OctoSLAM: A 3D Mapping Approach to Situational Awareness of Unmanned Aerial Vehicles

Joscha Fossel[1], Daniel Hennes[1], Daniel Claes[1], Sjriek Alers[1], Karl Tuyls[1]

Abstract— **The focus of this paper is on situational awareness of airborne agents capable of 6D motion, in particular multi-rotor UAVs. We propose the fusion of 2D laser range finder, altitude, and attitude sensor data in order to perform simultaneous localization and mapping (SLAM) indoors. In contrast to other planar 2D laser range finder based SLAM approaches, we perform SLAM on a 3D instead of a 2D map. To represent the 3D environment an octree based map is used. Our scan registration algorithm is derived from Hector SLAM. We evaluate the performance of our system in simulation and on a real multirotor UAV equipped with a 2D laser range finder, inertial measurement unit, and altitude sensor. The results show significant improvement in the localization and representation accuracy over current 2D map SLAM methods. The system is implemented using Willow Garage's robot operating system.**

INTRODUCTION

Multi-rotor Unmanned Aerial Vehicles (UAVs) are popular yet complex systems which have become widely available to the research community. A common vision is that such (semi-)autonomous airborne agents have a broad scope of applications and can be beneficial in numerous scenarios. The autonomous exploration of large, inaccessible or hazardous environments in the scope of urban search and rescue scenarios is a prominent example. A prerequisite to any such autonomous behavior however is the ability of the agent to perceive environmental elements and to localize itself.

Indoor situational awareness of UAVs is commonly tackled by either time of flight (e.g. laser range finder) or visual (e.g. depth cameras) simultaneous localization and mapping (SLAM) [1]. While visual sensors may provide 3D data, they are also lighting dependent, have a lower range and less precision than time of flight sensors. Visual SLAM also tends to be more computationally expensive than time of flight SLAM, which can be crucial on airborne robots which typically have limited payload and thus also limited processing capabilities. Therefore, we opt to use time of flight sensors. However, 3D time of flight sensors are usually heavy, expensive and have a high power consumption. Thus, to the best of our knowledge, only 2D laser range finders have been used on UAVs for time of flight SLAM so far. The established practice is to down-project the laser range data to 2D maps. This assumes that objects look the same regardless of the observation height. On the one hand, if this assumption holds, 2D SLAM [2], e.g. particle filter based SLAM [3], suffices for UAV SLAM. On the other hand, if the assumption does not hold, using 2D maps in the UAV domain may become impracticable. Therefore, we develop a planar 2D laser range finder based SLAM algorithm that operates on a 3D representation of the environment. We furthermore investigate whether and to what extend using 3D maps for SLAM outperforms using 2D maps.

The remainder of this paper is structured as follows. The next section provides background information on related work, which we combine and extend in this paper. Afterwards, we introduce our approach, called OctoSLAM, which advances the state-of-the-art low resource requiring SLAM frameworks for agents exhibiting 6D motion. This is followed by both simulated and real robot experiments. The paper then concludes with a brief discussion of our findings.

BACKGROUND

In this section we introduce research by other authors which we use in our approach, in particular

[1]The authors are with the Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands. Email: {j.fossel,daniel.hennes,daniel.claes,sjriek.alers, k.tuyls}@maastrichtuniversity.nl

Hector SLAM [4] and Octomap mapping [5].

Hector SLAM

Hector SLAM scan registration [4] is used for fast online learning of occupancy grid maps [6]. It registers 2D laser range finder input with a map in order to determine the 3D pose, and is designed to require low computational resources. The Hector SLAM scan registration algorithm consists of two steps that are repeated until the maximum number of iterations is reached. It operates on an occupancy grid map, i.e. a discrete map representation that stores the probability of a grid cell being occupied. The two steps are as follows:

1) Given a transformation $T = (t^x, t^y, \gamma)'$, an occupancy grid map M and a set of scan endpoints D the map gradient ∇M is determined.
2) Use the map gradient ∇M to update the transformation T, or stop if the maximum number of iterations is reached.

Optimally aligning the beam endpoints in D with map M translates to finding a transformation T that minimizes Equation 1 [4]:

$$T^* = \arg\min_T \sum_{i=1}^{N} [1 - M(T \otimes d_i)]^2 \qquad (1)$$

$T \otimes d_i$ refers to applying transformation T to scan endpoint $d_i \in D$, as defined in Equation 2. $M(T \otimes d_i)$ returns the map occupancy value at the coordinates provided by $T \otimes d_i$.

$$T \otimes d_i = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) \\ sin(\gamma) & cos(\gamma) \end{pmatrix} \begin{pmatrix} d_i^x \\ d_i^y \end{pmatrix} + \begin{pmatrix} t^x \\ t^y \end{pmatrix} \qquad (2)$$

To optimize the alignment of scan endpoints d_i with map M we seek to estimate transformation update ΔT that decreases the error:

$$\sum_{i=0}^{N} [1 - M(d_i \otimes (T + \Delta T))]^2 \to 0 \qquad (3)$$

Using a first order Taylor expansion of the map occupancy value M $(d_i \otimes (T + \Delta T))$ in Equation 3 leads to the following error term:

$$\left[1 - M(T \otimes d_i) - \nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T} \Delta T\right]^2 \qquad (4)$$

Here, $\nabla M(T \otimes d_i)$ is the map gradient at the position given by $T \otimes d_i$:
$\nabla M((T \otimes d_i)) = \left(\frac{\partial M}{\partial x}(T \otimes d_i),\ \frac{\partial M}{\partial y}(T \otimes d_i)\right)$

