WHERE TO PERCH?

AUTOMATIC IDENTIFICATION AND SELECTION OF PERCHING LOCATIONS FOR A DRONE-BASED MOTION CAPTURE SYSTEM

Simon Honigmann
Master of Robotics Candidate

Submitted: September 11, 2020

Supervisor: Dr. Fabrizio Schiano

Faculty Advisors:

Professor Dario Floreano

Professor Mark Cutkosky

Acknowledgements

I would like to take a moment to thank those who have supported me, stood by me, and have given me such incredible opportunities to explore, learn, and grow. It has been a tumultuous year to say the least, but I have been fortunate enough to be able to adapt and continue my studies thanks in no small part to my wonderful support network. Thank you to Professor Floreano for providing me with countless opportunities during my time at EPFL. To Professor Cutkosky, thank you for allowing me to visit your lab, however briefly. Even working remotely, the experience was invaluable. To Dr. Schiano, thank you for continued support and ever-useful feedback throughout this project. It goes without saying that none of this would have been possible without the generous and steadfast support of my parents, the encouragement of my partner, Paige, and of course my wonderful friends, colleagues, and mentors that I have had the pleasure of knowing at UBC, EPFL, and Stanford. Thank you.

Abstract

Aerial robots are ideal candidates for sensing tasks over long ranges or in challenging environments, however are inherently limited when it comes to physical interaction. Multi-rotor style vehicles, which have significant advantages in controllability, simplicity, and redundancy, are particularly afflicted by these limitations due to their inability to generate passive lift. One way to overcome these limitations is to enable the vehicle to perch or land when performing manipulation tasks or long term sensing tasks.

Perching is a behaviour that offers tremendous benefit to airborne animals and robots alike. Perching offers reprieve from costly aerial locomotion, allowing the environment to passively support weight of the flyer and its payload, rather than requiring active lift generation. In aerial robotic systems, passive perching techniques can extend mission duration almost indefinitely, and can allow vehicles to apply forces to the environment that are several orders of magnitude greater than while flying. A lot of work has been done in the field of aerial robotics to produce mechanisms which enable aerial robots to perch. However, to the author's knowledge, no published works have discussed perching location selection while considering broader mission context. Location selection and environmental context are exceedingly important when the aerial vehicles are being used as remote sensing nodes, as location can impact the completeness and quality of data being collected. Additionally, perching maneuvers are not trivial and can put the vehicle at risk. The perching success rate can be maximized by segregating between safe and high-risk locations. Thus, to maximize the probability of mission success, placement optimization is an important consideration.

This work presents a system capable of identifying potential perching locations and quickly select a near-optimal set of perching locations to establish an effective distributed sensing network in otherwise inaccessible environments. The developed system requires a dense reconstruction of the target environment as input, with optional semantic context. Computer vision and analytical geometry techniques are used to segment the environment and identify locations which offer necessary affordances for perching. Camera placement is investigated as a potential distributed sensing application, although the method presented is generalizable to any sensing modality for which a radial coverage function can be formed.

The proposed system is shown to process new environments and generate optimized camera arrangements in under 10 minutes. The system autonomously generates perch location proposals with an 85% pass rate, when subject to operator review, and can propose alternative solutions in seconds. Future improvements are suggested to enhance performance, but the work meets design goals as presented.

Contents

Acronyms and Abbreviations

Acronyms

BRIEF Binary Robust Independent Elementary Feature. v

CNN Convolutional Neural Network. 3

CPU Central Processing Unit. 7, 9, 68

DroCap Drone-Based Motion Capture System. 3, 5, 107

FAST Features from Accelerated Segment Test. v

FlyCroTug Flying Micro-Tug Winching Robot. 2, 4, 55

FOV Field of View. 7

GB GigaByte. 39

GNSS Global Navigation Satellite System. 2, 3

GPS Global Positioning System. 2

GPU Graphics Processing Unit. 3, 7, 9, 38, 68

GUI Graphical User Interface. 64, 66, 73, 92, 106

ICP Iterative Closest Point. 37

IMU Inertial Measurement Unit. 7, 17, 37

KD-Tree K-Dimensional Tree. 47–49

LIDAR Light Detection and Ranging (Device). 3, 8, 18

MAV Micro Aerial Vehicle. 2, 3, 7, 17, 37

MicroTug Micro-Tug Winching Robot. 2

OBB-Tree Oriented Bounding Box Tree. 49

ORB Oriented FAST and Rotated BRIEF. 37

PCA Principal Component Analysis. 13, 51, 62

PSO Particle Swarm Optimization. 15, 57, 59, 74, 80, 89, 99

RAM Random Access Memory. 38, 39, 68

RANSAC Random Sample Consensus. 12

RGB Red-Green-Blue. 17

RGB-D Red-Green-Blue-Depth. 3, 9, 16, 17, 34, 35, 37

RMS Root Mean Square. 23

RMSE Root Mean Square Error. 82

ROI Region of Interest. 64, 65

ROS Robotic Operating System. 67

RTK Real-time Kinematic. 2

SLAM Simultaneous Localization and Mapping. i, 6–9, 12

ToF Time of Flight. 18

TSDF Truncated Signed Distance Function. 37

VIO Visual Inertial Odometry. 7, 37

VO Visual Odometry. i, 7, 37

1 INTRODUCTION

Aerial robots are steadily finding their place in a number of remote sensing applications. While already commonplace in industries which require large area surveying, such as agriculture and power transmission, advances in hardware, control, and planning are making them increasingly viable for scenarios where a fast response to dangerous, remote, or inaccessible areas is required [1–5]. For instance, search and rescue teams can benefit considerably from aerial robots, which can deploy rapidly, traverse difficult obstacles with ease, and offer a birds eye perspective of the scene [6, 7]. Aerial robots can also provide vital support in disaster relief operations, when existing infrastructure has broken down. In these scenarios, decisions must be made quickly and must be executed with precision in order to maximize recovery efforts. For this reason, most researchers envision systems in which robots and humans collaborate to address the situation. However, at present, the role of the aerial robot is largely reserved to general reconnaissance - developing an understanding of the scene in order to inform human actions.

The use cases that have been formed for aerial robots have been largely formed by their fundamental physical limitations. The increased mobility comes at a cost: airborne vehicles are simply less capable of forceful interaction than their grounded counterparts due to the cost of generating forces through aerodynamic effects and the need to actively support the forces and their own weight [8]. Within the category of aerial vehicles, different classes have diverse strengths and weaknesses which help shape their individual application spaces[1]. Fixed-wing vehicles benefit from high aerodynamic efficiency and can achieve high speeds and long flight ranges. The large fixed airfoils that afford fixed-wing vehicles such high efficiency also limit the possible range of flight speeds and turn rates, making them far less maneuverable than multi-rotor vehicles. Multi-rotors in contrast, can be fully controllable, allowing them to navigate cluttered environments, hover in place, and perform precision maneuvers. They have also been shown to lift sizeable payloads in short bursts, but in practice are limited to about double their own mass [9, 10]. Multi-rotors are limited in their own right, however, with substantially reduced flight times and ranges owing to their high cost of transport.

Recent works have looked to extend application space of aerial vehicles by embracing multi-modal designs: robots capable of operating both in the air and on the ground. By transitioning from the mindset from designing purely aerial or purely ground-based robots, towards a hybrid approach which leverages the strengths of each mode, interesting capabilities become apparent [11–13]. Aerial transport is a clear winner for deployment and ensuring

maximal coverage. However, once on the scene, robots can transition to ground-based operations. This new paradigm can be used to extend the capabilities of robots for both sensing and manipulation tasks. By adhering the vehicle to fixed objects in the environment, forceful, controlled interaction become more tractable - particularly when the adherence is passively generated. Passive adherence to the environment can also extend the life of sensing missions considerably, as power can be reserved for sensing and communication, rather than maintaining flight [14].

FlyCroTug robots [13] are one such embodiment of a multi-modal approach. FlyCroTugs are MAV drones equipped with a dry-adhesive gripper to adhere to the ground and a high-force winch system. Estrada et al. demonstrated the ability of a small team of FlyCroTugs collaborate in manipulation tasks, such opening doors. The same manipulation technique was demonstrated by Christenesen et al., where a group of MicroTug robots were able to tow a car, using anchor points to pull with forces exceeding 200 times their own weight [15]. Such systems show considerable promise for disaster relief scenarios, and could be envisioned clearing debris to form openings. However, despite showing considerable mechanical promise, multi-agent robotic systems present large challenges in terms of sensing, control and planning. In both aforementioned works, robots were operated under manual control and a direct line of sight to the operators was always available.

In order to use multi-modal robot teams for practical manipulation tasks, several challenges must first be addressed to enable them to operate autonomously or semi-autonomously. First is the challenge of generating state information, both of the robots and of target objects. Once acceptable state information is available, the next challenges involve planning and coordination of robot teams in order to identify manipulation targets, place end-effectors, select pulling directions, and execute maneuvers. Developing a system capable of addressing the first point will be the topic of this thesis.

Aerial robots have been shown to perform extremely precise, aggressive maneuvers when provided with near-perfect state information. In controlled, indoor environments, near-perfect robot and object state information is commonly generated using multi-camera motion capture systems. In outdoor environments, the use of Global Navigation Satellite System (GNSS) in combination with inertial information is a popular method of generating accurate, low drift state estimates for mobile robots, but offer no direct measurement of the environment's state [1]. Recently, Real-time Kinematic (RTK) GPS systems have shown promise for providing centimeter-level positioning accuracy, approaching what is possible in indoor motion capture environments. While GPS is generally a low cost, reliable positioning option for outdoor flight, coverage can be unreliable indoors and near large structures. Other

robot localization methods have been suggested as an alternative, such as Wi-Fi time of flight localization [16]; however, as in the case of GNSS, these network-based approaches are susceptible to interference and provide only state information about the robots themselves and fail to provide environmental context or target state information. For this, a common approach is to take advantage of imaging modalities including cameras and LIDAR [17, 18].

Imaging sensors can provide rich sensory feedback, allow the possibility to generate maps of the environment and track objects in real time. Real time mapping and localization algorithms can be used to generate local, relative pose information in new environments, provided there are enough features in the scene to track. Objects can be detected and tracked as they are moved through a scene, relative to a camera. The latter point is becoming more and more reliable as Convolutional Neural Networks (CNNs) continue to develop [19, 20]. However, The rich sensory information takes a considerable amount of processing power to digest with the majority of CNN algorithms requiring a GPU to run in real-time. Aerial robotic platforms, and MAVs in particular, are often limited in terms of onboard computational power, though recent miniaturized computers, such as the Nvidia Jetson TX2 and the Intel NUC, have allowed increasingly complex algorithms to be run on-board [21, 22]. An alternative approach can be to perform processing off-board, using telemetry links or local Wi-Fi networks to offload computations to a more powerful base station computer. This can help keep weight down, making quadrotors more agile and maneuverable, at the cost of introducing bandwidth limitations and latency.

While a single imaging sensor can provide considerable information regarding the scene, there are many scenarios where it would be insufficient to extract all necessary elements. A simple example to consider is when the target volume is non-convex. In this case, parts of the scene are occluded and therefore unobservable. Additionally, stereo-based RGB-D cameras struggle to provide accurate depth information at distance [23]. In this case, tracking uncertainty can be inadequate unless fused with additional sensory information. Modern motion capture systems overcome these challenges using networks of cameras to cover the scene with diverse perspectives and using sensor fusion to generate state estimates that are accurate and robust to occlusion.

The proposed method of generating the necessary robot and object state information in arbitrary, unstructured environments is to create an in-situ, self-calibrating motion capture system using a team of perching drones to place cameras (dubbed DroCap) in select locations in the environment. This concept is illustrated in Figure 1. By equipping drones which are capable of perching on vertical and inverted surfaces with depth cameras, a remote camera network can be established in new environments. This camera network would

offer considerable benefit in making sense of new environments. A well established camera network could be used to identify and track target objects, grasping points, landing locations, plan manipulation and provide state updates for FlyCroTugs as they move around the scene. However, in order to do that, the drones must be able to solve the so-called relative localization problem. The latter consists of being able to estimate the position of each agent of a group with respect to the other agents. Additional complexity to this problem is added if the goal is to estimate the relative positions in a decentralized manner. A line of research tackling this problem is the one employing rigidity theory and specifically bearing rigidity theory [24–26].

Motion capture using airborne drones with monocular cameras was demonstrated by Saini et al. [27]. While the drones could re-position themselves to track mobile targets over large areas and simultaneously avoid obstacles, the fact that the drones were airborne introduced considerable uncertainty and challenge when performing pose matching between frames. Consequently, target pose estimation needed to be optimized offline to produce consistent results. The proposed system mitigates this challenge by giving cameras static positions in a mapped environment. This offers two substantial benefits. First, as the cameras are static in space except for well defined gimbal movements, the relative pose of cameras needs only to be solved once. Second, as cameras are being placed at select waypoints in the environment, the system has a-priori information regarding the relative pose of the cameras. The initial pose estimate provides an appropriate initial guess solution for pose correspondence algorithms, such as iterative closest point [28].

The proposed system also differs from commercial motion capture systems in two key ways. First, communication bandwidth will be inherently limited; commercial motion capture systems typically use high throughput Ethernet connections between each camera and a base computer, allowing for high frame rates and a large number of connected cameras. Second, the number of possible camera locations may be limited, as might be in the case in cluttered environments or in partially collapsed buildings. Both of these challenges compound to limit the feasible number of cameras placed in the environment. Because the number of cameras can be significantly limited, it becomes increasingly valuable to ensure that each camera is placed optimally, and maximizes coverage of the scene.

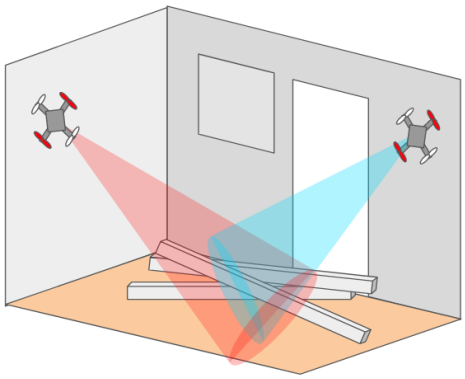


Figure 1: Potential realization of an impromptu, distributed, DroCap motion capture system made of several UAVs equipped with depth camera sensors.

State estimation is critical to the success of the subsequent perching, grasping and manipulation tasks. As such, it is imperative that cameras are placed in such a way that high-certainty estimates can be made throughout the operational space. This work investigates the challenges of camera placement for a perching drone motion capture system, and presents a system which addresses these challenges. The system is designed to meet the needs of the FlyCroTug aerial robots that were co-developed by Stanford's Biomimetic and Dextrous Manipulation Laboratory and EPFL's Laboratory of Intelligent Systems. First, the problem of perch location identification is explored. Methods of mapping new environments are compared to understand the strengths and limitations of modern algorithms. From the environment reconstruction, perching locations must be identified to form the camera placement search space. Segmentation algorithms are reviewed in order to identify planar perching targets. A PCA-based clustering and segmentation algorithm is proposed to segment the reconstruction into a set of large, planar surfaces. Different filtering techniques are explored to restrict the search space based on the requirements of the perching camera drones and FlyCroTug robots, to reduce the need for operator intervention. Next, the problem of optimal camera placement is addressed. Continuous and discrete search approaches are compared, and a heuristic search framework is developed to generate locally optimal camera placements in the identified search space. Finally, the perch placement system's performance is assessed in terms of speed as well as the quality of proposed camera arrangements.

2 BACKGROUND

To better understand and define the requirements for a camera placement system, it is important to understand the underlying constraints and requirements of robotic perching mechanisms and 3D reconstruction systems which are at the core of the input and output of the proposed pipeline. A brief review of advances in robotic mapping and reconstruction is presented, with a focus on techniques which would be relevant in the proposed disaster relief scenario. This is followed by a discussion of multi-camera networks and the notion of optimal camera placement. Finally, the state of the art for robotic perching is discussed.

2.1 Environment Mapping, Reconstruction and Partitioning

In order to enable autonomous or semi-autonomous robots to operate in new environments, being able to localize within the environment and understand the presence of obstacles and targets is paramount. A popular method to generate this understanding is through the use of Simultaneous Localization and Mapping (SLAM) [29, 30]. SLAM, which will be described in more detail in the next section, allows robotic systems to gain a spatial understanding of the environment. But in many instances, more context is required in order to make informed decisions in the environment. Rather than assuming all objects in an environment are static and uniform, it can be helpful to assign additional properties to objects and surfaces, depending on the application. There are many possible methods to generate this sort of semantic understanding of the scene. Some of these methods, along with their potential application to the perching camera system, will be discussed in Section 2.1.2.

2.1.1 Simultaneous Localization and Mapping (SLAM)

As the name suggests, SLAM systems enable robots to both create a map of an environment, and simultaneously localize themselves within the generated map. SLAM continues to be a very active research topic, as faster, more scalable, or more robust algorithms are tested. Neural networks are also beginning to find their place, but the study of their use is still in its infancy [31, 32]. For this reason, this project will focus on the well established techniques which use traditional computer vision methods to perform estimation and mapping operations.

Briefly, SLAM works as follows: sensor information is collected at key frames in time, and key points of the data are saved. These initial key points are used to initialize a map of the environment. Between each key frame, odometry is used to estimate the motion of the robot or camera relative to the previous key frame. Using odometry allows the robot to get an

a-priori estimate of its current pose. A subsequent update step compares new measurements to key points already on the map and refines the pose estimate. This a-posteriori estimate is then used to update the map and fuse new key points into the map[29].

SLAM requires several modules to run in parallel. In general, these consist of odometry, feature extraction, and feature matching. Odometry can take many shapes and forms. On wheeled robots, pose on a 2D surface can be estimated quite accurately using wheel encoders to track the distance travelled by each wheel. On aerial robots however, this is not possible. Instead, airborne odometry is often done through a combination of inertial and optical sensory input. Visual Inertial Odometry (VIO) fuse synchronized inertial information from an Inertial Measurement Unit (IMU) with the optical flow of key points in a camera's Field of View (FOV) to accurately estimate pose through time [33]. Purely Visual Odometry (VO) techniques also exist, but require additional information to scale the generated map. VO techniques are relatively lightweight and accurate for smooth, relatively slow camera movements. However, they can quickly lose track of points when subjected to high rotation rates. Because VIO techniques can still rely on inertial measurements to ground the pose updates in the absence of visual pose change estimates, they tend to be more robust, at the cost of additional computational load and hardware requirements [34]. Increasing performance and decreasing costs of cameras, CPUs, and GPUs have made the latter a moot point in all but the most restrictive situations, with several examples of VIO and SLAM algorithms running even on Micro Aerial Vehicle (MAV) [22].

SLAM algorithms can take many forms (as demonstrated in Figure 2), each with their respective strengths and weaknesses. To distinguish between different algorithms, one can consider the: sensor type, feature type, and map representation [35].

