Regular Article

Robotic Precision Harvesting: Mapping, Localization, Planning and Control for a Legged Tree Harvester

Edo Jelavic, Dominic Jud, Pascal Egli and Marco Hutter
Robotic Systems Lab, ETH Zurich, 8092 Zurich, Switzerland

Abstract: This paper presents a system for autonomously conducting precision harvesting missions using a legged harvester. Precision tree harvesting removes some trees selectively, while leaving neighboring trees intact. Our robot performs the challenging task of navigation and tree grabbing in a confined, GPS-denied forest environment. We propose strategies for mapping, localization, planning, and control and integrate them into a fully autonomous system. The mission starts with a human mapping the area of interest using a detachable, custom sensor module. Subsequently, a human expert selects the specific trees for harvesting. The sensor module is then mounted on the machine and used for localization within the created map. A planning algorithm searches for both an approach-pose and a path in a single path planning problem. We design a path-following controller exploiting the legged harvester's capabilities for negotiating rough terrain. Upon reaching the approach-pose, the machine grabs a tree with a general-purpose gripper. Our system has been tested in both emulated and natural forest settings. To the best of our knowledge, ours is the first robot to demonstrate such a level of autonomy on a full-size, hydraulic machine operating in a realistic environment.

Keywords: forestry, automation, autonomous robot, autonomous navigation, tree harvesting

1. Introduction

Efficient forest management is of interest to all humankind. Forests cover roughly 30% of the world's land surface [United Nations Food and Agricultural Organization, 2018]. Trees provide the human population with renewable raw materials and a significant source of both energy and oxygen. Besides, woodlands provide a habitat for animal wildlife and contribute to the fight against climate change.

The forestry industry contributes 1.2% of the global Gross Domestic Product (GDP) and employs about 30-45 million people [Renner et al., 2008]. In some countries forestry products have a significant share in the total value of exported goods, e.g., Finland (18%) and Latvia (16%) [Swedish Forest Agency, 2014]. Given the growing labor shortage [Hawkinson, 2017] and the classification of tree

Received: 16 April 2021; revised: 8 October 2021; accepted: 8 October 2021; published: 27 June 2022.

Correspondence: Edo Jelavic, Robotic Systems Lab, ETH Zurich, 8092 Zurich, Switzerland,
Email: edo.jelavic@mavt.ethz.ch, alternative email: edo.jelavic@gmail.com

DOI: https://doi.org/10.55417/fr.2022046

(a) Komatsu-895 forwarder **(b)** Komatsu-951 harvester

Figure 1. Different machines are typically used in modern forestry operations. They all have operator cabins to increase the comfort level in a possibly wet and muddy forest environment. Images were taken from [Komatsu Corporation, 2019].

(a) RCM Harveri harvester **(b)** eBeaver harvester

Figure 2. Smaller harvesters that are typically used for thinning operations. They do not have a cabin for the operator and are remotely controlled within the operator's line of sight. Images taken from [Jukka Hämekoski, 2016] and [Magnus Gustavsson, 2016].

harvesting as a *3D* job (dirty, difficult, and dangerous), automating forestry operations deserves a high economic priority.

1.1. Automation in Forestry

Forestry mechanization has increased productivity to the point that human operators have now become the critical bottleneck [Parker et al., 2016]. Since modern harvesting machines have many Degrees of Freedoms (DoFs), training efficient operators takes time and slows further productivity gains. To mitigate the problem, researchers have developed semi-autonomous modules employing Inverse Kinematics (IK) control for crane operation, such as those in Figure 1a and Figure 1b. Examples of such work can be found in [Westerberg, 2014; Ortiz Morales et al., 2014; Hyyti et al., 2018; Hellström et al., 2008].

Another common forestry task is thinning, a process wherein stewards selectively remove some trees to allow more space for others [Forestry Focus, 2018]. Thinning requires negotiating tight spaces (as opposed to conventional clear-cutting) and can be done with smaller machines that are remotely controlled within the operator's field of view. Figure 2 shoves two examples, Harveri and eBeaver. Despite their lower cost and reduced damage to the forest ground compared with large harvesters, machines such as Harveri are not very popular, since operators are reluctant to give up a

cabin's comfort and safety (personal communication with [Silvere, 2017]). Our system is developed to be suitable for both conventional clear-cutting and thinning.

1.2. Precision Forestry

Traditional forestry operations are still primarily based on fundamentals developed about 300 years ago [Von Carlowitz and von Rohr, 1732]. Nowadays, the trend is to move towards precision forestry, which practice uses automated data collection (versus manual measurement) and software-aided analysis to allow site- and tree-specific husbandry. Instead of compartment-based management relying on human experience and qualitative judgment, massive digital data enables much more granular and quantitative forest management. With the newest sensing technologies (e.g. Light Detection And Ranging (LIDAR)), precision forestry can even be done on the single tree level [Choudhry and O'Kelly, 2018], [Holopainen et al., 2014]. Compared to the traditional techniques, precision forestry diminishes the error in inventory estimates by 400 percent [Choudhry and O'Kelly, 2018]. Other benefits such as granular fertilizing, fire monitoring, pest and disease monitoring increase yields and labor productivity up to 10 times [Silvere, 2017].

1.3. Scope

This article describes a robotic system, comprising modified commercial hardware and custom software, designed specifically for tree-grabbing. We focus on automating the workflow until the moment of cutting the tree. Cutting and debranching are not investigated since there are existing harvesting tools for this process. We focus on executing an autonomous mission (referred to as harvesting in further text) and main aspects (other than cutting) that executing such a task entails, namely mapping, localization, planning, control, and tool positioning. Autonomy modules developed in this work are not task specific and could be used for either harvesting or thinning.

To this end, we develop a versatile sensor module for collecting data and localizing the machine. We present our hardware platform, an automated Menzi Muck M545 harvester [Jud et al., 2021] augmented with custom sensors and actuators (see Figure 3), and describe our technical approach to executing the critical, initial phase of a harvesting mission as specified by a human or algorithmic expert. Lastly, we present experimental results emulating a real harvesting mission in a forest and report our recommendations for integrating and tuning the system.

Figure 3. Legged excavator navigating to a tree and performing a grab. Human is in the cabin for safety reasons.

High-level decisions on which trees to cut and how to manage the forest inventory are beyond this article's scope. Nowadays, companies can create forest inventories and recommend future actions with aid from machine intelligence (e.g., [Silvere, 2017]), and our pipeline assumes using such a service.

1.4. Contribution

To our knowledge, we here present the first report of successfully deploying a large-scale, fully autonomous system for forest husbandry. Our approach enables operations under the forest canopy, which are significantly harder than clear cutting since the machine has to navigate among trees where Global Positioning System (GPS) signal may be unreliable. In summary, our work extends state of the art with the following contributions:

- Design of a portable sensor module with a sensor setup that enables mapping and localization. We test our sensor module in handheld operation, and we deploy it on a robotic platform.
- Development of a robust algorithm for converting raw point clouds into 2.5D elevation maps. We demonstrated the algorithm's applicability in a real forest and in several other environments. Our implementation is available as open-source[1].
- We develop an approach-pose planning algorithm suited for tight spaces and both structured and unstructured environments. The algorithm is evaluated on synthetic data and maps produced from real data in a planning scenario including large-scale harvesters.
- Development of a control procedure for driving a legged harvester in rough terrain. The algorithm performs path tracking and chassis stabilization at the same time.
- Both planning and path tracking algorithm implementations have been made publicly available for the community[2].
- We develop a lightweight method for tree detection based on LIDAR scans of local forest patches. Our method is purely geometry-based and operates directly on point clouds. It is suitable for online operation on the robot and we make it available for the community [3]
- We integrate all system components into a fully functioning autonomous system. Experimental verification of our pipeline is done on a full-size legged harvester in a realistic environment
- An evaluation of system components in the real-world scenario is presented. The evaluation includes chassis control, tree detection, approach-pose planning, mapping, localizing, and point-cloud to elevation-map conversion.

The rest of the paper is organized as follows: Chapter 2 introduces the related work. Chapter 3 describes the robotic platform and hardware used in the experiments. In Chapter 4, we give a high-level overview of the proposed harvesting system with brief explanations for each of the submodules. Chapter 5 describes mapping and localization, while Chapter 6 describes tree detection procedure. Control and planning are described in Chapters 7 and 8, respectively. Chapter 9 presents results and evaluation. Finally, Chapter 10 summarizes the paper and Chapter 11 discusses conclusions and future work.

2. Related Work

In this section, we give a brief overview of the work relevant for precision harvesting missions; a more in-depth survey on forestry robotics can be found in [Oliveira et al., 2021]. To the best of our four knowledge, no autonomous system for thinning or harvesting missions has been presented in the literature so far. The harvester presented in [Rossmann et al., 2009] is the closest to our work in the

[1] https://github.com/ANYbotics/grid_map/tree/master/grid_map_pcl
[2] https://github.com/leggedrobotics/se2_navigation
[3] https://github.com/leggedrobotics/tree_detection

sense that developed methods are showcased on a full-size machine. The authors develop a particle filter-based localization for forest environments; however, no attempt to automate other machine parts was made. Similarly, in [Li et al., 2020], a full-size harvester is fitted with a 3D LIDAR the authors develop a place recognition algorithm based on tree stems.

To acquire an overview about the forest and plan for harvesting, maps are typically first acquired by airborne surveying [Naesset, 1997], [Roßmann et al., 2009]. For mapping and localization, LIDAR sensors are a popular choice. GPS based localization can be inaccurate under a tree canopy, and vision-based sensors (cameras) can be sensitive to illumination changes in outdoor environments. For LIDARs, Iterative closest point (ICP) is a popular method for scan registration and has been used for mapping in forest surveying [Morita et al., 2018; Yue et al., 2018; Tsubouchi et al., 2014]. In [Babin et al., 2019], ICP is fused with GPS for mapping subarctic forests with sparse canopy. The ICP is also used for mapping in [Tremblay et al., 2020] and the authors propose a method for determining Diameter at Breast Height (DBH). Another possibility is to use filtering with trees as landmarks. [Roßmann et al., 2009; Rossmann et al., 2010] build a map using an airborne laser sensor and localize a harvester in *SE(2)* space using a particle filter. In [Miettinen et al., 2007], an Extended Kalman Filter (EKF) based Simultaneous Localization and Mapping (SLAM) is used on a skid steer robot to build a map and localize within it. The downside of relying on tree landmarks is that it can be susceptible to false positives.

However, LIDAR mapping does not have to rely on tree landmarks. Thus, mapping algorithms without tree stem detection can potentially handle the most general forest types (sparse, dense, thick vegetation). Many researchers have used LOAM ([Zhang and Singh, 2014; Zhang and Singh, 2015]) for LIDAR mapping, and numerous variants of LOAM have been tested in urban environments. LOAM is a LIDAR odometry and accumulates drift which may be problematic for large areas. Recently, loop closing has been added to LOAM ([Shan and Englot, 2018; Shan et al., 2020]), which enabled the algorithm to correct for the accumulated drift. The loop closure mechanism is based on ICP to map matching and cannot handle large drift. In [Chen et al., 2020; Nevalainen et al., 2020] LOAM has also been used in a forest. An alternative to LOAM is Cartographer [Hess et al., 2016] which has been shown to work well for large-scale mapping. We use Cartographer as a mapping algorithm of choice.

Once in possession of a map, one typically wants to estimate the biomass or classify the tree species, which requires segmentation of both the canopy and the stems. The forestry community has extensively studied techniques for tree segmentation and classification. Example of model-based approaches include [Zhang et al., 2019b], [Burt et al., 2019]. Model-based approaches typically rely on pointcloud processing such as euclidean clustering, surface normal computation, and RANSAC model fitting to segment out the trees. On the other hand, learning-based approaches train networks end to end to segment trees from point clouds [Ayrey et al., 2017], [Chen et al., 2021], [Bryson, 2017]. The need to accurately segment out whole trees in cluttered scenes renders most of the approaches complicated and with many steps. Hence, all algorithms are designed for offline operation, and some require powerful computational resources. In this work, a lightweight model-based approach operating online on the robot is used for tree detection.

A harvester can be treated as a big mobile manipulator. Planning and control for mobile manipulators is a well-studied problem in robotics. One can either treat the robot in a whole-body fashion [Kim et al., 2019; Giftthaler et al., 2017; Gawel et al., 2019] or decouple the planning and control for the arm and the base [Carius et al., 2018; Schwarz et al., 2017]. The latter approach has often been used for forestry automation. Controlling the harvester's (or forwarder's) crane is an integral part of forestry operations. Examples can be found in [Lindroos et al., 2015; Westerberg, 2014; La Hera et al., 2009; Kalmari et al., 2014]. A major effort was put towards semi-automating the crane operation since coordinating many DoFs is one of the hardest tasks for a human operator. Most crane control algorithms are based on IK or Inverse Dynamics (ID) [Siciliano et al., 2010]. Compared to classical robotic manipulators, the biggest difference is that harvester cranes were seldom designed with autonomous operations in mind. They often come without sensing capabilities which means they have to be retrofitted with sensors to estimate the end-effector position. Moreover,

the sensors have to be robust to withstand operation outdoors—such retrofitting results in increased automation effort. An additional difficulty is that hydraulic actuators are harder to control than their electric motors counterparts (nonlinearities, valve overlap, less bandwidth). To minimize the automation effort, [Morales et al., 2011; Ortiz Morales et al., 2014] investigate possibilities for purely open-loop control. More recently, an attempt has been made to develop arm motion planning for a feller-buncher machine [Song and Sharf, 2020], [Song and Sharf, 2021]. The authors rely on Zero Moment Point (ZMP) to compute a stable trajectory guaranteeing the machine's (tipping over) stability. The stability of the harvester becomes essential as soon as one uses an actuated end-effector for tree manipulation (as opposed to passive tools that let trees fall freely).

Little has been done to develop a fully autonomous system for forest environment missions; the robotic community has mostly focused on urban environments. [Mikhaylov and Lositskii, 2018] shows a very simplistic architecture without any validation. In [Georgsson et al., 2005], a GPS based tracking approach is presented and validated on a forwarder machine outside of a forest environment. [Tominaga et al., 2018] show experiments on an All Terrain Vehicle (ATV) without an arm in a small scale environment using GPS with Real-time Kinematic (RTK) correction. The authors use a global graph-based mission planner and a local state space sampling planner. A pure-pursuit algorithm is used for tracking. The proposed planning and control strategy cannot handle backward driving, limiting its ability to negotiate confined spaces. [Zhang et al., 2019a] presents an integrated system for navigation in a forest. The authors present a planner and a path follower that run on a small skid-steer robot without an arm. They show results in a structured forest (a rubber tree farm) where trees form a grid. This is reflected in the path generation algorithm, which involves heuristics to exploit the environment's structure and generates only straight-line paths.

[Wooden et al., 2010] develop a navigation system for the Big Dog robot; it was tested in a forest, and it features a local planner based on a graph search A* algorithm. [Hellström et al., 2008] discusses different planning algorithms (A*, elastic bands, potential fields) for forwarder machines; however, no results are shown since it is a pre-study only. More examples of forest navigation using graph search algorithms to compute paths can be found in [Tanaka et al., 2017; Mowshowitz et al., 2018]. Compared to the Unmanned Ground Vehicles (UGVs), much more work has been done for Unmanned Aerial Vehicle (UAV) path planning in forests, and examples can be found in, e.g., [Cui et al., 2014; Liao et al., 2016; Pizetta et al., 2018]. There is a research gap for UGV path planning in natural environments, which this article tries to bridge. We compute plans for a non-holonomic constrained vehicle using RRT* [Karaman and Frazzoli, 2011].

3. Hardware

The Hardware section introduces the robot used and it describes the sensor module we developed for the autonomous missions.

3.1. Platform

Our robotic platform is Hydraulic Excavator for Autonomous Purpose (HEAP). It is based on a Menzi Muck M545 multi-purpose legged machine often used for harvesting in challenging terrain. It is customized for teleoperation and fully autonomous operations. Besides forestry work HEAP, can be used for digging, landscaping and manipulation tasks as well (see [Jud et al., 2019], [Johns et al., 2020]) . We use the terms HEAP and harvester interchangeably in further text. Our machine is fitted with custom hydraulic actuators with pressure sensors and a high-performance servo valve. The actuators allow for precise force and position control, thus enabling active chassis balancing and adaptation to the uneven ground (see [Hutter et al., 2016]).

Proprioceptive Sensing: Two SBG Ellipse2-A Inertial Measurement Unit (IMU)'s (one in the cabin and one in the chassis) gather the inertial data that are primarily used to determine the chassis' roll and pitch angle. HEAP is equipped with a series of IMUs rigidly attached to each arm link. Measurements from these IMUs are fused together to estimate end-effector pose and joint angles

(a) Front sensors on the cabin **(b)** Localization sensor module in the back

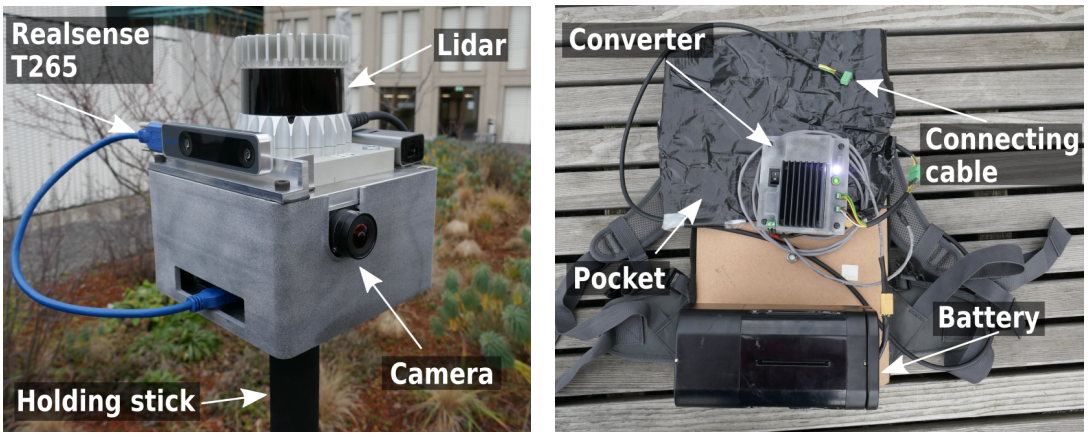
Figure 4. ***Left:*** two LIDAR sensors mounted on top of the cabin and used for scanning the area in front of the machine. They are primarily used for tree detection. ***Right:*** The localization module introduced in Sec. 3.2 mounted in the back of the machine.

(a) Sensor module mounts **(b)** Sensor module packed

Figure 5. ***Left:*** Close-up photo of sensor module mounts. The module is mounted onto the aluminum profiles (shown with arrows), clamped to the machine using the o-ring clamps (encircled with dotted line). The lever fasteners are used to adjust the angle of the module (encircled with full line). ***Right:*** The sensor module dismantled and packed after deployment.

in the machine frame. Note that, unlike in [Jud et al., 2019], we do not use externally mounted draw-wire encoders on the arm. This way, we avoid a potential entanglement of draw wires with tree branches.

Exteroceptive Sensing: Two Velodyne Puck VLP-16 LIDAR's are used for tree detection. It is important to note that one of the Pucks is rotated by 90° around the rolling axis (see Figure 4a) in order to get a better resolution when mapping the area in front of the machine. A sensor module (see Sec. 3.2) is mounted on the back of the machine, as shown in Figure 4b. A close-up image of the sensor mounts is shown in Figure 5a. The sensor module is mounted using aluminum profiles from *item*[item, 2021], and lever hinges from the same company. We attach it to the machine using o-ring clamps with rubber to mitigate the effect of vibrations. The mounts are rigid; hence the whole



(a) Sensor module **(b)** Backpack with power

Figure 6. ***Left:*** Head of the sensor module. The Central Processing Unit (CPU) and the IMU are inside the grey box. ***Right:*** Power supply for the sensor module mounted on the backpack.

module does not move w.r.t to the cabin frame. Note that the module is not mounted in the middle to reduce further engine vibrations (the engine is just below the sensor module).

Trees can snap when manipulated, resulting in heavy debris falling on the machine. Hence, for a tree felling application (as opposed to tree grabbing), one needs to make the sensor module rugged and add mechanical protection (e.g., bulletproof glass). Another option would be to place the sensors directly inside the cabin. It is important to place the module such that it has as large Field of View (FOV) as possible which is beneficial for the localization. Alternatively, one can use multiple synchronized sensors, each with a smaller FOV. For HEAP, the sensor module placement in the back as seen in Figure 4b has an effective FOV of about 180° (instead of 360°), which was enough for localizing.

