization similar to Fairfield (2009) or Harrison and Newman (2008).

T_w: Time dependent transform around a single axis of rotation to account for the rotation of the laser module around the sweep motor axis. This angle is calculated by assuming a constant sweep motor speed and interpolating the angle given by the motor encoder at the beginning and end of each line scan.

P_b: Position of the 3D point in the non-spinning base frame of the system. The origin of the base frame contains the sweep motor axis.

2.4.2 *Obstacle detection confidence*

We developed a model of the laser's obstacle detection performance to provide a confidence measure for the detection of variously sized obstacles at different vehicle velocities. Obstacles are modeled as thin, floating objects to account for a worst case senario such as a hanging vine or thin branch. The magenta line in Fig. 11(c) represents a horizontal obstacle. In our model, obstacles are defined only by their length, which is valid for the small tree branches that will pose significant danger to the rotorcraft above the river. We make two important assumptions: (1) if a scan line falls on the object, the object will be detected (i.e., the scan is dense) and (2) the rotation of the hatched scan pattern varies randomly from sweep to sweep. This random rotation variation could come from external perturbations in the vehicle's yaw or could be manually introduced by adding an angular offset to the motor for each sweep. The unseen angle between scans after one sweep is:

$$\theta_u = \sqrt{2}\frac{2\pi\lambda_w}{\lambda_s} \tag{23}$$

where λ_w is the sweep rate of the motor and λ_s is the line scan rate of the ladar. The factor of $\sqrt{2}$ expresses the greatest unseen angle, which is at the diagonal of the square in the hatched scan pattern. The visual angle θ_o of an obstacle with length l at a distance r is expressed as:

$$\theta_o = 2\arctan\left(\frac{l}{2r}\right) \tag{24}$$

The probability of detection after one sweep is simply the ratio of the obstacle's visual angle to the unseen angle, with a maximum probability of one. Here we consider a static obstacle directly in the path of the vehicle moving at velocity v from an initial starting distance D_0. The distance r and thus the visual angle will be reduced for each new sweep. The probability of an observation after k sweeps is

$$p_k(obs) = 1 - \left(1 - \min\left(\frac{\theta_o(r)}{\theta_u}, 1\right)\right)^k \tag{25}$$

$$r = D_0 - \frac{(i-1)v}{\lambda_w} \tag{26}$$

where the object's visual angle θ_o depends on r.

For a desired observation confidence, safe maximum velocities are found which satisfy the following equation with a fixed C-Space expansion L_C, reaction time t_a, and maximum vehicle acceleration a:

$$D_0 - L_C \geq \left(\frac{k}{\lambda_w} + t_a\right)v + \frac{v^2}{a} \tag{27}$$

The inequality states that the rotorcraft must observe the obstacle, react, and come to a stop in a distance equal to or less than the maximum laser range minus the C-Space expansion. Figure 12 plots safe rotorcraft velocities versus

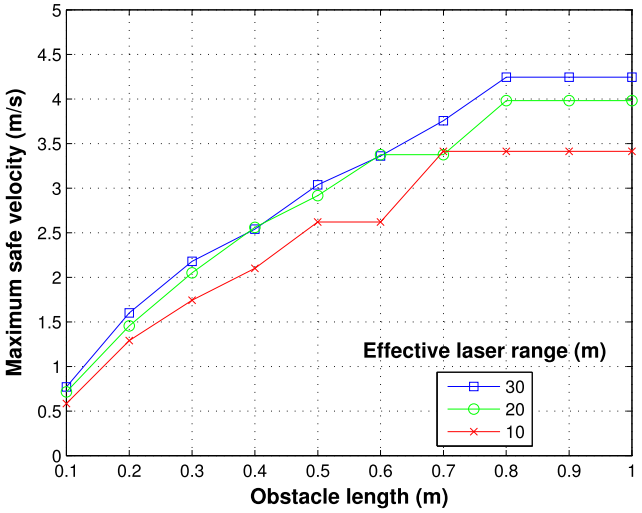


Fig. 12 Worst case analysis of maximum safe velocity for varying obstacle lengths. Maximum velocities increase in discrete steps since we require that a full sweep must be completed before an obstacle can be detected (Color figure online)

obstacle length from 0.1 m to 1 m for three different maximum laser ranges. We have found that the effective range of the Hokuyo laser scanner can be as low as 10 m or 15 m in bright, outdoor conditions. In this figure, obstacle detection confidence set to 95 %, C-Space expansion is 0.5 m, reaction time is 0.1 s, maximum acceleration is 5 m/s^2, scan rate is 40 Hz and sweep rate is 1 Hz. A velocity is considered safe if the rotorcraft can detect the obstacle and come to a stop before violating the C-Space expansion. Maximum velocities increase in discrete steps since we require that a full sweep be completed before an obstacle can be detected.

3 Results

We evaluate our results based on the ability of our sensing and perception algorithms to enable river navigation while avoiding obstacles and generating an accurate map of the river's course, width, and canopy height. Section 3.1 describes the sensor suite, datasets, and computing hardware and Sect. 3.2 details results from the state estimation system. Visual river detection in Sect. 3.3 shows mapping results for the course and width of the river as well as a direction of travel for the vehicle. We finish with ladar based evidence grid mapping in Sect. 3.4 and show final integrated mapping results for canopy height and shoreline structure in Sect. 3.5.

Finally, Sect. 3.6 reveals various insights into our system and analyzes our criteria for success.

3.1 Experimental setup

Sensor data to validate our system was collected on a stretch of the Allegheny river, Pennsylvania, USA, in a narrow passage created by an elongated island named Washington's Landing. This river passage is approximately 30 meters wide, with several car and railway bridges crossing overhead, and on either side of the passage there are tall trees and a steep embankment—all acting to create intermittent GPS coverage. To evaluate the river detection algorithm under varying conditions, we conducted additional testing across four datasets captured at different times of the day and year on two different rivers in southwest Pennsylvania; the Allegheny river and the Youghiogheny river.