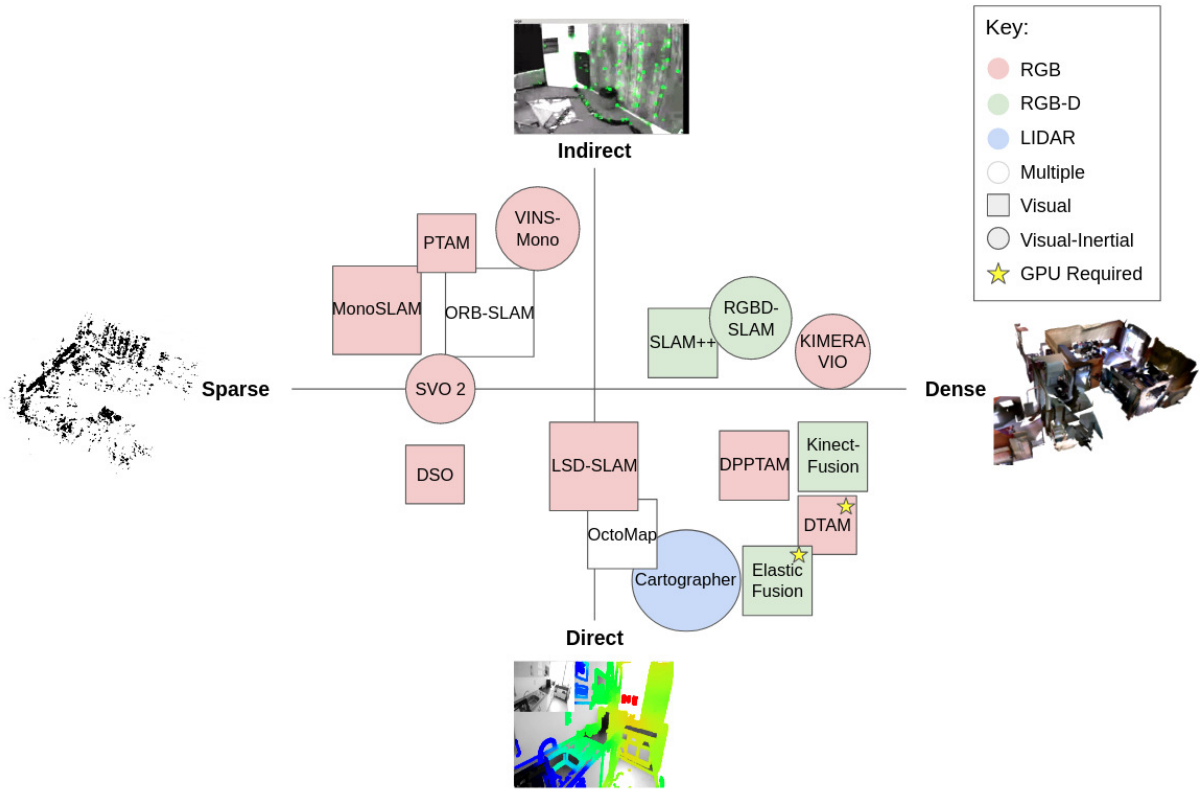


Figure 2: Comparison of select SLAM algorithms, inspired by [36]. Generally speaking, SLAM algorithms can be described by the features being tracked and the form of the map created. Each quadrant represents a specific classification (dense-direct, sparse-indirect, etc.)

It should be no surprise that most SLAM algorithms are sensor dependent. Different sensing modalities produce data in very different forms, with varying levels of quality and can extract a very different set of features. For instance, LIDAR produces excellent position information, but typically does not have integrated color sensing and frequently has a rolling shutter. Thus, LIDAR SLAM methods, such as GMapping [37], must extract features from purely geometric data, while potentially accounting for motion during the frame capture. In contrast, a monocular camera SLAM system does not directly generate any depth information; it must be inferred by tracking features between subsequent image frames. Depth cameras, such as time of flight cameras, stereo cameras, and structured light cameras, yield depth information that is less reliable than that of a typical LIDAR system, but also provide information about feature intensity in the scene. This can help distinguish between and track individual features. Many of the more popular algorithms do support several sensor formats, making them more generally applicable [38, 39].

SLAM algorithms can also be separated by their representation of the environment. In earlier algorithms and recent lightweight algorithms [40–42], lightweight representations are

used. These can be as simple as graphs of the pose estimate with a few key-frame images stored at key points to perform inference between frames [42], or 3D clouds of oriented features [38]. While the lightweight representation allows these algorithms to manage much larger environments with a given amount of memory and operate in real time on a CPU, the sparse maps make subsequent planning difficult. When a dense representation is used, it is possible to save all sensor data and post-process the scene using the camera's pose estimates using techniques such as BundleFusion [43].

Dense SLAM algorithms, such as RGB-D SLAM [44] and ElasticFusion [45] use surfels, voxels, or point clouds to track a large number of points in the environment. The result is akin to a digital reconstruction of the environment, and can directly support more advanced post-processing operations like object identification or obstacle avoidance.While there has been recent progress, allowing dense mapping algorithms to run in real-time, they often require GPU acceleration [39, 46] which can be a limiting factor on smaller mobile robots.

2.1.2 Semantic Labelling

Semantic labelling is the process of assigning non-geometric information to a map or image. Semantic labelling generally consists of two main steps. First, semantic information must be inferred from sensory data from the environment. Next, this data must be assigned to specific features in the map. Semantic information for an environment can take many forms. In the case of scene masking or material surface labelling, as demonstrated in Figure 3, information can be applied directly to geometric features [47, 48]. Alternatively, semantic information can be captured in the form of objects. As briefly mentioned when introducing object-centric SLAM, object recognition can provide relevant, explicit or learned information about objects within a scene, and is a popular route to gain more context about potential targets for manipulation, avoidance, and planning. In the context of robotic perching using dry adhesives (see Sec. 2.3.2), surface material and surface roughness provide some context regarding the viability of different mechanism types. Additionally, the presence of boundary between two materials would indicate a region may be unsuited for perching, despite the lack of a geometric boundary. For this reason, semantic surface labelling will be explored.

Many sensing modalities have been explored for both material identification and surface roughness measurement [50]. In the context of robotic perching, however, we are only concerned with methods that could potentially be mounted on an aerial vehicle, so methods like precision tactile sensing and interferometry methods that are well suited to lab environments but poorly suited to a mobile platform will not be discussed [50]. While simple tactile probes have been shown in the past to classify materials based on their roughness with relative

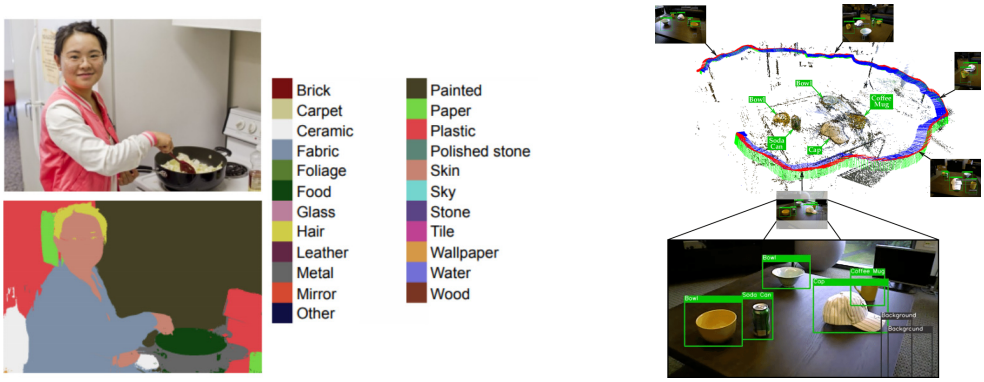


Figure 3: (left) Example of surface material segmentation from [47]. (right) Example of object recognition to generate semantic context. [49]

accuracy [51], a preference will be given towards non-contact approaches. Requiring surface contact would both slow down the collection of semantic data considerably, and introduce several operational risks.

Image-based methods are the most promising option given the problem constraints, as they provide rich sensory feedback in what is typically a lightweight, low power package. Image-based methods fall largely into the category of monocular and stereo imaging. Modern methods using monocular imaging almost exclusively use artificial neural networks to attempt to identify materials based on low and mid-level features [52, 53]. While these methods show impressive results, testing them "in the wild" [47] has demonstrated that material identification is still a challenging problem. However, these methods are only suitable for making broad material classifications (e.g. wood vs. metal). An alternative approach with more promise for millimeter-scale features is to use differential stereo imaging [54, 55]. By comparing multiple images with minute changes in lighting or perspective, there is more hope in capturing information about smaller features. These methods require quite close proximity to the target surface in order to generate sufficiently small discrepancies. In practice these methods only tell a partial story. Of interest to perching is the surface roughness (micro to millimeter scale) and asperity size (micrometer to centimeter scale). There is some hope of detecting large scale (i.e. cm scale) asperities with traditional cameras to infer suitability for microspine grippers. But when debating between smooth surfaces where gecko adhesives or suction cups are candidates, the scale of interest becomes impossibly small for macroscopic image capture [56].

There is no publicly available dataset specifically addressing microspine or gecko adhesive grasp quality on different materials. While there is hope that a neural network could be trained to identify quality perching locations from monocular or stereo images, this remains future

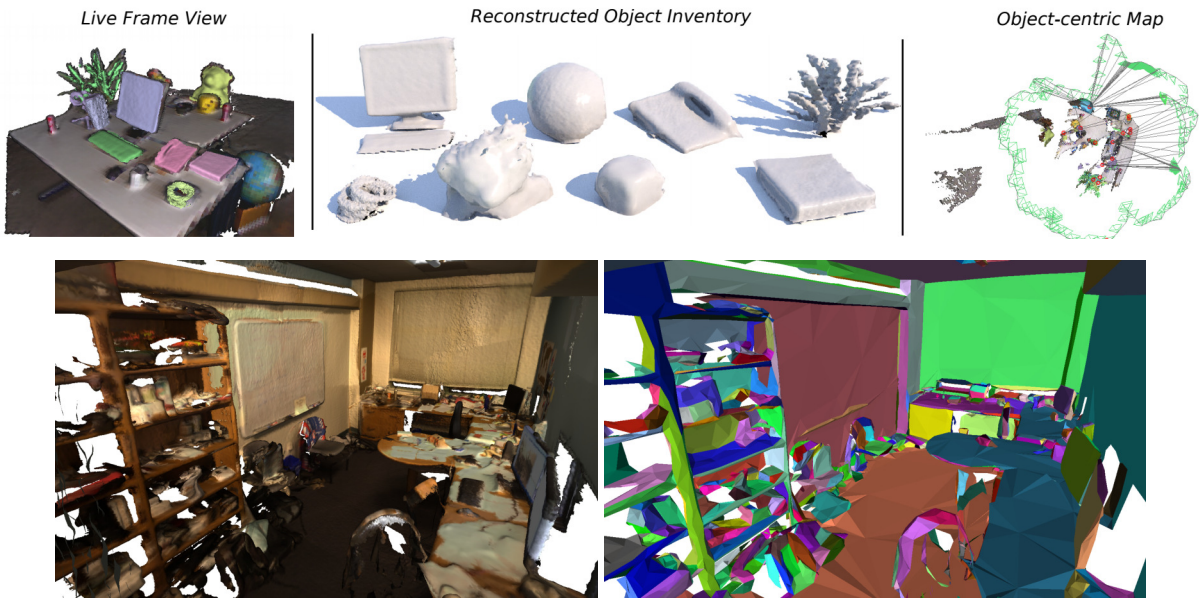


Figure 4: (top) Object partitioning by [60] where objects are recognized and tracked throughout a scene. (bottom) Plane partitioning based on [61] performed on the Office3 model from the BundleFusion dataset [43]

work. In this work, broad generalizations about surface suitability will be used, in conjunction with the operator's intuition regarding different surfaces.

2.1.3 Geometric Partitioning

As mentioned in Section 2.1.1, it is computationally advantageous to reduce dense environment representations into meaningful, simplified representations. This can take the form of predefined objects or geometric primitives, such as spheres, boxes, cylinders, and planes. While objects embed more semantic information in a scene and can result in more meaningful maps [57, 58], they require individual consideration of a finite set of predefined classes, like cars or chairs. In this sense, they can be less versatile for unknown environments than simpler representations. Identifying geometric primitives offers a lightweight alternative, which can still simplify the environment's representation considerably, while giving more context to the scene [59].

In the context of robotic perching, planar and cylindrical surfaces both offer important context regarding perchability. The specific mechanisms used to achieve robotic perching are discussed in greater detail in Section 2.3, and determine the specific features which should be represented. In this work, planar surfaces are the feature of interest as they are suitable for perching attempts using dry-adhesive grippers.

When navigating engineered environments, planar representations are a popular choice, as a large proportion of static structures consist of flat surfaces. Some SLAM algorithms extract planar geometry directly, creating lightweight maps which are particularly effective in indoor environments [62]. Even rounded features can still be represented with some fidelity as a collection of planar surfaces. When exploring perching and landing surface identification, this is a beneficial side-effect. Surfaces with excess curvature are necessarily represented by several small planar sections; to avoid high-curvature surfaces which may not be suitable for some perching mechanisms, surfaces can be quickly filtered by area or characteristic length. There are many well-known algorithms for identifying planar surfaces in geometric data [61, 63–65]. These can broadly fit into model fitting algorithms and region growing algorithms.

Model fitting algorithms employ various means to fit a provided model (e.g. a plane equation) to data. For plane identification, Random Sample Consensus (RANSAC) is a perhaps the most popular model fitting technique. RANSAC, sampling of random points to estimate the parameters of planar surfaces, then verifies the estimation by checking the consensus of the model with other randomly selected points [63]. RANSAC's popularity largely stems from its speed, probabilistic guarantee of yielding an acceptable solution, and its robustness to outlier data. While the algorithm has been adapted since its first inception [64, 66], RANSAC-based methods often ignore scene-level structures and local the regularity of points [67]. For these reasons, RANSAC is most commonly employed when the target consists of large, simple features, such as when identifying the faces of a building.

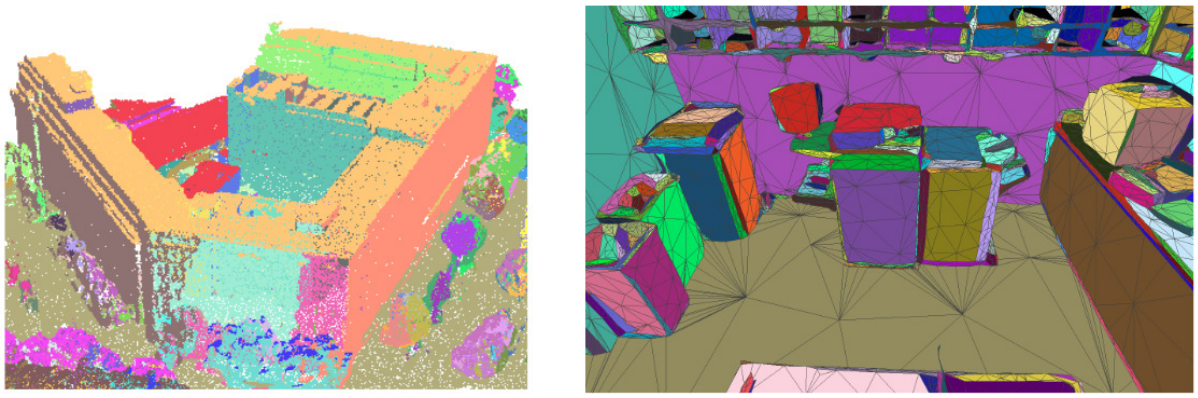


Figure 5: (left) Plane segmentation using RANSAC (courtesy of [68]). (right) Plane segmentation using PCA-Energy Minimization (courtesy of Wang et al.) [61]

In region growing, seed regions are initialized in the data, typically using a coarse segmentation function, then the segmentation is incrementally refined by comparing clusters to neighboring points and merged if similarity criteria are met. Region growing methods inher-

ently detect connectivity of clusters, which can add context to the reconstruction. They can also be suitable for plane detection in large scale models. However, region growing is highly dependent on seed selection and often requires some hand-tuning of methods, diminishing the robustness of the method [69]. Poorly initialized clusters can lead to over-segmentation when presented with free-form shapes [61].

Work by Cai et al. [70] addresses the over-segmentation challenge, presenting an asymptotically optimal region clustering method which iteratively merges regions which minimize a Principal Component Analysis (PCA)-based "energy" function defined in the work. By comparing the principal directions, or eigen-vectors, that are found using PCA, the orientation and aspect ratio of point clusters can quickly be determined.

PCA-based energy, E_{pca} is defined by Cai et al. for a cluster, C as follows:

$$E_{PCA}(C) = \frac{|U(C)|}{area(C)^4} \tag{1}$$

where:
$U(C)$ is the covariance matrix of the points in cluster C

PCA energy is thus minimized by maximizing the area of a cluster while minimizing its volume. The result is a linear piecewise representation of the entire surface, where vertices are more densely assigned to regions of increased curvature or complexity. Cai et al. show that this definition of energy results in asymptotically optimal behaviour.

Wang and Guo [61] extend the methods in [70] to also include a simplification step, capable of merging adjacent clusters who meet planar similarity criteria. This method results in lightweight, low-polygon meshes which preserve key geometric features from the original data. This method requires a mesh reconstruction of the environment as an input, compared to the previously mentioned methods which operate directly on point cloud data. The former concepts are more suitable for direct, real-time plane detection from raw depth data. In mapped environments where it is important to preserve geometric context, Wang and Guo's plane partitioning algorithm is shown to be effective and efficient. In this scenario, the goal will be to work with previously mapped environments, thus Wang and Guo's algorithm will be studied further.

2.2 Distributed Sensor Networks

Remote, distributed sensing has become increasingly viable as sensing, communication, and computing hardware continues to scale down in both size and price. By distributing multiple sensors throughout an environment, it is possible to fuse together the individual data streams into a single, more complete representation of the scene. Distributed sensor networks present unique challenges, however, when considering node placement, fusion, and communication. Strategies vary considerably with the type and rate of information being captured. As this work is focusing on camera placement, the following sections will discuss camera placement and fusion. Distributed control and communication strategies will remain as a topic for future study.

Multi-camera networks are well studied for applications in reconstruction, motion tracking, surveillance, and surveying [71–73]. In motion tracking applications, the goal is to place views of multiple cameras such that they intersect to generate precise state information that is robust to dynamic occlusion. Surveillance applications, in contrast, focus on maximizing coverage; the exact state information is less important than the ability to track objects through multiple views. Finally, surveying tasks opt for a balance of information quality and coverage; while scenes being surveyed are typically static, the surveyors will typically move to cover the scene.

As the goals for each task are quite varied, so they are their optimization functions and methods. The proposed perching camera problem largely resembles a motion capture camera placement problem, though some inspiration can still be taken from the methods used both in surveillance and surveying. In particular, most work studying motion capture assumes the captured environment is convex and that cameras can be placed in predefined, known locations on the perimeter. Surveillance and surveying works tend to focus more on optimization of coverage in the presence of static occlusions, which are important to consider in this case. Additionally, camera mobility can lead the way to control laws to improve upon initial solutions. The next section will discuss relevant works in camera placement optimization.

2.2.1 Camera Placement Optimization

Optimal camera placement is a deceptively challenging combinatorial optimization problem. It is considered to be an NP-Hard problem [74], suggesting that it is relatively simple to evaluate the quality of a proposed solution, but that a globally optimal solution cannot be solved in polynomial time. Despite this, many strategies exist to converge to local optima

with the hopes that they will approach global optimum. Many different strategies have been proposed for optimizing camera placement [71, 73, 75, 76].