To minimize Equation 4 the partial derivative with respect to ΔT is set to zero:

$$0 = 2 \sum_{i=1}^{N} \left[\nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T}\right]' + \left[1 - M(T \otimes d_i) - \nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T} \Delta T\right] \qquad (5)$$

Solving this equation for ΔT leads to the following Gauss-Newton equation for the minimization problem:

$$\Delta T = H^{-1} \sum_{i=1}^{N} \left[\nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T}\right]' \cdot [1 - M(T \otimes d_i)] \qquad (6)$$

with

$$H = \left[\nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T}\right]' \cdot \left[\nabla M(T \otimes d_i) \frac{\partial (T \otimes d_i)}{\partial T}\right] \qquad (7)$$

Using Equation 2 $\frac{\partial (T \otimes d_i)}{\partial T}$ can be resolved to:

$$\frac{\partial (T \otimes d_i)}{\partial T} = \begin{pmatrix} 1 & 0 & -\sin(\gamma) d_i^x & -\cos(\gamma) d_i^y \\ 0 & 1 & cos(\gamma) d_i^x & -sin(\gamma) d_i^y \end{pmatrix} \qquad (8)$$

Equation 6 can now be evaluated via $\nabla M(T \otimes d_i)$ and $\frac{\partial (T \otimes d_i)}{\partial T}$, yielding the desired transformation that optimizes the scan to map alignment.

As mentioned before, Hector SLAM operates on an occupancy grid map. This map representation is implemented using a two dimensional array, which stores the occupancy probability of the corresponding space. However, the discrete nature of this approach limits precision and does not allow for direct computation of map gradients. Therefore, an interpolation scheme that approximates occupancy probabilities for continuous coordinates is employed. Given a continuous coordinate $P_m = (p_m^x, p_m^y)$, the four surrounding discrete coordinates $P_{00..11}$ are used for linear interpolation along

x and y axis [4]:

$$\begin{aligned} M(P_M) \approx (p_m^y - p_{10}^y) \cdot [(p_m^x - p_{00}^y) \cdot M(P_{11}) \\ + (p_{11}^x - p_m^x) \cdot M(P_{01})] \\ + (p_{11}^y - p_m^y) \cdot [(p_m^x - p_{01}^x) \cdot M(P_{10}) \\ + (p_{11}^x - p_m^x) \cdot M(P_{00})] \end{aligned} \tag{9}$$

with

$$P_{00} = (p_{00}^x, p_{00}^y) = (\lceil p_m^x \rceil, \lfloor p_m^y \rfloor) \tag{10a}$$
$$P_{01} = (p_{01}^x, p_{01}^y) = (\lfloor p_m^x \rfloor, \lfloor p_m^y \rfloor) \tag{10b}$$
$$P_{10} = (p_{10}^x, p_{10}^y) = (\lceil p_m^x \rceil, \lceil p_m^y \rceil) \tag{10c}$$
$$P_{11} = (p_{11}^x, p_{11}^y) = (\lfloor p_m^x \rfloor, \lceil p_m^y \rceil) \tag{10d}$$

Using the four surrounding discrete coordinates the derivatives for the map gradient can also be approximated:

$$\begin{aligned} \frac{\partial M}{\partial x} = (p_m^y - p_{00}^y) \cdot [M(P_{11}) - M(P_{01})] \\ + (p_{01}^y - p_m^y) \cdot [M(P_{10}) - M(P_{00})] \end{aligned} \tag{11}$$

$$\begin{aligned} \frac{\partial M}{\partial y} = (p_m^x - p_{00}^x) \cdot [M(P_{11}) - M(P_{10})] \\ + (p_{11}^x - p_m^x) \cdot [M(P_{01}) - M(P_{00})] \end{aligned} \tag{12}$$

Unfortunately, with this non-smooth linear approximation of the map gradient, local quadratic convergence of the scan registration algorithm cannot be guaranteed. In practice however, the algorithm works with sufficient accuracy [4].

Octomap

Octomap [5] is an octree based approach for 3D mapping. It is designed to meet the following four criteria: *full 3D mapping*, *updatable*, *flexible* and *compact*.

Full 3D mapping means that no prior assumptions about the environment are required. Therefore mapping of arbitrary environments is possible. Furthermore, information about spaces can be encoded in the map as *occupied*, *free* and *unknown*.

Updatable refers to information encoded in the map being updatable and represented in a probabilistic manner.

Flexibility relates to a dynamic map size, which does not need to be defined in advance. In addition, the map provides multiple resolutions. It is however necessary to define the minimum resolution beforehand. If the task at hand requires variable minimum resolutions, octree hierarchies [7] can be used.

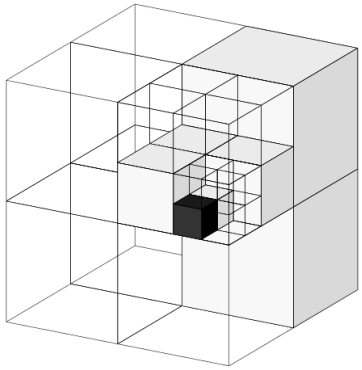

Fig. 1. This figure [5] gives an example for a concrete octomap map representation. It shows how occupied (black cube), free (shaded white cube) and unknown space (transparent cubes) is encoded.

Compactness requires the map to be both memory and disk-space efficient, i.e. optimized for random and sequential access memory.