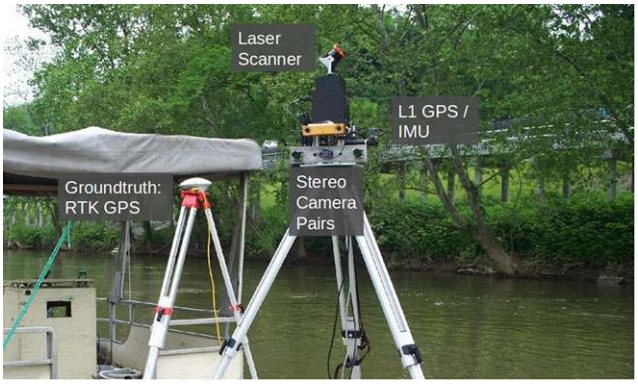


Fig. 13 Surrogate data collection system at the test site on the Allegheny river. Here we see the GPS, IMU, ladar and stereo camera pairs (one temporary COTS unit and the other a lightweight in–house device). There is also a high-end, heavy GPS unit that we use to groundtruth the experiment

3.1.1 *Sensing hardware*

For the complete system test, we deployed our sensor suite on a surrogate data collection platform, seen in Fig. 13. We have a flight ready version using the same sensors and embedded computing, as describe below, but at present we have not flown on a river because of the inherent risks involved with autonomous flight over a body of water. We have conducted flight tests on land as shown in Haines (2011).

The surrogate platform is substantially over the 1 kg payload of the rotorcraft. However, the excess weight comes from test computing and test mounts—whereas individually, the sensors, lightweight rotorcraft mounts, and embedded computing are within the total weight budget. Also seen in Fig. 13 is the high-accuracy L1/L2 GPS unit used for position information ground truth.

3.1.2 *Embeded computing*

The entire system runs at real-time on-board our rotorcraft, using an embedded Intel Core 2 Duo 1.86 GHz processor in the COM Express form factor with a custom designed carrier board based on the open-source PIXHAWK pxCOMex (Meier et al. 2011). To reduce the computational load on

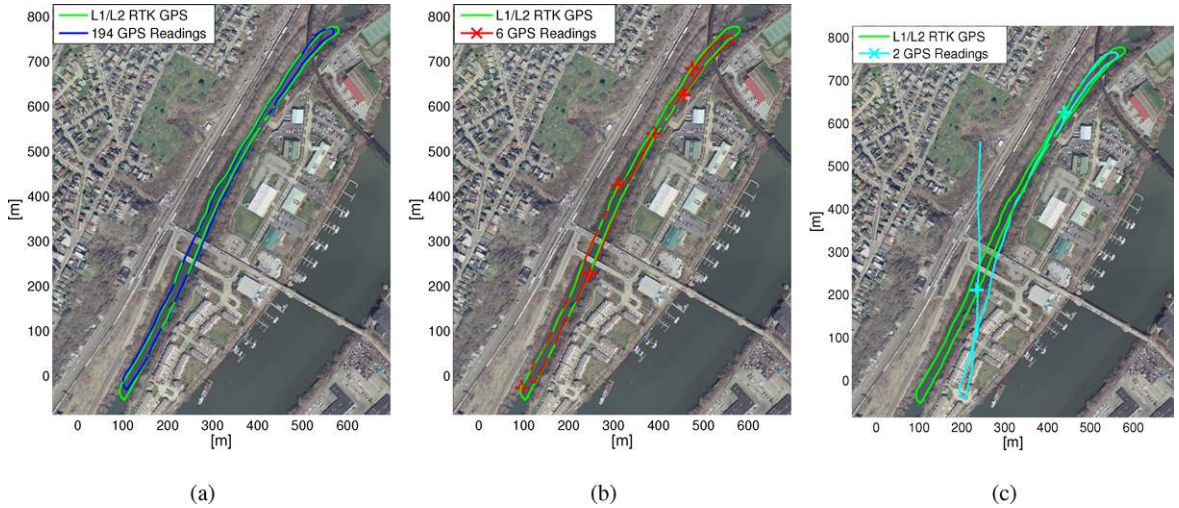


Fig. 14 Estimated path demonstrating the state estimation performance for different levels of GPS availability, overlaid onto an aerial map and a highly accurate L1/L2 GPS reference path. The path depicted in (**a**) was generated using 194 L1 GPS measurements. The estimations shown in (**b**) and (**c**) incorporated 6 and 2 global position measurements, corresponding to a rate of about a single GPS reading every 90 seconds and 270 seconds respectively (Color figure online)

the main embedded processor, we compute feature detection and matching for visual odometry using a small array of digital signal processors connected via USB (Haines 2011).

3.2 Vehicle state estimation

Figure 14 depicts the estimated vehicle positions for different levels of GPS availability and is overlaid onto an aerial map of the river. For ground truth, we acquired position information with a high accuracy L1/L2 GPS post-processed with RTKLIB (Takasu et al. 2007), which is shown in green. The sequence spans about 2 km and roughly 10,000 video frames. Figure 14(a) depicts results for a state estimation that incorporated measurements of a consumer-grade GPS receiver at a rate of roughly 0.3 Hz. As seen in Fig. 14(a) the estimated vehicle path closely resemble the ground truth path, despite employing a significantly lighter and less accurate GPS receiver. Figure 14(b) displays the same dataset, but GPS availability was synthetically degraded to 6 measurements over the entire course of the experiment. To simulate extensive GPS dropouts, we evenly subdivided the traverse into six segments and randomly sampled a single GPS reading from each segment. These GPS measurements are marked as red crosses in the figure. Even though GPS readings were incorporated only six times over 2 km, the resulting path is comparable to the results demonstrated with 194 GPS measurements in Fig. 14(a). This similar performance demonstrates that the state estimation system is able deal with seriously limited GPS coverage. Figure 14(c) depicts an estimation for the minimal case for which global heading is observable. The state estimation proves capable of inferring global position and heading up to reasonable accuracy, although the divergence of the estimated path from ground truth is more apparent. In the experiments shown in Fig. 14(b) and Fig. 14(c), the graph reduction scheme simplified the graph of initially 10,000 stereo pairs to roughly 170 active state nodes and solved the resulting reduced system in about 300 ms on our computer hardware.