When considering surveillance or scene coverage alone, this problem shares many similarities to the well known Art Gallery Problem [77]. The Art Gallery Problem poses the question of what the minimum number of guards required is to provide full visibility of an arbitrary scene. This problem was popularized in 2D, but has been extended to 3D as well. When considering static sensors, such as wall mounted cameras, several methods have been proposed to optimize camera coverage. Horster et al. proposed solving a mixed integer optimization problem using discrete camera positions, to optimize coverage of 2D environments [76]. A weighting system is proposed to prioritize coverage of key areas, based on a-priori information on the environment. Yildiz et al. suggested the use of bi-level linear programming to simultaneously optimize coverage, while minimizing the redundancy of the arrangement [78]. Other methods have opted for metaheuristic search techniques, such as Particle Swarm Optimization (PSO) and genetic algorithms [79].

Interesting work has been done to address a version of the Art Gallery Problem with distributed mobile sensors. For instance, Pimenta et al. proposed a distributed control law to provide Voronoi coverage of some non-convex environments using a heterogeneous team of networked robots [80]. Schwager et al. extended this idea to provide a control strategy that would converge to a local optimum in arbitrary environments [73]. These works, however, tend to use simplified 2D notions of coverage, looking at the coverage of a sensor projected onto a ground surface. The assumption is also made that sensors can achieve 360° coverage. This simplification can be useful when considering mobile robots or gimbal cameras which are free to rotate; however, with many sensing modalities, instantaneous coverage is limited to a given field of view frustum. While small targets could be tracked over a full rotation by servoing the camera, large targets may not be fully observable in a given frame. For this reason, it can be advantageous to consider field of view limitations. Additionally, these methods assume that the robots on which the cameras are mounted can search along a continuous work space and are able to stop, indefinitely, once a local optimum has been found. While this assumption is suitable for ground-based robots or drones with short mission-times, additional search space restrictions are required when considering long-term sensing with aerial vehicles.

In the realm of motion capture and reconstruction, however, reconstruction accuracy and dynamic occlusions are also key considerations. Chen and Davis discuss placement of monocular cameras for robust motion capture in simple environments, with known dynamic occlusions [71]. To ensure robustness in the presence of dynamic occlusions they include a

probabilistic occlusion model based on the known path of an occluder in the scene. Visibility and resolution are evaluated over discrete sampling points in a target region. A genetic algorithm is used to optimize camera pose, to be robust to discontinuous fitness landscapes. Rahimian and Kearny [75] took a similar approach to Chen and Davis, however they opted for a simulated annealing based solver rather than using a genetic algorithm. They also decided to discretize functions for camera reconstruction error due to convergence angle and camera range into binary functions with fixed thresholds. This simplification allowed them to increase search speeds, at the cost of granularity in solution fitness.

However, both works took a naive approach, attempting to place all cameras simultaneously, increasing the search space dimensionality N-fold. The search space dimension for meta-heuristic search methods, such as genetic algorithms, has a large impact on the search time. This is in part due to the credit assignment problem [81] where it is difficult to discern which change or changes to the solution proposal caused a noticed change in solution quality. This is particularly evident in the convergence time of 3-5 hours for a 17-cameras arrangement suggested by Rahimian and Kearny's method. While this is suitable for a fixed motion capture system whose life will far outweigh the original computation time, it would be unsuitable for an emergency disaster relief effort.

The curse of dimensionailty can be mitigated considerably with the realization that the camera placement problem is *sub-modular* in nature. Informally, a sub-modular problem is one where the function being evaluated exhibits diminishing returns; the added benefit of placing each additional camera decreases as the number of placed cameras increases. Sub-modular functions can be approximately optimized in a greedy-fashion [82, 83], i.e. by searching for the best position of one camera at a time, then fixing its position and solving for the next; solving in this fashion can be shown to obtain a solution that is at worst within a factor of $(1-1/e)$ of the global optimum [84]. Indeed, several works opt for greedy approaches to solve the camera placement problem. Chabra et al. optimized the placement of Microsoft Kinect V1 RGB-D cameras using greedy initialization followed by a fine-tuning step using simulated annealing. As Kinect V1 cameras are prone to inter-camera interference from the active point projection, a probabilistic point-wise data loss metric is applied to the fitness function. This work, however, optimized for the capture of known events (pre-defined medical procedures) in a known environment. The possibility of camera pan and tilt was not considered.

2.2.2 Camera Comparison

Many design considerations rely on the specific imaging hardware selected. The form of the environment map, as discussed in Section 2.1.1, is perhaps the most significant to this work. However, additional considerations, such as interference susceptibility for active cameras and mass when considering perching feasibility, are also important. As such, it was important to understand which cameras would be suitable for the task of setting up a perched drone camera network. Thirty-five 3D cameras and LIDAR systems were investigated in detail. A table summarizing key parameters was compiled, and is included in Appendix B for reference. Figures 6 and 7 show a subset of these cameras plotted with respect to key parameters such as mass, resolution, and range. Suitability for a perching MAV was used to quickly eliminate many options, which are simply too large or heavy. Sorting by mass, a shortlist of cameras could be generated.

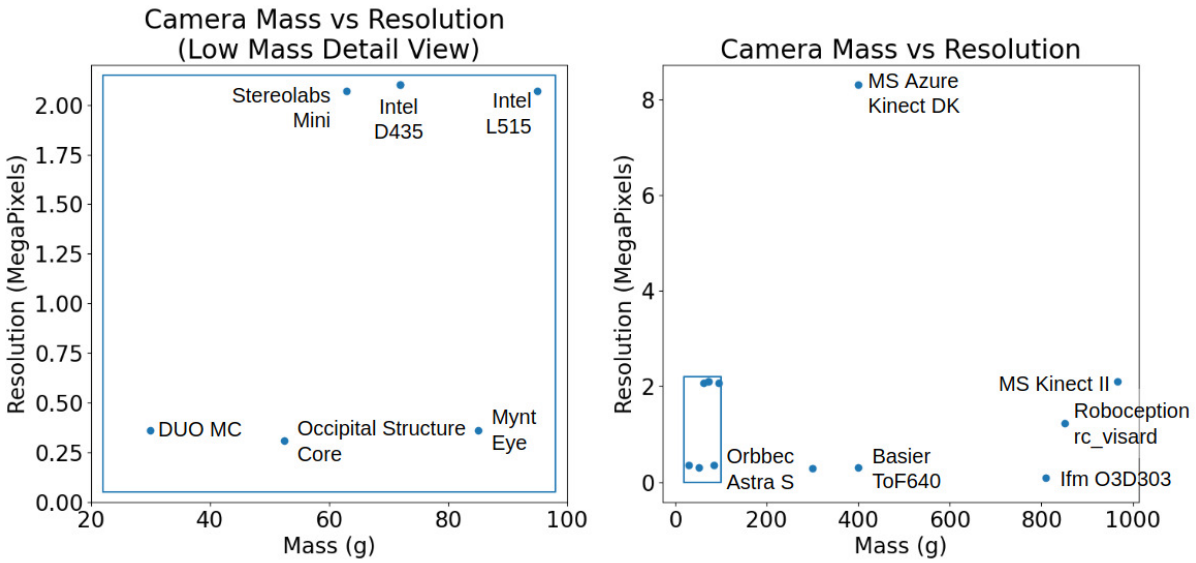


Figure 6: Mass-resolution comparison between common depth cameras of all masses (right) and cameras with masses between 20-100g (left). For airborne applications, the optimal camera would have low mass and high resolution (towards the top-left corner)

Several potential candidate solutions were identified. During the camera comparison, the most suitable camera for a perching MAV platform was found to be the Intel RealSense D435i Stereo RGB-D camera with IMU sensing. The D400 series RealSense cameras provides an excellent balance between imaging quality, mass, and power. Among the lightest depth cameras reviewed, the RealSense cameras boast reasonably high RGB resolutions, wide field of views, and long ranges for collecting depth data. Unlike the Kinect cameras which suffer

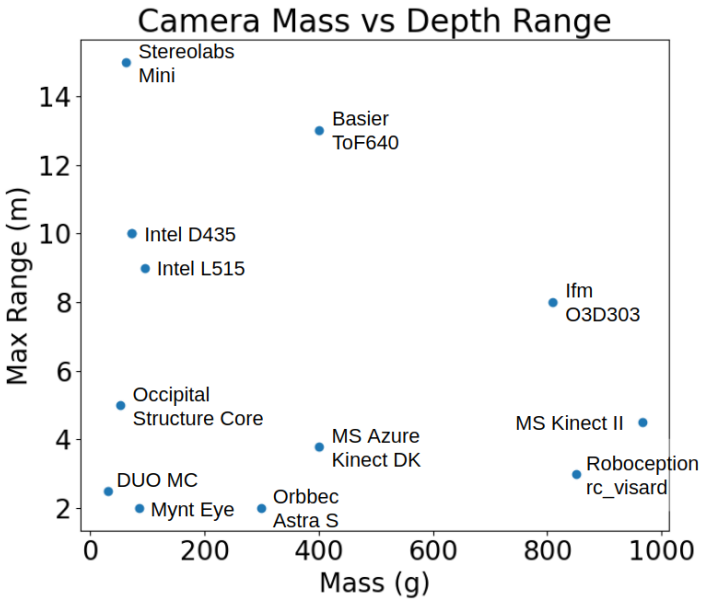


Figure 7: Mass-range comparison between common depth cameras. For airborne applications, the optimal camera would have low mass and a large working range (towards the top-left corner)

from inter-camera interference, depth sensing technique implemented by the D400 cameras is not affected by cross-talk between cameras. However, this comes at a cost. The depth information is more unreliable than Time of Flight (ToF) based cameras such as the Kinect 2 or LIDAR systems like the Velodyne Puck Lite. As the stereo baseline is relatively small, the accuracy in the depth axis is limited, and the depth map often presents large holes due to occlusions.

It was noted that the Intel L515 is potentially a more promising option, with more reliable depth information at a practical weight point, but could not be used for testing due to its Summer 2020 release date.

2.2.3 Camera Models

Several models, of increasing complexity have been proposed to represent camera optics. This section will discuss models which are useful to consider when evaluating the quality of a camera placement.

Pinhole Camera Model

The simplest model is the pin-hole camera model, based on early pin-hole cameras where light rays simply passes through a small hole in the camera to project an image onto film or

an image sensor. The pinhole camera model assumes the camera's lens acts as an ideal point focus. Features are linearly projected through this point onto the image plane. Each edge of a rectangular imaging sensor projects outwards through the focal point, creating a pyramidal volume, often referred to as the camera's visibility frustum. Points existing within the frustum will appear in the camera's image.

Using the pinhole camera model, a linear mapping between 3D world coordinates and 2D image coordinates is possible using the following equation:

$$sm' = A[R|t]M' \tag{2}$$

where:
s is a scalar conversion factor between world units (e.g. meters) and pixel units,
m denotes the position in the image frame,
M denotes the position in the world frame,
$[R|t]$ denotes the camera extrinsic with R representing the rotation matrix representing the transformation between the world coordinate frame and the camera coordinate frame, and t representing the translation of the camera frame with respect to the world origin, and
A denotes the camera intrinsic, defining its focal length and the location of the optical center in the image plane.

More explicitly, for a camera whose optical axis is defined as the $Z-axis$ of the camera frame, this formulation becomes:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

where:
u and v are the horizontal and vertical coordinates of a point in the image frame expressed in pixel units,
f_x and f_y are the focal lengths of the lens in the horizontal and vertical axes, expressed in pixel units, and
c_x and c_y are the pixel coordinates of the optical center of the lens in the image frame.

Visually, this model is summarized nicely by the graphic in Figure 8, courtesy of the OpenCV project's reference documentation. Note that this work will follow the frame convention where the camera's x-y-z axes will be defined by the camera's right-down-forward

directions.

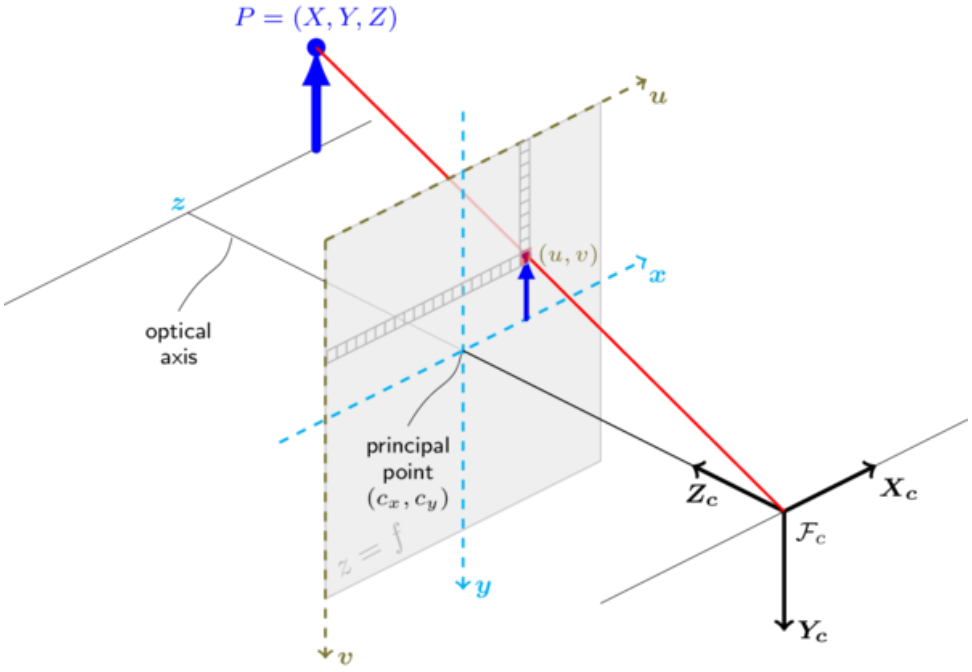


Figure 8: Graphical representation of the linear pinhole camera model, courtesy of OpenCV [85]

A linear pinhole camera model works well as a first order approximation for many cameras, and particularly when considering more expensive cameras with well aligned, precision optics. As cameras become more commonplace, their performance has approached the simple pinhole model. However, it can be worth applying more detailed models for low-cost cameras with poorly aligned or high distortion optics.

Brown-Conrady Distortion Model

When the ideal pinhole model is not sufficient, the next level of detail typically involves identifying non-linear radial and tangential distortion components for the lens. This is most commonly done using the Brown-Conrady camera distortion model, shown in Equations 4 and 5 [86]. Figure 9 demonstrates the manifestation of common distortion types using the Brown-Conrady equations. Radial distortion occurs when light rays are bent more near the edges of a lens than at the center. It can be broadly classified as barrel distortion (common in wide-angle lenses), pincushion distortion (common in low-end telephoto lenses), or mustache distortion (common on large-range zoom lenses). Tangential distortion arises when the image sensor plane is not properly aligned with the lens plane. This misalignment causes the projection distance to change across the length of the image sensor, resulting in a

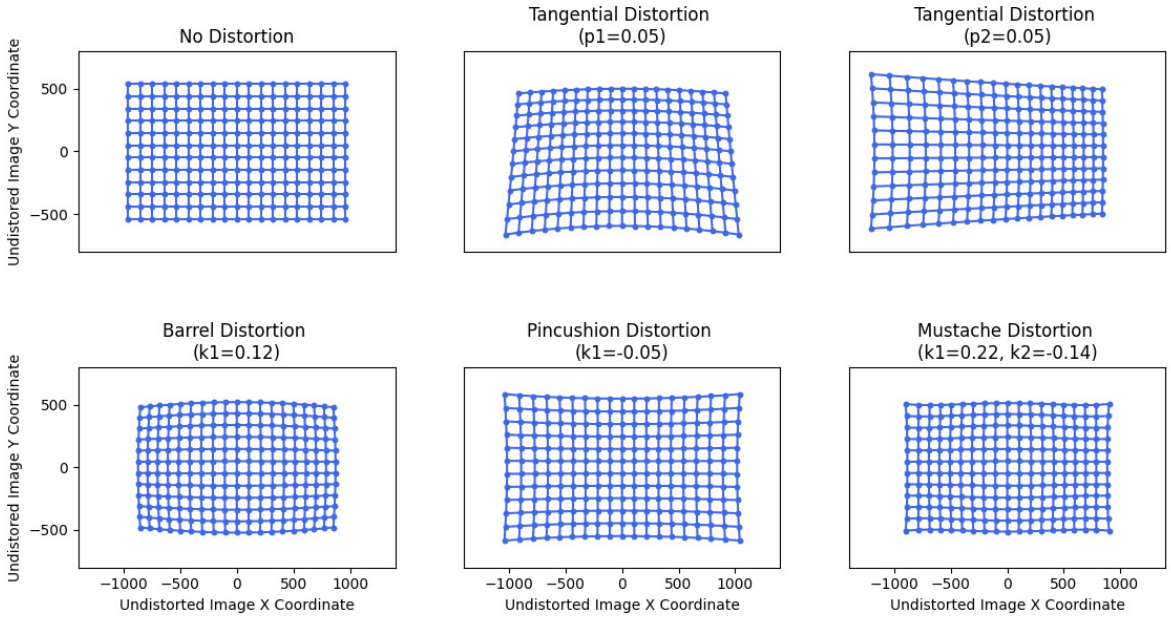


Figure 9: Visualization of different forms of camera distortion.

.

tapered image.

$$x_u = x_d + (\Delta x_d)(K_1 r^2 + K_2 r^4 + ...) + (P_1(r^2 + 2(\Delta x_d)^2) + 2P_2(\Delta x_d)(\Delta y_d))(1 + P_3 r^2 + ...) \quad (4)$$

$$y_u = y_d + (\Delta y_d)(K_1 r^2 + K_2 r^4 + ...) + (P_2(r^2 + 2(\Delta y_d)^2) + 2P_1(\Delta x_d)(\Delta y_d))(1 + P_3 r^2 + ...) \quad (5)$$

where:
(x_u, y_u) are the horizontal and vertical undistorted image point coordinates, in pixels,
(x_d, y_d) are the horizontal and vertical distorted image point coordinates, in pixels,
(x_c, y_c) are the horizontal and vertical distortion center coordinates, in pixels,
$(\Delta x_d, \Delta y_d) = (x_d - x_c, y_d - y_c)$,
$r = \sqrt{(\Delta x_d)^2 + (\Delta y_d)^2}$,
K_n is the n^{th} radial distortion coefficient, and
P_n is the n^{th} tangential distortion coefficient.