Octrees are trees with nodes that have eight children splitting the space into eight cubic volumes of equal size. An example is given in Figure 1 [5]. Therefore, by descending from root to leaf nodes resolution is increased. Point information inserted into the octomap are stored in the octrees leaf nodes, which represent spaces of minimum resolution size. Leaf nodes are generated on demand when inserting new information. This way the absence of leaf nodes encodes *unknown* for the corresponding position.

To store occupancy probabilities, the log-Odd representation is used.

$$logodd(p) \triangleq \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p). \tag{13}$$

The log-Odds representation allows for updating occupancies without using the inverse sensor model, which is less computationally expensive. To retrieve the probabilities, the inverse log-Odd function is used.

$$probability(t) \triangleq \left(\frac{1}{1+e^{-t}}\right), \tag{14}$$

where t is the log-Odd value for p. After updating the log-Odds occupancy value of a leaf node, the

maximum values are propagated up to the root node.

Furthermore, the octomap framework has been optimized in respect to efficiency through tree pruning and a smart implementation. Details and a discussion of alternative representations for 3D mapping can be found in [5].

OCTOSLAM

For 3D map SLAM we combine and extend both Octomap [5], an octree representation of the environment, and Hector SLAM [4], an algorithm for fast online learning of occupancy grid maps. To implement the proposed SLAM system, the robot operating system (ROS) [8] framework is used.

OctoSLAM determines the 6D pose of the UAV by both localization and direct sensor input. In our setup an inertial measurement unit (IMU) provides roll ϕ and pitch ψ, while altitude z is measured by an actuated downward facing distance sensor. The remaining three dimensions, i.e. translation in x, y direction and rotation around the yaw axis γ, are tracked through localization. At time t the position determined by scan registration is given by $T_t = (x_t, y_t, \gamma_t)$. The algorithm iteratively computes the displacement or the most recent pose change $\Delta T_t = T_t - T_{t-1} = (\Delta x_t, \Delta y_t, \Delta \gamma_t)'$. The laser range finder returns a vector of scan endpoints $D = (d_0, \ldots, d_N)$ in polar coordinates. Via the current 6D pose estimate $T_t^* = (x, y, z, \phi, \psi, \gamma)$, scan endpoints $d_i = (r_i, \theta_i)$ are transformed to the Cartesian coordinates of the map representation, i.e. to 3D. For ease of notation we write d instead of d_i and T instead of T_t in the following.

$$\begin{pmatrix} d^x \\ d^y \\ d^z \end{pmatrix} = \begin{pmatrix} r \cdot \cos\theta \\ r \cdot \sin\theta \\ 0 \end{pmatrix} \tag{15}$$

Afterwards, this vector is rotated by the yaw angle γ determined by previous scan registration:

$$\begin{pmatrix} d^x_\gamma \\ d^y_\gamma \\ d^z_\gamma \end{pmatrix} = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} d^x \\ d^y \\ d^z \end{pmatrix} \tag{16}$$

Subsequently, the vector pointing to the scan endpoint needs to be rotated according to ϕ and ψ, i.e. the roll and nick angles provided by the IMU.

$$\begin{pmatrix} d^x_{\gamma\phi} \\ d^y_{\gamma\phi} \\ d^z_{\gamma\phi} \end{pmatrix} = \begin{pmatrix} \cos\phi & 0 & sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \begin{pmatrix} d^x_\gamma \\ d^y_\gamma \\ d^z_\gamma \end{pmatrix} \tag{17}$$

$$\begin{pmatrix} d^x_{\gamma\phi\psi} \\ d^y_{\gamma\phi\psi} \\ d^z_{\gamma\phi\psi} \end{pmatrix} = \begin{pmatrix} \cos\psi & -sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} d^x_{\gamma\phi} \\ d^y_{\gamma\phi} \\ d^z_{\gamma\phi} \end{pmatrix} \tag{18}$$

Finally, the vector is translated by x, y and z, i.e. the x, y, position and altitude of the UAV.

$$\begin{pmatrix} d^x_{\gamma\phi\psi t} \\ d^y_{\gamma\phi\psi t} \\ d^z_{\gamma\phi\psi t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \end{pmatrix} \cdot \begin{pmatrix} d^x_{\gamma\phi\psi} \\ d^y_{\gamma\phi\psi} \\ d^z_{\gamma\phi\psi} \\ 1 \end{pmatrix} \tag{19}$$

This vector describes the scan endpoint position in map frame coordinates, which is requiered for inserting a point into, and retrieving occupancy values from the octree. We denote this transform by $T^* \otimes d_i$. Hector SLAM uses occupancy grid map gradients ∇M to align the scan D with the current map. OctoSLAM computes interpolated map gradients based on occupancy probabilities of the target and surrounding nodes of an Octomap. The pose change ΔT is computed via the following Gauss-Newton equation introduced by Hecto SLAM:

$$\begin{aligned} \Delta T = H^{-1} \sum_{i=1}^{N} & \left[\nabla M(T^* \otimes d_i) \frac{\partial (T^* \otimes d_i)}{\partial T} \right]' \\ & \cdot \left[1 - M(T^* \otimes d_i) \right]. \end{aligned} \tag{20}$$

We recall that the Hessian H is computed as follows:

$$\begin{aligned} H = & \left[\nabla M(T^* \otimes d_i) \frac{\partial (T^* \otimes d_i)}{\partial T} \right]' \\ & \cdot \left[\nabla M(T^* \otimes d_i) \frac{\partial (T^* \otimes d_i)}{\partial T} \right], \end{aligned} \tag{21}$$

where $\nabla M(T^* \otimes d_i) = (mv_i, dy_i, dx_i)'$ are again the bilinearly interpolated map value gradients. Similarly, the partial derivative $\frac{\partial (T^* \otimes d_i)}{\partial T}$ is computed as:

$$\frac{\partial (T^* \otimes d_i)}{\partial T} = \begin{pmatrix} 1 & 0 & -\sin(\gamma) d_i^x & -\cos(\gamma) d_i^y \\ 0 & 1 & cos(\gamma) d_i^x & -sin(\gamma) d_i^y \end{pmatrix}. \tag{22}$$

In contrast to Hector SLAM, which uses an array based occupancy grid, we use an octree based map. Therefore getting the interpolated map values and gradients is slightly different, as the voxel size throughout the octree is not constant.