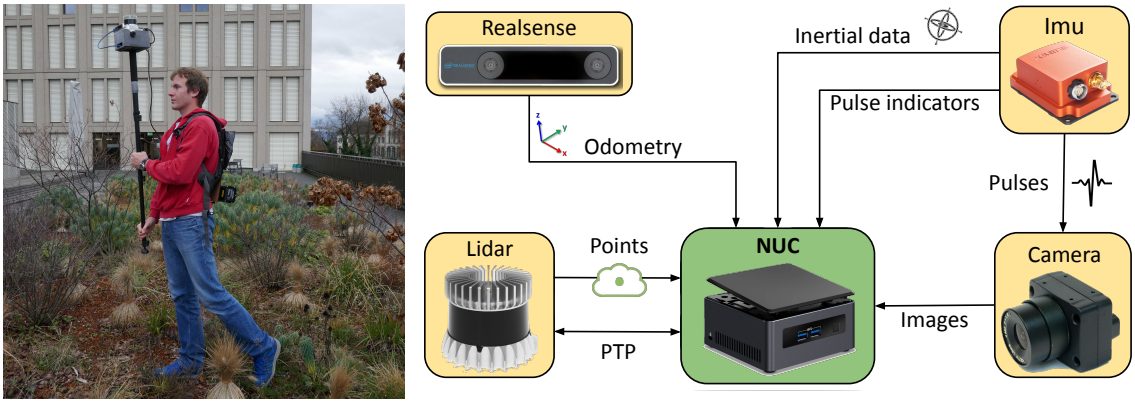
The planning, control, and tree detection software stack runs on one computer (Intel i7-5820K, 6x3.60GHz, Ubuntu 18.04, 32 GB RAM) installed in the cabin. The control loops work at 100 Hz and are triggered by the Controller Area Network (CAN) driver. All the algorithms presented are implemented using C++ with Robot Operating System (ROS) as integration middleware. For more details on HEAP, please refer to [Jud et al., 2021].

3.2. Sensor Module

We develop a sensor module for mapping and localization as displayed in Figure 6. The module can be used in the first step by a human to map the area of interest (see Figure 7a). In a second step, the system can be mounted on HEAP (Figure 4b) to localize the machine in the previously built map without GPS information. The whole module can be dismantled and packed, as shown in Figure 5b.

We design the sensor module to be lightweight such that handheld operation is possible. It has an integrated computer such that mission data can be collected and saved without an external laptop. Besides, one can run the SLAM directly on the module in real-time. The module has a LIDAR, an IMU, and two visual sensors. LIDAR is used as a primary sensing modality with aid from visual and inertial sensors. This way, the motion distortion in the point cloud can be corrected. Extrinsic calibration between the sensors is obtained from *Kalibr*[4] [Furgale et al., 2013] and *lidar_align*[5], both of which are available open-source. Lastly, all sensors installed on the module are time-synchronized. Time synchronization is essential for state estimation and mapping algorithms to function correctly.

[4] https://github.com/ethz-asl/kalibr
[5] https://github.com/ethz-asl/lidar_align



(a) Sensor module in action **(b)** Sensor module schematic

Figure 7. ***Left:***Human operator mapping an area of interest. ***Right:*** Schematic of the sensor module. It is composed of a camera, tracking camera (realsense), LIDAR and an IMU.

The sensor module in operation is shown in Figure 7a and sensor components in Figure 6a. We use an Ouster OS-1 LIDAR with 64 channels, an Intel Realsense T265 tracking camera, and a FLIR camera model BFS-U3-04S2M-CS. Inside the grey box in Figure 6a, one can find a computing unit (Intel NUC with i7-8650U processor) and an IMU (Xsens MTI-100). Total cost of our sensor module is about 10500 USD. We implemented two complementary solutions for Visual Inertial Odometry (VIO): FLIR camera synced with IMU and the Intel Realsense T265 tracking camera. The FLIR camera has an excellent low light performance, while the T265 has almost 180° FOV and higher frequency. Hence, one can choose the VIO sensor used depending on the application. In this work we use T265 because of large FOV and because higher frequency odometry estimates were beneficial for the LIDAR based mapping.

The sensor module can be mounted on a stick shown in Figure 6a. In this way, a human can walk around with the module and point it to the places of interest to map it in sufficient detail. The sensor module is powered from the backpack (shown in Figure 6b), equipped with a 36 V battery. We choose to use a 300 W h e-bike lithium-ion battery (BiX Power BX3632H) with an integrated battery management system. All smaller sensors like cameras and the IMU are powered through Universal Serial Bus (USB).

The schematic of the sensor module is shown in Figure 7b. The Intel Realsense camera provides software-synchronized VIO at a frequency of 200 Hz. The FLIR camera (20Hz) and the IMU (400 Hz) are synced using customized software. The LIDAR is connected directly to the computer, and it is configured to use Precision Time Protocol (PTP) for time synchronization [Eidson, 2006]. We modify the LIDAR's driver to send the packets as soon as they are received instead of batching them until the scan is completed. This way, the motion distortion is mitigated. We use the TICsync package [Harrison and Newman, 2011] to synchronize the IMU with computer's clock. TICsync estimates the bias and the drift between the IMU's internal clock and the computer's clock to recover the exact time when the measurement happened. To increase the robustness, it is beneficial to assign the real-time priority to the IMU driver (minimizes the timestamp jitter).

4. Approach Overview

Figure 8 presents an overview of the autonomous harvesting system and serves as a visual outline for the chapters in this paper. We briefly describe each subsystem and give a more detailed description in their respective chapters.

Mapping and Localization: We use a LIDAR based SLAM together with visual and inertial measurements to correct for scan distortion. The map is built by processing the data offline. At mission time, scans are registered against the existing map to compute the sensor module's pose.

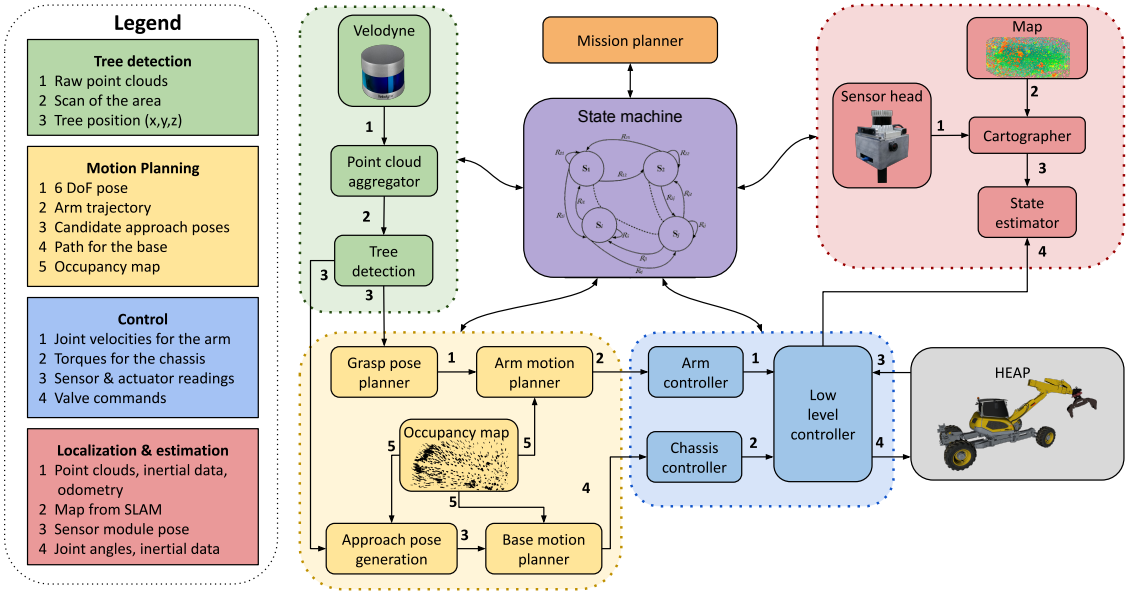


Figure 8. Overview of the system architecture deployed on HEAP. Components belonging to the same subsystem are shown in the same color. Different subsystems are shown in different colors. Arrows depict communication channels in the system, with some communication lines omitted for the sake of brevity.

Tree Detection: The LIDAR sensors mounted on the top of the cabin (see Figure 4a) deliver point clouds at 20 Hz. An intermediate node aggregates them and passes them to the tree detection module. Detection is done purely based on geometric features. The tree detection module forwards the detected tree's coordinates to the grasp pose planner and the approach-pose planner.

Control: The control subsystem is split into two parts. The arm controller ensures trajectory tracking by sending velocity commands to the arm joints (IK controller). The base controller ensures path tracking, and it regulates contact forces such that the base stays upright when driving over uneven terrain. The low-level controller transcribes higher level references (velocity, torque, positions) to the valve commands.

Motion Planning (Arm and Base): The planning stack comprises three planners. The base motion planner plans a path and an approach-pose for the base of the harvester. The grasp pose planner computes a gripper pose in 3D space that encompasses the tree trunk. Finally, the arm motion planner produces a collision-free spline trajectory to move the gripper into the desired pose.

Mission Planner: This module determines which tree to grab next, and it sends the target position to the state machine, which then sends it to the approach-pose planner.

State Estimation: The state estimator uses the sensor module pose and proprioceptive measurements (IMU, joint angles) to compute the complete state of the system (6 DoF base pose + joint angles).

State Machine: The state machine coordinates the execution of different tasks. It works on a handshaking principle where the state machine sends a request to a subsystem, and the subsystem responds with an acknowledgment once the task requested has been completed. Tasks that the state machine can request and the order of operations to grab a single tree are shown in Tab. 1.

5. Mapping and Localization

Precision forestry requires precise and consistent 3D geometric information about the forest. This data can be collected by a harvester itself, a smaller UGV or by a human carrying the sensors. The collected information can then be used for mission planning and forest inventory management. Moreover, the map serves as a reference for the harvester to localize. To retain flexibility, we would

Table 1. Order of operations and subsystems involved to grab a single tree.

Task order	Task Description	Subsystems responsible
1	Get the next tree position.	Mission planner
2	Plan an approach-pose and a path. If it fails, go to step 1.	Planning
3	Drive the machine to the approach-pose found in step 2. In case tracking fails, go to step 2	Control
4	Retract the arm (in case it is not already retracted)	Planning, Control
5	Scan the area around the expected tree location.	Control
6	Run tree detection algorithm and plan a grasp pose	Tree detection, Planning
7	Plan and track an arm trajectory to reach the grasp pose	Planning, Control
8	Plan and track arm trajectory back to default position (arm retracted all the way).	Planning, Control
9	Go to step 1.	State Machine

like to make no assumptions on the surroundings and allow the harvester to work in different types of forests (e.g., sparse, with vegetation, non-flat). Therefore we avoid using tree stems as landmarks (e.g., [Roßmann et al., 2009]) since they can be hard to detect (vegetation) or they may be scarce (e.g., a glade inside the forest).

We selected Google Cartographer [Hess et al., 2016] as the backbone of our mapping and localization pipeline. Cartographer is a grid-based SLAM approach that uses the scan to sub-map matching for loop closure detection and discards unlikely matches using the branch and bound method. At mission time, localization can be achieved simply by scan to sub-map matching. Compared to other SLAM systems available at the time, Cartographer has the advantage that it is fully open source, can cover large spaces, features loop closures that work robustly, and has a large community of users. Below we provide a brief description of the working principle.

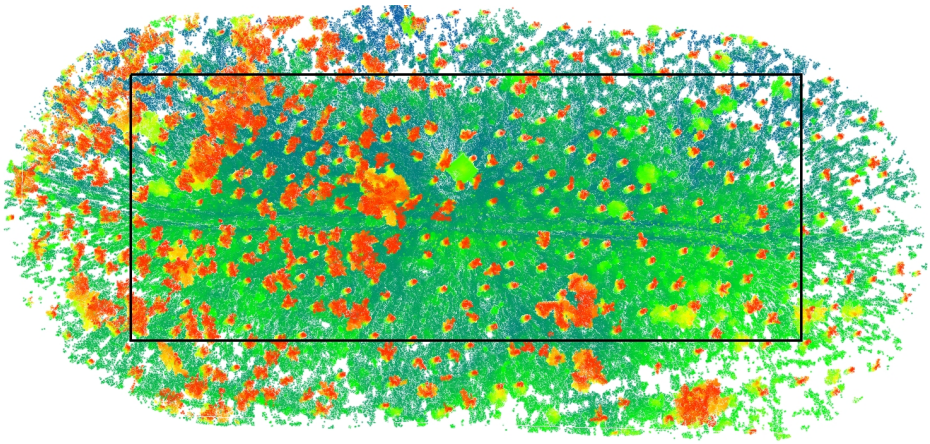
Cartographer divides the world into a raster of submaps where a number of accumulated scans determines the size of the submap. The submaps are grids (3D) where each cell is assigned a probability of being occupied. Before inserting a scan into a submap, the scan is voxelized, and each grid cell inside the scan is classified as *occupied* of *free*. Subsequently, the range scans is registered within the submap by solving a nonlinear least-squares problem which maximizes the probabilities at the scan points in the submap. This local, grid-based SLAM relies on a good initial guess which is achieved by extrapolating the previous pose using the inertial/odometry data.

Cartographer follows the Sparse Pose Adjustment approach of optimizing all scans, and submaps [Konolige et al., 2010]. Each range scan is associated with a trajectory node in the pose graph. In the background, all scans are matched to nearby submaps to create loop closure constraints. Suppose a sufficiently good match (user-defined minimum score) is found in a search window around the currently estimated pose. In that case, it is added as a loop closing constraint to the global optimization problem. The constraint graph of submap and scan poses is periodically optimized in the background (every few seconds). When localizing in a known map, Cartographer keeps only the latest N submaps, which are then considered for loop closures against the submaps in the known map. The known map is not updated.

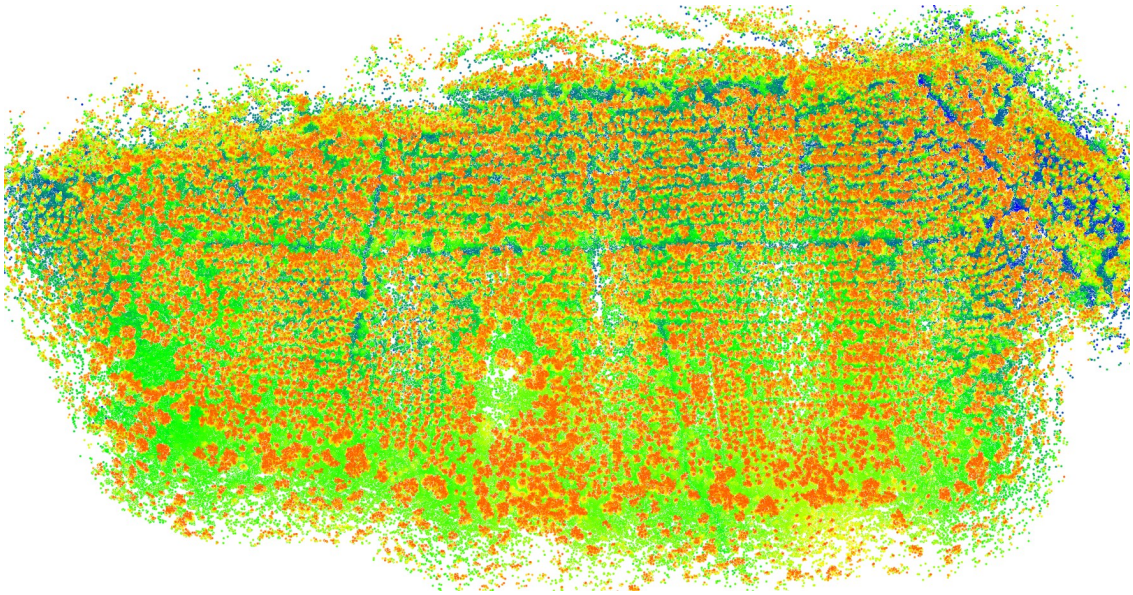
5.1. Mapping

Examples of maps generated with Cartographer are shown in Figure 9.

The map in Figure 9a shows a point cloud of a forest viewed from above. Blue and green colors correspond to a lower elevation (ground), whereas yellow and red colors correspond to a higher elevation (vertical structures like tree trunks and canopy). The map is about 140 m long and 60 m wide, and it shows a part of the forest where we conducted the experiments with HEAP. Another, larger map (340 m x 170 m) is shown in Figure 9b; the size of this map shows that Cartographer can map areas sufficiently large for an autonomous harvesting mission. Both maps have been processed offline and bundle adjusted. Compared to online processing, building the maps offline allows us to



(a) Testing forest patch



(b) Large forest area

Figure 9. Bird view of forest maps. Both maps are represented as point clouds. ***Top:*** Small forest patch where we conducted the experiments. Approximate size 140 m x 60 m The area encircled with a black rectangle was converted to an elevation map (Figure 10). ***Bottom:*** Map of a larger forest area (top view), where range, odometry and inertial data has been collected in multiple tours, concatenated and the processed with Cartographer, data courtesy of [Silvere, 2017]. Approximate map dimensions: 340 m x 170 m.

use more points from the LIDAR which results in denser maps, use finer resolution voxels, and sample for constraints between submaps more often. Furthermore, we can increase the search radius for loop closures and run the scan matcher optimization more often. All the changes above result in less drift and more loop closures, which produce more consistent maps.

Once we have a consistent 3D map of the space, we use it for localization at mission time. Since the planning is done in *SE(2)*, we convert the 3D map into a 2.5D map which the planner then uses. The 3D map is first converted into an elevation map, and from the elevation map, we compute a traversability map. Both maps are functions $f : (x, y) \to \mathbb{R}$ mapping the 2D coordinates to height or traversability. Below, we detail the conversion of the point cloud into an elevation map used by the planner.

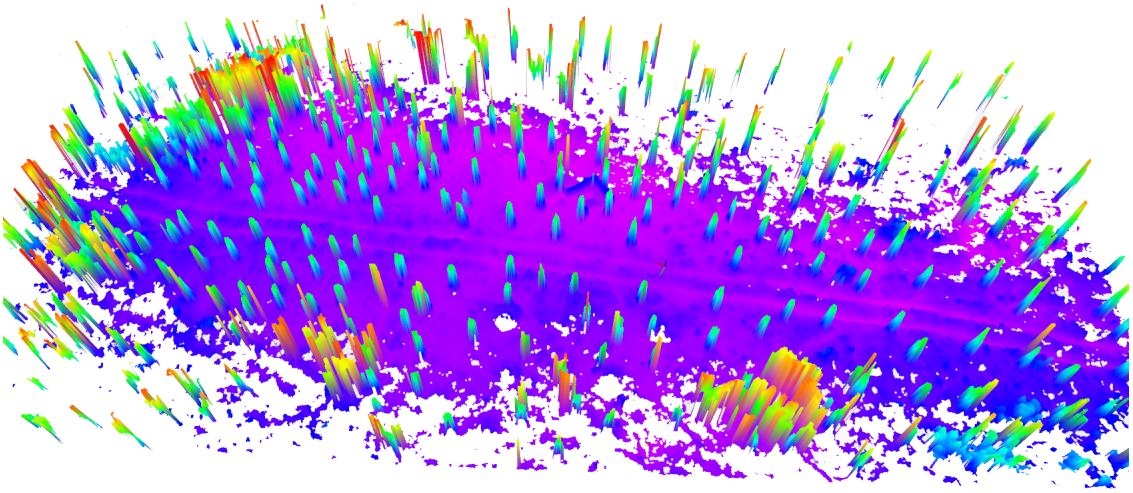


Figure 10. Elevation map of a forest patch encircled black in Figure 9a. Purple color corresponds to the lowest elevation and red to the highest. The resulting grid map size is 1419 by 1318 cells with a resolution of 10 cm. Note how tree trunks are clearly visible in the elevation map.

A grid map data structure [Fankhauser and Hutter, 2016] is used to store the elevation map. From the raw point cloud in Figure 9a, our algorithm builds a 2.5D elevation map of the area shown in Figure 10. Conceptually, the algorithm is similar to the one used in [Fankhauser et al., 2018]. In contrast to [Fankhauser et al., 2018], our algorithm can handle multiple elevations in the cell, i.e., multiple points with the same (x, y) coordinates and different z coordinates. Thus, it can filter out vegetation and clutter and recover true ground elevation more robustly. We successfully used the point cloud to elevation map conversion algorithm in both planar and non-planar environments. Implementation is available as an open-source package[6].