In practice lower order terms dominate the distortion, and only the first three radial and two tangential terms are considered [87], resulting in Equations 6 and 7.

$$x_u = x_d + (\Delta x_d)(K_1 r^2 + K_2 r^4 + K_3 r^6) + (P_1(r^2 + 2(\Delta x_d)^2) + 2P_2(\Delta x_d)(\Delta y_d)) \quad (6)$$

$$y_u = y_d + (\Delta y_d)(K_1 r^2 + K_2 r^4 + K_3 r^6) + (P_2(r^2 + 2(\Delta y_d)^2) + 2P_1(\Delta x_d)(\Delta y_d)) \quad (7)$$

These equations can easily be solved to predict the position of points on an undistorted image plane. This forward transformation, from distorted coordinates to undistorted coordinates is important when identifying or tracking features. The inverse transform, from the intersection with an undistorted image plane to the "real" distorted coordinates, is less commonly performed and cannot be solved directly. The inverse transformation is necessary both for estimating the distortion coefficients during calibration, and to predict whether a given ray will intersect the image sensor, given the optical distortion. A first order approximation can be performed by simply applying the forward transformation using negated. There are several open source libraries, such as OpenCV [88], which can quickly perform more accurate inverse transformations using Taylor series approximations. More complex distortion models have also been developed, but are generally not considered to be worth the additional computational complexity. The exception is perhaps with high distortion lenses, such as in fish-eye and omnidirectional cameras, which require some special consideration to minimize reprojection error [89].

Image Noise and Interference

Beyond distortion, it is also important to consider noise and interference, and their impact on tracking uncertainty.

Image noise can take many forms, and is a field of study itself. Briefly, the primary forms of image noise are Gaussian noise, quantization noise, salt-and-pepper noise, and shot noise [90]. Gaussian noise, also called heat noise, is approximately zero-mean, normally distributed, signal independent noise, mostly attributed to thermal noise in the camera electronics. Quantization noise is present in digital cameras due to the inherent discretization and quantization of information, both in terms of positioning, in the pixel space, and in terms of intensity, at the bit level. Both of these uncertainties are uniform over the image sensor and are signal independent, making them easy to model as sources of uncertainty. Other significant forms of image noise include salt-and-pepper noise and shot noise. While noticeable, these forms of noise are difficult to compensate as they are signal dependent, with shot noise being more prevalent in bright sections of an image, and salt-and-pepper noise dominating darker parts of an image.

Generally, noise is quite camera-dependent, environment-dependent, and difficult to

model [91]. Of interest to this work is how image noise may impact tracking uncertainty. When tracking objects in a scene, individual pixel values are typically less important than features like edges and areas. Because these features extend over multiple pixels, there is an inherent filtering effect, that helps diminish the impact of Gaussian noise. Salt-and-pepper noise and shot noise both may impair feature tracking, but cannot be accurately considered without a-priori knowledge of the environment. Additionally, modern cameras employ filtering and simply have better optics, mitigating the impact of these forms of noise. For this reason, they will be ignored when considering placement quality.

Quantization noise will be considered, however. The uncertainty of an edge location at the pixel-level corresponds directly to x-y (image plane) uncertainty in 3D space [92]. Figure 10 demonstrates how quantization error plays a role in tracking uncertainty. It is particularly important to note that the uncertainty in each axis is proportional to the radial distance. By use of similar triangles, Equation 8 can be used to represent the x-y uncertainty of a camera at a depth.

$$\sigma_{xy} = \frac{d e_{subpixel}}{f} \tag{8}$$

where:
d is the measurement distance,
$e_{subpixel}$ is the subpixel RMS error in pixels (camera dependent, typically 0.1),
$f = \frac{1}{2} R_x tan^{-1}(\frac{FOV_h}{2})$ is the camera's focal length in pixels,
R_x is the horizontal resolution of the camera in pixels, and
FOV_h is the horizontal field of view of the camera.

Uncertainty in the x-y plane varies from camera to camera, and in particular with any pre-processing and filtering performed on the images. Sub-pixel resolution is to be expected with modern algorithms, and it is not uncommon to have x-y uncertainty of less than 10% of the pixel width.

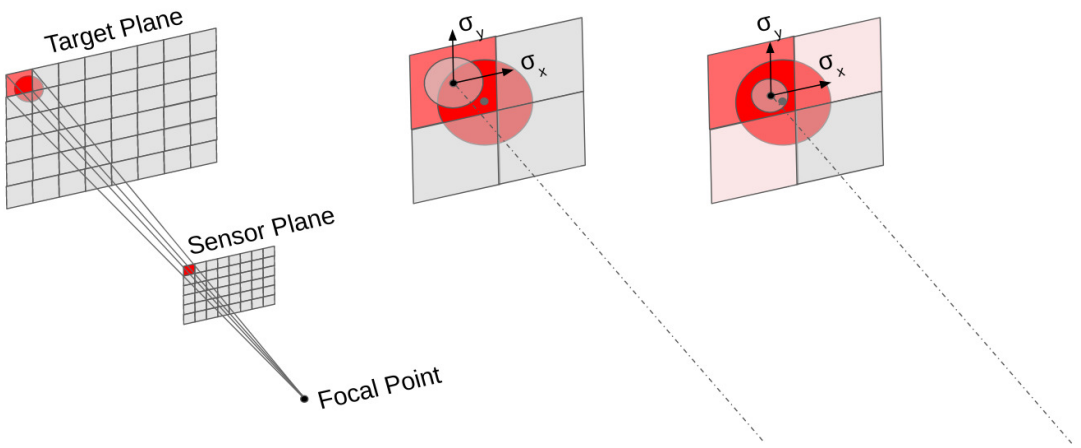


Figure 10: Simplified representation of pixel level matching error. (left) Pinhole projection through the sensor plane and target plane, demonstrating the growth in uncertainty as radial distance increases. (center and right) Close-ups of the individual pixels. The red circle represents the target being tracked. The middle view exaggerates the matching error, by assuming a naive binary thresholding approach. The right view shows a more realistic weighted average approach, where the error is reduced but is still non-zero.

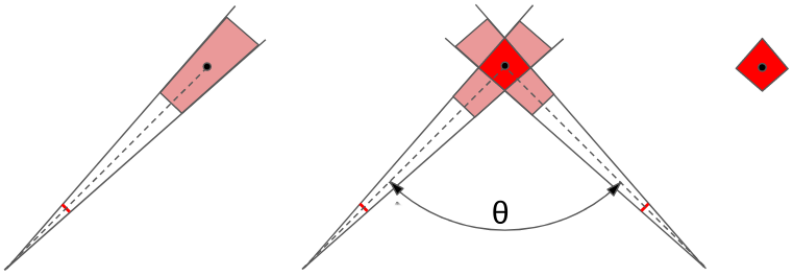
This provides a reasonable model for assessing tracking uncertainty in a camera's x-y plane. For depth cameras, similar considerations can be made for the z-axis. Depth uncertainty tends to be well defined by depth camera manufacturers. Different camera types will have different uncertainty models, largely to do with the mode in which depth information is generated. Focusing on stereo depth cameras, such as the selected Intel D435, the theoretical limit to depth accuracy is defined similarly to the x-y uncertainty, by Equation 9 [93].

$$\sigma_z = \frac{d^2 e_{subpixel}}{f L_b} \tag{9}$$

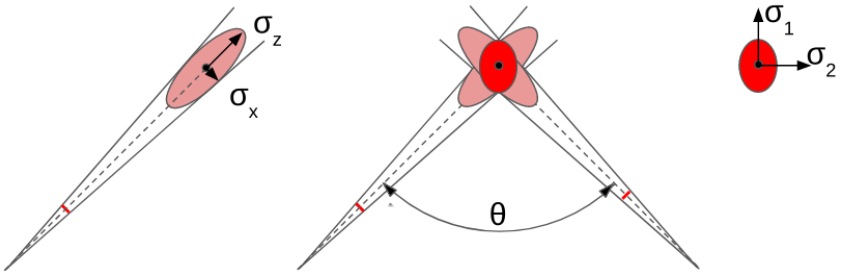
where:
L_b is the baseline distance between the optical center of each camera in the stereo pair.

The big difference between the x-y uncertainty and the depth uncertainty is the consideration of the camera baseline. As stereo matching relies on the distance between the two cameras to triangulate a point in 3D space, the accuracy is limited by the distance between the two cameras (known as the baseline). In practice, commercial stereo depth cameras operate on ranges of up to 10m with baselines of approximately 5-10cm. As a result, the depth accuracy is often an order of magnitude poorer than the x-y accuracy. However, with multiple stereo cameras, the overall uncertainty can be reduced considerably by fusing the two measurements together. Figure 11 demonstrates the propagation of matching error in the x-z

plane for one and two cameras. Because the depth accuracy of stereo cameras is an inherent limitation, creating a camera arrangement that maximizes the diversity of viewpoints (often referred to as the convergence angle between cameras [94]) helps in minimizing uncertainty.



(a) Discrete uncertainty volume representation



(b) Ellipsoidal uncertainty approximation

Figure 11: Matching error in the x-z plane. The left graphics show the uncertainty of a point tracked by a single camera. The middle graphics demonstrates how multiple cameras can be fused together to generate an improved state estimate. The right graphics show the isolated improved state estimates. The ellipsoidal representation allows uncertainty of multiple sensors to be estimated using a weighted least squares combination (bottom right).

By approximating the uncertainty bounds of a given point using a covariance matrix, data points from multiple cameras can be fused together. For zero-mean, uncorrelated noise, this is optimally done using weighted least squares combination. Uncertainty for a camera, in the camera frame, at a given point in space can be given by the following covariance matrix:

$$C_{camera} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_x^2 & 0 \\ 0 & 0 & \sigma_x^2 \end{bmatrix} \tag{10}$$

This covariance matrix can be reoriented into the world frame by applying the following transformation:

$$C_{world} = RC_{camera}R^T \tag{11}$$

where R is the rotation matrix transformation between the camera's local frame and the world

frame.

For C cameras observing a single point, the position uncertainty can then be expressed using a weighted least squares combination:

$$C_{point} = (\sum_{c=1}^{C} C_{world,c}^{-1})^{-1} \tag{12}$$

Interestingly, this formulation pairs nicely with a greedy search approach. The point-wise uncertainty for previously placed cameras does not need to be re-computed with each new camera placement. Instead, the covariance of the $N-1$ camera arrangement can simply be inverted, summed with the inverse of the covariance matrix for the new, proposed camera, and re-inverted.

2.3 Robotic Perching

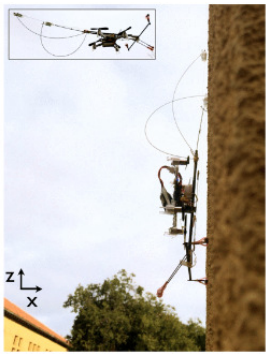
Perching is the act of landing or resting on a structure in the environment. In biology, perching can take many forms and serve many purposes. Many species of birds perch on elevated horizontal structures, such as tree branches, power lines, and buildings, to rest, nest, avoid predators, or search for food. Perching on vertical and even inverted surfaces is not uncommon, as exhibited by species such as the woodpecker (see Figure 12) and many bats species.

Figure 12: (left) A woodpecker perching on tree (courtesy of [95]). (right) A bio-inspired perching mechanism (courtesy of [96])

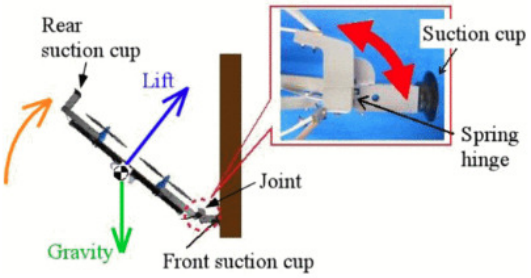
Perching for aerial robots has gained popularity among robotics researchers in recent years, for many similar reasons. For aerial robots, the potential benefit of perching is two-fold.

Intuitively, perching offers a simple and effective energy saving solution. Rather than actively maintaining flight continuously, taking breaks when assessing the situation or waiting for commands can increase the effective length of a mission by simply distributing the energy intensive flight components across discrete intervals [97, 98]. For robots which are designed interact with the environment rather than just survey it, adhering to static surfaces in the environment can allow the robot to exert forces which are an order of magnitude larger than what would be possible while airborne [12, 13]. As perching serves to enhance the possible operational modes of a vehicle, the perching mechanism is often tailored to a specific use case, resulting in a broad range of possible solutions, each with its own strengths and limitations.

With a diverse set of natural and artificial inspirations, researchers have explored a wide variety of solutions to the perching problem, designing unique mechanisms and control strategies to address the challenges of changing a vehicles operational mode. Perching mechanisms can take many forms, dependent on the target perching surface, vehicle size, and vehicle mission requirements. These can be broadly categorized into grasping mechanisms and adhesive mechanisms. For the purpose of this discussion, grasping mechanisms will be defined as mechanisms which rely on positive engagement with macroscopic features in the environment to support normal and tangential forces, including actuated claws and fingers. Adhesive mechanisms are those which rely on surface interactions at the microscopic scale to support normal and tangential forces; examples of which include gecko-inspired structured adhesives, microspine adhesives, electrostatic adhesives, and suction cups. Figure 13 demonstrates some of the methods that have been explored in recent works.

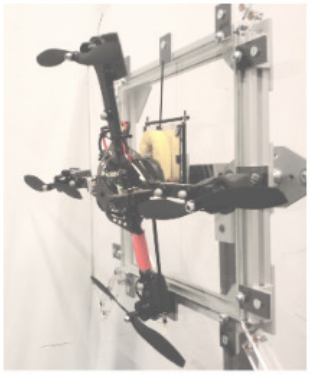


(a) Microspine Claws [99]

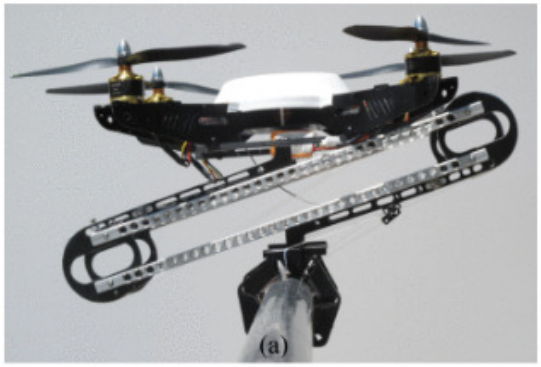


(b) Active suction cups [12]

(c) Harpoon perching [100]



(d) Gecko adhesive gripper [101]



(e) Passive avian gripper [96]

(f) Magnetic gripper [102]

Figure 13: Visualization of the different mechanisms used by recent works to achieve aerial robotic perching.

2.3.1 Grasping Mechanisms

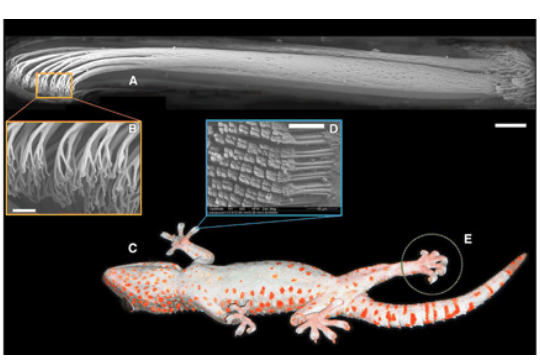
Many works investigating grasping and perching mechanisms take inspiration from the animal kingdom, opting for toe-like appendages which close around cylindrical objects such as pipes or branches, closely mimicking the structures and behaviours seen in birds and

bats [96, 103–105]. This can result in the mechanism being able to sustain larger forces than a comparably sized adhesive based approach, as the tensile strengths of most engineering materials are approximately three orders of magnitude larger than the maximum shear stress afforded by state of the art dry adhesives [106, 107]. However, finding a suitable perching location and subsequently achieving a robust grip are both non-trivial tasks and remain active fields of study. Additionally, grasping mechanisms are inherently limited to only grasp objects which meet specific size constraints. In structured environments, this can become a non-issue. For instance, in an urban environment, there may be regularly spaced street lights or overhead cables which can provide ample, reliable perching space [108]. For pipeline inspection, the pipeline itself acts as a reliable perching target [102]. However, in unstructured environments, such as dense forests or in indoor environments, grasping mechanisms may be overly restrictive.

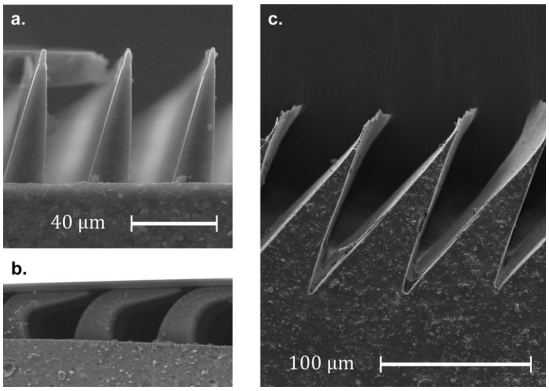
2.3.2 Dry Adhesive Mechanisms

Dry adhesive mechanisms form another sector of the explored perching design space. Dry adhesives are materials or mechanisms which can generate considerable adhesive forces while, unlike conventional "wet" chemical adhesives, being detachable and reusable over several contacts [109]. Dry adhesives come in many forms, with some of the most popular being gecko-inspired fibrillar adhesive [109], microspine adhesives [107], electrostatic adhesives [110], and suction cups [12]. Figure 14 demonstrates an up-close view of different dry adhesive mechanisms.

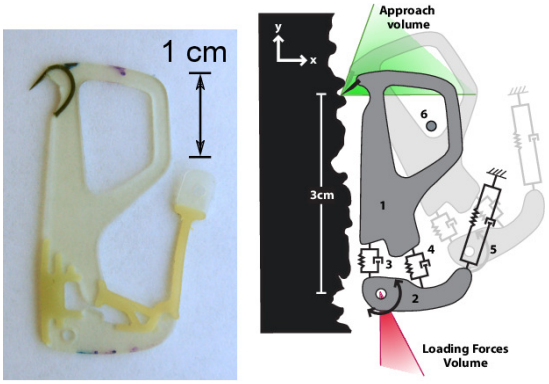
Geckos are able of generating tremendous adhesive forces using branching nano-scale fibrillar structures on their feet which are capable of generating Van der Waals intermolecular adhesive forces with the surface [106]. Their synthetic counterparts try to mimic these structures with micro-scale structures, such as mushroom caps [111], wedges [112], or hairs [113]. As the structures are anisotropic, the adhesives are directional, allowing for the adhesive to be engaged and disengaged using small directional displacements. Artificial gecko adhesives provide excellent adhesion on smooth surfaces, but have less success on rough surfaces with macroscopic surface features.



(a) Multi-scale view of the structures which give gecko feet their adhesive properties [114]



(b) Wedge shaped artificial gecko adhesive [115]



(c) Microspine gripper [107]

Figure 14: Detailed visualization of different dry adhesive mechanisms used in robotic perching.

Suction cups share much of the same use case as gecko adhesives, and adhere well to smooth, clean surfaces. Passive suction cups are simple and lightweight adhesive options, but tend to lose their grip as the cups gradually allow air to seep in, reducing the pressure differential. For this reason, they are particularly sensitive to rough or dirty surfaces. A more popular option in the field of robotic manipulation is to use active suction to maintain near-vacuum to ensure an effective grip is maintained. While this works well for industrial applications, aerial robots are sensitive to the added mass and power requirements to maintain active suction. For this reason, if the goal is extending mission time, passive solutions are often preferable.