Given a target coordinate $p = (p^x, p^y, p^z)'$, the voxels v_0, v_1, v_2, v_3 surrounding p have to be determined. To determine these voxels, p is rounded down to the next voxel center coordinate which is the voxel v_0:

$$\begin{pmatrix} v_0^x \\ v_0^y \\ v_0^z \end{pmatrix} = \begin{pmatrix} m_{res} \oplus p^x \\ m_{res} \oplus p^y \\ m_{res} \oplus p^z \end{pmatrix} \tag{23}$$

where m_{res} is the distance between the voxel centers, i.e. the resolution of the octomap which depends on the minimum resulution and on the tree search depth n.

$$m_{res} = 2^{n-1} \cdot min_{res} \tag{24}$$

The operator $m_{res}\oplus$ is defined as rounding down to the next voxel center. Algorithm 1 gives a pseudo code formulation for $m_{res}\oplus$. Correspondingly voxel v_1 is the voxel right to v_0, v_2 the voxel to the top of v_0, and v_3 the voxel right to v_2.

Algorithm 1 Round to Voxel Center

Require: Voxel resolution m_{res},
input coordinate in

```
function ROUND(m_res, in)
    voxelCenter ← 0
    mod ← in modulo m_res
    if mod larger m_res/2 then
        voxelCenter ← in + m_res − mod − m_res/2
    else
        voxelCenter ← in − mod − m_res/2
    end if
    return voxelCenter
end function
```

$$\begin{pmatrix} v_1^x \\ v_1^y \\ v_1^z \end{pmatrix} = \begin{pmatrix} v_1^x + m_{res} \\ v_1^y \\ v_1^z \end{pmatrix} \tag{25a}$$

$$\begin{pmatrix} v_2^x \\ v_2^y \\ v_2^z \end{pmatrix} = \begin{pmatrix} v_1^x \\ v_1^y + m_{res} \\ v_1^z \end{pmatrix} \tag{25b}$$

$$\begin{pmatrix} v_3^x \\ v_3^y \\ v_3^z \end{pmatrix} = \begin{pmatrix} v_2^x + m_{res} \\ v_2^y \\ v_2^z \end{pmatrix} \tag{25c}$$

The voxels $v_0..v_3$ are used to compute the interpolated map value mv as described in the previous section, with the addition of scaling with m_{res}, which is not necessary when using an array based map. The map value mv is hence defined as:

$$\begin{aligned} mv = &\frac{v_3^y - p^y}{m_{res}} \cdot \frac{v_1^x - p^x}{v_1^x - v_2^x} \cdot M(v_0) \\ &+ \frac{v_3^y - p^y}{m_{res}} \cdot \frac{p^x - v_2^x}{v_1^x - v_2^x} \cdot M(v_1) \\ &+ \frac{p^y - v_0^y}{m_{res}} \cdot \frac{v_1^x - p^x}{v_1^x - v_2^x} \cdot M(v_2) \\ &+ \frac{p^y - v_0^y}{m_{res}} \cdot \frac{p^x - v_2^x}{v_1^x - v_2^x} \cdot M(v_3) \end{aligned} \tag{26}$$

where $M(v_x)$ refers to the occupancy probability of v_x stored in the octree node, which can be found using Algorithm 2. This algorithm returns a list of octree nodes, which correspond to a certain x, y, z location. The map gradients can be found analogously to the map value:

$$\begin{aligned} \frac{\partial M}{\partial x} = &\frac{v_3^y - p^y}{m_{res}} \cdot (M(v_0) - M(v_2)) \\ &+ \frac{p^y - v_0^y}{m_{res}} \cdot (M(v_1) - M(v_3)) \end{aligned} \tag{27}$$

$$\begin{aligned} \frac{\partial M}{\partial y} = &\frac{v_1^x - p^x}{m_{res}} \cdot (M(v_0) - M(v_1)) \\ &+ \frac{p^x - v_2^x}{m_{res}} \cdot (M(v_2) - M(v_3)) \end{aligned} \tag{28}$$

Using the map values and gradients of the map for each scan endpoint, Equation 20 can be