The point cloud to elevation map conversion algorithm starts by removing the outliers and downsampling the input point cloud (implementation based on [Rusu and Cousins, 2011]). The main loop is parallelized to speed up computation. For each cell in the grid map, we pre-compute all the points contained within that cell (this enables lookup in $O(1)$ time), thus vastly speeding up the algorithm. Each cell is a rectangle in x, y coordinates. Once we have fetched the points inside the grid map cell, Euclidean clusters are computed. The cluster size is regulated with a cluster tolerance parameter (see [Rusu, 2010]). Any points withn the tolerance distance are considered the same cluster. After clustering, we calculate the centroid of each cluster. Finally, the centroid with the smallest z coordinate is deemed to be the ground elevation. Illustration of the process is shown in Figure 11. In Figure 11, grid map cells are extended in the z direction and form columns 1,2, and 3. Column 1 has one cluster which contains only the ground points (shown in light green), and the algorithm correctly extracts the ground height. In column 3, the algorithm extracts at least two clusters. The lower cluster is the ground (light green color), while branches and leaves form the upper cluster (dark green). Clusters are disjoint, and the algorithm correctly recovers the ground elevation as a mean value of the lower cluster. Column 2 contains one large cluster composed of both ground and the tree. In this case, we cannot recover ground information since the resolution of the map is too coarse; the height of the tree trunk determines the height. In this work, we use the resolution of 10 cm for the grid. We analyze algorithm's behaviour in presence of vegetation and clutter in the Appendix A.1.

Qualitative runtimes of our algorithm are shown in Table 2. The foreseen use case is to run the algorithm once offline to construct a global elevation map that can be used for planning. The times

[6] https://github.com/ANYbotics/grid_map/tree/master/grid_map_pcl

Table 2. Qualitative runtimes of the point cloud to elevation map conversion. All the conversions were done on the same map with the size of 1419 by 1318 cells at 10 cm resolution.

Point cloud size	less than 10 million points	40-60 million points	100 - 140 million points
Algorithm runtime	1-2 minutes	5-15 minutes	30-60 min

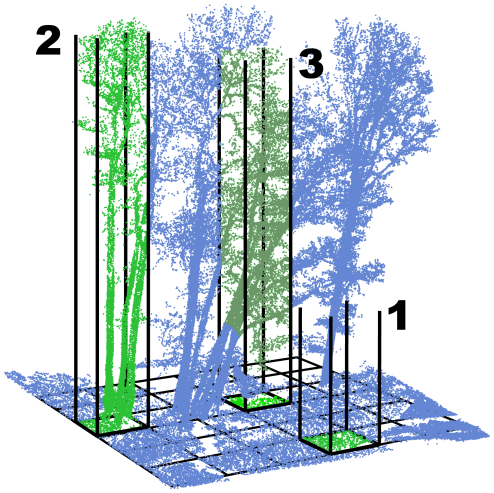


Figure 11. Generating a grid map from a raw point cloud. The grid is shown with black lines, and the input point cloud is shown in blue color. Clusters of points that the conversion algorithm might extract are shown in green. Note that image is not drawn to scale.

shown in Table 2 were obtained on an Intel Xeon E3-1535M (2.9 GHz, 4 cores). The algorithm is not limited to application in forests only but also generalizes to non-forest environments (see https://github.com/ANYbotics/grid_map/tree/master/grid_map_pcl).

5.2. Localization

Apart from mapping, the sensor module from Sec. 3.2 is used to localize HEAP within the map. The localization pipeline consists of components encircled in red color in Figure 8. The sensor module is mounted on the excavator (as shown in Figure 4b). The sensor module's clock is synchronized to the harvester's computer clock using Network Time Protocol (NTP). PTP and NTP run on different networks. We forward all the measurements to the main computer running Google Cartographer in the localization mode (see [Hess, 2017]).

Cartographer gives a pose estimate of the sensor module in the map frame used to compute the excavator's entire state. Extrinsic calibration of the complete sensor module mounted on HEAP is obtained from Computer Aided Design (CAD) model and manual measurement. Accurately calibrating sensors mounted on heavy machinery remains a challenging problem, and to increase overall localization accuracy one should use more advanced methods (e.g. [API, 2021]). Note that sensors on the sensor module itself are calibrated w.r.t. each other using techniques mentioned in Section 3.2. The setup with the sensor module mounted in the back resulted in end-effector position accuracy (coupled errors from sensor module localization and robot kinematics) of about 30 cm. For evaluation, we have asked the harvester to grab the same tree blindly multiple times. Such a level of accuracy may not be enough for high precision harvesting, and we mitigate the problem by detecting the grabbing target in a locally built map (see Sec. 6).

6. Tree Detection

The tree detection subsystem's responsibility is to detect a tree trunk and compute its position. n contrast to the methods mentioned in Chapter 2, our method is lightweight, suitable for online

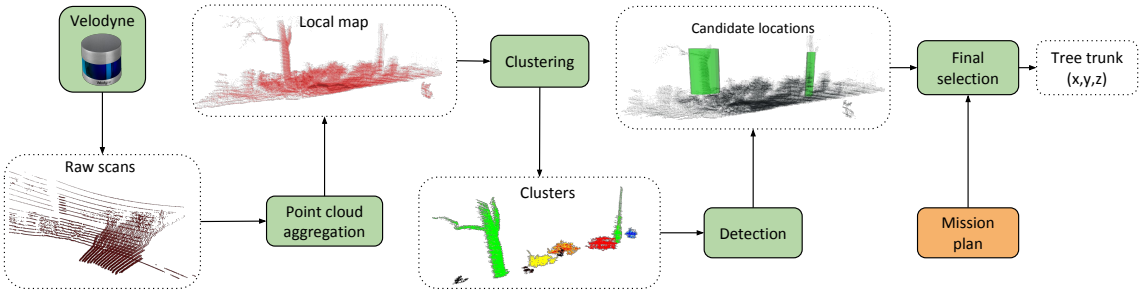


Figure 12. Flowchart of the tree detection subsystem procedure.

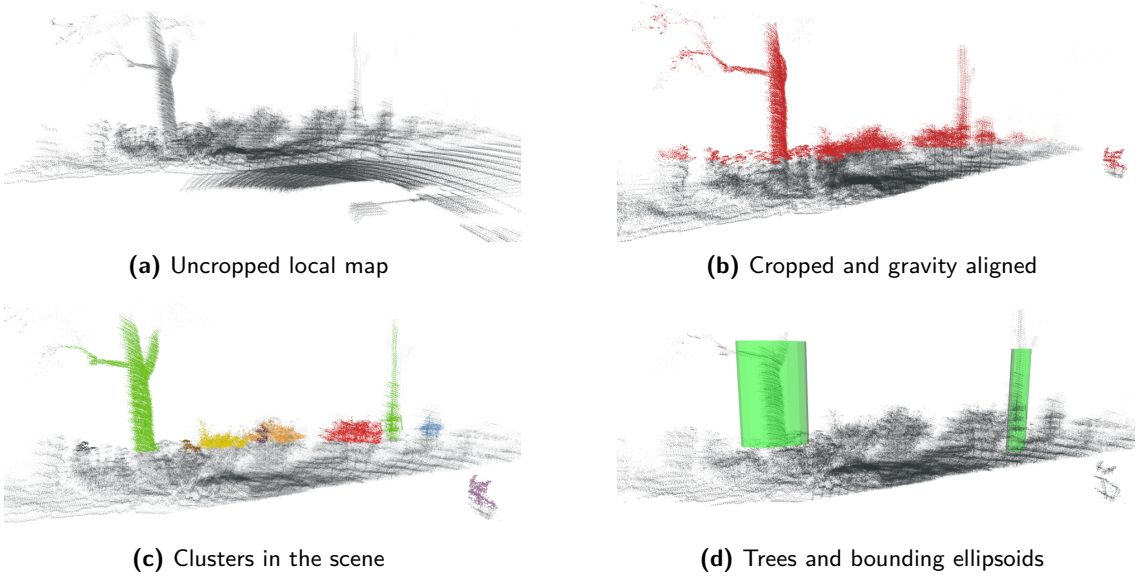
operation, and can be implemented in a dozen lines of code (available as open source[7]). While learning-based methods can still be fast to evaluate, we opted for a model-based method to avoid data labeling and for ease of implementation. Note that we are not interested in complete tree segmentation like most of the related work, but we are only interested in detecting a good grabbing spot. This circumstance allows simplifying the detection algorithm. A schematic of the subsystem with some intermediate processing steps is shown in Figure 12. Once HEAP is positioned close to a tree, the state machine initiates a scanning maneuver to create a map of the scene in the local frame. Tree detection in the local frame is less affected by inaccuracies in the global localization system. It is worth noting that the tree detector also works on global maps and can be used to aid mission planning; this is discussed in Sec. 9.3.1.

Tree detection starts with a scanning maneuver by rotating the cabin while the chassis is kept steady. The cabin is initially pointing in the direction where the harvester expects the target tree to be (known from the mission planner). The two LIDAR's mounted on the front of the machine (see Figure 4a) stream the point clouds at 20 Hz. Note that the harvester's arm (or legs) might appear in the point cloud which is undesired. We use the *robot_self_filter* package from the ROS software stack to filter them out. The *robot_self_filter* uses robot's geometry (meshes or geometric primitives), state (joint angles, pose) and range sensor's extrinsic calibration to identify points belonging to robot's links in the point cloud. Subsequently, these points are filtered out.

A receiver node stitches filtered point clouds together into a local map. The relative transformation between subsequent scans is recovered from the chassis's roll and pitch angle together with the cabin joint angle measurement. Note that in the absence of joint angle measurements or odometry, once could use point cloud registration methods (e.g. ICP). An example of a local map is shown in Figure 13a. The area scanned in Figure 13a is somewhat larger than necessary to visualize steps in the tree detection algorithm better. We turn the cabin in its yaw angle $\pm 30°$ to scan the area and build a local map during the deployment. This corresponds to double the horizontal FOV of the tilted Velodyne LIDAR (see Figure 4a) such that a dense local map can be built. We found that vertical Velodyne was more important for building dense maps, hence if one sensor is used, recommendation is to use it tilted.

The filtered scans from LIDARs are transformed into the cabin frame and cropped to speed up the subsequent steps. We use a box filter to crop the point clouds. The $x - y$ limits are set to drop all the points farther than the arm reach. Once cropped, the scans are transformed into a gravity-aligned frame where they are concatenated. The maximal density of the concatenated cloud is constrained to further speed up the computation. We use *libpointmatcher* library for cropping, density filtering and concatenation [Pomerleau et al., 2013]. Point cloud assembled from cropped scans is shown in black in Figure 13b (black). The assembled cloud is then gravity aligned and cropped again to filter out the ground plane and the tree crown (shown in red color, Figure 13b). Again, we use a box filter to remove all the that are lower than the center of the highest wheel. The

[7] https://github.com/leggedrobotics/tree_detection



(a) Uncropped local map

(b) Cropped and gravity aligned

(c) Clusters in the scene

(d) Trees and bounding ellipsoids

Figure 13. Intermediate results in the tree detection pipeline. ***Top Left:*** Scan of the area, without any cropping. One can observe the large ground plane in the point cloud. ***Top Right:*** Cropped point cloud after first and second cropping (red color). ***Bottom Left:*** Clusters in the scene. Each of these clusters is considered by the tree detection module. ***Bottom Right:*** Final tree detection with the resulting bounding ellipsoids.

red point cloud is sent to the tree detection module, which selects all prominent trees in the scan, as shown in Figure 13d.

The tree detection algorithm first computes Euclidean clusters in the input point cloud (shown in Figure 13c). Euclidean clusters are searched using the algorithm from [Rusu, 2010]. We discard the clusters with too few points. Since most trees grow vertically, a point cloud of a tree trunk should have a majority of the points spread out along the z axis. Hence, Principal Component Analysis (PCA) is performed on each cluster, and we only keep clusters with a significant principal component along the z axis. Note that we can exploit the verticality assumption since our point cloud is gravity-aligned. The gravity alignment score ($\in [0, 1]$) is defined as the dot product of the largest principal component with a $[0\,0\,1]^T$ vector. Lastly, we check for the minimum height of the tree. The final detection result is shown in Figure 13d. Sizes of bounding ellipsoids are computed based on principal components' eigenvalues in x and y direction. The final tree location is the ellipsoid's center. In the case of multiple tree detection (such as in Figure 13d), the algorithm extracts coordinates of the tree closest to the expected tree position (from the mission planner).

7. Control

In this work, chassis control is responsible for locomotion and arm control for tree grabbing (harvesting). The respective control subsystems (shown in blue, Figure 8) are described in more detail in this section.

7.1. Chassis Control

HEAP has four legs with wheels allowing it to drive and adapt to the terrain (see Figure 14.) Thereby, the goal is to optimally distribute the four wheels' load to ensure traction and minimize terrain damage.

Terrain adaptation controller (also named Hip Balancing Controller (HBC)) is based on a virtual model control principle where the controller computes a net force/torque on the chassis to achieve the desired orientation (roll, pitch) and height. An optimal contact force distribution is computed

(a) Chassis schematic **(b)** Chassis in action

Figure 14. ***Left:*** Illustration of the HEAP's chassis. The cabin and the arm are not shown for the sake of clarity. Steering joints axes are shown with blue arrows, flexion joint axes with red, and abduction joint axes with green color arrows. The path following controller actuates the steering joints while the HBC actuates the flexion joints. Abductions joints are not used. ***Right:*** Chassis control system overcoming a stump during the deployment.

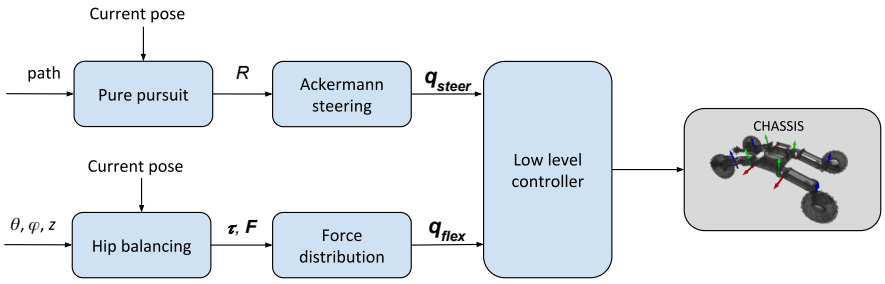


Figure 15. System diagram of the chassis control module deployed on HEAP.

from the net force/torque for all the legs. Contact force tracking is achieved via force tracking on the hydraulic actuator level. For more details, refer to [Hutter et al., 2016]. The described chassis controller can keep the base leveled while overcoming large irregularities in the terrain without getting stuck. Terrain adaptation is achieved using proprioceptive measurements only (joint sensing, IMU). We have extensively tested HBC performance in our previous work ([Hutter et al., 2015], [Hutter et al., 2016]). A video showing the machine overcoming challenging terrain can be found online[8].

For path tracking, we use a pure pursuit controller similar to the one outlined in [Kuwata et al., 2008]. The pure pursuit controller calculates the turning radius (denoted with R) required to bring the machine on the path computed by the planner. The Ackermann steering module then calculates steering joint angles for each leg. Steering angles are computed such that all four wheels point to the same center of rotation with the radius R. The computation is implemented as a quadratic optimization that satisfies all the joint limits and finds a feasible center of rotation. For more details on the Ackermann steering module, the reader is referred to [Jud et al., 2021].

The HBC and the pure pursuit controller are combined to locomote the harvester, as shown in Figure 15. Both controllers close the feedback loop over pose: HBC regulates the roll, pitch, and height, whereas pure pursuit ensures that x, y, and yaw are tracked correctly. The Ackermann steering module computes position reference for the steering joints ($\boldsymbol{q}_{steer}$ in Figure 15), while the force distribution module computes joint torques assuming quasi-static conditions ($\boldsymbol{q}_{flex}$). The low-level controller translates the joint quantities (torques, positions) into the valve commands. In Figure 14b, the proposed controller is driving over a stump. Note how the controller retracts the left hind leg to maximize the traction. In parallel, the pure pursuit controller controls the driving direction.

[8] https://youtu.be/5_Eq8CxKkvM

Table 3. Task priority inside the hierarchical optimization for the arm inverse kinematics controller.

Priority	Task
1	Equations of motion
2	Pump flow limit
3	Cylinder force limits
3	Cylinder velocity limits
3	Cylinder position limits
4	End-effector orientation
4	End-effector position

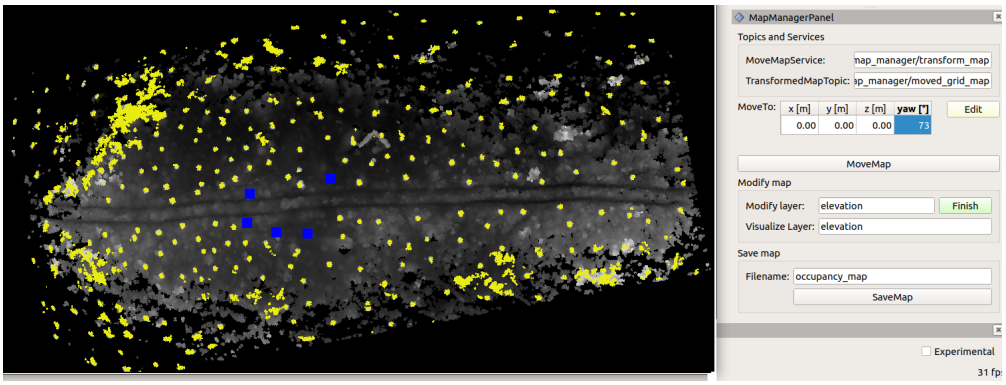


Figure 16. Graphical User Interface (GUI) used for mission planning. It is a panel in *Rviz* and it uses a *Qt* front-end. An elevation map of previously mapped environment is shown in gray-scale. All points higher than 2 m are colored in yellow; hence, the yellow color corresponds to the tree trunks and parts of the canopy. In this example, a total of five trees (marked with blue squares) are selected to be cut.

7.2. Arm Control

Reaching the end-effector target position determined from the tree detection module is achieved by following a trajectory from the planner described in Section 8.3. The trajectory following is done using an IK controller (see [Siciliano et al., 2010]) which uses the Hierarchical Optimization (HO), based on the implementation from [Bellicoso et al., 2016]. The main difference is that we tailor the tasks for HEAP instead of a quadruped robot. Besides tracking, HO computes joint velocities, enforces kinematic limits, and ensures that all flow constraints for hydraulic actuators are satisfied. The set of tasks optimized by the HO is given in Table 3, where *1* is the highest priority task. Opening and closing the gripper is controlled directly by the state machine and it is done in a purely open-loop fashion.

8. Planning

In this section we describe the mission planner and the motion planning stack in more detail. The motion planing stack is divided into three components shown in Figure 8: base motion planner, grasp pose planner and the arm motion planner.

8.1. Mission planner

To conduct the experiments, we design a mission planner which determines which tree to grab next. Before the mission, a human manually selects the trees to be cut. Selection is accomplished using the GUI shown in Figure 16, thus mimicking an algorithm for tree inventory management. When clicked on, a tree gets added to the tree list for harvesting with position coordinates extracted in

the map frame. The mission planner passes the tree coordinates to the state machine in the same order as they were selected. A mission planner for optimizing some user-given objective and finding an optimal cutting order remains to be investigated in the future.

8.2. Base Pose Planning

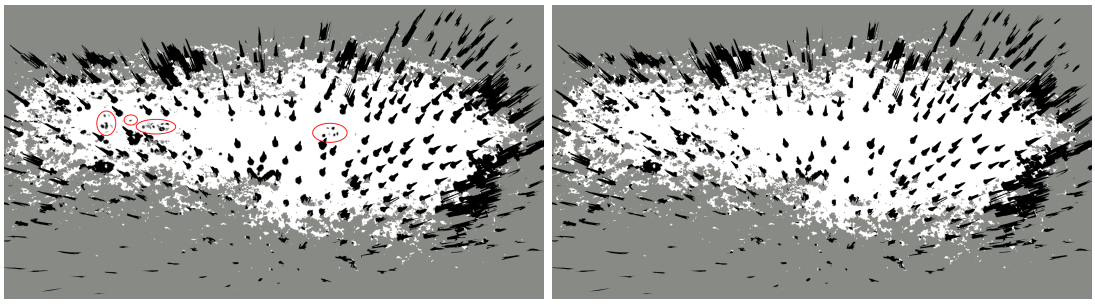
We do not make any assumptions on the terrain properties; thus, our planner accounts for irregular terrain. Path planning in rough terrain is a complex problem, and in general, it has not been solved yet. In this work, we split it into two more manageable subproblems: planning in *SE(2)* (instead of *SE(3)*) and traversability estimation.