3.3 River detection

We tested the river detection algorithm on four datasets captured on different rivers under varying conditions. Three of the datasets were collected on the Allegheny River near Herr's Island in Pittsburgh, PA. The first (Allegheny Day) was collected on a summer afternoon, the second (Allegheny Dusk) was captured near dusk, and the third (Allegheny Fall) was collected in autumn around noon. The forth dataset was collected at Perryopolis, PA on the Youghiogheny River. The Youghiogheny dataset was collected around noon during the summer. Each dataset contains 120 to 150 images captured from a small motorboat.

All the images in these datasets were manually segmented into three labels (river, shore and sky) to provide ground truth for performance evaluation. Since our algorithm generates only two labels and differentiating between the shore region and the sky is not our goal, we treat them as a single class for the purpose of evaluating performance. The performance metric we used is the mean error rate, which is the percentage of pixels misclassified by the algorithm compared to the ground truth labeling. This includes both river pixels misclassified as shore and shore pixels misclassified as river.

We compare our self supervised approach to a fully supervised alternative.

Table 1 Supervised Segmentation Performance: Mean Error Rate (percent). Each row corresponds to a test dataset and each column corresponds to a training dataset. A supervised approach to river segmentation performs well when trained and tested on the same environment, but often suffers significant performance degradation when run on an environment different from the one it was trained on

Dataset	Allegheny day	Allegheny dusk	Allegheny fall	Yough	All but self	All
Allegheny day	2.54	5.60	4.97	4.19	4.19	3.48
Allegheny dusk	7.95	2.32	12.51	8.48	7.93	3.71
Allegheny fall	10.04	5.42	2.64	7.51	10.33	4.14
Yough	4.12	4.36	6.24	2.59	4.62	3.44

3.3.1 Fully supervised river detection

As a point of comparison against our self supervised classifier, we evaluate the ability of a fully supervised classifier to generalize to images from previously unseen environments. The fully supervised classifier evaluated was a linear SVM with identical input features to the self supervised algorithm. From each dataset, a third of the images were picked at random as training examples and the remaining images were used as test data. Each dataset was then classified using six different supervised classifiers. The first four were trained on the training data from each of the four datasets. The fifth classifier (All but Self) was trained using the training images from the datasets other than the test dataset and the sixth classifier (All) was trained on all the training data across datasets. The error rates for these classifiers were averaged over 10 trials and are tabulated in Table 1. The supervised approach works well when used on datasets seen during training (the diagonal entries and the 'all' column in Table 1) but performance degrades when run on new datasets that were not seen while training (off diagonal entries and the 'All but Self' column in the table). This suggests that a supervised approach does not generalize well to new environments. Similar lack of generalizability to new datasets has been observed in many vision tasks as reported by Torralba and Efros (2011). It should be noted that when one third of the images from a dataset are used for training, every test image from the same dataset has a number of very similar images in the training data. This means that the supervised classifier does not have to generalize much to perform well when the training data contains images from the same dataset as the test data.

We also investigated the effect of changing the fraction of data used for training on the performance of the fully supervised algorithm. We repeated the same experiment with varying splits between the amount of data used for training and testing. The change in classifier error rates when using training data from the Allegheny Day sequence are shown in Fig. 15. When as little as one tenth of the images are used for training, the classifier performance saturates and adding new training data has little effect on performance. These results suggest that in the feature space being used, it is difficult to find a single, static linear decision boundary that can separate river and non-river regions well across datasets and generalize well to unseen data. A more complex, non linear decision boundary learnt using a more complex algorithm (such as a kernelized SVM) may be able to deliniate the classes adequately but would be difficult to use in real time. This motivated our decision to learn dynamic, linear decision boundary that changes over time to adapt to variations in appearance of the river and its surroundings.

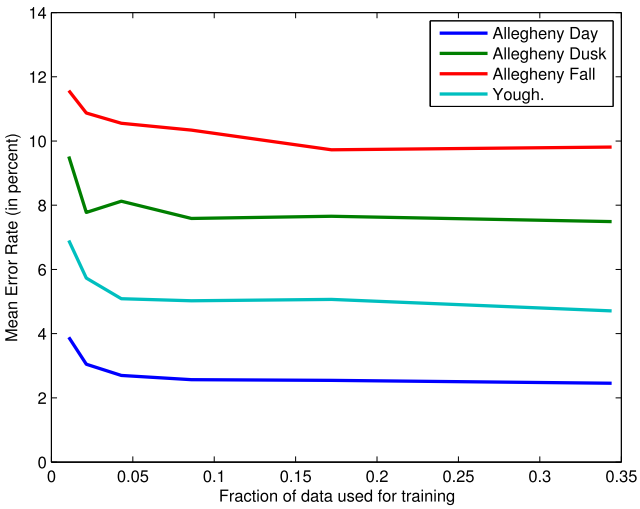


Fig. 15 Effect of Increasing Training Data: Varying amounts of data from the Allegheny Day dataset were used to train a supervised river segmentation. As the fraction of training data is increased, the performance of the fully supervised algorithm saturates very quickly on all the test datasets (Color figure online)

3.3.2 Self supervised river detection

The performance of the self supervised algorithm described in Sect. 2.2 was evaluated on the four datasets. Error rates are shown in Table 2. On all four datasets the self supervised algorithm outperformed a supervised classifier that did not see images from the test sequence during training. Even when the supervised classifier was trained on a subset of images from the dataset it was tested on, it did not perform significantly better than the self supervised algorithm. This is significant considering how the self supervised algorithm has no off-line training phase and does not depend on a set

Table 2 Self supervised segmentation performance

Dataset	Mean error rate
Allegheny day	2.75 %
Allegheny dusk	4.15 %
Allegheny fall	2.73 %
Youghiogheny	3.22 %

Table 3 The mean error rate of various modified versions of the self supervised algorithm

Dataset	Chow Liu tree only	Perfect training	Full algorithm
Allegheny day	4.72 %	2.73 %	2.75 %
Allegheny dusk	8.14 %	4.09 %	4.15 %
Allegheny fall	3.82 %	2.68 %	2.73 %
Youghiogheny	3.24 %	3.23 %	3.22 %

of images with manually annotated groundtruth. Our belief for the performance improvement is due to the ability of our self supervised system to react almost instantly to the present point in the sequence, whereas the fully supervised classifier tries to learn a single appearance model of the river for the entire sequence.