Algorithm 2 Octomap Search

Require: target coordinates tC,
search depth tD,
octree $tree$

function OCTOSEARCH($tC, tD, tree$)
  $list \leftarrow \{\}$
  $path \leftarrow$ calculatePath(tC)
  $curNode \leftarrow tree$.getRoot()
  $i \leftarrow tree$.getMaxTreeDepth()-1
  for i down to 0 **do**
    $list \leftarrow \{list, curNode\}$
    if $i+1$ equals tD **then**
      return $list$
    end if
    $nextNode \leftarrow path$.getNode(i)
    if $curNode$.hasChild($nextNode$) **then**
      $curNode \leftarrow nextNode$
    else
      if $curNode$.hasNoChildren() **then**
        return $list$
      else
        $list \leftarrow \{list, -1\}$
        return $list$
      end if
    end if
  end for
  return $list$
end function

resolved to:

$$\Delta T = \left[\sum_{i=0}^{n} \begin{pmatrix} \sqrt{dy_i} & dy_i \cdot dx_i & dy_i \cdot \gamma_i' \\ dy_i \cdot dx_i & \sqrt{dx_i} & \gamma' \cdot dx_i \\ dy_i \cdot \gamma' & dy_i \cdot dx_i & \sqrt{\gamma_i'}) \end{pmatrix} \right]^{-1} \cdot \sum_{i=0}^{n} \left[(1 - mv_i) \begin{pmatrix} dy_i \\ dx_i \\ \gamma_i' \end{pmatrix} \right] \tag{29}$$

with

$$\begin{aligned} \gamma_i' = &(-\sin\gamma \cdot d_i^x - \cos\gamma \cdot d_i^y) \cdot dy_i \\ &+ (\cos\gamma \cdot d_i^x - \sin\gamma \cdot d_i^y) \cdot dx_i \end{aligned} \tag{30}$$

where dy_i and dx_i refer to the map gradients for the i-th scan endpoint.

If a scan endpoint is located in such a way that all surrounding voxels are unoccupied or so far unknown, no gradients can be determined. In this case, the voxel size m_{res} used for determining the interpolated map value and gradients is increased by reducing the search depth n in Equation 24. Since parent nodes in octrees aggregate the information stored in the according child nodes, increasing voxel size neither increases memory nor computational requirements. By using this heuristic the coverage along the x and y as well as the z dimension is increased. Thus, the probability of recovering from mislocalization due to hardware lag or sensor noise is increased, and the probability of getting stuck in a local minima decreased. The downside of this heuristic is that an increased voxel size impairs localization accuracy.

A pseudo code formulation for the whole octomap scan registration is given in Algorithm 3. Here the function *getMapValueGradients* refers to $\nabla M(T \otimes d_i)$. The functions *updateH* and *updateDet* relate respectively to the first and second factor of the multiplication in Equation 29. Using these three functions, ΔT is approximated and subsequently added to T in every iteration.

Algorithm 3 Octomap Scan Registration

Require: Scan endpoints $scan$,
initial pose estimate $pose$,
number of iterations $maxIter$

function MATCH($scan, pose, maxIter$)
  $T \leftarrow pose$
  $n \leftarrow 0$
  for n to $maxIter$ **do**
    $H \leftarrow 0$
    $dtr \leftarrow 0$
    $i \leftarrow 0$
    for i to $scan.size()$ **do**
      $d \leftarrow$ scan.get(i)
      $d \leftarrow d \oplus T$
      $M \leftarrow$ getMapValueGradients(d)
      $H \leftarrow$ updateH(H, M)
      $det \leftarrow$ updateDet(det, M)
    end for
    $T \leftarrow T + (H^{-1} * det)$
  end for
  return T
end function

EXPERIMENTS

In this section we present simulated and real robot experiments, and the corresponding results in order to evaluate the performance of OctoSLAM.

For simulated experiments we use Gazebo [9], a simulator capable of simulating articulated robots in three dimensional environments. It generates realistic sensor feedback and supports various simulated sensors, such as laser range finders, IMUs, and ultrasonic sensors. We pollute sensor readings with noise generated by the Gaussian probability distribution with mean 0 and standard deviation 0.01. We test OctoSLAM in three simulated environments with different degrees of variation over the z axis. To measure performance, the root mean squared error (RMSE) of the euclidean distance between localized and true pose is averaged over all trials. Every trial consist of recording the sensor data of a manually controlled flight with an approximate duration of two minutes. Hector and OctoSLAM are then applied to the recorded data. The latter is executed with both 2D and 3D maps and with treshold and constant mapping. Threshold mapping refers to updating the map only if the pose has changed by a certain amount and is commonly employed when using 2D maps. Constant mapping in contrast refers to updating the map with every obtained sensor reading, which is necessary to build a 3D map from 2D laser range finder data. The RMSEs for all trials/approaches are plotted in a bar graph. In this graph the central line within each bar marks the mean. The inner box around the mean represents the 95% confidence intervals for the mean. Therefore, non-overlapping confidence intervals indicate a significant difference for a p-value of 5%. The outer box marks one standard deviation.

For real robot experiments we use a *Mikrokopter Oktokopter XL*. We modify the stock version with a *Hokuyo URG-04LX* laser range finder, a downward facing *Parallax PING* ultrasonic sensor and a *Pololu MiniIMU-9* inertial measurement unit. As the true pose is not available to us in the real robot experiments, no quantitative analysis of the results can be given. Instead we perform a qualitative analysis on maps generated by 2D and 3D map SLAM.

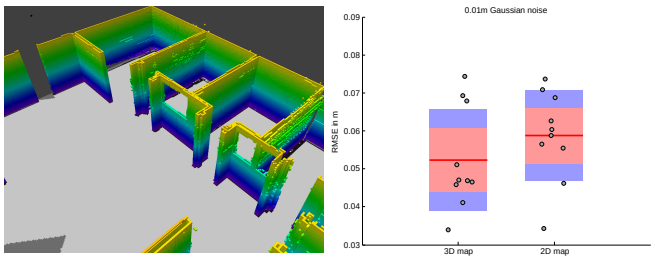

Fig. 2. On the left, a map for the first environment is shown. On the right the results for localization using the 3D map shown here, and for using a 2D down-projection of it, are given.