8.2.1. Traversability Estimation

The traversability estimation module discerns which areas can be driven versus which ones should be avoided. From the elevation map we compute a function $f : (x, y) \rightarrow [0, 1]$ that tells us for each coordinate (x, y) where HEAP can go (e.g. 1 - safe to drive, 0 - not safe). Traversability estimation has been previously studied, and different approaches for various sensing modalities exist. In this work, we use purely geometric traversability estimation, which is directly applied to elevation maps without any additional processing. We follow the approach presented in [Wermelinger et al., 2016] that evaluates three criteria: terrain slope, roughness, and step height for local terrain patches (0.3 m radius). The final result is a weighted sum of all three components. We chose to mainly rely on step criterion (80 %) since the chassis control system introduced in Sec. 7.1 can overcome slopes and drive over rough terrain. The remaining 20 % was assigned to the slope criterion to prevent the harvester from driving on very steep slopes, which could result in slipping.

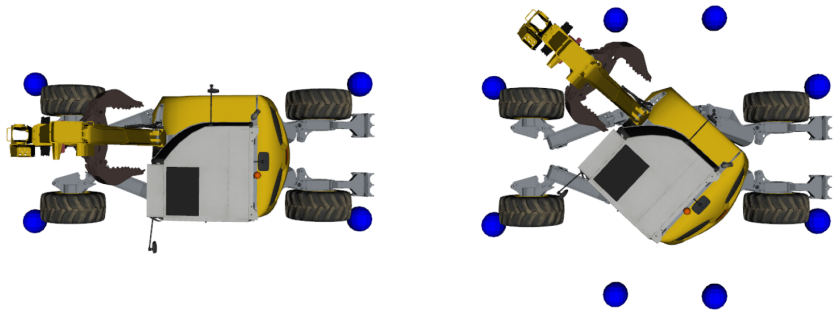
For storing the traversability map we use the grid map [Fankhauser and Hutter, 2016] data structure. Figure 17a shows the traversability map, a 2.5D map with a traversability layer; incorrectly classified areas (classified untraversable while it is traversable) are encircled red. Thick vegetation that occludes the ground from the LIDAR sensor during the mapping phase is the main culprit for false negatives. Since the overall misclassified area is small, a human can manually correct the errors. We used the GUI from Figure 16 and the correction lasted about 2 minutes. Figure 17b shows the corrected traversability map; note how the algorithm automatically classifies tree trunks as untraversable. Presented traversability estimation has an upside that is easy to implement; there is no need for elevation map processing or segmenting out the tree trunks explicitly. Fully automating the proposed pipeline would require additional work to eliminate the correction step and be implemented in future work.

The traversability map (see Figure 17b) is converted into an occupancy map. Any traversability value smaller than 0.5 is deemed an obstacle, and higher values are regarded as free space. Motion planners use the occupancy map, a 2.5D map with an occupancy layer.



(a) Traversability Classification errors **(b)** Corrected traversability

Figure 17. Traversability maps computed by our approach shown against gray background. The maps shown are 2.5D maps with different layers (traversability layer displayed). White areas represent fully traversable terrain, whereas black areas represent terrain that is not traversable. ***Left:*** Red areas have been incorrectly classified as untraversable by the algorithm (false negatives). ***Right:*** Traversability map after applying the manual correction.



(a) Path planning footprint **(b)** Approach-pose planning footprint

Figure 18. Top view of HEAP with the arm retracted with the footprint vertices shown in blue. Note: not drawn to scale. ***Left:*** Footprint used for path planning (see Alg. 2). ***Right:*** Footprint used for the approach-pose generation (see Alg. 1) is wider in the middle, thus allowing for cabin turning.

8.2.2. Base Approach-Pose and Path Planning

The planning subsystem (shown in yellow color in Figure 8) gets a tree position from the mission planner. The tree position is an approximate target location for the end-effector. The harvester should not reach the tree position itself because this would result in a collision with the tree. To this end, we develop an algorithm for joint path and approach-pose planning. We evaluate the algorithm in both simulations and natural environments. Functionality described in this subsection corresponds to blocks *Approach-Pose Generation* and *Base Motion Planner* in Figure 8.

Existing planning algorithms typically plan from starting pose to the goal pose. However, in our case, we do not know the goal pose (only approximate end-effector position in x and y). Hence, the proposed planning algorithm is split into two stages: first, the planner generates candidate poses and checks for their feasibility. Secondly, the planner attempts to compute a path to any of the feasible candidates. Thus the approach-pose planning is reduced to a common path planning problem, and we leverage a Rapidly-exploring Random Tree (RRT)* algorithm for a joint path and approach-pose computation. Our implementation leverages Open Motion Planning Library (OMPL), [Sucan et al., 2012]. We chose the OMPL because it is available as open-source and comes with efficient implementations of various sampling-based planners. We do not rely on optimization-based planners since forests are cluttered environments with many local minima, which present a challenge for optimization.

The approach-pose generating subroutine is shown in Alg. 1. It starts by getting a target/tree location $\boldsymbol{x}_T \in \mathbb{R}^2$ (line 1), and computing a set of approach-poses around it (lines 13 - 23). We compute $M \times N$ approach positions in polar coordinates around $\boldsymbol{x}_T$ by pairing M different distances to the tree with N uniformly distributed polar angles. Lastly, the approach positions are paired with K yaw angles (lines 20-22) to generate a total of $N \times M \times K$ candidate approach-poses in *SE(2)*. To be a valid candidate, an approach-pose must be collision-free (lines 6-7), the harvester's arm must be able to reach the target (lines 6-9), and all heuristics must be satisfied (lines 10-11).

The blue points shown in Figure 18 represent the harvester's collision footprint. The harvester is in a collision if there is an obstacle inside the blue points convex hull. Collision checks inside the Alg. 1 use the footprint shown in Figure 18b. Note how the hull is wider in the middle to allow for cabin turns. The real-world map is abstracted away from the planner, and it only sees the obstacles computed by the traversability estimation step. Such a simplification is warranted by using a control system able to overcome slopes and rough terrain.

The target being reachable from an approach-pose means that the harvester can safely extend the arm to reach the goal position (see Alg. 1, line 8). We check whether there is a collision-free line of sight from the base to the target goal. Because of HEAP's kinematic structure, the arm always stays within a slab in the $x - z$ plane (in the cabin frame). Hence, we require no obstacles inside the slab spanned by the target tree position and the base position. Therefore, there is no need

Algorithm 1. Candidate approach-pose generator

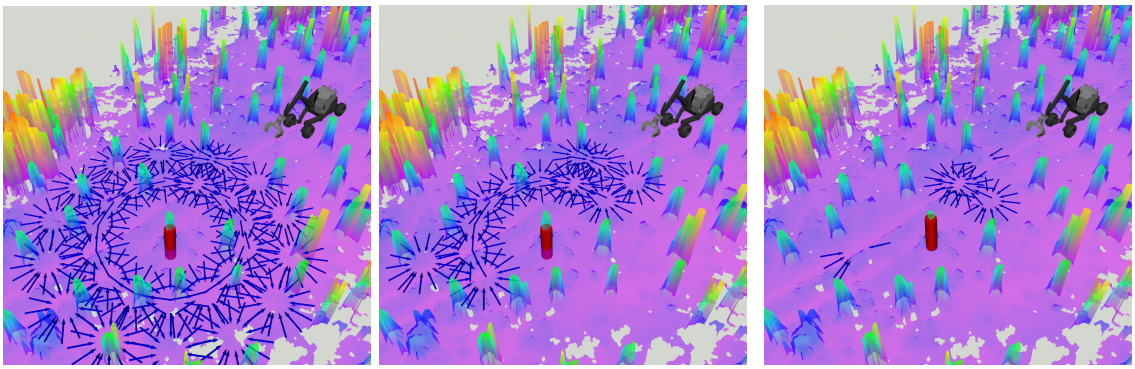
```
1: x_T, y_T ← GETNEXTTARGETLOCATION()
2: isUseHeuristic ← READFROMCONFIGFILE()
3: procedure COMPUTECANDIDATEAPPROACHPOSES(x_T, y_T)
4:     candidateApproachPoses ← GETAPPROACHPOSESAROUNDTARGET(x_T, y_T)
5:     for each pose in candidateApproachPoses do
6:         if ISINCOLLISION(pose) then
7:             candidateApproachPoses.DELETE(pose)
8:         if ¬ISTARGETREACHABLEFROM(pose) then
9:             candidateApproachPoses.DELETE(pose)
10:        if isUseHeuristic AND ¬ISHEURISTICVALID(pose) then
11:            candidateApproachPoses.DELETE(pose)
12:    return candidateApproachPoses
13: procedure GETAPPROACHPOSESAROUNDTARGET(x_T, y_T)
14:    distances ← {d_1, d_2, ..., d_M}
15:    polarAngles ← {φ_1, φ_2, ..., φ_N}
16:    headings ← {ψ_1, ψ_2, ..., ψ_K}
17:    approachPoses ← ∅
18:    for each d_i in distances do
19:        for each φ_i in polarAngles do
20:            (x_i, y_i) ← (x_T + d cos(φ_i), y_T + d sin(φ_i))
21:            for each ψ_i in headings do
22:                approachPoses.ADD([x_i, y_i, ψ_i])
23:    return approachPoses
```

Algorithm 2. Path and approach-pose planner

```
1: procedure COMPUTEPATHANDAPPROACHPOSES(x_T, y_T)
2:     x_T, y_T ← GETNEXTTARGETLOCATION()
3:     p_s ← startingPose
4:     rrt.INITIALIZE(p_s)
5:     candidateApproachPoses ← COMPUTECANDIDATEAPPROACHPOSES(x_T, y_T)
6:     while t_cpu < T_max do
7:         rrt.GROWTREE()
8:         for each p_i in candidateApproachPoses do
9:             rrt.CONNECTTOTREE(p_i)
10:    return (p_i, rrt.GETPATH())
```

for more complicated algorithms that are popular in mobile manipulation literature (e.g., [Zucker et al., 2013]). An example of the approach-pose generation in a forest environment is shown in Figure 19.

In the second stage, the planner checks whether the feasible approach-poses are attainable, i.e., can the harvester drive to them. Note that a feasible approach-poses such as the red one in Figure 20 is not necessarily attainable. The subroutine for attainability checking is summarized in Alg. 2. We build upon a standard RRT* planner that grows a random tree, as described in [Karaman and Frazzoli, 2011]. Instead of trying to connect the single goal pose (as standard RRT*) to the rest of the tree, we attempt to connect every candidate approach-pose. Planning terminates when the allotted planning time runs out. We found 5 s to be an adequate compromise between computation time vs mission progress time. The footprint used for collision checking inside Alg. 2 is shown in Figure 18a. As a steering function inside the RRT*, we use the Reeds-Shepp (RS) curves



(a) All approach-poses **(b)** Goal target reachable **(c)** Goal reachable & no collision

Figure 19. Approach-pose generation in the forest environment. The elevation map with the target tree (red cylinder) and candidate approach-poses are shown with blue arrows. Some approach-poses have been omitted for the sake of clarity. ***Left:*** In total, 450 goal approach-poses are generated. So far they have not been checked for feasibility and this example we do not apply any heuristics. ***Middle:*** Out of 450 poses 155 do not satisfy the target reachability criterion (line 8 in Algorithm 1). The arm can extend and grab the target tree from 195 remaining poses. ***Right:***. Out of 195 poses, 168 of them are in collision with the environment (line 6 in Algorithm 1). The remaining 27 approach-poses both satisfy the reachability criterion and are not in collision. These remaining 27 approach-poses are then sampled inside the RRT* to determine which ones are attainable.

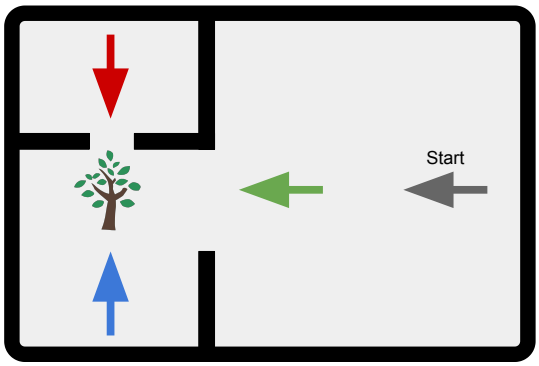


Figure 20. Example environment for approach-pose planning. All approach-poses are feasible; however, the red pose is not reachable from the start. The green pose is better than the blue pose since it is easier to reach.

[Reeds and Shepp, 1990] which allows us to respect the minimal turning radius constraint (just like any car, HEAP cannot turn in place). The turning radius parameter was set to 8.3 m.

The rationale behind the planner's second stage is that an attainable approach-pose has a high probability of having a collision-free connection to the rest of the random tree. For example, the green pose is a better approach-pose compared to the blue one in Figure 20. The environment around the green pose is less cluttered, so the probability of a successful connection is higher. A beneficial feature of the described approach (Alg. 2) is outsourcing the final approach-pose selection to the RRT*. Note that using the RRT* in its standard form would require the user to pick an approach-pose, a challenging task since we do not know which ones are attainable *a priori*. Since fewer approach-poses ensure faster convergence, the proposed framework allows the use of heuristics for pruning the approach-pose candidate set. Pruning heuristics can be a wide array of constraints and rules, thus allowing for great flexibility (e.g., discard approach-poses where heading changes more than 90°).

The proposed approach-pose planning takes into account almost all DoFs that HEAP has to offer. It allows the machine to turn the arm and approach trees from any angle. Furthermore, the harvester can utilize both driving directions when navigating to the goal target. One could still

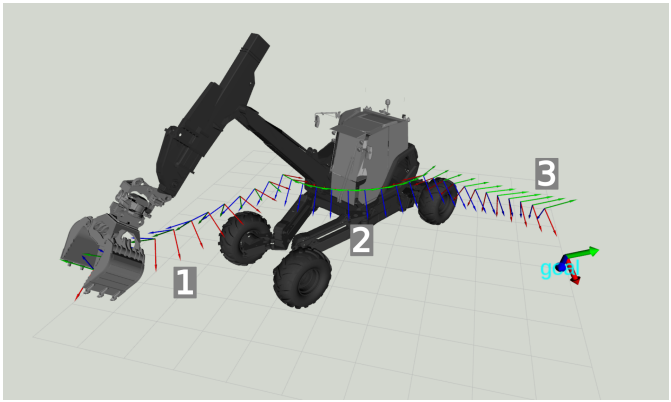


Figure 21. Plan for the arm end effector. Intermediate poses are visualized with coordinate systems. Each axis (x, y, z) corresponds to a different color (red, green, blue). A thick coordinate system and "goal" sign denote the final desired pose. Numbers correspond to different maneuver stages: 1) retract, 2) turn, 3) extend.

improve the approach-pose generation to anticipate the arm turning direction. Algorithm 1 uses collision footprint shown in Figure 18b which is clearly conservative since the arm doesn't have to make a full 360° turn. Anticipating the turning direction would allow shrinking the collision footprint, which is very beneficial in cluttered environments such as one shown in Figure 19. Another possible improvement is to adapt the number of approach-pose candidates based on the environment. Generating too many approach-pose candidates slows down the planning while marginally contributing to finding better solutions when the obstacle density is low.

8.3. Arm Grasp Pose and Motion Planning

The grasp pose planner receives a tree position from the tree detection subsystem and computes the desired gripper pose. Functionality in this section corresponds to *Approach-pose generation* and *Base motion planner* blocks in Figure 8. Since for our demonstration, we use a gripper instead of a standard tree cutting tool (such as [Menzi Muck , 2020]), we emulate the same behavior by fixing the roll and pitch of the gripper and by computing the yaw angle such that the cabin faces the tree. The kinematic properties of HEAP and harvester machines in general with an arm that only moves in a plane allow us to come up with a simple arm planning algorithm. To reach the desired grasp pose, we design a three-stage maneuver that requires minimal space:

1. Retract the arm
2. Turn the cabin
3. Extend the arm

The approach-pose planner (see Sec. 8.2.2) ensures that there is enough space for the whole arm maneuver. An exemplary arm plan is shown in Figure 21 (stages of the maneuver are indicated with numbers). Intermediate poses (waypoints) are visualized with coordinate systems. We compute Hermite polynomials between the adjacent waypoints to form a trajectory, and we limit the average linear and angular velocity along the spline. The Hermite polynomial tracjecory is then tracked using the inverse kinematics controller, as described in Section 7.2.

For tree felling, the proposed planning method would have to be augmented to consider tree geometry when the harvester is holding one. This could be achieved by adding visual sensors and tracking the falling tree such that the planner can optimize the pulling direction. Besides, one could also estimate the weight and adapt the arm controller tuning or add a feedforward command. The current arm controller is robust concerning weight changes in the gripper. We experimented with stone stacking in [Johns et al., 2020] and we were able to manipulate stones between 400 kg and 2400 kg

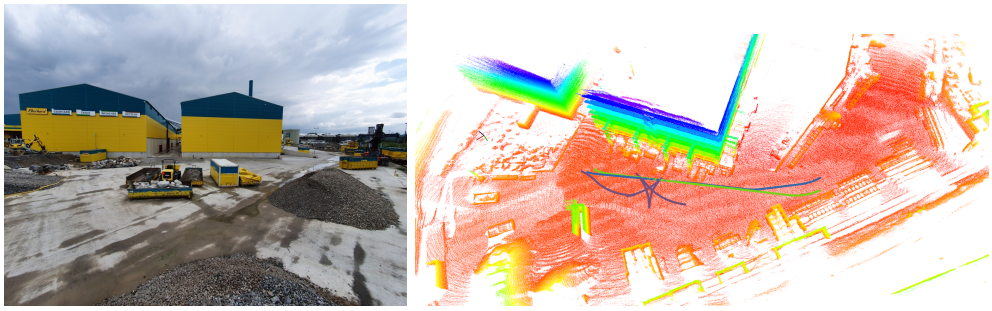
(1040 kg on average) which is enough to harvest a small tree. Lastly, one could extend approaches such as [Song and Sharf, 2020] to prevent the machine from tipping over when holding a tree.

9. Results

The presented system components are individually tested and evaluated for complete harvesting missions in the forest and on our testing field. It is worth noting that we implemented the whole system in simulation first to catch as many mistakes as possible before field testing. As the simulation environment, we use *Gazebo* (with ROS integration) which is based on the Open Dynamics Engine (ODE) physics engine. For visualizing (e.g. trajectories, point clouds) we rely on *Rviz*, a ROS tool for visualization.

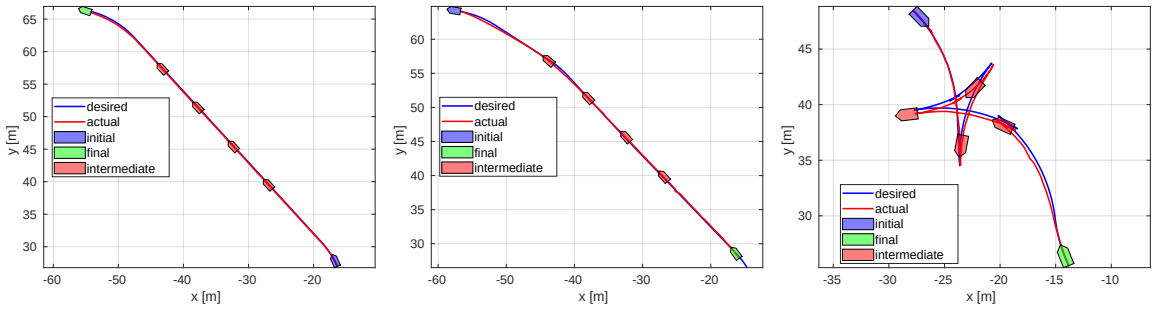
9.1. Path Tracking

We evaluated the tracking performance of the proposed control approach in Oberglatt (Switzerland), where HEAP was located for another project at the time of writing this paper. The site is shown Figure 22; it is a construction site with a mix of concrete and gravel surfaces. We evaluate the



(a) Oberglatt site **(b)** Site map with overlaid paths

Figure 22. Testing site where tracking experiments were performed. The image and the map were taken few weeks apart. ***Left:*** Photo of the testing site where HEAP was located at the time of writing this paper. We were allowed to drive on the site to test the tracking controller's performance. ***Right:*** Map of the testing site with some example plans overlaid. The green path corresponds to the one in Figure 23a, the blue path can also be seen in Figure 23b and the purple one can be seen in Figure 23c.



(a) Forward driving **(b)** Reverse driving **(c)** Tight maneuvering

Figure 23. Example trajectories from tracking accuracy evaluation. Blue color shows output from the planner; red color denotes the actual tracked path. HEAP's initial orientation is shown in blue color and the final one in green. The size of the vehicle is not drawn to scale. ***Left:*** HEAP achieved 9 cm average tracking error over 55 m of forward driving distance. ***Middle:*** HEAP achieved 13 cm average error over 57 m of backward driving distance. ***Right:*** HEAP achieved an average error of 18 cm over 62 m distance for tight maneuvering scenario. In this particular example, HEAP does 7 cusps (changes of driving direction).