Electrostatic adhesives use high voltages to generate modest electrostatic adhesive forces. Electrostatic adhesion generated scales with Voltage squared, surface area, and the square of the inverse of the distance between electrodes [116], and can typically generate on the order of 200 Pa [110] - several orders of magnitude lower than other methods. However, these adhesives can be made to be very lightweight and have found use in many micro robotic applications. Finally, there are microspine grippers, which use arrays of spines or needles

with microscopic tip radii to hook onto small asperities in a surface. These spines are also capable of embedding themselves in softer materials [117, 118]. In man-made environments, soft materials such as wood and carpets and rough materials such as stucco, roof shingles, or unpolished concrete can provide these types of mechanism excellent purchase [111, 119, 120]. These surfaces can also be readily found in natural environments, with rough stones and trees being suitable. However, the strength of the surface can be a limiting factor on the force limits of the perch [14]. Microspines are also prone to dulling over time, reducing their effectiveness on smaller asperities.

While the mechanisms employed in these different types of dry adhesives are very different, they translate to similar requirements for a perching robot. They require relatively flat, planar surfaces which the mechanism can conform to, with a suitable surface characteristic. Dry adhesive mechanisms are also gravity independent, allowing for robots to perch on vertical and inverted surfaces. Electrostatic adhesives are unique in that they do not require any directional preloading of the adhesive to be effective. Conversely, suction cups, microspine grippers, and gecko adhesive pads all allow require some amount of normal or tangential preload to be applied, resulting in a non-zero contact force constraint.

2.3.3 Perching Maneuvers and Control

The control complexity of a perching maneuver is largely dependent on the type of vehicle and the type of mechanism selected, which determine the required approach maneuver. For instance, most of the grasping mechanisms developed for drones employ a simple top or bottom side gripper which can attach to perch points when the drone is flown upwards into or downwards onto the target [103, 105, 108, 121] . Similarly, mechanisms with front-facing adhesives substantially simplify the process of perching on vertical surfaces by simply requiring the drone to fly forward into the surface until contact is made [11, 12, 97, 122, 123] . These approaches can rely on visual servoing techniques to perform low-level control corrections on the approach. However, since only small corrections will likely be required, standard linearized controllers based on the Euler angles of the vehicle are sufficient.

Things become more complicated when the drone must pitch up to intersect vertical surfaces. For aggressive maneuvers, linearized controllers based on the small angle approximation will no longer be sufficient [101]. Additionally, controllers based on a representation of attitude through Euler angles result in singularities due to gimbal lock. For this reason, alternate representations of attitude through quaternions or rotation matrices are required. Geometric control techniques were popularized for quadrotors by Lee et al. and can result in nearly globally attractive controllers [124, 125]. As shown in Figure 15, drones with gecko

adhesive or spine grippers mounted along their vertical axis can benefit from these advanced control techniques to execute fast and accurate maneuvers to transition from level hovering into a ballistic trajectory towards the target. The momentum of the drone can be repurposed to engage the adhesives with the surface and ensure the drone does not bounce off. While embracing perching on horizontal and vertical surfaces affords the robot an increased perching space, it is by no means easy, even when modern control techniques are implemented.

Figure 15: Stroboscopic image showing the ballistic trajectory of a vertical surface perching maneuver with a downward facing gecko adhesive gripper, from [101]

Microspine adhesives are inherently stochastic, with the quality of the grip depending on the local variations in material surface and the relatively unpredictable engagement of spines on surface asperities [126]. Alignment with the surface is critical, as angular deviation from the surface normal can prevent the perching mechanism from successfully maintaining contact. Outriggers can help correct modest misalignment prior to contact, but accurate state estimates are still required. Aggressive perching maneuvers require the drones to make split-second guesses as to whether or not the attempt was successful in order to have time to take corrective action if the perch attempt was a failure [101, 127]. Previous experiments [127] with perch recovery showed that an average of 0.8 m needed to detect and correct a failed perching attempt, when using inertial sensing only. More advanced detection models and sensors could likely yield shorter recovery drops, and remains an active research topic.

2.3.4 Perch Location Identification

While much of the focus in literature has been on the hardware and control aspects of the perching problem, few have discussed autonomous detection and selection of perching locations, with the majority opting instead for manual control or preprogrammed waypoints. To the author's knowledge, [108, 121] are the only works directly addressing the identification of perching locations. These works discuss the use of computer vision feature detectors to

identify and servo towards basic geometric shapes such as rectangular plates, box corners, or cylindrical tubes. While these works show that visual servoing can be an effective and computationally efficient method of approaching and perching on surfaces, the scenarios presented are still idealistic, with only one obvious perching location made available to the robot. Unlike previous works, this work will address perch location identification and selection in unstructured and cluttered environments.

3 System Objectives and Requirements

The detailed review of existing hardware and software solutions for environment mapping and reconstruction, segmentation, and camera placement, as well as discussions with project stakeholders, helped to define the following system objectives and functional requirements.

System Objectives

1. The system should be able to identify potential perching locations, given a densely reconstructed environment mesh.
2. The system should maximize coverage over regions of interest.
3. The system should minimize tracking and reconstruction error over the regions of interest.
4. The system should minimize the workload of the human operator.
5. The system should leverage the operator intuition when appropriate.

Functional Requirements

1. The system must generate relevant perching poses for a quadrotor with an RGB-D camera mounted below the frame, and an adhesive gripper mounted above the frame.
2. The system must be able to identify potential perching locations on vertical and inverted surfaces.
3. The system must be able to identify potential landing locations on horizontal surfaces.
4. Selected perching locations must leave adequate free space to allow a drone to be able to complete its perching maneuver.
5. Selected perching locations must consist of a material that is compatible with the selected perching gripper.
6. If multiple drones are being placed, sufficient spacing must be maintained between selected positions to avoid perching conflicts.
7. The system must be compatible with any realistic room shape, target volume, drone parameters, and camera parameters.

4 METHODS

This section will give an overview of the system that was developed to address the Where To Perch problem. The system implementation and key design decisions will be discussed in detail. Subsequently, the methods used to validate the system will be listed.

4.1 System Overview

In order to address the goals outlined in Section 3, the following pipeline (demonstrated in Figure 16) was developed.

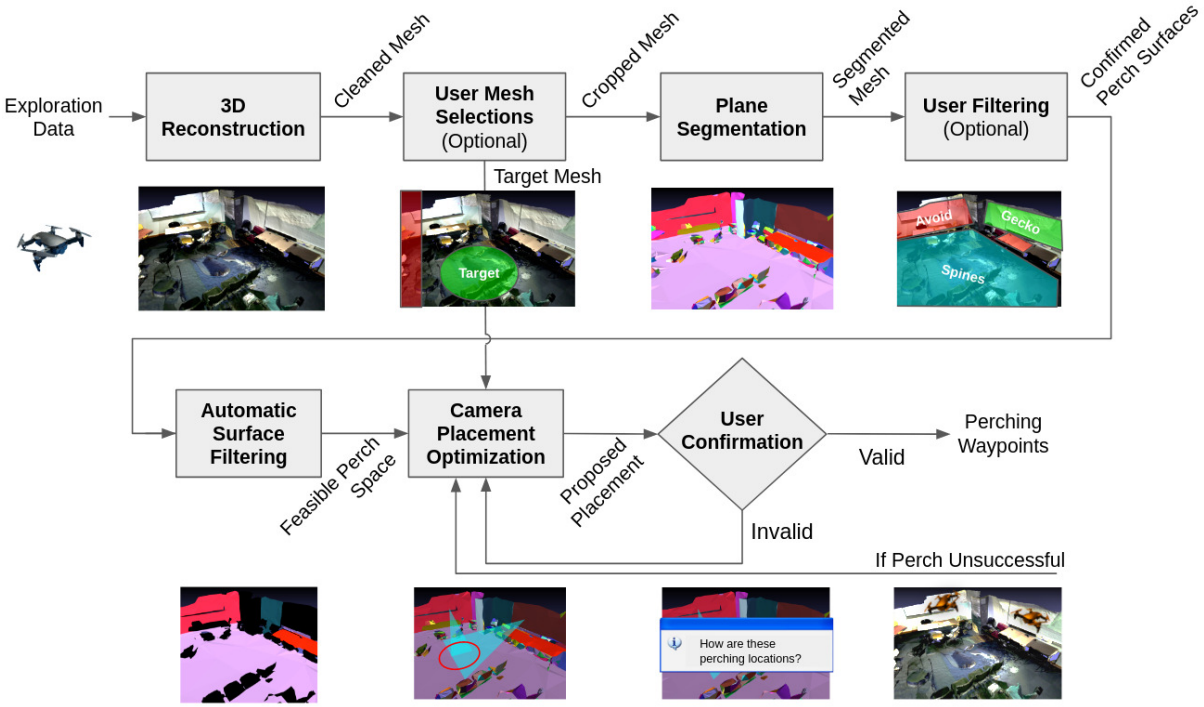


Figure 16: Top level representation of the developed camera placement solution

In layperson's terms, the steps are as follows:

1. A robot enters and explores an environment while collecting RGB-D video and inertial sensor information. This information is passed into the pipeline as the primary input.

2. The dense reconstruction node is run to generate a 3D triangulated mesh of the environment that is aligned with the quadrotor's world frame. Basic mesh cleaning processes are automatically performed to remove islands and reduce the file size as necessary.

3. The mesh reconstruction is displayed to the user. The user can optionally crop out sections of the mesh that are irrelevant to the camera placement search (e.g. surfaces from along the flight path leading to the target environment). The user can also select a portion of the mesh to set as the motion tracking target volume. If no region is selected, the entire mesh volume will be used as the target volume. The cropped mesh will be passed to the plane segmentation node. If a target mesh is created, it will be passed to the camera placement optimization node.

4. The plane segmentation node segments large planar surfaces from environment. These are passed to a user-verification node, where invalid surfaces can be eliminated from the search set, based on the user's discretion.

5. A simple graphical interface displays the reconstructed environment and highlights target regions in the mesh for the user. The user gives a pass/fail assessment for the subset of largest surfaces in the scene. If a surface has been poorly segmented, the user can perform basic cropping operations on the surface to improve the segmentation prior to optimization. The winnowed set is passed to the surface filtering module. The graphical interface also allows the user to manually select a target region in the environment, to narrow the search efforts.

6. The surface filtering node evaluates each planar surface based on user specified requirements for perching, such as perch direction, minimum landing window or minimum distance to obstacles. The minimized set of perching locations is passed to the camera placement optimizer.

7. The camera placement optimizer runs a meta-heuristic search to place cameras, one-by-one. Optionally, basic tests are run to inform the user of the quality of the found solution.

8. The placement locations are passed to the user interface and are used to confirm the automatic selections. A rejected placement will remove surfaces within a small radius of the selection from the search space, and the search will try again. If the placement is approved, a waypoint is published to the drone dispatcher.

9. If the selected perch location results in a failed perching attempt, the user will have the option to remove the location from the search space and re-attempt optimization.

The following sections will discuss each component in more detail.

4.2 Environment Reconstruction

While environment reconstruction was not the focus of this work, many powerful open source tools exist which can achieve this task. In order to make sure the system could perform under realistic conditions and prevent over-fitting to published data sets, it was important to be able to generate new data for new environments using flight-ready sensors.

A deep search of available depth cameras was conducted, and is discussed in Section 2.2.2. Of sensors which were currently available, the Intel RealSense D435i proved the most promising for MAV applications. Unfortunately, access to lab hardware was severely limited due to the COVID-19 crisis. While it was not possible to access a D435i camera for testing, a D435 camera was made available; the only difference between the two cameras is that the D435i (inertial) model includes a clock synchronized IMU, making it ideal for VIO applications. While it is recommended, for an aerial robotic system to take advantage of its onboard inertial sensing and employ a VIO algorithm, it is still possible to perform dense 3D reconstruction using purely visual odometry. Validating the system using purely visual state information will guarantee that a system with visual-inertial state information will perform at least as well.

In future work where inertial sensing is available, a modern package such as Kimera [39] would be recommended, as it allows for simultaneous mapping, reconstruction, and semantic labelling. The latter requires considerable computational hardware to run in real-time, and would likely not be suitable to run onboard a MAV.

4.2.1 VO Dense Reconstruction

To satisfy hardware restrictions during testing, environment reconstructions used in following experiments are generated using Intel's Open3D Library [128], which provides a dense reconstruction module. The Open3D dense reconstruction module uses only visual information. VO is performed using ORB-SLAM 2 [38]. Image frames in the input RGB-D stream are first clustered into small video fragments. For each fragment, each frame is oriented using the estimated camera pose and is then integrated into a point-cloud fragment using a scalable Truncated Signed Distance Function (TSDF) volume integration. Each local 3D fragment is registered to the global frame. This is done in two parts. First, global registration is performed to identify and close loops and provide an initial coarse alignment between fragments. Then, local registration methods are performed using Iterative Closest Point (ICP) registration to fine-tune the registration. The final step is scene integration, fusing the graphical content from the original RGB-D frames into the aligned geometry to create a final, textured mesh.

Dense reconstruction is accelerated using an unofficial GPU enabled fork of the Open3D library [129]. With GPU acceleration, accurate dense scene reconstructions are possible in nearly real-time (i.e. 1s of video input takes approximately 1s of processing). Faster speeds have been reported when run on systems with more available RAM and GPU memory [129].

4.2.2 Mesh Post-Processing

The dense mesh reconstruction is then automatically post-processed prior to passing it to the next node perch placement pipeline. Post-processing is performed using MeshLab, a powerful, open source, 3D geometry processing system developed by Cignoni et al. [130]. MeshLab includes fast, parallelized, C++ implementations of several common mesh processing filters. Filters can be applied automatically to geometry but running a headless MeshLab server instance; thus post-processing can be performed without requiring cumbersome user interface interactions.

Post-processing consists of two operations. First, small, disconnected islands smaller than 0.3 m are removed from the mesh using MeshLab's "Remove Isolated pieces (wrt Diameter)" filter. These islands are typically the result of poor registration of fringe points, and can safely be removed to reduce mesh complexity and improve plane segmentation efficacy.

Next, if the mesh contains more than 100,000 faces, it is simplified using MeshLab's "Quadric Edge Collapse Decimation" filter. Quadric decimation is a technique which iteratively collapses mesh edges and merges vertices to simplify a mesh [131]. Edges are selected such that the reconstruction error quadrics are minimized. The edge collapse procedure is applied iteratively until a target number of faces remain in the mesh. For reference, the quadric decimation algorithm is described in Appendix C. Figure 17 shows visually how quadric decimation works and minimizes error compared to other approaches.

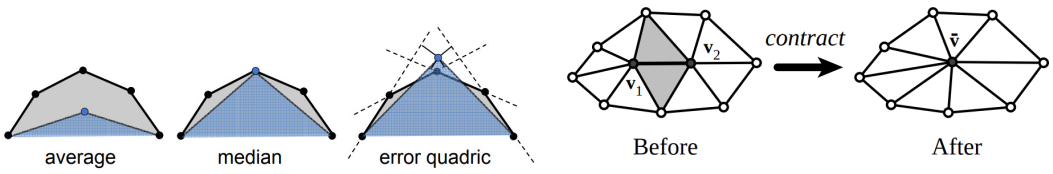


Figure 17: Left: Visualization of quadric decimation, compared to mean and median decimation approaches (courtesy of [132]). Right: Visualization of edge contraction (courtesy of [131])

Optionally, the pipeline can accept user input to crop the reconstruction to limited regions of interest for perching. This will be discussed in greater detail in Section 4.6.

Mesh size reduction is necessitated by memory limitations of the test hardware during

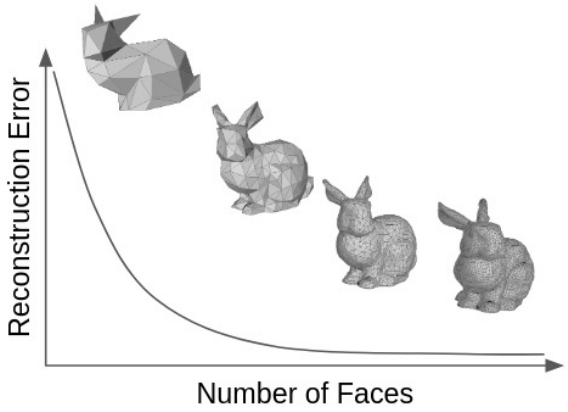


Figure 18: Mesh decimation can substantially reduce the complexity of the mesh, without introducing too much error.

testing (see the hardware discussion in Section 4.8 for more detail). Wang et al.'s plane partitioning algorithm (discussed more in Section 2.1.3) requires on the order of 20 GB of RAM memory for an input mesh with 1,000,000 faces [61]. The target number of faces can be scaled with the available system memory. On the test system with 8 GB RAM, 300,000 faces was the approximate limit. Beyond this point the system will not have enough memory to perform the place segmentation step, and all subsequent mesh operations will be bottlenecked by disk read/write operations.

Note on reconstruction density:
It may appear to be redundant to first generate a dense, detailed mesh, then run a simplification procedure. Indeed, if the approximate size of the environment is known, it is possible to simply reconstruct the environment at a lower density by increasing the base voxel size. The base voxel size specified for a reconstruction denotes the minimum volumetric element to be tracked between frames. For instance, specifying a voxel size of 5cm would result in a a reconstruction which tracks 5cm cubes, resulting in a mesh where the minimum element size is 5cm. Intuitively, voxel size determines the number of volumes tracked across frames and has a significant impact on computation time and memory requirements.

There is no free lunch, however. First, reduced computation complexity comes hand in hand with reduced model complexity, potentially causing small relevant features to be lost. Following camera placement, the next objective is manipulation of objects within the scene using FlyCroTug robots. The reconstruction resolution should preserve features of interest for grasping and manipulation tasks, to better inform subsequent planning algorithms without requiring an additional reconstruction step. Second, the mesh generated using larger voxels

will have higher reconstruction error than a comparably sized mesh which was decimated from a dense reconstruction. This is because changing the voxel size increases the minimum feature size indiscriminately. All features, regardless of complexity or curvature are impacted. Quadric decimation, in contrast, reduces complexity optimally, targeting low curvature regions first. For these reasons, and because the test computer has sufficient available memory, voxel size is held constant in this work. Voxel size augmentation can be considered in future work on systems with a more limited system memory. This may be particularly important if the reconstruction node is to be run on an embedded computer onboard the aerial robot.

4.3 Plane Segmentation

Once a dense reconstruction has been generated and preliminary filtering steps are complete, the next step is to extract relevant perching features. As mentioned in Section 3, this work will focus on perching with an upward facing dry adhesive gripper. As a result, the drone will be able to land on horizontal surfaces and perch on vertical and inverted surfaces. Surfaces should be nearly planar and large enough to support a perching attempt with imprecise state information.

To identify potential perching locations, plane segmentation is performed using a modified form of the plane partition algorithm presented by Wang et al. [61]. Briefly, Wang et al.'s mesh partitioning algorithm first initializes clusters for each vertex in the input mesh. Clusters are merged iteratively, using the process of PCA-energy minimization suggested by Cai et al. (see Eq. 1) [70]. Finally, clusters are merged by applying simple threshold criteria, comparing cluster position and orientation. Merging is made more efficient by re-using calculations from the PCA-energy minimization. Namely, the 3rd eigenvector of the cluster is equivalent to the normal vector of the best fit plane through the data. The mean position of the cluster is also computed when performing PCA. This can be re-used when comparing cluster center positions.

Starting with a dense, un-clustered mesh, Algorithm 1 is used to partition the environment into planar regions.