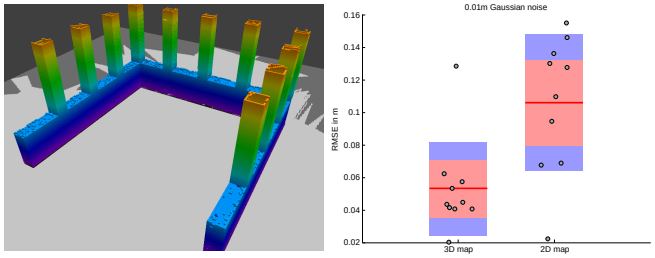

Fig. 3. On the left, a map for the second environment is shown. On the right the results for localization using the 3D map shown here, and for using a 2D down-projection of it, are given.

Localization

To evaluate the localization performance using 3D maps compared to using 2D maps we generate 0.05m resulution maps of the simulated environments using the true pose. Those maps are then used for localization without mapping.

The results for the first environment are given in Figure 2. On the left the map used for localization, and on the right the results for 3D and 2D map OctoSLAM, are shown. Using a 3D map yields an approximately 11% lower average RMSE. However, as can be seen in the figure, the 95% confidence intervals overlap. Hence, using localization on a 3D map does not give a significant advantage over using a 2D map, in this environment. As this map mostly consists of orthogonal walls, one would intuitively expect this result. Furthermore, because the voxel size of the octomap is 0.05m, we consider a localization RMSE of 0.05m to 0.06m to be very accurate, as the representation resolution itself can introduce inaccuracies in this range.

Figure 3 shows the second environment, which introduces slightly more variation over the vertical axis. Similar to the first one, the RMSE for 3D map localization ($\sim$0.05m) is approximately equal

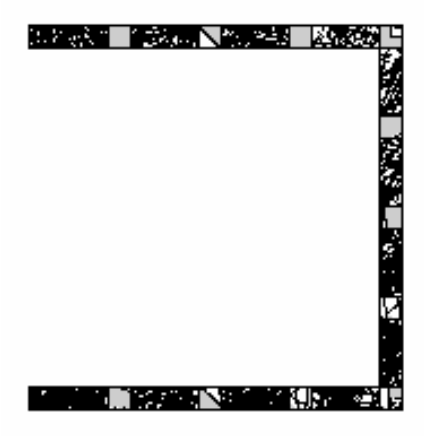

Fig. 4. This figure shows the second simulated world map with voxel size 0.05m down-projected to 2D.

to the voxel size and can be considered a good result. Using a 3D map yields an around 51% lower average RMSE. The confidence intervals do not overlap, i.e. using a 3D map for localization yields significantly better results than using a 2D map. This result can be explained by the fact that a 2D map makes it hard to distinguish whether a scan hits a pillar or the top of the lower wall. To illustrate the issue, Figure 4 shows a 2D map of this environment.

The third environment and the corresponding results are given in Figure 5. As in all previous environments, the absolute RMSE for both 2D and 3D maps is again in the same order as the voxel size. However, 3D map localization yields an approximately 51% lower RMSE than 2D map localization. As visualized in the figure, this is a significant improvement. This improvement results from the fact that if facing the tilted wall in the top right, 3D map localization can determine the

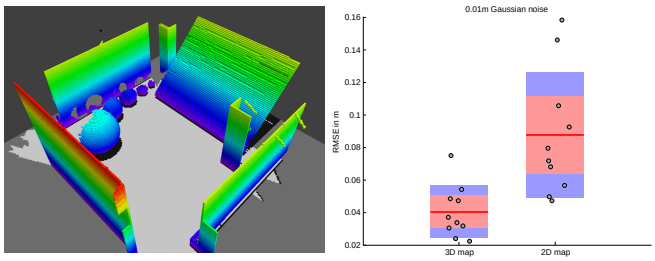

Fig. 5. On the left, a map for the third environment is shown. On the right the results for localization using the 3D map shown here, and for using a 2D down-projection of it, are given.

position reliably as it can utilize the height of scan endpoints. 2D map localization on the other hand can not benefit from the height information of the scan endpoint, and hence can not extract reliable map values corresponding to the tilted wall.

SLAM

The results for simultaneous localiztion and mapping in the first environment are given in Figure 6. Furthermore, the 3D 0.05m^3 voxel size octomap generated by the median trial is also provided. While both OctoSLAM 2D A and B perform better than Octo SLAM 3D and Hector SLAM, the difference is not significant. While 3D map OctoSLAM and Hector SLAM have a similar RMSE, 0.090m and 0.087m respectively, Hector SLAM has double the standard deviation. Hence, Octo slam seems to perform more reliably. However, seeing that the RMSE is below 0.10m for all SLAM system, we consider them all adequate for the Willow world. The trial used to generate the 3D map has a RMSE of 0.079m. Unfortunately, there is quite some clutter along the walls. However, even using ground truth data for mapping (Figure 2) instead of localization does not produce a very clean map of this environment.

Figure 7 visualizes the results for SLAM in the second environment. Similar to the localization results for this environment, 3D map Octo SLAM significantly outperforms the other SLAM approaches. It scores an average RMSE of 0.11m, while other approaches feature at least double that RMSE. 3D map OctoSLAM also has the lowest standard deviation of 0.038m, while the tested 2D map approaches have one at least three times as big. The octomap shown is generated by a trial with a RMSE of 0.109m. The walls in this map are rather clean, and the pillars are clearly visible and distinguishable from the lower wall.