To provide a deeper understanding of the self supervised algorithm and which steps are the most crucial to its working, we evaluated the performance of a few modified versions of the algorithm. One interesting question is: how important is it to learn a discriminative SVM classifier on top of the self supervised appearance modeling? The appearance modeling uses Chow Liu trees to calculate the probability of each image patch being a part of the river or the shore. In our algorithm, these probabilities are used to select patches for training a river/shore classifier. This SVM step can be omitted and the appearance modeling output can be used directly by labeling all pixels that have a $P(R \mid X)$ value of more than 0.5 as part of the river. The performance of the resulting classifier is shown in the 'Chow Liu Tree Only' column of Table 3. It can be seen that for most of the datasets, a steep reduction in performance occurs. This indicates that using a discriminative classification step which can exploit a richer feature descriptor and image position information is important to getting good performance.

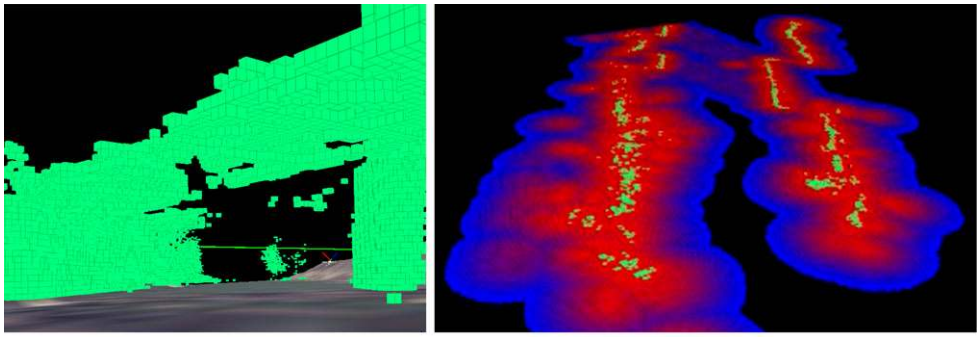
Another point of interest is investigating how mistakes made at the self supervision step effect performance. Patches with very high values of $P(R \mid X)$ are marked as river training examples and patches with very low values are used as shore class examples. But it is possible that some of these automatically selected training samples will have incorrectly inferred labels. If these labels inferred from $P(R \mid X)$ are replaced with groundtruth labels and the rest of the algorithm is unchanged, we observed that there is very little change in the overall performance (the 'Perfect Training' column in Table 3). This indicates that there are few mislabeled patches in the automatically selected training sets to begin with and that the SVM is able to generalize over them. Table 3 repeats the performance of the actual self supervised algorithm under the 'Full Algorithm' column for comparison.

3.4 Obstacle mapping

We tested the obstacle mapping algorithms on the river dataset described above. We assume the vehicle will be primarily traveling horizontally, and therefore the grid and distance transform's dimensions were 6 times the nominal ladar range in the horizontal directions and 2 times in the vertical direction. With a nominal sensor range of 30 meters and a cell size of 0.5 meters this meant a $360 \times 360 \times 120$ cell grid that required significant grid scrolling over the length of the dataset. Figure 16 shows a visualization of the occupancy grid and distance transform. The occupancy grid display shows only cells with an occupancy likelihood above 50 %. The distance transform display is only a 2D slice of the full 3D transform.

Table 4 shows the time taken to update the distance transform with an expansion of 20 cells (10 meters) and update frequency of 5 Hz, averaged over the length of the dataset. The simple mask algorithm is shown to be relatively slow. Our improvements show a small reduction in average update time compared to the Lau algorithm with the bug removed (see Sect. 2.3.1). This dataset had well registered laser data and a static environment resulting in very few obstacle cell removals, roughly 1 removal per update on average. As our key improvements are for obstacle removal we expect that in environments where more occupied cells must be removed (for example obscurants or dynamic environments) our algorithm would show further speed gains. Table 4 also shows that our algorithm visits fewer cells, adds fewer cells to the priority queue, and removes the need to modify the queue. Table 5 shows the memory usage of the distance transform algorithms. The Lau algorithm uses at minimum 10 bytes per cell, 2 for distance and 8 for a pointer (an additional queue index may also be required to allow for queue modifications). Our algorithm uses only 4 bytes by replacing the obstacle pointer and distance with an index into a lookup table (no queue index is required). The use of scrolling significantly reduces the memory required to cover the length of the dataset.

The key parameter affecting the scrolling algorithm is α, which determines how often the grid is scrolled. Figure 17 shows the average time required per scroll event and the total time taken over the entire dataset doing scroll operations for different values of α. An α value of 1 means that the scroll is performed as late as possible and a value of 3, for this grid, means scrolls are carried out at every cycle. As the scroll



(a) (b)

Fig. 16 Visualization of the evidence grid and distance transform. The grid visualization (*left*) shows the test rig passing under a bridge and only displays cells considered occupied. The *green line* in the background indicates the current edge of the scrolling grid map. The distance transform visualization (*right*) shows a 2D slice from a high viewpoint. *Green* indicates an obstacle, distance is mapped from *red* (low distance) to *blue* (high distance), cells with maximum distance are not shown. The slice is taken just above water level, the two lines of obstacles (*green*) are the river banks. Some areas are *red* with no visible obstacle because the obstacle is out of plane, e.g. the *two bars* of *light red* across the river at the *top* of the image are bridges (Color figure online)

Table 4 Average time, number of cells visited, and number of additions & modifications to the priority queue taken to update the distance transform. Each update requires the addition or removal of a different number of occupied cells, the averages and standard deviation is calculated per update. The max time reflects the time taken on the update with the most work to be done, for this dataset

Algorithm	Average time (s)	Standard deviation (s)	Max time (s)	Average cell visits	Average queue additions	Average queue modifications
Simple mask	0.2960	1.2349	13.2203	39516000	no queue	no queue
Lau et al. (2010)	0.0213	0.0150	0.1333	303017	11693	772
Our algorithm	0.0200	0.0144	0.1319	283552	10928	0

Table 5 Approx. memory required to store the distance transform. The memory requirement for the lookup table component of our algorithm is displayed in parentheses. The memory required without map scrolling is calculated for a grid covering the entire mission area