Table 4. Path tracking performance evaluation in Oberglatt site. All the errors are transitional errors in $x - y$ coordinates. *Avg length fwd* denotes the average distance traveled forward during the maneuver (analogously for the backward distance, *avg length bck*). *Traveled total* is the cumulative distance traveled across all trials.

scenario	num trials	avg track error [m]	standard deviation [m]	avg length fwd [m]	avg length bck [m]	avg num cusps	traveled total [m]
fwd driving	10	0.23	0.04	56.05	0.52	0.6	565.84
bck driving	8	0.16	0.04	0.76	55.93	0.75	453.56
maneuvering	14	0.13	0.03	17.99	17.44	2.93	496.19
total	32	0.17	0.02	24.93	24.63	1.43	1515.59

(a) test field SE view **(b)** aerial view

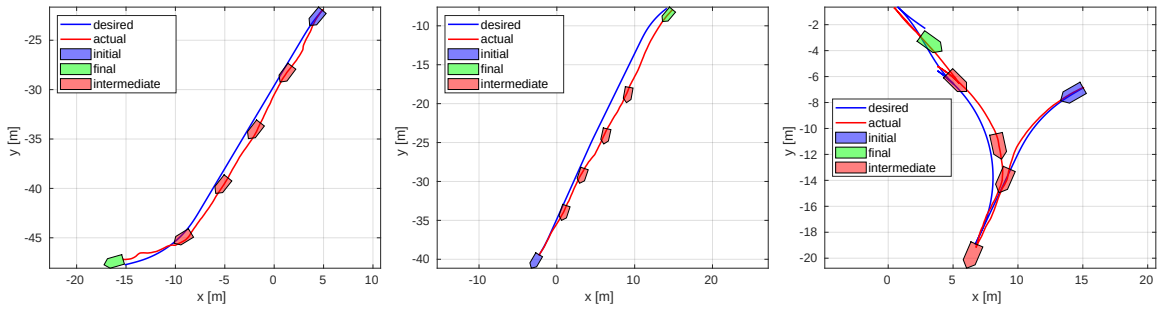
Figure 24. Photos of our testing field, not all images were taken at the same time. ***Left:*** View looking southeast with the machine parked. The area shown is flat. ***Right:*** Aerial view of the field with different classes of terrain labeled. We overlay the planned (blue) and executed (red) path from one tracking experiment (see Figure 25a).

tracking performance using RTK GPS to measure HEAP's position accurately. HEAP was asked to track three different types of paths which consisted of: mostly forward driving, mostly backward driving, and tight maneuvering. An example of the forward driving path is shown in Figure 23a, the backward driving path is shown in Figure 23b and tight maneuvering path is shown in Figure 23c. The maneuvering scenarios require the machine to change its orientation in a tight space. Hence, the planner comes up with paths containing cusps and tight turns; we asked HEAP to change its orientation (heading) for either 90° or 180°.

Our controller only tracks the path in $x - y$ coordinates, and hence, all the errors are computed in $x - y$ coordinates. The controller does not track the z coordinate since the planner plans in *SE(2)*. The tracking performance was evaluated over about 30 trials and about 1.5 km of driving. We summarize the tracking performance in Table 4. Across all trials, the path tracking algorithm achieves the tracking error of about 17 cm. We note that the tracking errors achieved in the Oberglatt site are somewhat optimistic since tests took place on flat surfaces that have good traction. It is to be expected that the tracking performance deteriorates as the surfaces get more slippery. This effect we illustrate in a set of experiments conducted on our test field.

The testing field is shown in Figure 24. HEAP was commanded to follow three different paths shown in Figure 25. Figure 25a and Figure 25b show tracking a long path on an inclined terrain (see Figure 24b). A combination of inclined and wet terrain caused the HEAP to slide sideways. This can also be observed in the path visualization: the heading is not tangential to the path. The machine is pointing towards the slope to compensate for sliding to the side. In Figure 25c we asked HEAP to reorient itself on the flat part of the testing field.

The chassis controller was able to achieve mean absolute tracking error of 0.461 m over 34.1 m distance for the run shown in Figure 25a and for the run shown in Figure 25b the error was 0.684 m over distance of 44.28 m. While performing reorientation on the flat terrain, the mean absolute error



(a) Forward on lateral slope **(b)** Reverse on lateral slope **(c)** Turns and cups on even ground

Figure 25. Experiments were conducted to determine the tracking accuracy of our chassis control approach. In blue color, the planner's desired plan has been shown; red color denotes the actual tracked path. The initial orientation of the excavator is shown in blue color and the final one in red. The size of the vehicle is not drawn to scale.

was 0.222 m over distance traveled of 44.48 m (with all direction changes). One can observe that inclined terrain presents a bigger challenge for the tracking controller than direction changes. We note that the test field was fairly wet during the experiments, which negatively affected the amount of traction available. Forestry operations are typically performed when the ground is either dry or frozen, which reduces these slipping issues. Hence, the set of results from our test field presents a hard case scenario which is not often encountered in regular harvesting operations.

9.2. Planning

We evaluate the approach-pose planning pipeline proposed in Sec. 8.2.2 in simulated scenarios and in two experiments on different terrain: test field and forest patch. We ask the planner to compute both path and an approach-pose to grab the selected target trees. All the map visualizations in this section have been created using *Rviz*.

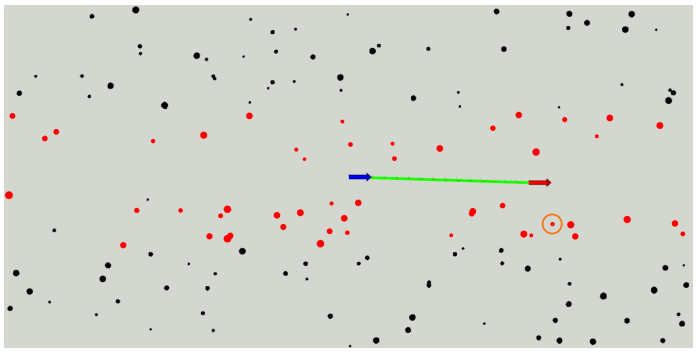
We created a simulated forest by sampling the number of trees from a Poisson distribution and their positions and radii from a uniform distribution. We consider two scenarios: in the first one, there is a forest alley that can be used for driving (see Figure 26a). This resembles the situation encountered during the field experiments. There is no forest alley in the second scenario, and the harvester has to navigate between the trees (see Figure 27a). Such a scenario is common when using a smaller machine such as one shown in Figure 2a. For each forest density, we run 10 planning trials and average the metrics. Within one trial, we to compute plans for several target trees; the number of trials is shown in Table 5.

For the forest alley scenario, we ensure the alley at least 2.8 m wide. For comparison, the harvester path planning footprint has a width of 2.4 m and the approach-pose planning footprint is 4.8 m wide at its widest point. Note that as the forest gets denser, there might not be enough space to turn the cabin, and therefore no feasible approach-poses. We choose several trees (max 50) at random within the 6 m distance from the forest alley middle (HEAP has a reach of about 8 m) to be the targets. Black dots represent trees while target trees are colored red in Figure 26a. Each target is deemed feasible if we can find a path to it within 30 s of planning. An example path is shown in green; the starting pose and the planned approach-pose are denoted with a blue and red arrow, respectively. In this particular example, the plan requires the harvester to drive forward and turn the cabin to grab the tree encircled orange. Quantitative evaluations of the planner are shown in Figure 26, we compute the metrics only for feasible targets.

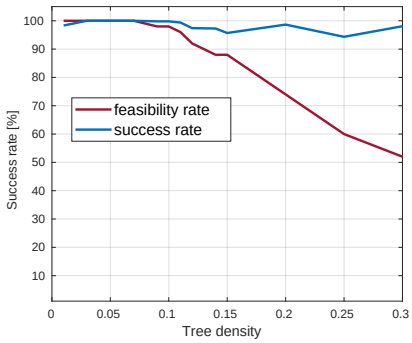
The percentage of feasible targets (see Figure 26b) drops as the forest's density grows. However, one can see that the planner maintains a high success rate. Besides the success rate, we evaluate times required to generate candidate approach-poses $t_{approach}$, time until the first solution inside the RRT* planner t_{init} and we show the total planning time t_{total} (Figure 26c). One can observe

Table 5. Number of feasible (attainable) targets together with the total number of planning attempts for each forest density. We show the data for forest alley and unstructured forest scenario.

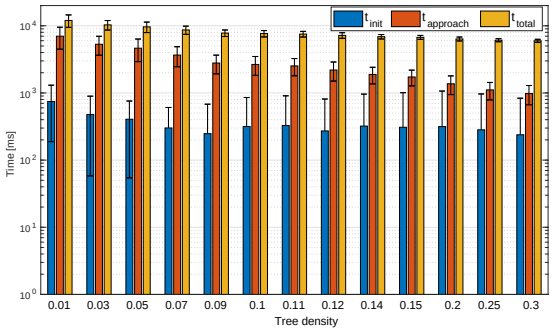
	forest alley			unstructured forest		
density	**n targets**	**n feasible targets**	**n (successful) planning attempts**	**n targets**	**n feasible targets**	**n (successful) planning attempts**
0.01	6	6	(59) 60	23	23	(230) 230
0.03	18	18	(180) 180	50	50	(499) 500
0.05	23	23	(230) 230	50	50	(498) 500
0.07	38	38	(380) 380	50	49	(486) 490
0.09	50	49	(489) 490	50	44	(398) 440
0.1	50	49	(489) 490	50	36	(332) 360
0.11	50	48	(477) 480	50	32	(283) 320
0.12	50	46	(448) 460	50	27	(269) 270
0.14	50	44	(428) 440	50	26	(251) 260
0.15	50	44	(421) 440	50	23	(224) 230
0.2	50	37	(365) 370	50	9	(90) 90
0.25	50	30	(283) 300	50	7	(70) 70
0.3	50	26	(255) 260	50	7	(70) 70



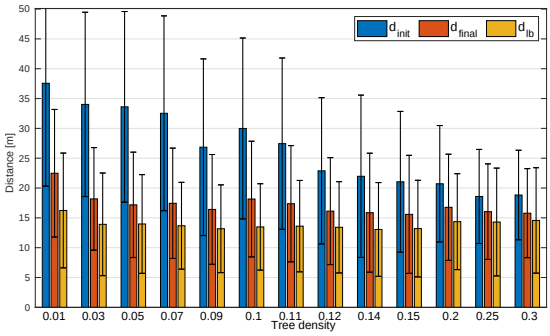
(a) Simulated forest alley for tree density 0.1



(b) Success rate

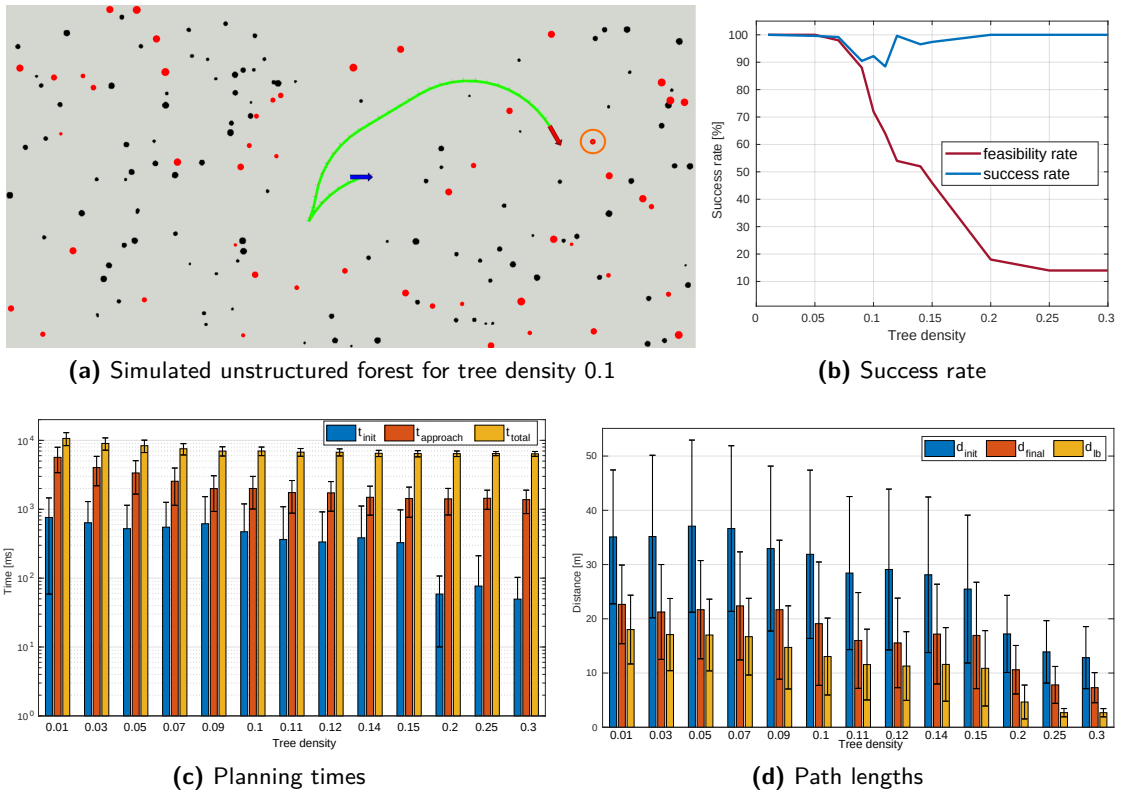


(c) Planning times



(d) Path lengths

Figure 26. Joint path and approach-pose planning for the forest alley scenario. Tree density is defined as a number of trees per m^2. ***Top Left:*** Top view of the random forest with the path and approach-pose planned. The harvester starts from the blue arrow to grab the tree encircled orange. Path is shown in green and the final approach-pose in red. Black dots represent the trees and red dots are the remaining target trees. ***Top Right:*** Percentage of feasible targets and the success rate as the forest density increases. Success rate is calculated as number of successful planning attempts divided by the number of feasible targets. ***Bottom Left:*** Planning times against varying forest density, note the logarithmic scale on the y axis ***Bottom Right:*** Path lengths as a function of varying forest density.



(a) Simulated unstructured forest for tree density 0.1

(b) Success rate

(c) Planning times

(d) Path lengths

Figure 27. Joint path and approach-pose planning for the unstructured forest scenario. ***Top Left:*** Top view of the random forest with the path and approach-pose planned. See Figure 26 for explanations of the colors in the image. ***Top Right:*** Success rate as the forest density increases. Feasibility rate and success rate are calculated the same way as for Figure 26. ***Bottom Left:*** Planning times against varying forest density, note the logarithmic scale on the *y* axis ***Bottom Right:*** Path lengths as a function of varying forest density.

that the RRT* planner finds the first solution rather quickly (worst case in 700 ms). The most expensive part of the pipeline is the approach-pose generation which in the worst case takes about 7 s. Planning times tend to shorten as the forest density (number of trees per m^2) increases since many approach-poses can be discarded in the early stage of collision checking (early termination). In contrast, for low forest densities, almost all approach-pose footprints have to be checked for collisions fully (no early termination).

For the experiments presented, the planner considers 14060 approach-pose candidates in total (no pruning heuristics were applied to showcase the generality of the approach). In this work, we use single-threaded implementation; however, the approach-pose generation can be easily paralleled to decrease computation time. Lastly, we measured the distances between the starting pose and the planned approach-poses (see Figure 26d). We show the length of the first found path d_{init} and the length of the optimized path d_{final} (after running RRT* for 5 s). Lastly, the graph shows Euclidean distance between start and finish d_{lb} which is a lower bound on the path length. We can see that the planned path is close to the lower bound in all cases. The paths tend to shorten with the increasing density since more feasible trees lie on the inside of the forest alley. Hence, HEAP does not turn to the side, which shortens the overall path length.

As a second scenario, we evaluated planning performance while navigating an unstructured forest. The forest was generated using the same probability distributions as in the scenario above. We leave a clear area (10 m by 10 m) in the middle of the map to ensure a feasible starting pose. The target trees were selected at random with a maximum of 50 trees. Again, we check the feasibility for each target by running the planner for 30 s and perform 10 planning trials.

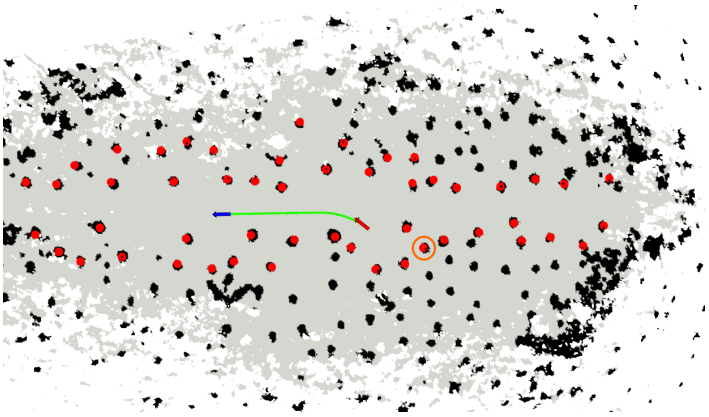


Figure 28. Obstacle map for the forest patch scenario. Obstacles are shown in black color whereas free space is shown in gray color. The targets are denoted with red cylinders

Table 6. Planning metrics for the forest patch scenario (averaged over 10 trials).

num targets	51
success rate	0.98
time until first solution [ms]	(120 ± 555)
approach-pose generation time [ms]	(1625 ± 621)
total planning time [s]	(6.6 ± 0.6)
initial path length [m]	(18.38 ± 10.86)
optimized path length [m]	(14.3 ± 9.84)
length lower bound [m]	(13.53 ± 9.48)

Performance measures shown in Figure 27b - Figure 27d are calculated only for feasible targets. One can observe a much faster drop in the percent of feasible targets (compared to the forest alley scenario). In most cases, our planner finds the solution rather quickly (800 ms in the worst case). We can also notice that the success rate drops compared to the previous scenario since there is no forest alley. The planner has to find a path through narrow passages (hard for sampling-based planners). The forest density of 0.1 seems to be especially hard, i.e. the harvester can still fit through the trees, but just barely. The success rate for densities between 0.05 and 0.15 can be increased with longer planning times. For forest densities of 0.15 and higher the HEAP harvester simply dos not fit between the trees. Hence the set of feasible targets comprises the trees around the clear patch in the middle of the simulated forest. In this case, the success rate goes again to 100 %.

To evaluate the planning performance under realistic conditions, we planned on a map of our testing field and a map of forest patch where we conducted the tests. The forest patch map with target trees is shown in Figure 28. Black dots represent trees and other obstacles, while target trees are denoted with red dots. We manually selected the trees along the forest alley, discarding any tree without path computed within 30 s of planning. The forest density at the testing site was estimated to be about 0.14 (trees/m^2). The planner achieved a success rate of about 98 % which is almost the same as the simulated scenario (95 %). Furthermore, the time $t_{approach}$ (1625.15 ms) is close to the simulated one (1728.75 ms). t_{init} (120.62 ms) is lower than the simulated one (308.36 ms); which would indicate that it has fewer narrow passages than the simulated forest.. On the testing field, the planner achieved the success rate of 100 % (omitted for the sake of brevity).