Figure 8 shows the third environment and the corresponding results. OctoSLAM 3D significantly outperforms the other three SLAM approaches with a RMSE of 0.133m compared to RMSEs ranging from 0.260m to 0.844m respectively. Hector SLAM scores the worst RMSE average and also has the highest standard deviation. The trial used for mapping has a RMSE of 0.146m.

The real robot test environment, as well as the corresponding octomaps, are shown in Figure 9.

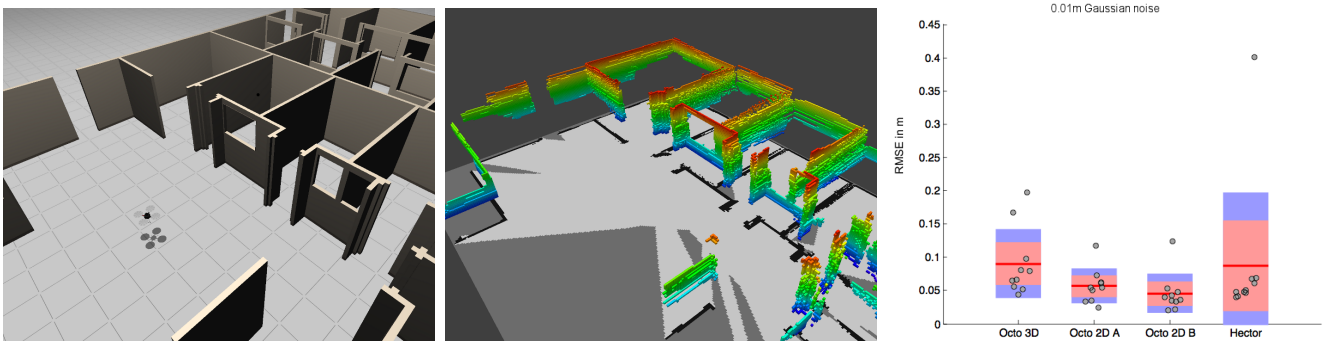

Fig. 6. On the left the first environment is shown. In the middle the 3D map generated by the OctoSLAM 3D median trial can be seen. On the right the RMSEs for SLAM in this evironment are given. (A) refers to threshold mapping, (B) to constant mapping.

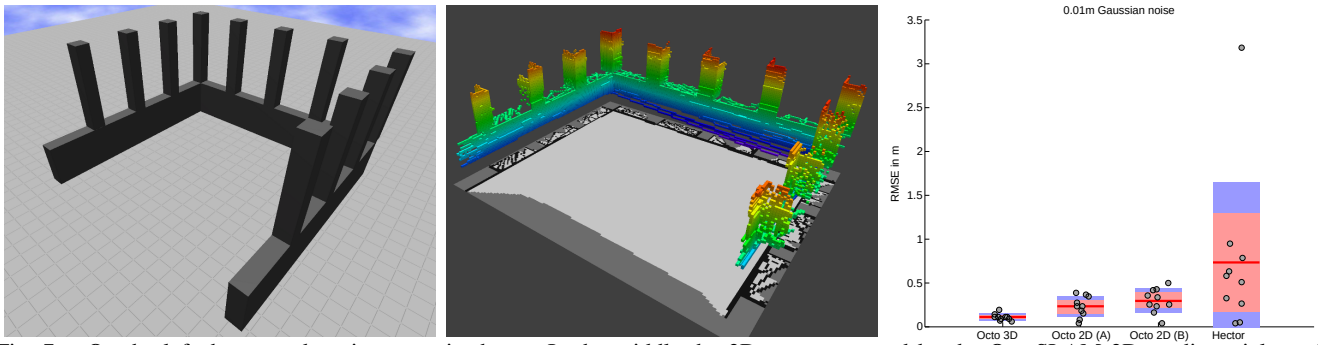

Fig. 7. On the left the second environment is shown. In the middle the 3D map generated by the OctoSLAM 3D median trial can be seen. On the right the RMSEs for SLAM in this evironment are given. (A) refers to threshold mapping, (B) to constant mapping.

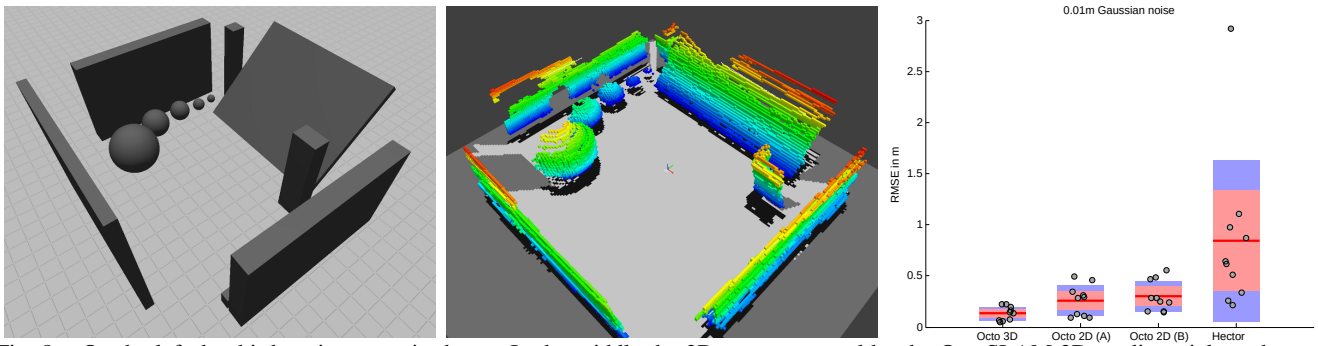

Fig. 8. On the left the third environment is shown. In the middle the 3D map generated by the OctoSLAM 3D median trial can be seen. On the right the RMSEs for SLAM in this evironment are given. (A) refers to threshold mapping, (B) to constant mapping.