Algorithm	Memory (Mbytes)	Memory, no scrolling (Mbytes)
Simple mask	31	960
Lau et al. (2010)	156	4800
Current algorithm	62 + (0.5)	1920 + (0.5)

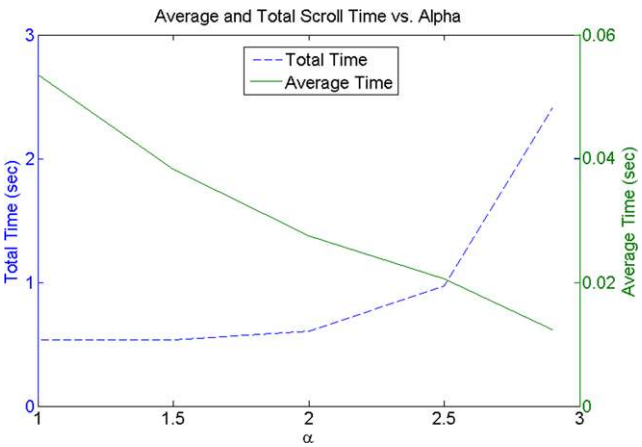


Fig. 17 Time required to scroll the grid as α, a parameter controlling how often the grid is scrolled, is varied. Average time is calculated per scroll event, total time is measured over the entire dataset. An α value of 1 means scrolls are performed as late as possible, a value of 3 means they are performed every update cycle (Color figure online)

frequency increases the per scroll time required decreases linearly however the total work required starts to increase rapidly.

3.5 Integrated mapping

Here we present a visual verification of the maps generated by our system against globally registered satellite images. Figure 18 shows examples of the 3D reconstruction built from the back projected ladar as the vehicle moves through the environment. Each laser scan is globally registered and placed into a world map by using the graph-based state estimate system. Since the laser scans occur at a higher frequency than the state estimates, an intermediate state is found by interpolating between neighboring state estimates. The state estimate is locally smooth and accurate enough to generate clean 3D reconstructions. The terrain mesh seen in

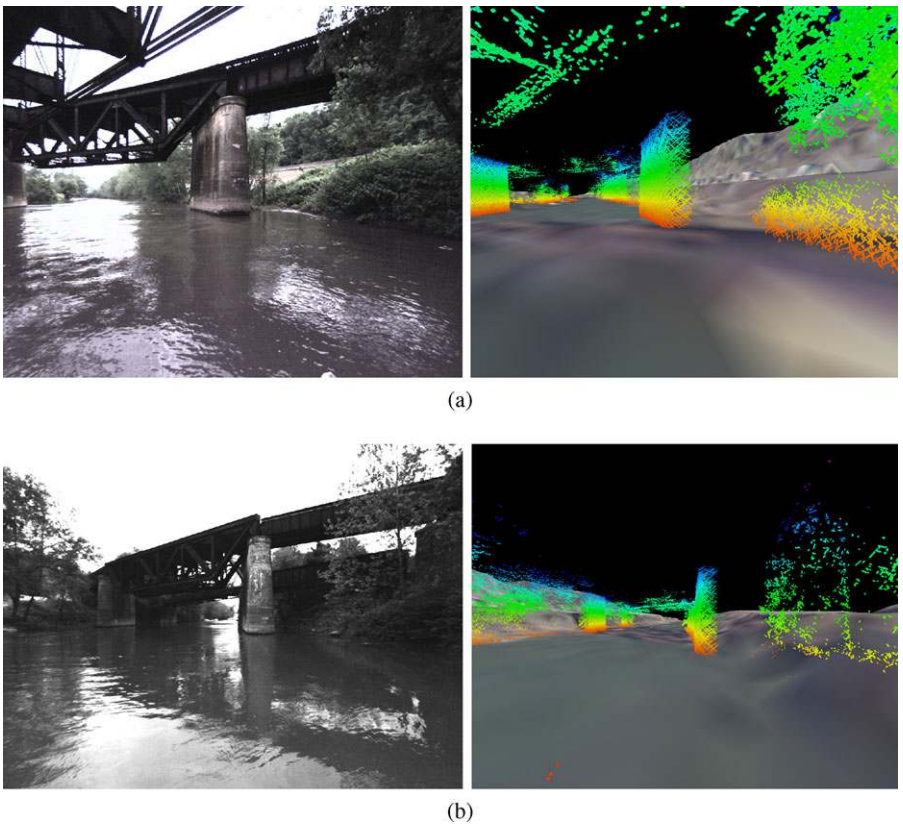


Fig. 18 Examples of the 3D point cloud maps. The images on the left show the raw image from the camera. On the right one can see the reconstructed point cloud generated from our laser range scans registered by our state estimate. The 3D point cloud is rendered within a visualization of an aerial image overlaid on terrain elevation data. The visualization is rendered from a virtual camera approximately at the location where the raw image was captured. The point cloud is colored by height of the point, ranging from red points that are low to blue points that are high

the reconstruction is build from elevation data and orthoimagery provided by USGS.[1] The point cloud produces a measurement of the river canopy height and a dense structure of the shoreline. The complete point cloud map generated from the 2 km traverse along the river channel can be seen in Fig. 19(a) as a top-down view overlaid on a satellite image.

The map in Fig. 19(b) is generated from the visual river classification algorithm described in Sect. 2.2 and is a 2D representation of the river course and width. The map is created by integrating the classified images at 2 Hz into an evidence grid in a global coordinate frame.