9.3. Tree Detection

We evaluated tree detection offline by mimicking local maps assembled from harvester sensors during the scanning maneuver. We select 6 m by 6 m patches inside the map shown in Figure 9a and ask

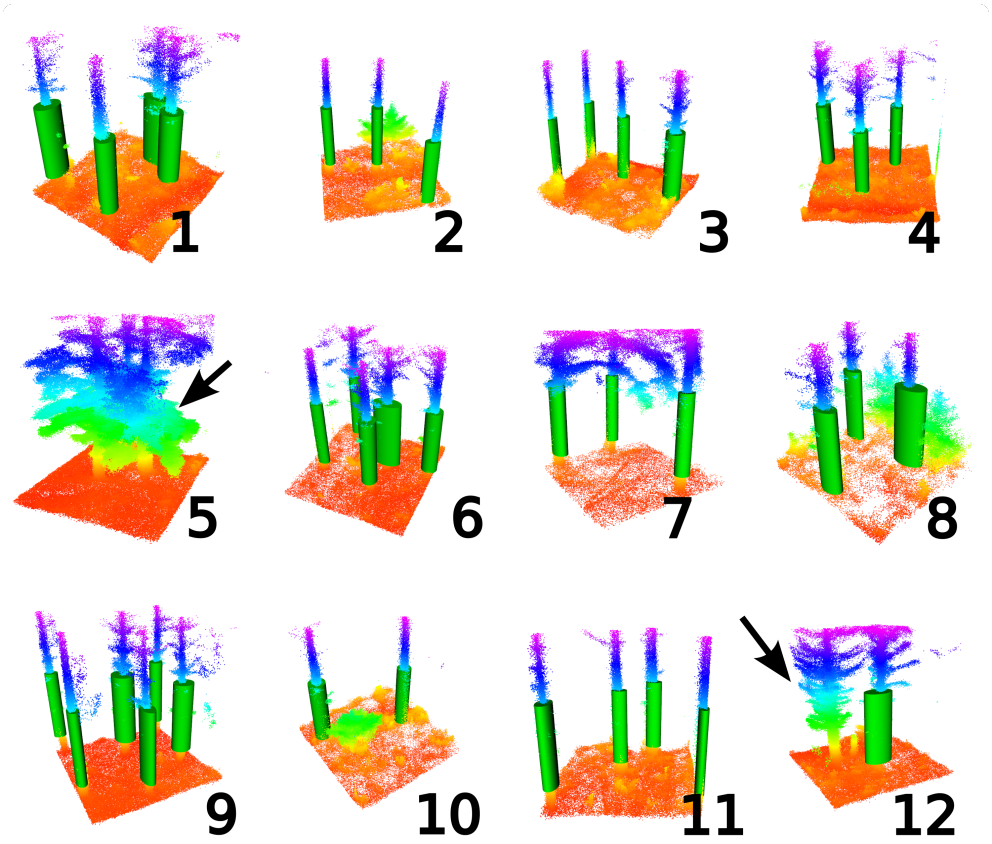
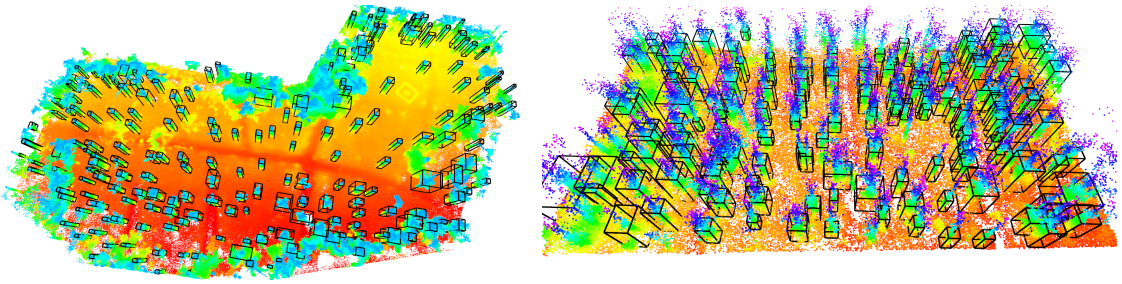


Figure 29. 6 m by 6 m point cloud patches from the testing site. Red corresponds to the lowest and purple to the highest elevation. Tree detections are shown with green cylinders. Tree detection has failed for snapshots 5 and 12 (marked with black arrows).

Table 7. Tree detection evaluation.

map	num trials	num trees	recall	precision
Figure 32b	32	4.13 ± 1.91	0.88 ± 0.2	0.92 ± 0.16
Figure 37a	33	4.78 ± 2.39	0.82 ± 0.25	0.94 ± 0.11
Figure 32c	32	3.28 ± 1.71	0.95 ± 0.14	1
total	97	4.06 ± 1.17	0.88 ± 0.12	0.95 ± 0.07

the tree detector to detect all trees inside the cropped map. The patch size roughly corresponds to the area covered by vertical Velodyne sensor frustum after the scanning maneuver. The detection procedure discarded clusters with diameter bigger than 2.5 m, fewer than 1000 points or with gravity alignment score less than 0.8 (as defined in Sec. 6); the result is shown in Figure 29. One can observe successful tree detections even in the presence of vegetation. Heavy clutter in the scene, such as in patch five and patch twelve, causes the detector to reject segmented tree clusters, incorrectly producing false negatives. In case no trees are detected close to the target, the harvester resorts to blind grabbing at the target location (known *a priori* from the map). We evaluated the detection accuracy on three different forest patches (the ones shown in Figure 32b, Figure 32c and Figure 37a). For evaluation, we selected 6 m by 6 m patches (about 30 of them) at random from each map and run the tree detection on them. The true number of trees was determined manually by inspecting the point cloud patch. The quantitative evaluation is shown in Table 7. The fail cases and extended analysis for the tree detector are presented in the Appendix A.2.



(a) Mixed forest with detected trees **(b)** Spruce forest with detected trees

Figure 30. Maps and tree detections for two forest patches. Detected trees are denoted with black bounding boxes. Red/yellow color corresponds to the lowest elevation and blue/purple to the highest. ***Left:*** Forest with both evergreen and deciduous trees. Note the heavy clutter towards the edges of the map. ***Right:*** Forest with mainly spruce trees.

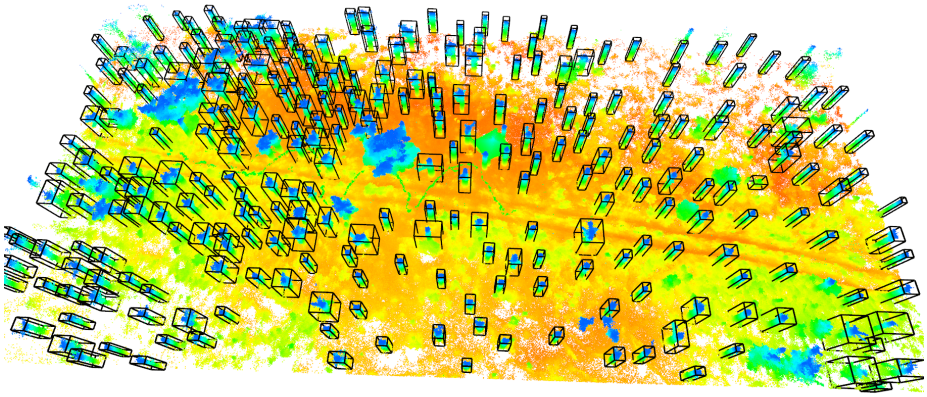


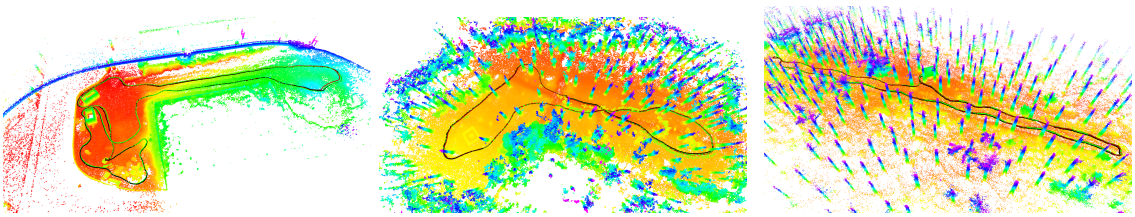
Figure 31. Forest patch and detected trees denoted with black bounding boxes. The forest patch consists of deciduous trees.

9.3.1. Tree Detector for Mission Planning

In this section, we present the use of a tree detector as a mission planner. Although the tree detector was designed to be used on small local maps, it can also be used to extract tree coordinates for clear-cutting missions or as a mission planning aid. We ran the tree detection offline on three forest patches of different styles, all shown in the figures below. The ground plane was filtered out, and the procedure presented in Sec. 6 was used. We manually identify the trees in each point cloud to obtain the actual number of trees.

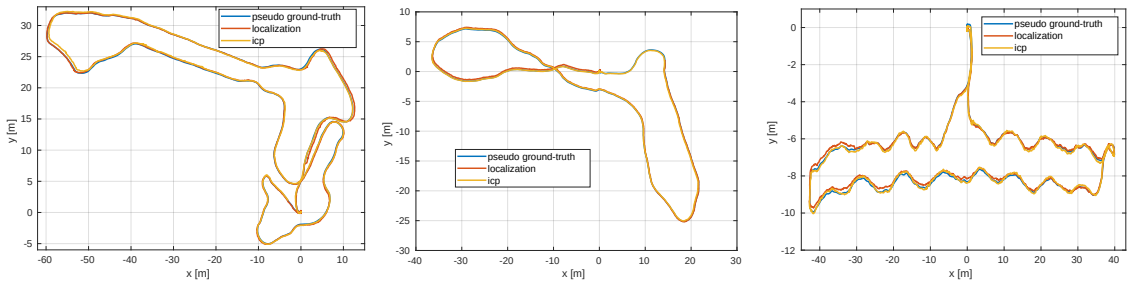
Figure 30a shows the first forest patch together with tree detections. The patch has about 246 (both evergreen and deciduous) trees, and the tree detector correctly detected 210 of them. There were 14 false positives, and 36 trees were not detected (false negatives), which amounts to a precision of 94 % and recall rate of about 85 %. The point cloud of the second forest patch and detected trees are shown in Figure 30b. In total, there are 163 (spruce) trees, and the tree detector correctly detected 143 of them. There were six false positives, and 20 trees were not detected, which amounts to a precision of 96 % and recall rate of about 88 %. The last forest patch is shown Figure 31. The Forest patch has about 285 trees (all deciduous trees). The tree detector detected in total 266 trees. Four detections were false positives, whereas the number of false negatives was 19. These numbers give the tree detector precision of 98 % and recall rate of 93 %. For a pointcloud with 26 million points, it takes our method about 300 s to complete.

We want to conclude that using our tree detector as a mission planner may yield inferior results than other methods. However, the strong points of our method are that it is lightweight, available as open-source, and can be used online on the robot. For tree detection on large forest areas,



(a) Testfield **(b)** Forest 1 **(c)** Forest 2

Figure 32. The trajectories from the localization field experiments overlaid with the map.



(a) Testfield **(b)** Forest 1 **(c)** Forest 2

Figure 33. The trajectories from the localization field experiments viewed from above. Forest 1 and Forest 2 are datasets from two different forests.

one is better off using some of the more advanced approaches, such as [Burt et al., 2019] which achieve recall rates up to 96 % on tropical forest datasets. One should note, however, that the method is computationally expensive; it takes two days to process a map with 17 million points on a computer with 24 cores. Some of the best results that we have seen have been obtained using the forestry module inside the LiDAR360 commercial software for pointcloud processing. Unfortunately, LiDAR360 is not available for free.

9.4. Localization

We evaluated the localization of the proposed sensor module running Google Cartographer SLAM in three different terrains: two forest patches and our testing field. We collect the dataset with the sensor module presented in 3.2, and we run the SLAM offline, which produces a consistent, optimized map. Subsequently, we set the Cartographer in the localization mode and tune it such that both the LIDAR odometry and loop closures run in real-time. Note that Cartographer running in the localization mode does not try to build a global map (see [Hess, 2017]). We are interested in quantifying performance degradation when running in real-time localization mode without joint map and trajectory bundle adjustment. We localize in a previously built map and compare the localized pose against the bundle adjusted trajectory (pseudo-ground-truth). Paths overlaid inside the map can be seen in Figure 32 while the top view plots of the trajectories can be seen in Figure 33.

We evaluate the localization error as the euclidean distance between the corresponding points on the localized trajectory and pseudo ground-truth trajectory. For the test field, we obtain a translational error of 0.11 ± 0.09 meters; the trajectory length was 255.8 m. Trajectory length for the first forest dataset was 158.37 m, and Cartographer localizes with the error of 0.17 ± 0.28 meters. Inside, the second forest trajectory was 183.27 meters long, and the localization error was 0.25 ± 0.28 meters. The presented evaluation is biased since the cartographer (the same method) is used for both map building and localization. Hence, we evaluate localization performance using a different method, the ICP (using point to plane error metric). The ICP is used to register LIDAR scans in a map built using the cartographer (we do not have the ground truth). Subsequently, the translational

(a) Wangen and der Aare (CH) **(b)** Forest 3

Figure 34. Locations for mapping ground truth evaluation. ***Left:*** Aerial photo of the emergency responders training ground, Wangen and der Aare. Image taken from https://www.mediathek.admin.ch/media/image/da1c1259-8d9a-430a-893d-d325de139149 ***Right:*** Forest patch where the mapping accuracy was evaluated. The forest in the photo is an old forest.

error between the ICP localized trajectory and the pseudo-ground truth from the cartographer is computed. For the first forest, the error is 0.085 ± 0.038 meters, and in the second forest, the error is 0.078 ± 0.035 meters. For the test field, we obtain an average error of 0.13 ± 0.11 between the pseud-ground truth and the ICP.

To better understand the overall accuracy, we compare the mapping and localization performance against ground-truth data in two different environments: a forest and an urban setting (emergency services training ground in Wangen an der Aare, Switzerland). Both areas are shown in Figure 34. The ground-truth data was produced using Leica's RTC 360 3D laser scanner. It is a rotating laser scanner that aligns point clouds and uses VIO to record moves from station to station for scan pre-registration automatically. For both the forest and the training ground, the RTC360 reported sub-centimeter accuracy.

We align the ground truth map and the cartographer map using the Cloudcompare software alignment tool. The aligned maps for the Wangen area are shown in Figure 35. The area has dimensions of 294 x 577 x 66 meters (W x L x H). The RMS error from the registration was 0.578 m with the average distance between the points of 0.76 ± 0.52 meters (see below for a discussion). To map the area using our proposed setup, about 10 min of walking around the site was required. To obtain the ground truth map, we made 55 scans which required a whole afternoon of work.

Besides the map quality, we also evaluate cartographer's trajectory quality (pseudo-ground truth in Figure 32 and Figure 33). To obtain the ground truth trajectory, we register laser scans in the ground truth map from the Leica scanner using the ICP. Then we align the trajectories estimate from the cartographer with the ground truth trajectory (using the transform obtained during the map alignment step). The trajectory length was 823 m and the overall error was 0.41 ± 0.28 meters, overlay visualized in Figure 38a. This level of accuracy is acceptable for navigation; however, blind grabbing might not be accurate enough, which is why our method does an extra detection step in the local map. We note that the overall errors (both map and trajectory estimation) computed are pessimistic (worst case) because of the changes in the environment between the data collection. We observed new (or differently parked) vehicles, cranes, trailers, tents, and even a large water pool left on the main road of the training ground. All of these changes in the environment negatively influence the accuracy of scan matching and make the correct data association harder.

In addition to the urban environment, we collected the ground truth in two forest patches near Wangen an der Aare in Switzerland. Again, the Leica RTC360 3D scanner was used for generating the map. The cartographer map was registered against the ground truth map using Cloudcompare

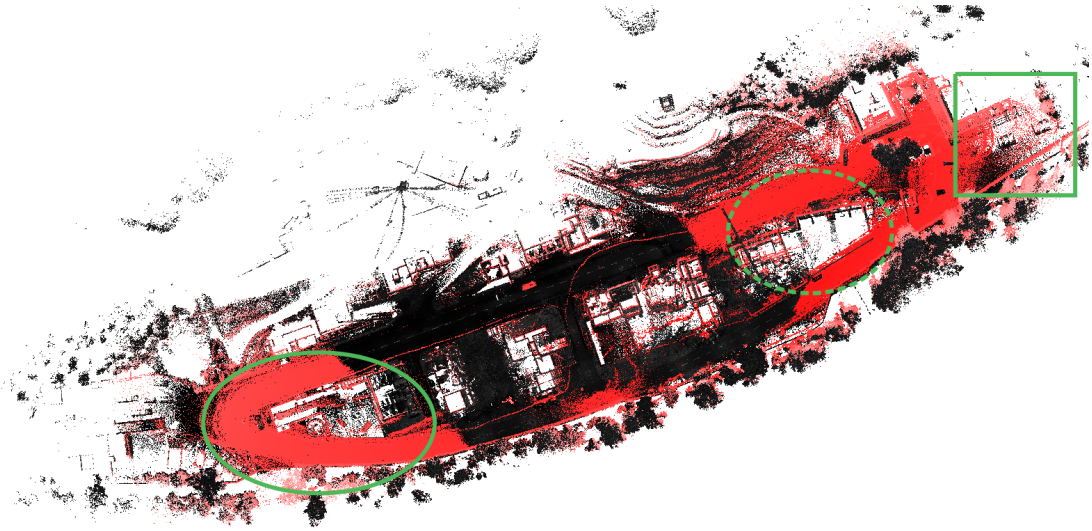
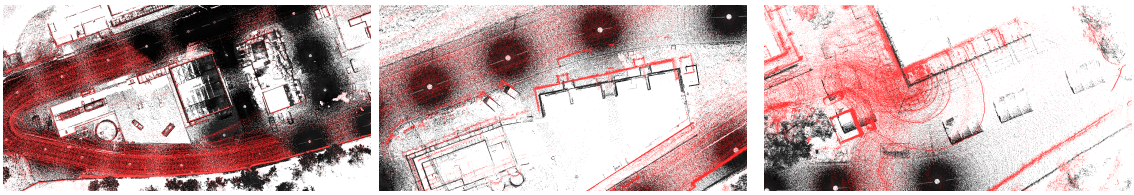


Figure 35. Map of the emergency responders training ground, Wangen an der Aare, Switzerland. The ground truth map is shown in black(from Leica RTC360), and in red, the cartographer map is shown. The green circle shows an example of good alignment, the dotted circle shows an example of bad alignment, and the green square shows an example of changes between two mapping sessions. The data was collected a couple of weeks apart.

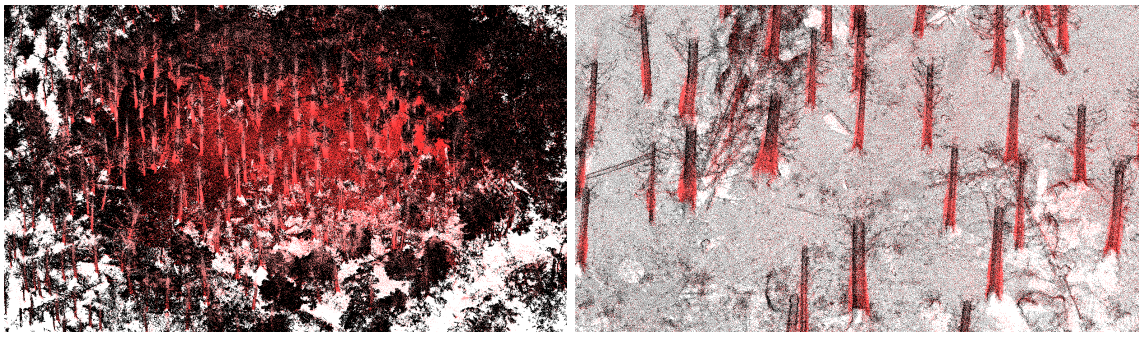


(a) Good alignment **(b)** Poor alignment **(c)** Changes between data collection

Figure 36. Close up view of encircled areas from Figure 35. Ground truth is shown in black and cartographer map shown in red. ***Left:*** Example of good alignment. Note how building edges from different maps snap well onto each other. ***Middle:*** Example of poor alignment. Note how the building walls are far apart from each other. ***Right:*** Changes between the dataset collections. Note the black tents in the ground truth map and the red firefighting trailer in the cartographer map.

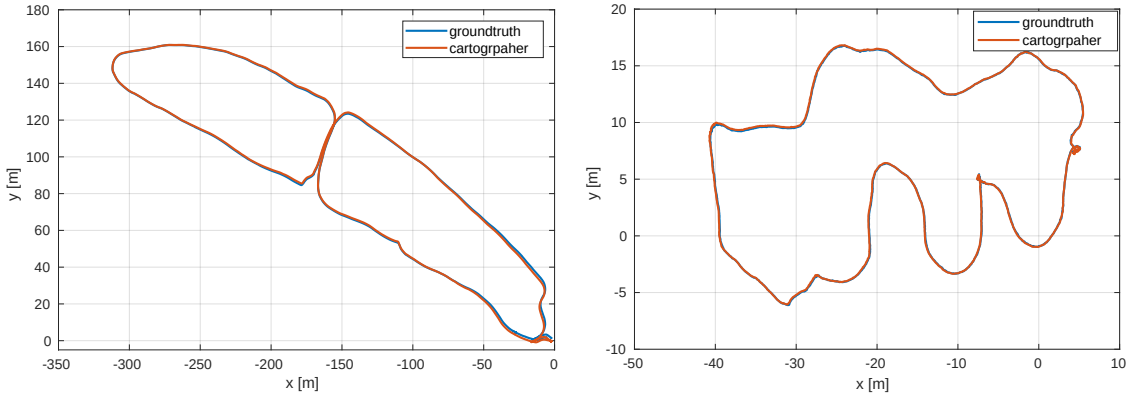
software. For the first forest patch, the RMS registration error was 0.042 m, and the distance between registered point clouds was 0.06 ± 0.05 meters. Dimensions of the forest area are 243 x 257 x 43 meters (W x L x H). The aligned maps are shown in Figure 37. The close-up shot shown in Figure 37b shows a snug fit between the aligned maps. One can also notice how the cartographer map is significantly noisier than the ground truth map from the RTC scanner. This comes to no surprise since RTC360 comes with more than an order of magnitude more accurate distance measurements (less than 5 mm at 40 meters) compared to the Ouster range sensor (5 cm at 40 meters).