In the algorithm proposed by [61], planar clusters are merged when they meet geometric criteria. The criteria are as follows:

- the angle between the normals of each cluster must be sufficiently small;
  (i.e. $|\vec{n}_1 \cdot \vec{n}_2| < cos(\theta_{max})$),

Algorithm 1: Plane Partition Algorithm

```
load mesh;
for face in mesh faces do
    create Cluster from face;
    find Cluster neighbors;
    add unique(neighbor edge) to EdgeList;
    add Cluster to ClusterList;
end
for edge in EdgeList do
    compute edge contraction energy;
    add edge to Heap
end
while # clusters > target # and heap size > 0 do
    pop minimum energy edge from Heap;
    apply edge contraction;
    update EdgeList, ClusterList;
end
for Cluster in ClusterList do
    for N in Cluster Neighbors do
        if can merge(Cluster, N) then
            merge(Cluster, N);
            remove N from ClusterList;
        else
            remove N from Cluster Neighbors;
            remove Cluster from N Neighbors;
        end
    end
end
save clustered mesh;
for Cluster in ClusterList do
    if Cluster area > threshold then
        save cluster;
end
```

- the vector connecting the center of the clusters must have a sufficiently large angle with respect to the normal of each cluster. This prevents parallel planes from merging and creating ill-formed geometry;
  (i.e. $(\vec{c_1} - \vec{c_2}) \cdot \vec{n_1} > cos(\theta_{min})$ and $(\vec{c_1} - \vec{c_2}) \cdot \vec{n_2} > cos(\theta_{min})$)
- the distance between the planes must be sufficiently small;
  (i.e. $mindist(C_1, C_2) < d_{max}$)

The criteria are shown in Figure 19 for reference. While there is no hard rule for how these thresholds should be set, in practice the following values are effective:

- $\theta_{max} = 15°$
- $\theta_{min} = 70°$
- $d_{max} = 0.2m$

More conservative thresholds will result in a larger number of clusters being preserved, and fewer conservative thresholds will results in a smaller number of clusters being preserved.

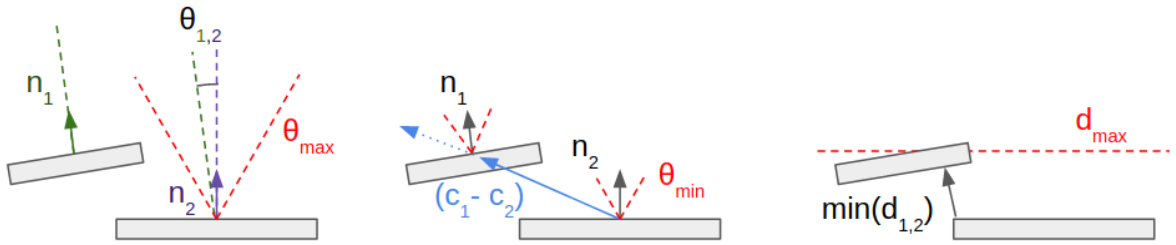


Figure 19: Plane merging criteria. (left) relative angle criterion, (middle) center angle criterion, (right) relative distance criterion

This algorithm performs well to segment distinct geometric surfaces. However, there are many instances where geometric information alone is not sufficient. An example of such a case is a door or window that is flush or nearly-flush with the containing wall. Especially in a sparsely generated representation of the environment, the geometric distinction of the surface may be insufficient to distinguish the surface boundary given the noise threshold of the data. These scenarios are important to consider in the context of robotic perching for several reasons. First, the surface material may have a significant impact on the maximal adhesive force. Subtle changes in the possible adhesive force could result in changes in the required perching maneuver, which can have cascade effects on the required perching window. More pronounced changes in perching adhesion could invalidate a surface altogether. In cases

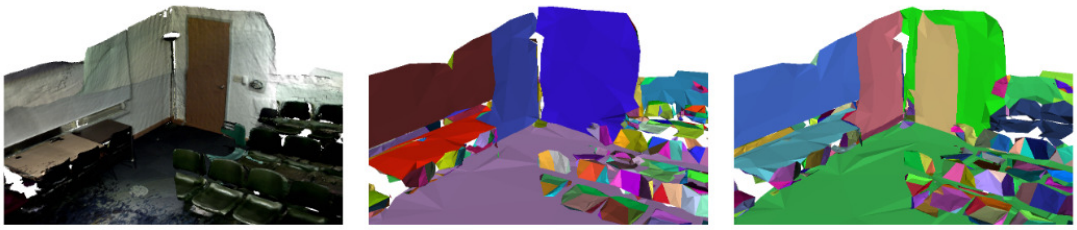


Figure 20: Example of plane partition results with and without color thresholding. While the planar regions extracted are mostly comparable, a key difference is the door. When color thresholding is disabled, the door and wall are included in the same cluster. When color thresholding is enabled, the door and wall are kept in separate clusters.

where geometric segmentation is not sufficient, adding an additional segmentation function, such as comparing semantic labels, can be beneficial.

In lieu of proper semantic labelling using machine learning techniques, this work briefly experiments with a simple approach of color based thresholding. In addition to geometric thresholds, adjacent clusters are only merged if the mean colors of points within each cluster are within a certain distance of each other. Color separation is measured using euclidean distance between the colors. Figure 20 shows the difference in partition boundaries when a color threshold is set. Color thresholding is very much a heuristic approach, and is less robust than a trained feature detector. However, in combination with an operator who can quickly review segmentation results, color thresholding offers a first step towards generating better plane segmentations.

4.4 Automatic Surface Filtering

The final step prior to optimization is to perform preemptive feasibility checks of the filtered surfaces. There are several geometric criteria that surfaces must meet in order to be feasible for perching. These feasibility checks are easily automated using computational geometry, and ensure that the optimization search does not waste time searching invalid regions.

This node applies the following checks to filter input perching surfaces:

1. Sort surfaces by their orientation and discard surfaces which do not meet the requirements for the selected perching gripper.

2. Remove vertical surface regions which do not meet a minimum perch-failure-recovery height.

3. Remove surface regions that are within a minimum radius of environment obstacles.

4. Remove surface regions which do not have a direct line of sight to the target volume.

5. Remove surface regions which do not afford a large enough window for perching.

The first filter is the simplest to compute, and is thus performed first. The latter 4 can all be processed simultaneously, as discussed below.

4.4.1 Orientation Filtering

It is useful to bin surfaces into distinct categories, prior to trying to make decisions. In the case of perching, it is useful to distinguish between surfaces that are oriented upwards (floors), horizontally (walls), and downwards (ceilings). Depending on the selected gripper, ceiling or wall perching may not be possible. The camera configuration on the drone may also be a limiting factor. For instance, if the drone is not equipped with a gimbal and the camera is mounted below the drone, landing on floor surfaces would not provide any useful camera information.

For these reasons, surfaces are first filtered by their orientation. Valid orientations are predefined given the user. Surfaces are then classified by determining their normal orientation. The surface normal is already known from the plane segmentation. However, the scene's context is important in determining the correct orientation of the vector. Prior to labelling, all surface normal vectors are re-oriented, if necessary, so that they point towards the center of the environment, using the simple relation:

$$\vec{n}_i = \begin{cases} \vec{n}_i & if\ \vec{n}_i \cdot (\vec{c}_{env} - \vec{c}_i) >= 0 \\ -\vec{n}_i & else \end{cases} \tag{13}$$

where $\vec{n}_i$ is the normal vector of the best fit plane through the ith surface, $\vec{c}_{env}$ is the centroid position of the environment mesh, and $\vec{c}_i$ is the centroid position of the mesh.

This way, the normal vectors for floor surfaces will point upwards, ones for ceilings will point downwards, and ones for walls will point inwards.

Surface i's orientation in the world frame is found by finding the angle, θ_i, between the normal and the vertical unit direction, $\vec{k}$.

$$\theta_i = cos^{-1}(\vec{n}_i \cdot \vec{k}) \tag{14}$$

Surfaces are separated using the following binning equation:

$$class_i = \begin{cases} floor & if\ \theta_i <= \theta_{threshold} \\ wall & if\ 90° - \theta_{threshold} < \theta_i° | <= 90° + \theta_{threshold} \\ ceiling & if\ \theta_i >= 180° - \theta_{threshold} \\ intermediate & else \end{cases} \tag{15}$$

A boolean flag exists for each surface classification in the pipeline's configuration file. In tests, $\theta_{threshold}$ is set to 15°.

During this step, boundary surface is also identified. Of particular interest is the floor boundary surface, whose position is used to define the target volume, and for height-based thresholding (discussed in Section 4.4.2). The boundary surfaces for a given direction, $\vec{d}$, are defined as the set of surface clusters that are approximately normal to $-\vec{d}$ and have the most extreme coordinate in the direction $\vec{d}$. First, the most extreme boundary surface is found using the following relation:

$$\arg\min_i (\vec{c}_{env} - \vec{c}_i) \cdot \vec{d} \mid cos^{-1}(-\vec{d} \cdot \vec{n}_i) <= \theta_{threshold} \tag{16}$$

Once the most extreme boundary surface is found, other surfaces which meet the orientation criteria are compared, and merged if they meet the criteria presented in Section 4.3, even if the meshes are physically disconnected. This is useful for merging sections of the floor or ceiling that are otherwise disconnected, but still useful when considering the scene's context.

4.4.2 Surface Section Removal

The remaining filters are processed in tandem, as they all involve removing regions from a mesh. First, regions that do not meet specified criteria are identified. Then, all regions are simultaneously removed from the mesh. Each filter will be discussed, then the method with which to remove flagged regions will be demonstrated.

Minimum Recovery Height Filtering

The next step of filtering is to ensure that the drone would be able to successfully recover from a failed perching attempt at every location in the search space. For perching on horizontal surfaces, the drone stays level and maintains control authority throughout the maneuver. Perching on vertical and inclined surfaces requires the drone to orient itself so that its thrusters are not acting in the direction of gravity. As a result, the drone temporarily loses control authority in the vertical direction. Thus, if the drone's perching attempt fails, it will inevitably fall some distance before being able to correct its orientation and stop its descent.

For vertical surfaces to be considered for perching, enough space must be allocated for this corrective maneuver, as shown in Figure 21.

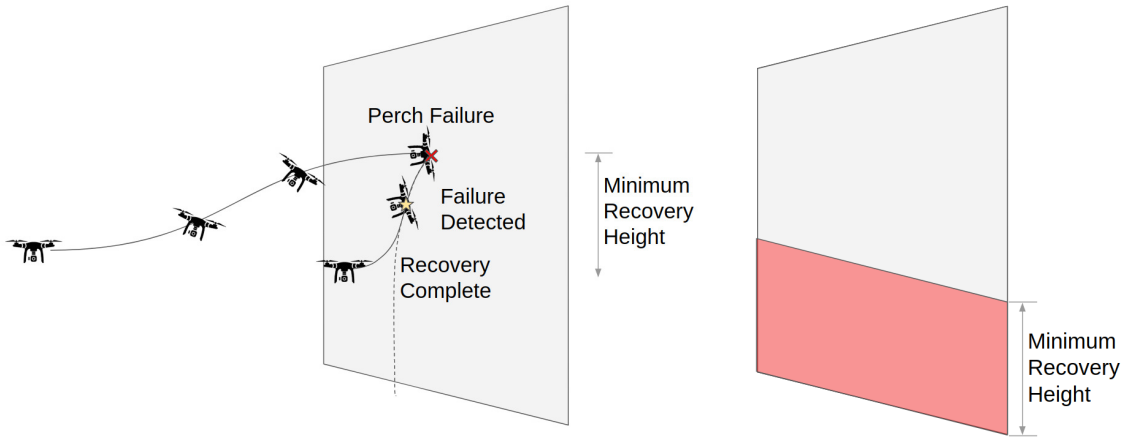


Figure 21: (left) Graphic showing the path and position of a drone recovering from a perch failure. (right) The minimum recovery height (red) transposed to correctly align with ground level. Perching on vertical surfaces is not always successful. In case of a failure, some vertical space is required to facilitate a successful recovery maneuver and avoid crashing into the ground. The minimum recovery height filter, removes surfaces which do not meet this threshold. In the case above, the red section would be removed from the search space.

This is measured as the distance between the floor boundary surface centroid, $\vec{c_f}$, identified in the previous section, and each point in the surface mesh, $\vec{p_{i,j}}$. In other words, $|(\vec{p_{i,j}} - \vec{c_f}) \cdot \vec{k}| >= h_{min}$, where h_{min} is the minimum recovery height. This filter is applied by simply removing faces from a surface mesh if all 3 vertices are lower than the threshold height. If only 1 or 2 vertices are lower than the threshold, they are simply moved upwards to the threshold height, and the face is left in the mesh. The area of the face is computed after this step. If the triangle is degenerate, or has zero-area, it is removed.

In practice, h_{min} depends on the perching hardware and how tight the control response of the drone is. In this work, 0.8m will be used as the threshold, based on the results demonstrated in [127].

Proximity Filtering

Of surfaces which have survived previous filters, the next step is to remove regions which do not meet a proximity threshold to nearby obstacles and obstructions. To do this, a neighborhood search is performed for each remaining surface in the search set. At each point in the surface, a nearest-neighbors radius search is performed. All faces within the nearest neighbors search are then projected onto the surface along the surface's normal direction,

and removed from the surface, as demonstrated in Figure 22.

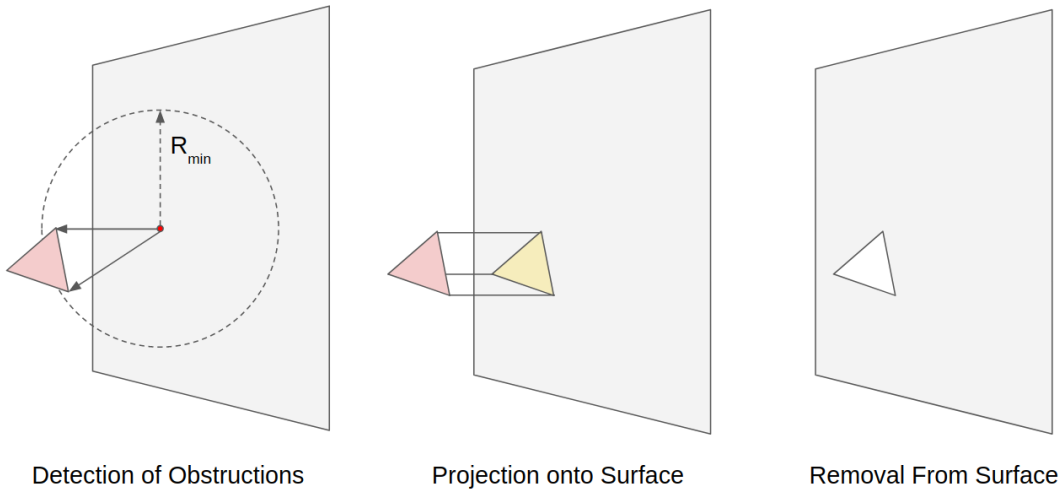


Figure 22: Graphic depicting the projection of detected obstacles. At each point in a mesh, a neighborhood radius search is performed to find any obstacles which are at least partially in the search sphere (left). Obstacles are the projected onto the surface along the surface normal (middle), and the region is removed from the search space (right).

Proximity search is a computationally expensive task, but it can be made more efficient by taking advantage of geometrically partitioned data structures. For a nearest-neighbor search, a KD-Tree is an effective choice [133]. A KD-Tree is a K-dimensional tree-like data structure which preserves the spatial relations between its nodes. The KD-Tree is formed by iteratively partitioning the data set using (K-1)-dimensional hyperplanes (as demonstrated in Figure 23). Each hyperplane is selected such that it passes through the median point of the subset in the axis normal to the plane, thus resulting in two equal half-spaces. The normal direction of the partitioning planes is selected in a cyclic fashion. For example, in 3D, the first plane placed would be aligned with the x-axis, then y, then z. The partitioning process continues until each partition only contains a single data point. This method results in a balanced k-d tree, which has maintained the relative spatial context of the points it is storing.

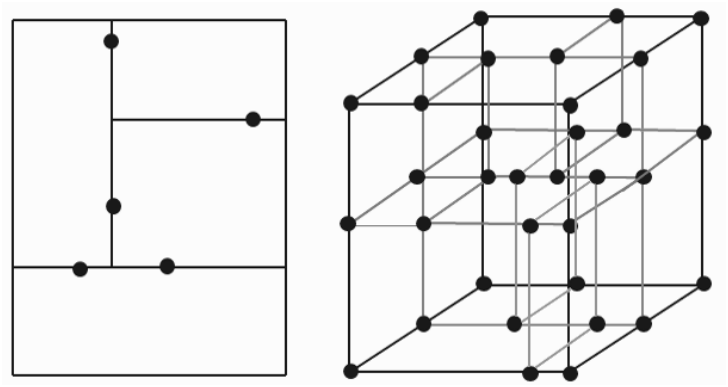


Figure 23: 2D and 3D examples of KD-Trees. Figure courtesy of [134]

Constructing a KD-Tree from unsorted data is relatively slow, with a time complexity of $O(nlog(n))$ [135], this is a one-time operation that occurs prior to the search process. This is a modest ask given the highly reduced form of the data. All subsequent searches within the tree occur in $O(log(n))$ time on average [133]. Thus, when evaluating proximity or obstruction for all remaining mesh preprocessing steps and placement evaluation steps, the processing is much more efficient.

Line of Sight Filtering

Line of sight filtering involves removing regions from the search space if they do not have a line of sight to the specified target region. Starting from the center of the target volume, regions are removed if they are invisible to the target. This process is demonstrated in Figure 24. In particularly cluttered environments, this can quickly narrow the search space to regions of interest which at least a minimum level of visibility of the target. As it would be extremely computationally expensive to check visibility for every point in a target volume, this filter is simplified to only consider visibility of the target centroid. In this way, bad solutions are quickly filtered out, and the optimizer can search within the remainder.

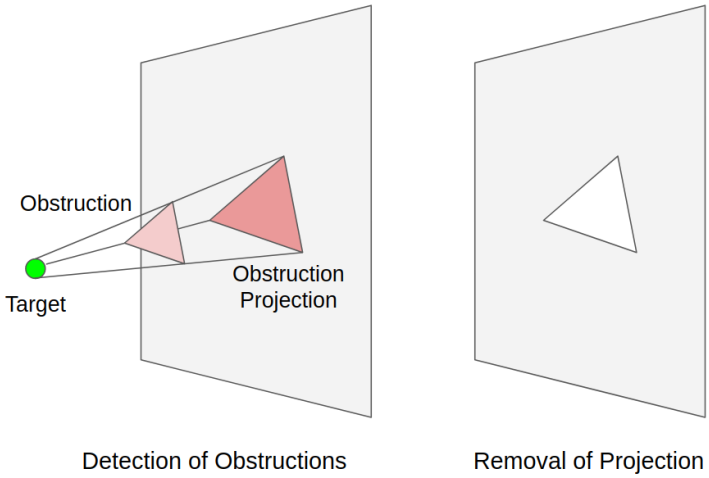


Figure 24: Projection filter demonstration. (left) Obstructions (light red) are found between each point on a surface and the target centroid (green). Obstructions are then projected along the ray extending between each obstruction vertex and the target centroid, onto the surface (dark red). (right) The projection is then removed from the search space.