3.6 Discussion

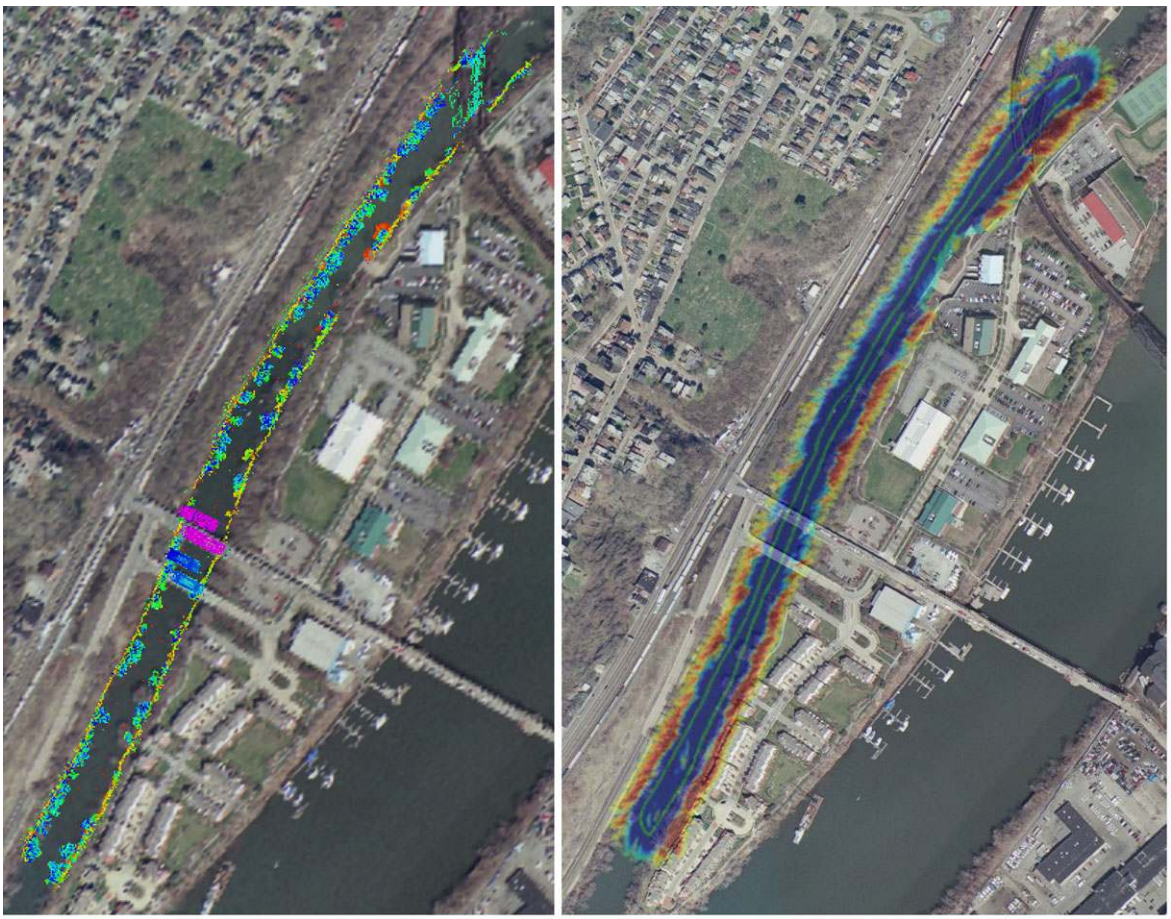
Here we discuss some insights into our results and finish with an overall assessment of our system in the context of criteria necessary to enable autonomous river mapping.

Firstly when analyzing the state estimation system, an ongoing issue that we are dealing with is a bias towards underestimating distance-traveled in the visual odometry system. This is observable in Figs. 14(a) and 14(b), where the estimated trajectory clearly falls short at both ends of the loop in the river. The underestimation bias is caused by the large distance to the features on the river rank detected in the stereo camera; more details on this issue are presented in Rehder et al. (2012). The impact of the underestimation on the resulting maps can be seen in Fig. 19(a) where we see errors introduced causing ghosting effect of the bridges. The bridges are seen twice by the laser scanner, once on the traverse upstream and once on the downstream traverse, due to the underestimation problem in the visual odometry described above.

Figure 19(b) shows the 2D river map, which was generated by integrating classified images with limited effective range. The effective range depends on the height of the camera above the water surface, which determines the angle of the projection of the images onto the evidence grid. Here the data was collected approximately 2.5 m from the water surface and the current reliable maximum range for integrating the classification results into the map is about 30 m. Both river banks are not within range at the same time, which requires a traverse along one bank and return traverse along the other bank. The effective range will improve as our flight altitude increases from the current 2 m to about 6–8 m. At this height the projection of the image into the map will be at a less oblique angle and more reliable at a longer range.

Now we describe and discuss our criteria of success in two ways; first is the ability of our sensing and perception algorithms to automatically navigate the river and second is

[1] http://seamless.usgs.gov/.



(a) 3D map of the river generated from laser scanner.

(b) 2D map generated from self-supervised visual river classification.

Fig. 19 River maps generated from a 2 km traverse. *Left*, the 3D point cloud rendered on top of a satellite image of the river. The point cloud is colored by height of the point, ranging from *red* points that are low, to high points that are *colored* magenta. *Right*, the 2D map of the river course and width generated from the self-supervised river classification algorithm Sect. 2.2. The classification algorithm generates an evidence grid in world coordinates colored as high probability of river (*blue*) and low probability of river (*red*) (Color figure online)

the capability to generate a map of the river course, width, and canopy height.

To autonomously navigate we need the following: an accurate state estimation for vehicle control, a heading to follow the river, and a reliable method to sense obstacles. We have a demonstrated a real-time system for state estimation, which is locally consistent and globally accurate. The output of the river detection system gives a direction to travel in to follow the river course. Finally, we have demonstrated real-time algorithms adequate in both execution time and in memory required to generate an obstacle cost map from laser scanner. Considering all of the above we regard our system design as suitable for autonomous navigation of a river.

Now we analyze our design for ability to generate appropriate river maps. The 3D river map seen in Fig. 19(a) illustrates that despite sparse GPS data, it is still possible to create a globally registered 3D map of the environment and in particular the canopy height as seen in Fig. 18. Looking at the 2D river map presented in Fig. 19(b), we see further proof for the success of the river classification algorithm and, importantly, that it is able to generate a map of the river course and width.

4 Conclusions

We have described a lightweight perception system for autonomously navigating and mapping a river from a low-flying rotorcraft.

The system incorporates a global state estimation system that is both locally consistent—necessary for vehicle control—and globally referenced—a requirement for the resulting river maps. The state estimation combines visual odometry, inertial measurement, and sparse GPS readings in a graph-optimization algorithm.

A self-supervised visual river classification algorithm is developed to determine the direction to travel along the river and also for mapping river course and width (2D map).