ICP was used to register LIDAR scans against the ground truth map and create a ground truth trajectory. We compare it to the pseudo ground truth from the cartographer. Overall translational error was 0.05 ± 0.03 meters and the overlaid trajectories are shown in Figure 38b. This is an order of magnitude lower error compared to the urban environment, which can be explained by the fact that we collected the data in succession, and hence both maps look the same. This allows for better scan registration. Additionally, the forest areas are smaller, which also leads to better accuracy. We verified the obtained error in another forest patch with younger trees (map size 100 x 96 x 17 meters). Comparing the map with the ground truth map, we obtained similar numbers: the RMS registration error was 0.078 m, the distance between the registered maps was 0.12 ± 0.1 meters.



(a) Registered maps **(b)** Close up of the alignment

Figure 37. Aligned forest maps. Ground truth map shown in black color where the cartographer map has been shown in red color. ***Left:*** Top view of the aligned maps. ***Right:*** Close up image of the aligned maps. Note how tree trunks nicely fit together.



(a) Wangen training site **(b)** Forest 3 (near Wangen)

Figure 38. Top view trajectory estimate from cartographer (brown) aligned with the ground truth trajectory (blue). ***Left:*** Trajectory estimates for data collection at Wangen training site. Note how the error is bigger in the area where maps in Figure 35 are not well aligned. ***Right:*** Trajectory estimates for data set collection in the forest 3.

The error between the ground truth trajectory and pseudo-ground truth from the cartographer was 0.04 ± 0.03 meters over the 100.34 m distance traveled (images and a map of the forest patch omitted for brevity).

Both cartographer's localization and the ICP can run in real-time on the robot, however ICP consumes less CPU. The comparisons between cartographer's localization mode and pure ICP based localization suggest the superior performance of ICP. Hence, we would like to conclude that one could also use ICP for localization during harvesting missions. The authors will switch to the ICP based localization in the future. Our ICP implementation is available as open-source for the community[9].

9.5. Fully integrated system

This section presents snapshots of a fully integrated system performing a harvesting mission. We tested our system on a test field where HEAP was commanded to grab "tree trunks". One tree trunk is composed of three wooden logs strapped together such that they can stand straight up

[9] https://github.com/leggedrobotics/icp_localization

Figure 39. HEAP approaching wooden logs at our testing field. We do not show the rest of the maneuver for the sake of brevity. The reader is encouraged to watch the video.

(see the video or Figure 39). We show the approaching sequence on our test field in Figure 39 and omit the rest of the maneuver for the sake of brevity since the full maneuver is already shown in Figure 40. In addition to our test field, we have performed the second set of experiments in a small forest alley in a real forest with fully grown adult trees. We show snapshots of 1 operational cycle between grasping two trees in Figure 40. The reader is encouraged to watch a video accompanying this submission since it offers a better insight[10].

It takes about 1-2.5 minutes for the machine to complete the entire cycle, depending on how far the cabin has to turn and how far it has to drive. The cabin turns and driving are tuned conservatively (slow), and the time could be improved drastically by speeding them up. The proposed system was able to run without any changes in both the test field and the forest. Furthermore, we were able to successfully detect both tree trunks and wooden logs and perform a grab. A human operator was often inside the cabin for safety.

10. Summary

We present the first (to the best of our knowledge) demonstration of a full-sized harvester performing autonomous precision tree grabbing. We do not investigate tree cutting, which could be done with an appropriate harvesting tool. Components off our system have been evaluated individually and integrated into a complete autonomous system. We show evaluation of the control system, approach-pose planner, mapping accuracy, localization accuracy, point cloud to elevation map conversion algorithm, and the tree detector.

A lightweight and versatile sensor module is presented; it can be used for mapping and later mounted on the harvester for localization. The sensor module and SLAM system together generate an *a priori* map of the mission area. Subsequently, the HEAP harvester can localize itself at mission time, which enables it to navigate under the forest canopy without relying on a GPS signal.

Our system can plan approach-poses in confined spaces, and it can negotiate challenging terrain by combining the chassis-balancing and path-following controllers. Our planner relies on traversability maps that we estimate from elevation maps, which, in turn, we calculate directly from point clouds using our conversion algorithm. We rely on a human expert to specify target trees for harvesting. To combat localization error and enable precision harvesting, we plan grasping poses in the local frame, using a geometric detection algorithm that operates on the laser point-cloud. Lastly, we make parts

[10] https://youtu.be/1FLD0djPFgU

Figure 40. Snapshots of an operational cycle between grabbing two trees. HEAP re-positioning the base close to the tree (snapshots 1-6) followed by scanning the environment and building a local map (snapshots 7-9). Finally, HEAP grabs the tree by extending the arm (snapshots 10-12) and retracting it (snapshots 13-15). After retracting the arm, the harvester is ready to grab the next tree and whole cycle repeats.

of our planning[11], mapping[12], localizing[13] and tree-detecting[14] software stack open source for the community.

11. Discussion & Outlook

Each of the modules (e.g., planning, mapping) has been first tested and benchmarked individually before integrating them into a complete precision harvesting mission. For example, we first tested planning and control with RTK GPS before adding the SLAM system. Stepwise integration was

[11] https://github.com/leggedrobotics/se2_navigation
[12] https://github.com/ANYbotics/grid_map/tree/master/grid_map_pcl
[13] https://github.com/leggedrobotics/icp_localization
[14] https://github.com/leggedrobotics/tree_detection

essential to isolate problems. We also note that having a simulation environment was beneficial, since we could first test the functionality in simulation and then focus on tuning on the machine. State-machine behavior, for example, can be almost completely verified in simulation. Another lesson learned from this work was that state machines get complicated when they must include recovery behavior (e.g. re-plan if no solution is found). In the future, it would be beneficial to look into using behavior trees [Winter et al., 2010], which are typically more compact and implicitly integrate recovery behaviors.

Experimental verification of our approach during the wet season was unfortunate, and the biggest problem occurring was that the machine would occasionally get stuck in the mud. This problem was somewhat mitigated by using chains and putting wooden logs and branches in the mud to improve traction. We do not see this as a limitation of our system since forestry operations typically find a place during dry seasons.

We point out that pure, geometric localization (or detection) can fail in a real forest. For example, a crowd of humans observing the machine can look very similar to tree trunks in a point cloud, a situation that, on one occasion, caused the SLAM module to generate spurious pose estimates. Furthermore, we had a few cases of sun rays directly shining on our camera, which condition caused the visual odometry to diverge. Subsequently, the whole state estimation pipeline diverged despite the pure LIDAR odometry working well. This problem seeks an approach that can adequately detect sensor degradation to ensure robust mapping and localizing in all cases. We also believe that it is important to perform a quantitative evaluation of different SLAM systems under similar conditions in forest areas. This procedure would involve testing different approaches (e.g. LOAM variants, Cartographer, ICP) in varied forest styles with a high-accuracy ground-truth map (e.g. from the 3D scanner). The research community could thus focus the development efforts on the most promising approach.

An immediate improvement to the localization system wold be to use ICP as opposed to Cartographer's localization mode (see Section 9.4). Other improvements we are planning will focus on utilizing complementary sensors where applicable and implementing health-checking systems to mitigate state-estimating problems. For example, one could run both vision and LIDAR-based SLAM to increase robustness since these two modalities complement each other [Khattak et al., 2020]. Furthermore, traversability estimation should use visual information, since pure geometry can be misleading in natural environments.

Robust handling of heavy clutter would be especially beneficial, since such situations constitute the primary conditions of tree-detection failure. Detecting trees in the local map could also benefit from using visual information. In case of thick vegetation or heavy branch clutter, one could also look into using radar sensors for determining a grabbing pose on a tree stem.

We plan to improve the approach-pose planner by incorporating footprint adaptation and adjusting the number of generated approach-poses based on the environment, as noted in Section 8. Furthermore, it would be interesting to benchmark different planners (e.g., Probabilistic Roadmap (PRM) or hybrid A* algorithm [Dolgov et al., 2010]). Additionally, the implementation of approach-pose generation can be parallelized. For tree felling, the arm planner can be enhanced to include tree geometry, once the gripper is holding a tree trunk. Lastly, the tree trunk weight could be included in the arm controller to improve the arm-plan tracking.

Besides improving the mission-execution and motion planners, an exciting area to pursue is to develop a more intelligent mission planner. Instead of just relying on an operator-provided order of execution, one could optimize some objective (e.g., minimal time). Such a planner becomes especially relevant if one wants to maximize efficiency in a scenario involving multiple harvesters.

Apart from algorithmic improvements, one crucial aspect for future forestry missions is to have rugged sensors. We used an umbrella to protect our sensor module from water during the experiments (see Figure 3, in the back of the machine); however, this is not a viable solution for a fully autonomous harvester operating in a harsh environment. Furthermore, the sensors must be protected from branches, vegetation clutter, and branch debris falling onto the machine. Since harvester machines need to deal with clutter and vegetation, it is advisable to have redundant sensors such that localization can be robust to occlusions.

Incorporating the improvements outlined above will bring us one step closer to fully automated precision forestry that will increase yields and relieve humans from tedious labor.

Acknowledgement

This research was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 852044 and through the SNSF National Centre of Competence in Digital Fabrication (NCCR dfab). We would also like to thank Alexander Reske for his help with Leica RTC 360 3D laser scanner during the data collection. We also thank to Silvere and Simo Gröhn for their help with Cartographer.

A. Appendix

A.1. Point Cloud to Elevation Map Conversion

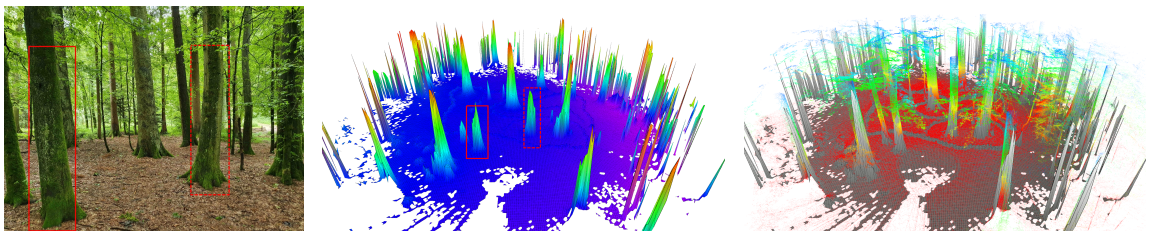
We analyze how the point cloud conversion algorithm presented in Section 5.1 reacts to clutter and vegetation. The data was collected with a handheld sensor, and we build small maps of forest patches. The forest patches vary in dimension between 17x18 meters up to 29x32 meters (width x length). We convert the point cloud to the elevation map and show the results below. Unfortunately, we cannot provide rigorous quantitative analysis since the ground truth (actual ground elevation) is extremely hard to collect in the presence of vegetation. Furthermore, it is hard to define the ground truth since there are cases where we want more than just ground information in the elevation map (e.g., trees, stumps). However, looking at the images and maps presented below, one can draw some conclusions about the conversion algorithm's behavior.

All the elevation maps shown below have a resolution of 10 cm. One can observe (especially from Figure 42) that thin tree trunks present a problem for the elevation map and are often not well captured in it. One could increase the map's resolution to capture them better; however, there is a trade-off between detail and computation time. Furthermore, to avoid holes and artifacts, one needs denser point clouds with the increasing elevation map resolution (smaller cell size). Dense branches and low and medium clutter do not seem to pose a big problem, and the resulting elevation maps have no or low artifacts. Thick tree trunks (30 cm diameter and more) are visible well in the map, and we did not experience any problems with the tree canopy or branches. This holds as long as forest ground is captured in the point cloud. One limitation of the approach is the ability to deal with heavy clutter (e.g., Figure 46). Indeed thick vegetation and branches touching the ground will often appear as blobs in the elevation map.

In Figure 48 we illustrate the influence of a cluster tolerance parameter on the elevation map. Both maps correspond to the cluttered forest patch shown in Figure 46a. One can note that the lower value of cluster tolerance better filters out the clutter and recovers the ground elevation. However, there is a trade-off since it also filters smaller trees out of the map (denoted with a red rectangle). Large tree trunks remain unaffected; in case of a cluttered environment and larger tree trunks, smaller values are recommended. In a case the point cloud is very dense, smaller values usually yield better results.

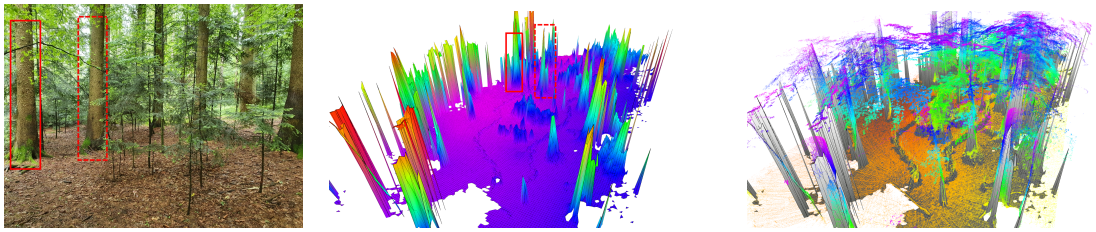
A.2. Tree Detection from Local Maps

In this section, we present some tree detection failure cases on the local map patches. Example failure cases are shown in Figure 49. The most common cause of failure is a false negative when no tree is detected, and there is one in the map. False-negative examples can be seen in snapshot 1,2,3,4,5,7,8,9,10,11 and 12. What all these fail cases have in common is the presence of vegetation and clutter in the scene. Some patches are so cluttered (e.g., 1 and 12) that it is extremely challenging for a human to find the trees in the scene. Since our approach extracts Euclidean clusters, any branches or canopy that touches other trees presents a problem. Touching canopies (from two different trees) cause the algorithm to extract two or more trees (together with all the branches)



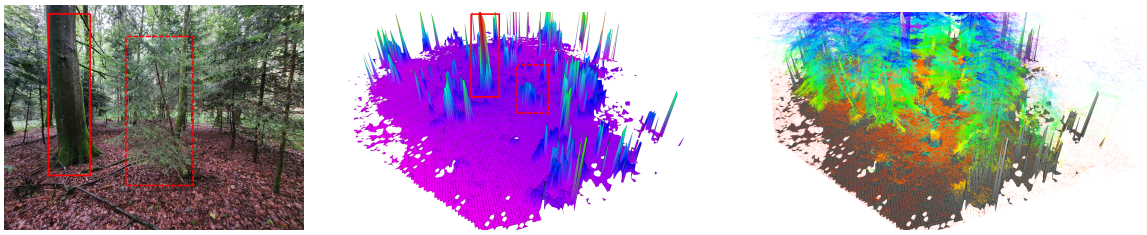
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 41. Forest scene with large tree trunks, no clutter and low branch density. The elevation map nicely captures the tree trunks and the forest ground.



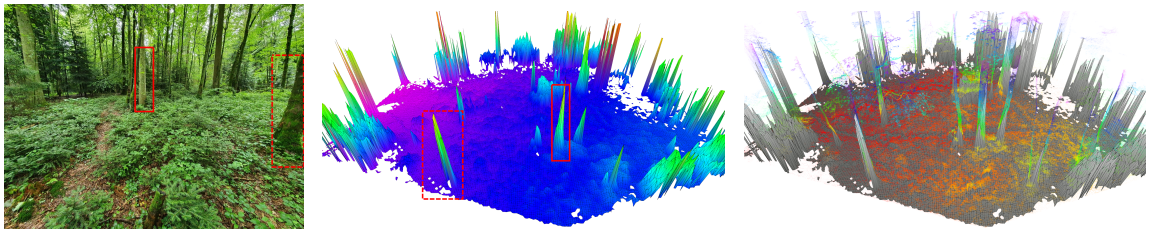
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 42. Scene with no clutter, but higher branch density compared to the previous scene. Besides the big tree trunks many smaller trees with trunk diameter less than $10\,\mathrm{cm}$ are present. One can clearly see a limitation of our algorithm, since many of the smaller trees are not well captured in the map at $10\,\mathrm{cm}$ resolution.



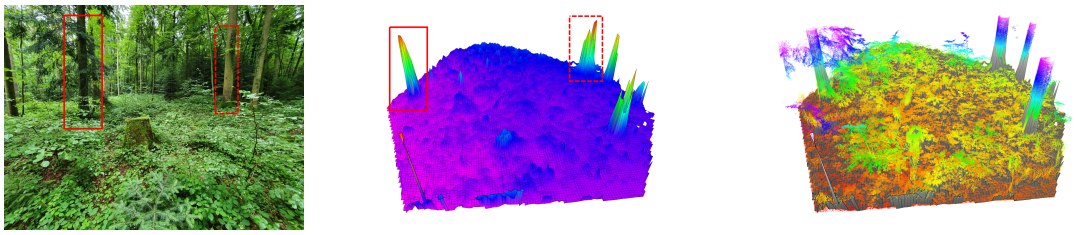
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 43. Cluttered scene without vegetation. Compared to the Figure 42, tree trunks in this scene are thicker and better captured in the elevation map. High branch density does not seem to pose the problem for the conversion algorithm as long as there is enough ground data in the point cloud.



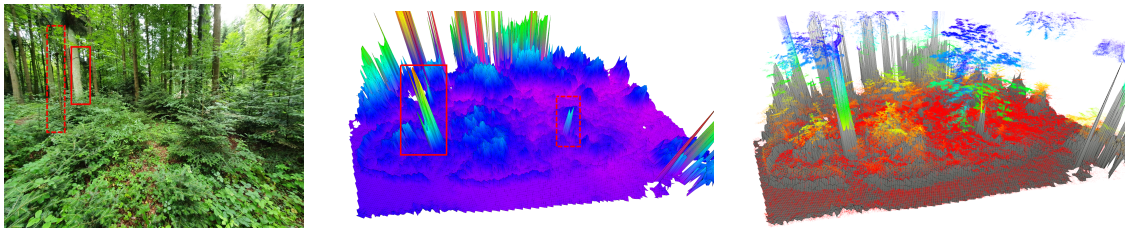
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 44. Scene with low vegetation and low clutter. One can see that the elevation map does not contain any artifacts. The vegetation causes the map to be somewhat rougher compared to the forest ground without the vegetation.



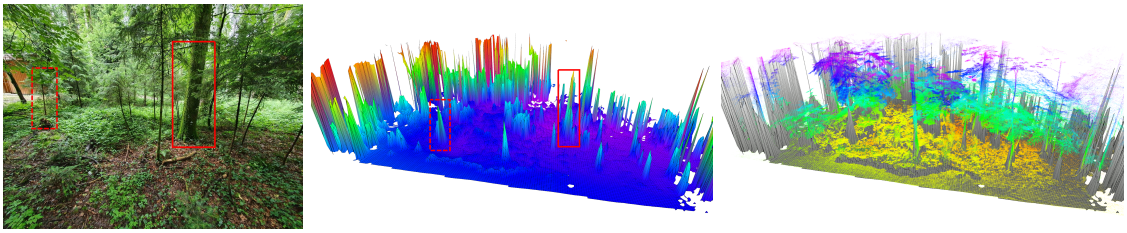
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 45. Medium clutter forest patch with ground vegetation. We can see that the surface of the elevation map looks somewhat rougher compared to the Figure 44. The algorithm successfully captures the tree trunks and filters out some thin trees.



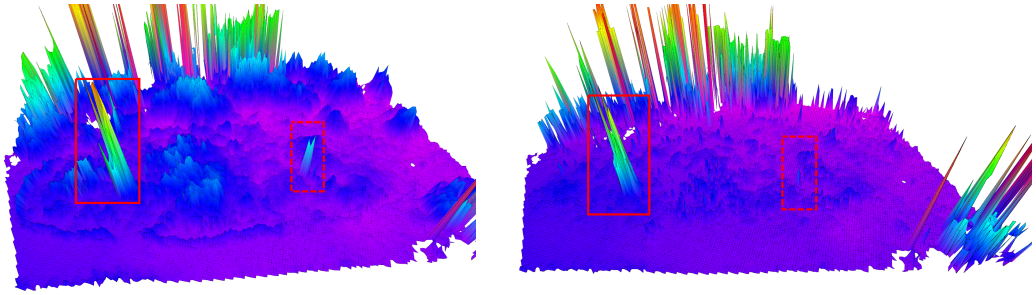
(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 46. High clutter scene with lots of vegetation. We can see that the thick vegetation corrupts the elevation map and causes bumps and roughness in it. Thick tree trunks are still well visible in the map.