Line of sight filtering shares many similarities with proximity filtering. The key difference is that both the obstruction search and the surface projection are performed radially, from the target centroid. As a result, rather than using a KD-Tree, an OBB-Tree is a more suitable choice. An OBB-Tree, or Oriented Bounding Box tree, is a common data structure used in computer graphics and game design when detecting collision between objects and performing ray tracing [136]. Rather than storing point data in nested space partitions, mesh data is stored in a recursive set of increasingly fine oriented bounding boxes. For reference, Figure 25 demonstrates this process. This reduces potentially complex meshes into manageable sizes (only 8 points per box), and allows most of the environment to be quickly ruled out, prior to searching individual faces for intersections. This structure is built in $O(nlog(n))$, with collision detection running in $O(log(n))$ on average [136].

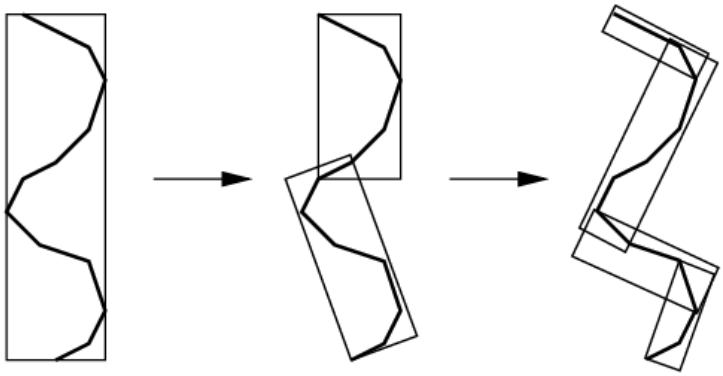


Figure 25: A simple 2D example of an OBB tree being recursively constructed, courtesy of [137]. Each subsequent level of the tree encapsulates a smaller volume and results in less error with respect to the original geometry.

Minimum Perch Window Filtering

Perch window filtering is the last operation that is applied to the set of surfaces prior to being passed to the camera placement optimization node. Here, perching window is used to describe a bounding box for the final position of any part of the drone that is perching on a surface. This box can be thought of as the Minkowski sum between a region denoting the confidence interval of the final drone centroid position and the footprint of the drone itself, as demonstrated in Figure 26.

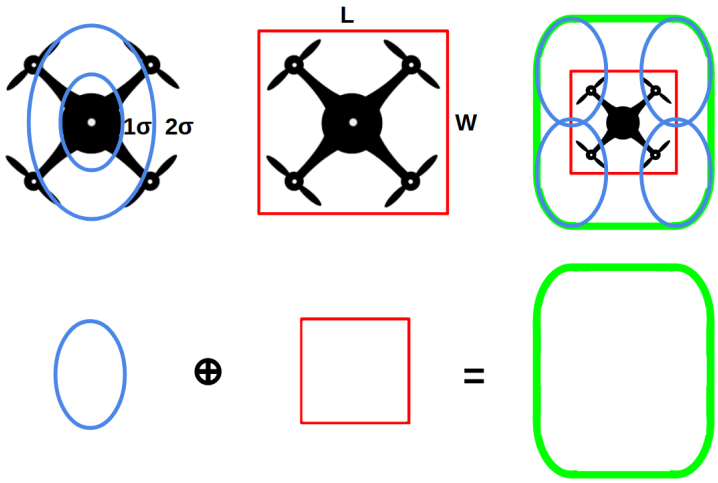
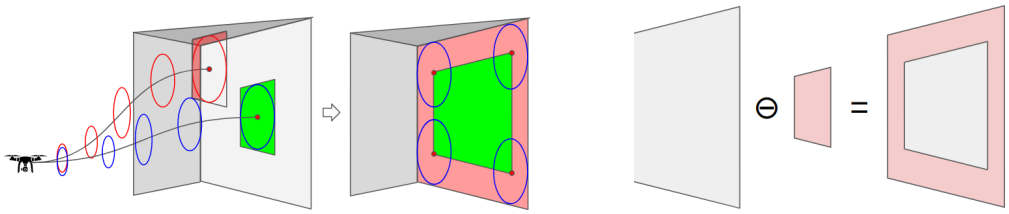


Figure 26: Graphical representation of determining the minimum perch window. The minimum perch window can be represented as the Minkowski sum of the position confidence interval (blue ellipse) and the drone's bounding box (red). The resultant region (green) must be free in order to be confident that the drone will not encounter obstructions when perching.

To guarantee, within some confidence interval, that a surface will have enough space to

facilitate a perching attempt, ideally one would take the Minkowski difference between the surface and the perching window. Figure 27 shows how certain regions would be unsafe for perching due to their proximity to other surfaces or boundaries conditions.



(a) Perching window requirement **(b)** Minkowski difference representation

Figure 27: Demonstration of the need for perch window filtering, and a way of representing this filter as a Minkowski difference.

There is, to the author's knowledge, no algorithm that computes a Minkowski difference on arbitrary, non-convex, meshes which contain holes. The approximate method employed in this work will be discussed next.

Mesh Region Removal

In the discrete space, however, the Minkowski difference operator is easily approximated. In image processing, Erosion is a common morphological operation, which is closely tied to the Minkowski difference. Erosion eliminates all regions within an image which will not contain the selected structuring element. Erosion is quickly computed by convolving a structuring element, or kernel, over a binary representation of the image. In this work, the regions of interest are planar, so 3D meshes can be projected onto their best-fit planes without introducing too much error.

Projecting a 3D point, $\vec{X}$, into coordinates on a 2D plane, $\vec{x}$, is trivial, as PCA has already been performed on the cluster. The 2D planar representation is found as follows:

$$\vec{x} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} (\vec{X} - \vec{\bar{X}}) \tag{17}$$

where: v_1 and v_2 are the two principal eigenvectors of the cluster, and $\vec{\bar{X}}$ is the cluster mean position.

Once in this form, the meshes (now 2D polygons) can be rasterized (i.e. converted into pixelated images) by filling regions of an empty image with the mesh regions. To do this, first a pixel density factor, ρ_{px} is selected, determining the number of pixels assigned per unit length. The image coordinates, u and v, of each point can be found as follows:

$$\vec{c} = \begin{bmatrix} u \\ v \end{bmatrix} = \rho_{px}\vec{x} + \bar{c} \tag{18}$$

where $\bar{c}$ is the center coordinate of the image, in pixels.

The image size is selected to match the bounding box size of the projected surface, and is simply the length and width of the bounding box, multiplied by ρ_{px}. Polygons can be represented with higher fidelity by increasing the number of pixels assigned per unit length, though making this value too high increases the computation time of image operations. In practice, $\rho_{px} = 50\frac{px}{m}$ (or 2cm per pixel) provides a nice balance of accuracy and speed. Additionally, the features of interest for perching are generally quite large. Inaccuracies for small features has low expected impact on system performance.

Once the mesh has been rasterized, the erosion operation can be applied to eliminate regions which do not meet the minimum perching window requirement. The structuring element for the erosion operation is generated by multiplying the dimensions of the minimum window by ρ_{px}, and generating either a rectangular or elliptic image. Regions identified in previous filtering steps can also be removed here, at low computational cost.

After the image operations is applied, the image can be converted back into a polygon, by using a contour extraction algorithm. This work uses the OpenCV [88] implementation of Teh and Chin's k-cosine curve detection algorithm [138]. Figure 28 demonstrates this procedure from start to finish. If desired, the points in this new mesh could be re-projected into 3D, by projecting to the nearest face from the original mesh. However, the planar representation proves convenient for using continuous search methods (this point will be discussed further in Section 4.5.3). Thus, the re-projection step is skipped.

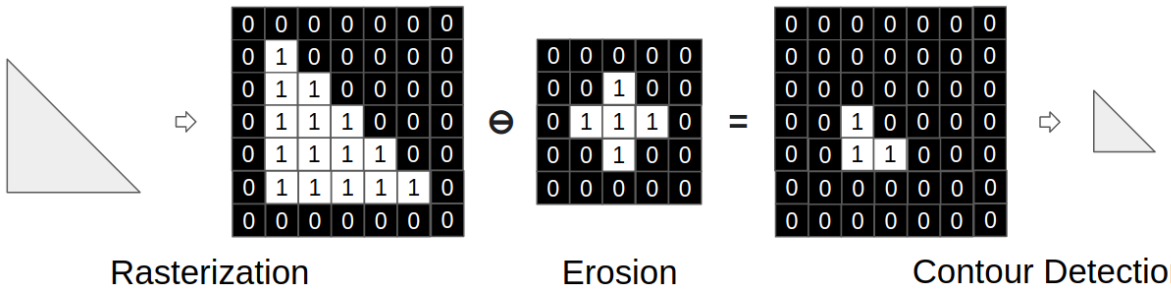


Figure 28: Highly simplified representation of polygon erosion. Polygons can be rasterized, by converting their planar projection coordinates into pixel coordinates. The image can then be processed to remove regions directly, or apply an erosion operation to remove a perching window. In this case, a cross shaped structuring element is shown (middle), but rectangular or elliptic structuring elements are also possible. Finally, the image is converted back into a polygon.

After all filters have been applied, the reduced set of perching surfaces is passed to the *Camera Placement Optimization* node, described in the next section.

4.5 Camera Placement Optimization

Given a mesh environment, the camera placement node automatically finds a locally optimal camera arrangement that satisfies camera and perching hardware limitations. There are three key aspects to this node. First is the camera simulator, used to evaluate the quality of proposed arrangements. Second is the optimization search algorithm. Finally is the camera placement fitness function used to assess the arrangements.

4.5.1 Camera Simulation

A camera simulator was developed in Python, to allow for the rapid assessment of new camera arrangements with different parameters, environments, and search methods. Mesh environments can be loaded into the simulator. Camera coverage and tracking uncertainty can then be assessed at arbitrary camera poses.

Configuration files are used to specify the parameters of each camera. Parameters include resolution, field of view, range, gimbal motion limits (if applicable), camera intrinsic, extrinsic, and distortion parameters. Cameras are then simulated using the Brown-Conrady distortion model [86] (see Section 2.2.3 for details), when assessing visibility of points.

Camera range is further modified from the manufacturer specifications, prior to simulation. As the goal of the camera arrangement is to detect and track potentially small objects throughout a scene, it is necessary that a minimum resolution is enforced. For instance, if a particular scenario involves grasping the end of a small pipe, to pull it out of a debris pile, a feature detector must be able to detect the pipe in the image frame. This imposes a limit on the minimum size of the target object in the image frame, feature detectors require at least a certain feature area. The number of pixels associated with a given area decreases according to the inverse of the square of the distance to the camera, as shown in Figure 29. Thus, given the minimum size of the tracked object, L_{min}, and the minimum number of pixels that would still result in a positive detection, p_{min}, the camera's maximum observation range (R_{max}) can be set. While the actual number of pixels required will depend on the tracking algorithm, a reasonable starting point used in this work is $p_{min} = 10px$ to give a 10x10 pixel minimum area. Through discussion with project collaborators, it was decided that the minimum feature to be grasped would have a length of 4cm, the approximate diameter of a golf ball.

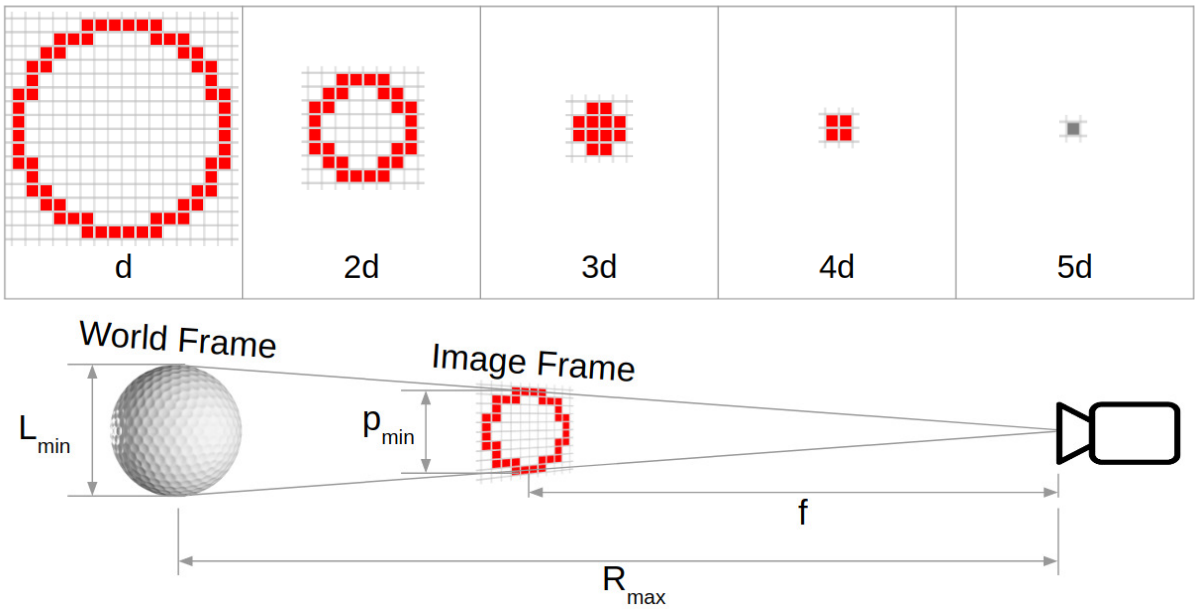


Figure 29: (top) Demonstration of the loss of resolution with range while tracking a round feature at distance, d. (bottom) Demonstration of the camera range limitation as a function of minimum image feature size using similar triangles between the image and world frame.

By considering similar triangles, maximum working range can be expressed as:

$$R_{max} = \frac{f L_{min}}{p_{min}} \tag{19}$$

where f is the focal length of the camera, in pixels.

For the values suggested above of $p_{min} = 10px$ and $L_{min} = 4cm$, and the focal length of the D435 camera ($f = 638px$) this works out to a maximum working range of 3.2 m, compared to the manufacturer's specified limit of 10 m.

The Intel D435 was made available for physical testing and is therefore used to set relevant parameters for the simulation testing as well. It should be noted, however, that the simulation is largely camera independent. The same principles apply to LIDAR and monocular cameras as well, though the depth sensing error formulation would change. Some cameras which use active projection to sense depth information, such as the Microsoft Kinect, are prone to inter-camera interference which can result in loss of depth data and increased depth uncertainty. The D435 is not impacted by inter-camera interference, thus it was not modeled.

4.5.2 Fitness Function

Camera placement optimality is impacted by the following:

- camera position feasibility,
- scene coverage,
- pixel information density,
- tracking uncertainty, and
- robustness to occlusion.

Camera position feasibility refers to the feasibility that a camera drone can be placed in the target location. All suggested positions must be feasible. Scene coverage refers to the proportion of a target scene that can be captured by the camera arrangement. The ideal arrangement has 100% coverage of the scene. Pixel information density is defined as the area in the world frame that is associated with a single pixel. To capture the most detail in a given area, perhaps counter-intuitively, the pixel information density should be minimized, equivalent to zooming in to the details of the target. Tracking uncertainty, discussed in Section 2.2.3, refers to the level of uncertainty, due to depth error and matching error, associated with tracking a point in 3D space. Robustness to occlusion refers to the ability of the system to continuously track objects in dynamic scenes, as obstructions move and potentially occlude points of interest.

In this project, the list of metrics can be reduced considerably. The previous nodes, presented in Section 4.4, filter out the majority of infeasible positions. Any infeasible positions that survive filtering can still be removed by user oversight (see Section 4.6 for more details). Thus, there is no need to include a feasibility metric in the fitness function. Additionally, in the case of FlyCroTug manipulation, it can be assumed that object motion in the scene will be quasi-static. As such, dynamic occlusion is not anticipated to play a significant role in determining the tracking performance of the cameras. Finally, information density and tracking uncertainty can be bundled into a single metric. Both information density and tracking uncertainty are related to the target distance. Thus, a system that minimizes tracking uncertainty will also minimize inherently minimize pixel information density.

The proposed fitness function thus considers only coverage and tracking uncertainty. Rather than attempt to integrate non-linear functions over complex intersecting regions representing each camera's field of view, calculations are simplified by discretizing the target space into N uniformly sampled points. This simplicity is key to ensuring that a large number of potential arrangements can be evaluated as quickly as possible.

At each n of the N points, coverage, and visibility can be easily evaluated, as demonstrated by Figure 30. Proximity is evaluated on a per-point basis by taking the euclidean distance

between the camera's optical center and the point. Coverage is by performing ray-tracing intersection checks using OBB-Tree search, as discussed in Section 4.4.2. Tracking uncertainty is evaluated according to Equations 20-24.

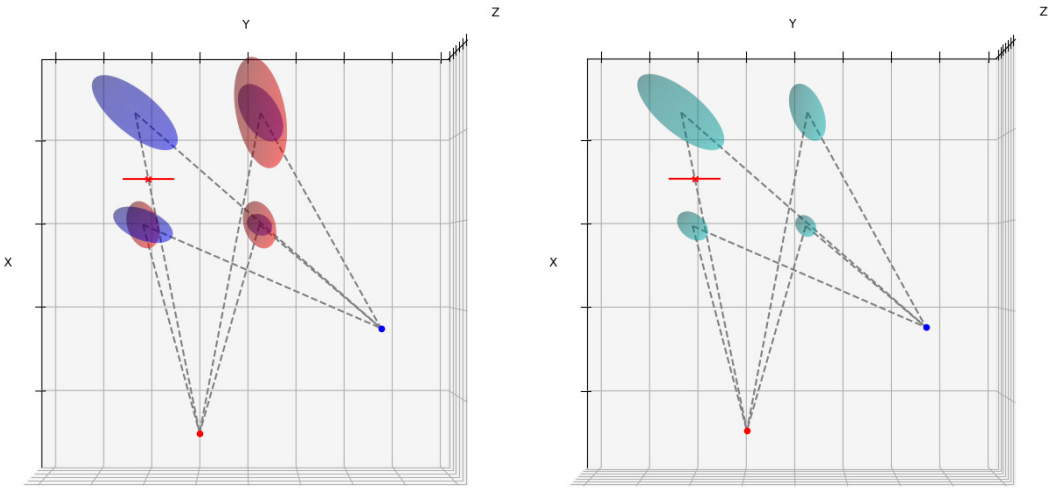


Figure 30: Top views of point-wise uncertainty ellipsoids for a two camera arrangement. An obstacle (red line) blocks the view of the red camera from the top, left point. The red and blue points indicate the camera locations. (left) The red and blue ellipsoids indicate the tracking uncertainty of each camera. Note that the axial tracking uncertainty grows with distance at a faster rate than the tangential uncertainty, resulting in more a oblong ellipsoid. (right) The light blue ellipsoids represent the weighted-least-squares combination of the uncertainty from each camera. The displayed covariance ellipsoids are exaggerated for clarity.

Each camera, c, is assessed at each sampling point, n, in the target volume as follows:

$$C_{c,n} = \begin{cases} R \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_x^2 & 0 \\ 0 & 0 & \sigma_x^2 \end{bmatrix} R^T & if\ visible \\ NaN & else \end{cases} \tag{20}$$

In Figure 30, $C_{c,i}$ would be one of the red or blue ellipsoids.