A unique, lightweight, off-axis, spinning laser scanner is used for sensing obstacles and mapping the river bank structure and canopy height. The laser scans are efficiently processed to compute the obstacle map and distance transform necessary to avoid obstacles in real-time. An analysis is presented of the laser scanner to compute the maximum safe velocities for the rotorcraft to guarantee obstacle avoidance.

The experimental results over a 2 km traverse along a river show that mapping the river course, width, and canopy height are all feasible from lightweight sensors that are available on a micro aerial vehicle.

In future work, we will evaluate our system on different rivers, with even more degraded GPS coverage, and over longer distances on the order of 10 km. We seek to improve the state estimation accuracy and latency, to demonstrate high performance vehicle control and completely autonomous flight for sustained periods, and finally to demonstrate aggressive maneuvers around complex obstacles such as tree vines. We also want to investigate the possibility of perceiving the river flow rate from the rotorcraft.

Acknowledgements The authors gratefully acknowledge Lyle Chamberlain, and Justin Haines for their feedback and help with hardware. The work described in this paper is funded by the Office of Naval Research under grant number N00014-10-1-0715.

References

Achar, S., Sankaran, B., Nuske, S., Scherer, S., & Singh, S. (2011). Self-supervised segmentation of river scenes. In *Proceedings international conference on robotics and automation (ICRA)*.

Andert, F., & Goormann, L. (2007). Combined grid and feature-based occupancy map building in large outdoor environments. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (pp. 2065–2070).

Andert, F., Adolf, F. M., Goormann, L., & Dittrich, J. S. (2010). Autonomous vision-based helicopter flights through obstacle gates. *Journal of Intelligent & Robotic Systems*, *57*(1–4), 259–280.

Bryson, M., & Sukkarieh, S. (2007). Building a robust implementation of bearing-only inertial SLAM for a UAV. *Journal of Field Robotics*, *24*(1–2), 113–143.

Chambers, A., Achar, S., Nuske, S., Rehder, J., Kitt, B., Chamberlain, L., Haines, J., Scherer, S., & Singh, S. (2011). Perception for a river mapping robot. In *Proceedings of the 2011 IEEE/RSJ international conference on intelligent robots and systems (IROS'11)*.

Chow, C., & Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, *14*(3), 462–467.

Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., & Bradski, G. (2006). Self-supervised monocular road detection in desert terrain. In *Robotics science and systems conference (RSS'06)*.

Reid, I., Davison, A., Molton, N., & Stasse, O. (2007). Monoslam: real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1052–1067.

Desai, A., & Huber, D. (2009). Objective evaluation of scanning ladar configurations for mobile robots. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*.

Eustice, R., Singh, H., & Leonard, J. (2005). Exactly sparse delayed-state filters. In *Proceedings of the IEEE international conference on robotics and automation, ICRA 2005* (pp. 2417–2424). New York: IEEE Press.

Fairfield, N. (2009). Localization, mapping, and planning in 3D environments. Ph.D. Thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Folkesson, J., & Christensen, H. I. (2007). Graphical SLAM for outdoor applications. *Journal of Field Robotics*, *24*(1–2), 51–70.

Geiger, A., Ziegler, J., & Stiller, C. (2011). StereoScan: dense 3D reconstruction in real-time. In *IEEE intelligent vehicles symposium*, Baden-Baden, Germany.

Grzonka, S., Grisetti, G., & Burgard, W. (2009). Towards a navigation system for autonomous indoor flying. In *IEEE international conference on robotics and automation, ICRA'09* (pp. 2878–2883). New York: IEEE Press.

Haines, J. (2011). Vision-based control of an aerial vehicle. Master's thesis, Carnegie Mellon University.

Harris, C., & Stephens, M. (1988). A combined corner and edge detector. In *Alvey vision conference*, Manchester, UK (vol. 15, p. 50).

Harrison, A., & Newman, P. (2008). High quality 3D laser ranging under general vehicle motion. In *Proc. IEEE international conference on robotics and automation, ICRA'08*, Pasadena, California.

Holz, D., Droeschel, D., Behnke, S., May, S., & Surmann, H. (2010). Fast 3D perception for collision avoidance and SLAM in domestic environments. In A. Barrera (Ed.), *Mobile robots navigation* (pp. 53–84). Vienna: IN-TECH Education and Publishing.

Hrabar, S., & Gaurav, S. (2009). Vision-based navigation through urban canyons. *Journal of Field Robotics*, *26*(5), 431–452.

Kalra, N., Ferguson, D., & Stentz, A. (2006). Incremental reconstruction of generalized Voronoi diagrams on grids. In *Proc. of the international conference on intelligent autonomous systems*.

Koenig, S., & Likhachev, M. (2002). D* Lite. In *Eighteenth national conference on artificial intelligence*.

Konolige, K. (2004). Large-scale map-making. In: *Proceedings of the national conferences on artifical intelligence* (pp. 457–463). Menlo Park/London:AAAI Press/MIT Press

Konolige, K., & Agrawal, M. (2008). Frameslam: from bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics and Automation*, *24*(5), 1066–1077.

Kuemmerle, R., Grisetti, G., Strasdat, H., Konolige, K., & Burgard, W. (2011). g2o: a general framework for graph. *Optimization Journal of Autonomous Robots*, *30*(1), 25–39.

Lau, B., Sprunk, C., & Burgard, W. (2010). Improved updating of Euclidean distance maps and Voronoi diagrams. In *IEEE/RSJ international conference on intelligent robots and systems, IROS, 2010* (pp. 281–286).

Laws, K. (1980). Rapid texture identification. In: *SPIE* (pp. 376–380).

Leedekerken, J., Fallon, M., & Leonard, J. (2010). Mapping complex marine environments with autonomous surface craft. In *12th International symposium on experimental robotics*

Li, S. Z. (1994). A Markov random field model for object matching under contextual constraints. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition* (pp. 866–869).

Lourakis, M. (2010). Sparse non-linear least squares optimization for geometric vision. *Computer Vision–ECCV*, *2010*, 43–56.

Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11*(2), 431–441.

Martin, M. C., & Moravec, H. (1996). *Robot evidence grids*. Tech. Rep. CMU-RI-TR-96-06, The Robotics Institute, Carnegie Mellon, Pittsburgh, PA.