(a) Forest patch **(b)** Elevation map **(c)** Elevation map + point cloud

Figure 47. Combined scene with medium vegetation clutter and trees. Branches reaching all the way to the ground effectively inflate the trunk of the tree which can be seen in the middle of the elevation map (big blobs to the left of the tree trunk).



(a) Map computed with higher cluster tolerance **(b)** Map computed with lower cluster tolerance

Figure 48. Elevation maps computed using our point cloud to elevation map conversion algorithm with different cluster tolerance parameter. One can observe that the lower cluster tolerance better filters out the vegetation and clutter.

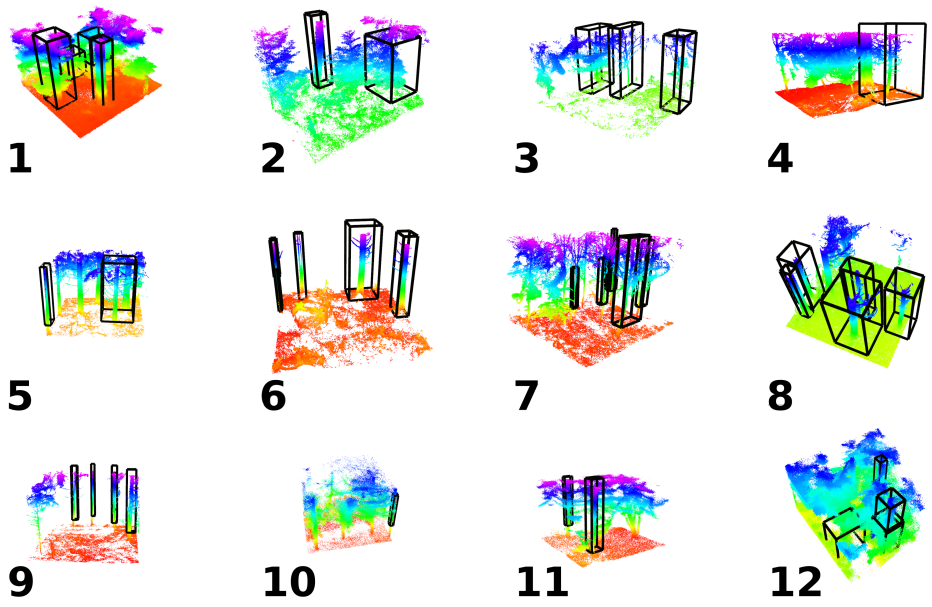


Figure 49. 6 m by 6 m point cloud patches where detection algorithm failed. Red corresponds to the lowest and purple to the highest elevation. Tree detections are shown with transparent boxes. The patches come from three different maps that were used for detection accuracy evaluation (see Sec. 9.3.

as one cluster. Subsequently, the whole blob of points will be rejected because it is either too wide or the gravity alignment is not vertical enough. Ultimately, since our detection approach is purely geometric, it runs into limitations in the presence of high clutter and vegetation. The clutter problem could be mitigated by changing the cluster tolerance in the Euclidean cluster extraction phase. If the point cloud is very dense smaller values (less than 0.1 m) usually yield better results. Values that are too small will result in a single tree being split into multiple clusters, whereas large values will result in multiple trees merged into a single cluster. This calls for a detection procedure involving a tree model, which would help filter out the branch clutter (e.g., RANSAC). A learning-based approach could also be an option. Alternatively, one could use a different sensor modality that will penetrate through the clutter and allow for direct tree trunk detection (e.g., a radar).

The second failure mode is mistaking a branch for a tree or detecting the same tree twice (false positive). Examples of such behavior can be seen in snapshots 6 and 12. This type of failure is not as common as the false negative. One could extend our approach with simple sanity checks (e.g., if two clusters have similar x and y coordinates than most likely they belong to the same tree) to increase the performance. Another option would be to filter out branches and vegetation, which can be mistaken for a tree. A visual sensor would also help in this case since it is relatively easy to distinguish between vegetation and a tree trunk for a human.

ORCID

Edo Jelavic https://orcid.org/0000-0002-1757-5543
Dominic Jud https://orcid.org/0000-0001-8445-6344
Pascal Egli https://orcid.org/0000-0002-4432-1047
Marco Hutter https://orcid.org/0000-0002-4285-4990

References

API (2021). Api heavy machinery industry, measurement and calibration services. https://services.apimetrology.com/manufacturing-industry-measurement-calibration-services/heavy-machinery/.

Ayrey, E., Fraver, S., Kershaw Jr, J. A., Kenefic, L. S., Hayes, D., Weiskittel, A. R., and Roth, B. E. (2017). Layer stacking: a novel algorithm for individual forest tree segmentation from lidar point clouds. *Canadian Journal of Remote Sensing*, 43(1):16–27.

Babin, P., Dandurand, P., Kubelka, V., Giguère, P., and Pomerleau, F. (2019). Large-scale 3d mapping of sub-arctic forests. *arXiv preprint arXiv:1904.07814*.

Bellicoso, C. D., Gehring, C., Hwangbo, J., Fankhauser, P., and Hutter, M. (2016). Perception-less terrain adaptation through whole body control and hierarchical optimization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 558–564. IEEE.

Bryson, M. (2017). 7.1 pointclouditd: A software package for individual tree detection and counting. *Deployment and integration of cost-effective high resolution remotely sensed data for the Australian forest industry*, page 154.

Burt, A., Disney, M., and Calders, K. (2019). Extracting individual trees from lidar point clouds using treeseg. *Methods in Ecology and Evolution*, 10(3):438–445.

Carius, J., Wermelinger, M., Rajasekaran, B., Holtmann, K., and Hutter, M. (2018). Deployment of an autonomous mobile manipulator at mbzirc. *Journal of Field Robotics*, 35(8):1342–1357.

Chen, S. W., Nardari, G. V., Lee, E. S., Qu, C., Liu, X., Romero, R. A. F., and Kumar, V. (2020). Sloam: Semantic lidar odometry and mapping for forest inventory. *IEEE Robotics and Automation Letters*, 5(2):612–619.

Chen, X., Jiang, K., Zhu, Y., Wang, X., and Yun, T. (2021). Individual tree crown segmentation directly from uav-borne lidar data using the pointnet of deep learning. *Forests*, 12(2):131.

Choudhry, H. and O'Kelly, G. (2018). Precision forestry: A revolution in the woods. Retrieved November 27 2019 from https://www.mckinsey.com/industries/paper-forest-products-and-packaging/our-insights/precision-forestry-a-revolution-in-the-woods.

Cui, J. Q., Lai, S., Dong, X., Liu, P., Chen, B. M., and Lee, T. H. (2014). Autonomous navigation of uav in forest. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 726–733. IEEE.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The international journal of robotics research*, 29(5):485–501.

Eidson, J. C. (2006). *Measurement, control, and communication using IEEE 1588*. Springer Science & Business Media.

Fankhauser, P., Bloesch, M., and Hutter, M. (2018). Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3019–3026.

Fankhauser, P. and Hutter, M. (2016). A universal grid map library: Implementation and use case for rough terrain navigation. In *Robot Operating System (ROS)*, pages 99–120. Springer.

Forestry Focus (2018). Thinning. Retrieved November 26 2019 from https://www.forestryfocus.ie/growing-forests-3/establishing-forests/thinning/.

Furgale, P., Rehder, J., and Siegwart, R. (2013). Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286. IEEE.

Gawel, A., Blum, H., Pankert, J., Krämer, K., Bartolomei, L., Ercan, S., Farshidian, F., Chli, M., Gramazio, F., Siegwart, R., et al. (2019). A fully-integrated sensing and control system for high-accuracy mobile robotic building construction. *arXiv preprint arXiv:1912.01870*.

Georgsson, F., Hellström, T., Johansson, T., Prorok, K., Ringdahl, O., and Sandström, U. (2005). *Development of an Autonomous Path Tracking Forest Machine: a status report*.

Giftthaler, M., Farshidian, F., Sandy, T., Stadelmann, L., and Buchli, J. (2017). Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3411–3417. IEEE.

Harrison, A. and Newman, P. (2011). Ticsync: Knowing when things happened. In *2011 IEEE International Conference on Robotics and Automation*, pages 356–363. IEEE.

Hawkinson, D. (2017). Forestry industry. Retrieved January 22 2021 from https://forestresources.org/resources/woods-to-mill/item/870-addressing-the-labor-shortage-in-the-forestry-industry.

Hellström, T., Lärkeryd, P., Nordfjell, T., and Ringdahl, O. (2008). Autonomous forest machines: Past present and future.

Hess, W. (2017). Cartographer online documentation. Retrieved December 06 2019 from https://google-cartographer.readthedocs.io/en/latest/#.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE.

Holopainen, M., Vastaranta, M., and Hyyppä, J. (2014). Outlook for the next generation's precision forestry in finland. *Forests*, 5(7):1682–1694.

Hutter, M., Leemann, P., Hottiger, G., Figi, R., Tagmann, S., Rey, G., and Small, G. (2016). Force control for active chassis balancing. *IEEE/ASME Transactions on Mechatronics*, 22(2):613–622.

Hutter, M., Leemann, P., Stevsic, S., Michel, A., Jud, D., Hoepflinger, M., Siegwart, R., Figi, R., Caduff, C., Loher, M., et al. (2015). Towards optimal force distribution for walking excavators. In *2015 international conference on advanced robotics (ICAR)*, pages 295–301. IEEE.

Hyyti, H., Lehtola, V. V., and Visala, A. (2018). Forestry crane posture estimation with a two-dimensional laser scanner. *Journal of Field Robotics*, 35(7):1025–1049.

item (2021). item industrietechnik gmbh. https://www.item24.com/.

Johns, R. L., Wermelinger, M., Mascaro, R., Jud, D., Gramazio, F., Kohler, M., Chli, M., and Hutter, M. (2020). Autonomous dry stone. *Construction Robotics*, 4(3):127–140.

Jud, D., Kerscher, S., Wermelinger, M., Jelavic, E., Egli, P., Leemann, P., Hottiger, G., and Hutter, M. (2021). Heap-the autonomous walking excavator. *Automation in Construction*, 129:103783.

Jud, D., Leemann, P., Kerscher, S., and Hutter, M. (2019). Autonomous free-form trenching using a walking excavator. *IEEE Robotics and Automation Letters*, 4(4):3208–3215.

Jukka Hämekoski (2016). Practical and universal tree harvester to increase the possibilities of logging machines. Retrieved July 28 2021 from https://www.ins-news.com/fi/100/779/1403/Practical-and-universal-tree-harvester-to-increase-the-possibilities-of-logging-machines-.htm.

Kalmari, J., Backman, J., and Visala, A. (2014). Nonlinear model predictive control of hydraulic forestry crane with automatic sway damping. *Computers and Electronics in Agriculture*, 109:36–45.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894.

Khattak, S., Nguyen, H., Mascarich, F., Dang, T., and Alexis, K. (2020). Complementary multi–modal sensor fusion for resilient robot pose estimation in subterranean environments. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1024–1029. IEEE.

Kim, S., Jang, K., Park, S., Lee, Y., Lee, S. Y., and Park, J. (2019). Whole-body control of non-holonomic mobile manipulator based on hierarchical quadratic programming and continuous task transition. In *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 414–419. IEEE.

Komatsu Corporation (2019). Harvesters - product catalogue. Retrieved November 26 2019 from https://www.komatsuforest.com/forest-machines/our-wheeled-harvesters.

Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., and Vincent, R. (2010). Efficient sparse pose adjustment for 2d mapping. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 22–29. IEEE.

Kuwata, Y., Teo, J., Karaman, S., Fiore, G., Frazzoli, E., and How, J. (2008). Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7166.

La Hera, P. M., Mettin, U., Westerberg, S., and Shiriaev, A. S. (2009). Modeling and control of hydraulic rotary actuators used in forestry cranes. In *2009 IEEE International Conference on Robotics and Automation*, pages 1315–1320. IEEE.

Li, Q., Nevalainen, P., Peña Queralta, J., Heikkonen, J., and Westerlund, T. (2020). Localization in unstructured environments: Towards autonomous robots in forests with delaunay triangulation. *Remote Sensing*, 12(11):1870.

Liao, F., Lai, S., Hu, Y., Cui, J., Wang, J. L., Teo, R., and Lin, F. (2016). 3d motion planning for uavs in gps-denied unknown forest environment. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 246–251. IEEE.

Lindroos, O., Ringdahl, O., La Hera, P., Hohnloser, P., and Hellström, T. H. (2015). Estimating the position of the harvester head–a key step towards the precision forestry of the future? *Croatian Journal of Forest Engineering: Journal for Theory and Application of Forestry Engineering*, 36(2):147–164.

Magnus Gustavsson (2016). The radio-controlled bio-energy harvester forest ebeaver. Retrieved July 28 2021 from https://www.ebeaver.se/En/Default.aspx.

Menzi Muck (2020). Menzi muck construction equipment accessories. Retrieved March 30 2021 from https://www.menzimuck.com/en/product-groups/menzi-construction-equipment-accessories/.

Miettinen, M., Ohman, M., Visala, A., and Forsman, P. (2007). Simultaneous localization and mapping for forest harvesters. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 517–522. IEEE.

Mikhaylov, M. and Lositskii, I. (2018). Control and navigation of forest robot. In *2018 25th Saint Petersburg International Conference on Integrated Navigation Systems (ICINS)*, pages 1–2. IEEE.

Morales, D. O., Westerberg, S., La Hera, P., Mettin, U., Freidovich, L. B., and Shiriaev, A. S. (2011). Open-loop control experiments on driver assistance for crane forestry machines. In *2011 IEEE International Conference on Robotics and Automation*, pages 1797–1802. IEEE.

Morita, M., Nishida, T., Arita, Y., Shige-eda, M., di Maria, E., Gallone, R., and Giannoccaro, N. I. (2018). Development of robot for 3d measurement of forest environment. *Journal of Robotics and Mechatronics*, 30(1):145–154.

Mowshowitz, A., Tominaga, A., and Hayashi, E. (2018). Robot navigation in forest management. *Journal of Robotics and Mechatronics*, 30(2):223–230.

Naesset, E. (1997). Determination of mean tree height of forest stands using airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 52(2):49–56.

Nevalainen, P., Li, Q., Melkas, T., Riekki, K., Westerlund, T., and Heikkonen, J. (2020). Navigation and mapping in forest environment using sparse point clouds. *Remote Sensing*, 12(24):4088.

Oliveira, L. F., Moreira, A. P., and Silva, M. F. (2021). Advances in forest robotics: A state-of-the-art survey. *Robotics*, 10(2):53.

Ortiz Morales, D., Westerberg, S., La Hera, P. X., Mettin, U., Freidovich, L., and Shiriaev, A. S. (2014). Increasing the level of automation in the forestry logging process with crane trajectory planning and control. *Journal of Field Robotics*, 31(3):343–363.

Parker, R., Bayne, K., and Clinton, P. W. (2016). Robotics in forestry. *NZ Journal of Forestry*, 60(4):9.

Pizetta, I. H. B., Brandao, A. S., and Sarcinelli-Filho, M. (2018). Control and obstacle avoidance for an uav carrying a load in forestal environments. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 62–67. IEEE.

Pomerleau, F., Colas, F., Siegwart, R., and Magnenat, S. (2013). Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3):133–148.

Reeds, J. and Shepp, L. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393.

Renner, M., Sweeney, S., and Kubit, J. (2008). *Green Jobs: Towards decent work in a sustainable, low-carbon world*. UNEP.

Roßmann, J., Krahwinkler, P., and Bücken, A. (2009). Mapping and navigation of mobile robots in natural environments. In *Advances in Robotics Research*, pages 43–52. Springer.

Rossmann, J., Krahwinkler, P., and Schlette, C. (2010). Navigation of mobile robots in natural environments: Using sensor fusion in forestry. *J. Syst. Cybern. Inform*, 8(3):67–71.

Rossmann, J., Schluse, M., Schlette, C., Buecken, A., Krahwinkler, P., and Emde, M. (2009). Realization of a highly accurate mobile robot system for multi purpose precision forestry applications. In *2009 International Conference on Advanced Robotics*, pages 1–6. IEEE.

Rusu, R. B. (2010). Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348.

Rusu, R. B. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE.

Schwarz, M., Rodehutskors, T., Droeschel, D., Beul, M., Schreiber, M., Araslanov, N., Ivanov, I., Lenz, C., Razlaw, J., Schüller, S., et al. (2017). Nimbro rescue: Solving disaster-response tasks with the mobile manipulation robot momaro. *Journal of Field Robotics*, 34(2):400–425.

Shan, T. and Englot, B. (2018). Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE.

Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., and Rus, D. (2020). Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: modelling, planning and control*. Springer Science & Business Media.

Silvere (2017). We know your trees. Retrieved November 27 2019 from https://silvere.com/english.

Song, J. and Sharf, I. (2020). Time optimal motion planning with zmp stability constraint for timber manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4934–4940. IEEE.

Song, J. and Sharf, I. (2021). Stability constrained mobile manipulation planning on rough terrain. *arXiv preprint arXiv:2105.04396*.

Sucan, I. A., Moll, M., and Kavraki, L. E. (2012). The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82.

Swedish Forest Agency (2014). Swedish statistical yearbooks of forestry. Retrieved November 26 2019 from http://klimatetochskogen.nu/en/information-sources/reports/161-sks2014-forestry-yearbook-2014/.

Tanaka, K., Okamoto, Y., Ishii, H., Kuroiwa, D., Yokoyama, H., Inoue, S., Shi, Q., Okabayashi, S., Sugahara, Y., and Takanishi, A. (2017). A study on path planning for small mobile robot to move in forest area. In *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2167–2172. IEEE.

Tominaga, A., Eiji, H., and Mowshowitz, A. (2018). Development of navigation system in field robot for forest management. In *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 1142–1147. IEEE.

Tremblay, J.-F., Béland, M., Gagnon, R., Pomerleau, F., and Giguère, P. (2020). Automatic three-dimensional mapping for tree diameter measurements in inventory operations. *Journal of Field Robotics*, 37(8):1328–1346.

Tsubouchi, T., Asano, A., Mochizuki, T., Kondou, S., Shiozawa, K., Matsumoto, M., Tomimura, S., Nakanishi, S., Mochizuki, A., Chiba, Y., et al. (2014). Forest 3d mapping and tree sizes measurement for forest management based on sensing technology for mobile robots. In *Field and Service Robotics*, pages 357–368. Springer.

United Nations Food and Agricultural Organization (2018). 2018 the state of the world's forests. Retrieved November 26 2019 from http://www.fao.org/state-of-forests/en/.

Von Carlowitz, H. C. and von Rohr, J. B. (1732). *Sylvicultura oeconomica*.

Wermelinger, M., Fankhauser, P., Diethelm, R., Krüsi, P., Siegwart, R., and Hutter, M. (2016). Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189. IEEE.

Westerberg, S. (2014). *Semi-automating forestry machines: Motion planning, system integration, and human-machine interaction*. PhD thesis, Umeå Universitet.

Winter, K., Hayes, I. J., and Colvin, R. (2010). Integrating requirements: the behavior tree philosophy. In *2010 8th IEEE International Conference on Software Engineering and Formal Methods*, pages 41–50. IEEE.

Wooden, D., Malchano, M., Blankespoor, K., Howardy, A., Rizzi, A. A., and Raibert, M. (2010). Autonomous navigation for bigdog. In *2010 IEEE international conference on robotics and automation*, pages 4736–4741. Ieee.

Yue, Y., Yang, C., Senarathne, P., Zhang, J., Wen, M., and Wang, D. (2018). Online collaborative 3d mapping in forest environment. *Forest*, 2(1.2710):4–0915.

Zhang, C., Yong, L., Chen, Y., Zhang, S., Ge, L., Wang, S., and Li, W. (2019a). A rubber-tapping robot forest navigation and information collection system based on 2d lidar and a gyroscope. *Sensors*, 19(9):2136.

Zhang, J. and Singh, S. (2014). Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9.

Zhang, J. and Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181. IEEE.

Zhang, W., Wan, P., Wang, T., Cai, S., Chen, Y., Jin, X., and Yan, G. (2019b). A novel approach for the detection of standing tree stems from plot-level terrestrial laser scanning data. *Remote sensing*, 11(2):211.

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., Bagnell, J. A., and Srinivasa, S. S. (2013). Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193.

How to cite this article: Jelavic, E., Jud, D., Egli, P., & Hutter, M. (2022). Robotic precision harvesting: Mapping, localization, planning and control for a legged tree harvester. *Field Robotics, 2*, 1386–1431.