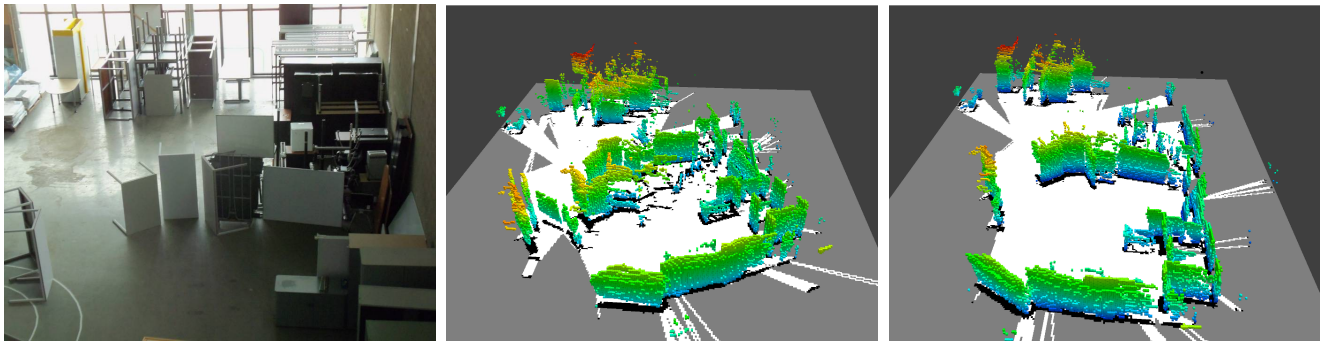

Fig. 9. This figure shows a photograph of the real robot test environment on the left. In the middle a 3D map recorded using 2D map OctoSLAM is shown. On the right a 3D map resulting from 3D map OctoSLAM is given.

In the middle, the result for 2D map OctoSLAM is shown. For better comparability to 3D map OctoSLAM the map was created in 3D, but the localization is performed on a down-projection of that map. As can be seen, there are several severe errors in the map. A good indicator for the low performance is that the right wall has been mapped twice. Partly vertical as it is in the real world, and partly rotated counter clockwise. Additionally, there is quite some offset both vertically and horizontally, and the wall seperating the upper and lower parts of the environment is mapped very inaccurately. The resulting map for 3D map OctoSLAM is given in the right part of Figure 9. While 2D map SLAM has failed for this trial, using a 3D map returns a feasible result. Features like table legs in the center can be recognized. The wall to the right is also approximately vertical as in the real world. There is however some horizontal offset, as the lower part seems to be shifted slightly to the left. Nevertheless, this result shows that in certain environments using a 3D map for localization outperforms using a 2D map, making the difference between SLAM beeing feasible or infeasible.

CONCLUSIONS

We have presented OctoSLAM, a novel situational awareness approach based on Octomap mapping and Hector SLAM. OctoSLAM fuses various sensor data, in order to allow for indoor 3D mapping from 2D planar scans and simultaneously performing localization. This requires the use of heuristics which introduce inaccuracy to the localization, which in return decreases the quality of the generated map. Nevertheless, the conducted experiments show that using 3D instead of 2D maps can significantly improve the SLAM performance in the UAV domain. In simulation, we found that in environments which tend not to differ over height, 2D map OctoSLAM seems to perform better than 3D map OctoSLAM. However, 2D map OctoSLAM it did not significantly outperform 3D map OctoSLAM. In contrast, if the environment does not mostly consist of straight walls, 3D map OctoSLAM did perform significantly better than 2D map OctoSLAM. The real robot experiments, while not evaluated quantitatively, reinforce these findings. We have not tested OctoSLAM outdoors, but we expect a similar performance if sufficient features are in range (e.g. in urban environments). In case of deployment on open fields however, we expect OctoSLAM to be infeasible. Overall, the presented experimental results clearly demonstrate the effectiveness and applicability of 3D map OctoSLAM in the indoor UAV domain. And while only a certain degree of SLAM accuracy is required for many UAV tasks, the increased performance gained by using 3D maps can enable SLAM in environments where regular 2D map methods fail. To that extent we believe that further research into 2D laser range finder based 3D map SLAM is justified. Especially the combination with visual approaches, e.g. optical flow odometry, could help to overcome the limitations of using 2D sensors for 3D map SLAM.

REFERENCES

[1] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D SLAM system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012.

[2] I. Dryanovski, W. Morris, and J. Xiao, "An open-source pose estimation system for micro-air vehicles," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[3] G. Grisetti, C. Stachniss, and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005.

[4] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *Proceedings of IEEE International Symposium on Safety, Security and Rescue Robotics*, 2011.

[5] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010.

[6] H. Moravec and A. E. Elfes, "High Resolution Maps from Wide Angle Sonar," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1985.

[7] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of Octrees for Efficient 3D Mapping," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[8] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[9] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proceedings of the IEEE/RSJ International Conderence on Intelligent Robots and Systems*, vol. 3, 2004, pp. 2149–2154.