Matsumoto, M., & Yuta, S. (2010). 3D laser range sensor module with roundly swinging mechanism for fast and wide view range image. In *IEEE conference on multisensor fusion and integration for intelligent systems, MFI, 2010* (pp. 156–161).

Meier, L., Tanskanen, P., Fraundorfer, F., & Pollefeys, M. (2011). PIXHAWK: a system for autonomous flight using onboard computer vision. In *IEEE international conference on robotics and automation, ICRA, 2011* (pp. 2992–2997). New York: IEEE Press.

Meijster, A., Roerdink, J., & Hesselink, W. (2000). A general algorithm for computing distance transforms in linear time. In *Mathematical morphology and its applications to image and signal processing* (pp. 331–340).

Rankin, A., & Matthies, L. (2010). Daytime water detection based on color variation. In *IEEE intl. conf. on intelligent robots and systems, IROS'10*.

Rankin, A., Matthies, L., & Huertas, A. (2004). Daytime water detection by fusing multiple cues for autonomous off-road navigation. In *24th Army science conference, ASC'04*.

Rathinam, S., Almeida, P., Kim, Z., Jackson, S., Tinka, A., Grossman, W., & Sengupta, R. (2007). Autonomous searching and tracking of a river using an UAV. In *American control conference, ACC'07* (pp. 359–364).

Rehder, J., Gupta, K., Nuske, S., & Singh, S. (2012). Global pose estimation with limited GPS and long range visual odometry. In *IEEE international conference on robotics and automation, ICRA, 2012*. New York: IEEE Press.

Scherer, S., Singh, S., Chamberlain, L., & Elgersma, M. (2008). Flying fast and low among obstacles: methodology and experiments. *The International Journal of Robotics Research*, *27*(5), 549–574.

Scherer, S., Ferguson, D., & Singh, S. (2009). Efficient C-space and cost function updates in 3D for unmanned aerial vehicles. In *IEEE international conference on robotics and automation, ICRA'09* (pp. 2049–2054). New York: IEEE Press.

Shen, S., & Kumar, Michael N. V (2011). Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *IEEE international conference on robotics and automation, ICRA, 2011* (pp. 20–25).

Strasdat, H., Montiel, J., & Davison, A. (2010). Real-time monocular SLAM: why filter. In *IEEE international conference on robotics and automation, ICRA, 2010* (pp. 2657–2664). New York: IEEE Press.

Takasu, T., Kubo, N., & Yasuda, A. (2007). Development, evaluation and application of RTKLIB: a program library for RTK-GPS. In *GPS/GNSS symposium*.

Torralba, A., & Efros, A. A. (2011). Unbiased look at dataset bias. In *CVPR* (pp. 1521–1528).

Viquerat, A., Blackhall, L., Reid, A., & Sukkarieh, S. (2007). Reactive collision avoidance for unmanned aerial vehicles using Doppler radar. In *Proceedings of the international conference on field and service robotics*.

Weiss, S., Achtelik, M., Kneip, L., Scaramuzza, D., & Siegwart, R. (2011). Intuitive 3D maps for MAV terrain exploration and obstacle avoidance. *Journal of Intelligent & Robotic Systems*, *61*, 473–493.

Wulf, O., & Wagner, B. (2003). Fast 3D scanning methods for laser measurement systems. In *International conference on control systems and computer science, CSCS14* (pp. 2–5).

Yang, J., Rao, D., Chung, S., & Hutchinson, S. (2011). Monocular vision based navigation in GPS-denied riverine environments. In *Proceedings of the AIAA Infotech@Aerospace conference*, St. Louis, MO.

Sebastian Scherer is a Systems Scientist at the Robotics Institute (RI) at Carnegie Mellon University (CMU). His research focuses on enabling unmanned rotorcraft to operate at low altitude in cluttered environments. He has developed a number of intelligent autonomous rotorcraft. Dr. Scherer received his B.S. in Computer Science, M.S. and Ph.D. in Robotics from CMU in 2004, 2007, and 2010.

Joern Rehder is a Masters (Diplom) student in Electrical Engineering at the Hamburg University of Technology, Germany. He is currently with the Robotics Institute at Carnegie Mellon University and has been a visiting scholar at the University of California, Berkeley in 2007/2008. His research interests include computer vision and SLAM.

Supreeth Achar is currently a research programmer at the Robotics Institute in Carnegie Mellon University working in the Field Robotics Center. He received a M.S. in robotics from Carnegie Mellon University in 2010. He is interested in using computer vision and machine learning techniques to build mobile robots that can figure out where they are, where they need to go and how to get there.

Hugh Cover is a Masters student at the Robotics Institute (RI) at Carnegie Mellon University (CMU). His research interests include mapping and motion planning for autonomous air vehicles. He received his B.E. in Mechatronic Engineering from the University of New South Wales, Australia (UNSW) in 2009.

Andrew Chambers is a master's student at Carnegie Mellon University's Robotic Institute. He obtained his B.S. in Electrical Engineering from the University of Southern California. His research interests include perception and control for UAVs.

Stephen Nuske is a Project Scientist at the Robotics Institute, Carnegie Mellon University since 2010. Before that he was a Post-Doc at the Robotics Institute from 2008–2010. He received a Ph.D. and a Bachelor of Software Engineering from the Universtiy of Queensland, conducting much of the doctoral research at the Autonomous Systems Laboratory at the Commonweath Scientific and Industry Research Organization in Queensland, Australia. He spent three months as a visiting scholar at the eMotion lab at INRIA, Grenoble, France. His research is in computer vision systems applied to outdoor robotics applications such as heavy industry, surveillance and agriculture.

Sanjiv Singh is a Research Professor at the Robotics Institute, Carnegie Mellon University. His recent work has two main themes: perception in natural environments and multi-agent coordination. He has led projects in both ground and air vehicles operating in unknown or partially known environments, in applications such as mining, agriculture, emergency response, surveillance and exploration. He is also actively involved in the automation of complex tasks, such as the assembly of large space structures, that cannot be addressed by single agents and must necessarily be performed by teams. Prof. Singh received his B.S. in Computer Science from the University of Denver (1983), M.S. in Electrical Engineering from Lehigh University (1985) and a Ph.D. in Robotics from Carnegie Mellon (1995). He is the founder and Editor-in-Chief of the Journal of Field Robotics.