The point-wise uncertainty is assessed using a weighted least squares combination of the non-NaN camera uncertainties.

$$C_n = (\sum_{c=1}^{C} C_{c,n}^{-1})^{-1} \tag{21}$$

In Figure 30, C_i would be one of the cyan ellipsoids.

The fitness of a given camera arrangement is thus computed as follows:

$$f = \frac{1}{N_p} \sum_{n=1}^{N_p} (1 - \frac{|C_n|}{|C_{max}|}) \tag{22}$$

where: f is the fitness of a given camera arrangement
N_p is the number of sampling points over a target volume
$|C_n|$ is the matrix norm of the tracking covariance matrix for point n
$|C_{max}|$ is the matrix norm of the worst-case tracking covariance matrix (i.e. when the point is viewed by a single camera at its maximum working range).

4.5.3 Optimization Search

Now that a method to quickly evaluate potential camera arrangements has been developed, the next step is to define a way to navigate the search space to effectively optimize this heuristic. Several solutions have been proposed using metaheuristic search techniques including particle swarm optimization [79], genetic algorithms [71], and simulated annealing [75], as discussed in Section 2.2.1, but opted limited their search efforts to a discrete grid of points around the environment. This work investigates the use of Particle Swarm Optimization (PSO) with a hybrid (continuous and discrete) search space, to provide rapid convergence on high quality, initial solutions, and to exploit fitness gradients between neighboring particles to fine tune later solutions.

Particle Swarm Optimization (PSO)

PSO is a metaheuristic search algorithm inspired by social flocking and swarm behaviours observed in biology [139]. Particles represent points moving through the solution space. Much like in biological swarms [140], the motions of particles and the aggregate motion of the swarm are interconnected, and result in interesting emergent behaviours. In PSO, particles are moved through space according to a velocity update function that attracts the particle to a weighted combination of their own personal (local) best solution, and the swarm or neighborhood best solution. The weights of each velocity update are randomized, to encourage exploration. By moving particles through space according to local and global minima, particles can converge on optimal solutions, without requiring explicit computation of the fitness gradient.

Algorithm 2 briefly describes the pso algorithm.

Note that the personal and global best updates shown in Algorithm 2 are treated as a maximization of the fitness function, to remain consistent with the results presented later

Algorithm 2: Particle Swarm Optimization Algorithm

```
% Initialize Swarm
for each particle, p, in swarm, S do
    Initialize particle position, x_p, randomly in the N-dimensional search space;
    Initialize particle local best positions, b_p;
    Initialize swarm best position, g ⟵ max(b);
    Initialize particle velocity, v_p, randomly in the N-dimensional space;
end
% Run Optimization
while searching do
    for each particle, p, in swarm, S do
        Generate random numbers, r_l, r_g, in the N-dimensional space;
        Update particle velocity according to: v_p = ω v_p + c_p r_l (b_p − x_p) + c_g r_g (g − x_p);
        % where:
        % ω = particle inertia
        % c_p = individual weight
        % c_g = social weight
        Update particle position according to: x_p = x_p + v_p ;
        if f(x_p) > f(b_p) then
            Update the the local best, b_p ⟵ x_p if f(x_p) > f(g) then
                Update the the global best, b_p ⟵ x_p
    end
end
```

in this work. Many works opt instead to minimize a cost function, and will have slightly different notation. Algorithm 2 represents a global-best PSO algorithm. Global-best search can be prone to converging on local minima without adequately searching the space [141]. An alternative is to use a Local-best search strategy. Rather than being attracted to the global optimum, particles are assigned to a neighborhood and are attracted towards the neighborhood's best. Topologies can be predefined and static, such as using a ring topology to assign each particle two neighbors, or dynamic, such as using the particle's nearest neighbors in the search space [141].

PSO has several hyper-parameters which can impact convergence speed, and balance between exploration and exploitation. The hyperparameters are described briefly below, to understand the impact that they have.

Hyperparameters include:

- Social weight (c_g) defines the relative importance of the neighborhood best solution
- Local weight (also referred to as cognitive weight) (c_l) defines the relative importance of a particle's personal best solution
- Particle inertia (ω) sets a decay rate for the particle's velocity
- Swarm size, S, determines the number of particles in a swarm.
- Neighborhood size (k) determines how many neighbors are considered in the neighborhood best fitness (in local-best search only)
- Number of Iterations (N_{its}) sets how many generations will be tested prior to returning the local best.

The camera placement environment will likely feature a highly discontinuous fitness landscape due the disconnected nature of the search surfaces and scene obstructions, and as a result, a large number of local maxima are anticipated. For this reason, local-best search is implemented. Particle neighborhoods are defined using nearest neighbor grouping with a Euclidean distance function over the search space. To speed up development time of the pipeline, the `pyswarms` library [142] was used to provide the back-end for particle swarm optimizations. This helped shift the focus of this work away from the specific implementation details of PSO, and towards the more novel challenges associated with creating a perching drone camera network.

The hyperparameters included in Table 1 indicate the default values used during testing. There is no universally accepted metric for setting the optimizations hyper parameters

[143], and parameter tuning for PSO remains an open research question [144–146]. To not distract from the main goals of this work, no formal hyperparameter tuning was performed. Parameters used are were recommended by [142], and were deemed acceptable in early tests.

Table 1: Default hyperparameters used during testing of PSO camera placement optimization

Hyperparameter	Value
c_g	0.4
c_l	0.5
ω	0.7
S	50
k	5
N_{its}	40

Given the fitness function and the optimization method, the final requirement is to develop a strategy to encode camera state in a compact form which can be used by the optimizer.

State Encoding While previous works used discrete methods to solve the camera placement problem[71, 75] to solve discrete versions of the 3D camera placement problem, this work investigates the the use of hybrid continuous-discrete state vector. This section will discuss briefly each search dimension and its encoding. The perching drone will be assumed to have a motorized gimbal capable of altering camera pan and tilt, and a motorized lens, capable of altering zoom. These individual options can be disabled in the camera configuration file, mentioned in Section 4.5.1.

Under these conditions, a naive approach would be to assign each possible particle 9 dimensions. Six dimensions would fully define the position of the drone in translation and rotation, and the remaining 3 dimensions would define the pan, tilt, and zoom parameters. However, this would be unnecessarily complex and would ignore the constraints associated with perching on a planar surface. In fact, to fully define an object that is in contact with a plane, only 3 dimensions are required; 2 dimensions define the position along the surface, and the third defines rotation about the yaw axis.

A final reduction is possible with the realization that rotation on the surface is unnecessary information. Two cases are relevant here. If the drone is perching on a vertical surface, it can be assumed that the angle will be nearly-constant. Holding yaw constant during a perching maneuver simplifies the motion planning algorithm, and allows the preferred direction of a potentially an-isotropic dry adhesive gripper to be loaded with gravity. Thus, on vertical

surfaces, the drone is assumed to perch such that its local forward direction is pointing downwards, along the gravity vector. On horizontal surfaces including floors and ceilings, the approach maneuver allows the drone to maintain full controllability over yaw. Surfaces are not anticipated to have a preferred alignment, thus, the drone is assumed to orient itself such that its forward direction points towards the target centroid. Thus, only 5 dimensions are required to fully define a drone perching on a planar surface, as demonstrated in Figure 31.

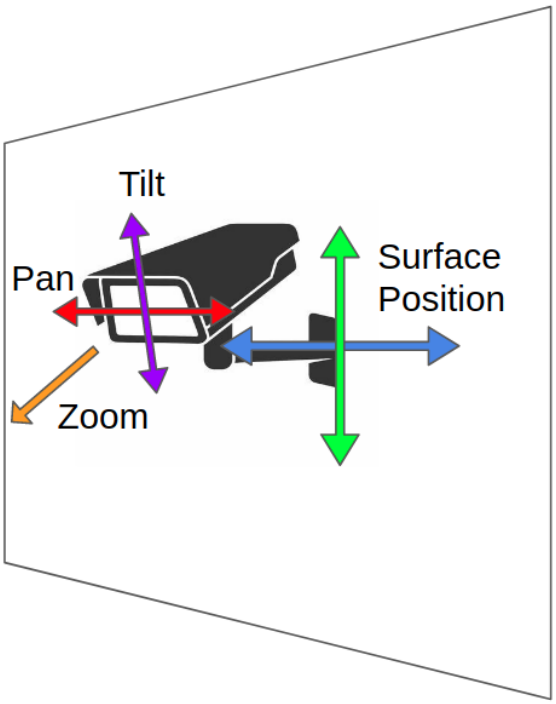


Figure 31: The five different colored arrows here show the minimum 5 degrees of freedom required to fully define a pan-tilt-zoom camera placed on a 2D surface.

But what about a collection of planar surfaces? One idea is to map each position using spherical coordinates, centered about the target center. Thus, each position within line of sight of the target center can be defined uniquely by its relative azimuth and elevation angle. This idea goes hand in hand with the line of sight filtering discussed in Section 4.4.2. By encoding in this way each visibility of the target center is guaranteed and spatial continuity is maintained.

This approach was rejected due to two limitations. The first is that this the density of perching locations in this spherical space is coupled with its radial distance from the target. While this could be seen as beneficial, as closer surfaces will make up a larger proportion of the search space, there are many instances where distance from the target center is not inherently a limitation. There are also cases where surfaces may be too close for a camera

to effectively capture the target volume; in these cases, it would make no sense to prioritize nearby surfaces. The second reason is that the set of perching surfaces in the perching space is highly discontinuous. Using spherical coordinates would result in large dead-zones with zero fitness, making it more difficult for particles to navigate the space. Instead, one additional parameter is proposed as a discrete surface selector, with a value of [0, 1). This range is partitioned into S ranges, where S is the number of surfaces in the search space. The partition size for each surface is weighted according to its net surface area. Once a surface is defined, two approaches are explored to define position.

The first approach to define surface position is to use a discrete face selector, much in the same way as the surface selector. The value is partitioned into F ranges, where F is the number of faces in the surface's mesh representation. Given a selected face, the drone's position can be set to be the face's center position. This method will be referred to as *Mesh Encoding*. This discrete representation provides a dimensionality reduction, which may increase search speed, but also limits the search space and potentially impacts the achievable solution quality. This method also has the consequence of removing the spatial context between points, thus potentially making it more difficult for particles to navigate the space and converge on locally promising areas.

An alternative approach, dubbed *Plane Encoding*, takes advantage of the planar representation of each surface. In this method, the particle position is defined on the surface using the first two principal axes, v_1 and v_2, from the PCA decomposition. These values are then bounded over the limits of points in the mesh within this projection space denoted $[v_{1,min}, v_{1,max}]$ and $[v_{2,min}, v_{2,max}]$ respectively. This renders the spatial search dimension in a continuous fashion and preserves the relative spatial context of positions, while adding one dimension. For highly non-convex surfaces, this representation can also lead to dead-zones in the search space.

Pan, tilt and zoom are easily represented by continuous variables. The bounds for each of these parameters are set according to the available ranges defined in the camera's configuration file.

Table 2 summarizes the state encoding methods implemented. Note that all particle components are normalized to a range of [0, 1) prior to optimization. This ensures the particle distance, used in the nearest neighbors search is consistent.

Table 2: Summary of the two possible state encoding methods developed

Parameter		**Method**	
		Mesh Encoding	**Plane Encoding**
Drone Pose	Surface Selection	$0 \leq s < S$	
	Position Selection	Face Selector, $0 \leq f < F$	Principal axis 1, $v_{1,min} \leq v_1 \leq v_{1,max}$
			Principal axis 2, $v_{2,min} \leq v_2 \leq v_{2,max}$
Camera State	Pan	$-\frac{1}{2}\theta_{g,p} < \theta_p < \frac{1}{2}\theta_{g,p}$	
	Tilt	$-\frac{1}{2}\theta_{g,t} < \theta_t < \frac{1}{2}\theta_{g,t}$	
	Zoom	$1 \leq z < z_{max}$	
State Representation		$p = [s, f, \theta_p, \theta_t, z]$	$p = [s, v_1, v_2, \theta_p, \theta_t, z]$
Particle Dimension, D		5	6

where: $\theta_{g,p}$ and $\theta_{g,t}$ are the pan and tilt ranges of the camera gimbal.

Search Strategies As discussed in Section 2.2.1, camera placement can be optimized using greedy search, as the coverage and tracking uncertainty metrics used to define optimality are sub-modular functions. In greedy search, cameras can be placed one-by-one in the optimal location, reducing the dimensionality of the search considerably.

Another search strategy implemented was to take advantage of a-priori information to inform the pan and tilt placements. Assume for a moment that the perching drone has a gimbal which is able to point along any angle in spherical coordinates. When perched on a flat surface, one hemisphere of the search space can immediately be ruled out, as there is no benefit to pointing the camera towards the drone itself and the wall. For a camera with a non-zero field of view, a further restriction can be made by limiting the pointing angle of the camera such that the vertical edge of the camera frustum is parallel to the surface. A final suggested restriction is to limit the camera pose to a small range of angles around the vector which points directly to the target center. While some variation in pan and tilt will likely be required to center the camera and ensure full coverage of larger scenes, having knowledge of the target volume center provides a very useful starting point for the search. Figure 32 demonstrates each of these search space restrictions. In practice, the intersection of set of achievable gimbal angles, G, and the set of angles about the target direction T is used. In the case that the sets do not intersect, the gimbal angle closest to the desired target angle is returned. This technique is referred to as Gimbal Limited Targeting. This targeting mode is used in tests.

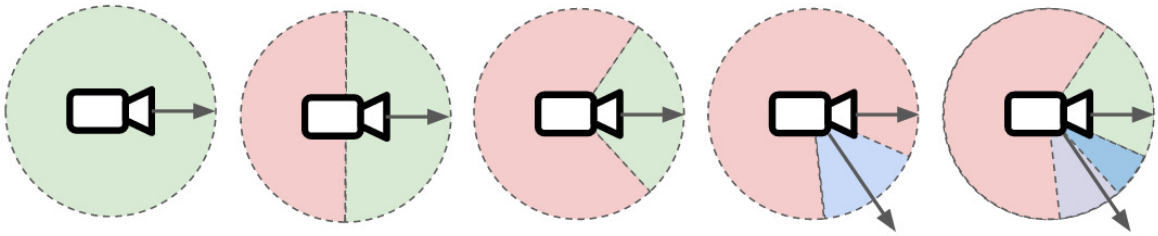


Figure 32: (from left to right) Unrestricted search space, in green. In this mode, particles can represent all angles. Next is the wall restricted search space, where the vertical line represents the perching surface and the red region represents angles that are removed from the search space. Third is the the gimbal limited search space, where the green wedge indicates the range of angles achievable by the gimbal. In target limited search (fourth), the set of angles is defined around some deviation from the target direction. Angles are selected from a set about this angle. Finally, in gimbal limited targeting, the search space is set to be the intersection of the gimbal limited and target limited sets, shown in dark blue. Note that the set does not necessarily include the target direction.

4.6 User Interface

One goal of the FlyCroTug project was to investigate human-robot collaboration in disaster relief scenarios. Considering the perching camera problem, leveraging human intuition can be beneficial in several ways. First and foremost, in a disaster response scenario, the intuition of the first responders is invaluable. While heuristics could be used to predict where regions of interest *may* be, asking the operator directly removes all uncertainty. Additionally, while computer vision and computational geometry can automate feature extraction and target identification, these algorithms are not perfect and can yield undesirable results when presented with noisy, real-world data. Keeping a human in the loop can help the system quickly identify and correct faulty outputs. This is especially important considering the risk of catastrophic failure if a perching target was set to an unreachable position. It is important, however, to not overburden the operator. As such, operator decision-making is limited as much as possible.

When human input is required, the prompts include a simple Graphical User Interface (GUI) with 3D visualizations to provide situational context efficiently. The visualizations can be manipulated using standard mouse controls to change the view perspective and zoom in on Region of Interest (ROI)s. Rendering and user interaction are handled using Musy et al.'s `vedo` library [147]. User action is limited to simple rectangular selections and boolean "yes/no" responses. Figure 16 shows that user input can be requested at 3 points in the system pipeline. Each instance will be discussed in detail below.

4.6.1 User Mesh Selections (Optional)

The first instance is in the *User Mesh Selections* node. In this node, the user is prompted with a 3D rendering of the dense environment reconstruction. The user is given the option to perform two basic operations: mesh cropping and target identification (demonstrated in Figure 33). Cropping simply refers to the removal of mesh sections which are irrelevant to camera placement, such as hallways or incomplete geometry. The user can simply click and drag to select a rectangular region of interest. Multiple selections can be combined if desired. Cropping is used to limit the geometry that is passes along to the *Plane Segmentation* node, limiting the perch location search space. The original mesh is stored and can be used as necessary in later planning algorithms. In addition to cropping, the user is able specify a target volume. The target volume is used by the *Camera Placement Optimization* node to specify where camera coverage is desired. If no target volume is specified, the entire, cropped environment volume is used. This node is completely optional, and serves to focus the search procedure when environmental context is available. This can be particularly advantageous in large, complex environments where only a subsection is relevant for subsequent manipulation tasks.

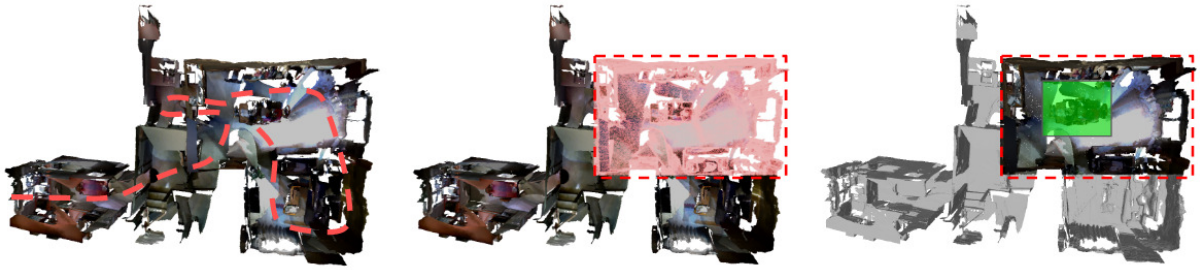


Figure 33: Demonstration of a case where cropping the reconstruction can benefit the reconstruction accuracy in the region of interest. (left) path visualization of an exploratory robot entering a complex building environment. (middle) user selected ROI narrows the search for perching and camera placement to the top-right room. (right) original environment model is preserved for future path planning tasks. ROI is used in the camera placement search. The user can further specify a region (green box) where camera coverage is desired. (building model from [43]).

4.6.2 User Filtering (Optional)

The next instance of user interaction involves surface filtering. The operator may have a better understanding of the capabilities of the perching mechanism, or contextual intuition which can help narrow the search. If the *User Filtering* node is enabled, the top N planar surfaces are displayed to the user. Because context within the environment is often needed

Figure 34: Demonstration of the *User Filtering* node GUI with a few characteristic cases. (left) A well segmented whiteboard surface which would be suitable for perching without alteration, (middle) a wall that is mostly suitable for perching but the posters in the bottom right may prevent perching and could be cropped out, (right) a surface which is large in area, but could a poor candidate for perching due to its irregular shape and narrow aspect ratio.

to decide, the surfaces are shown within the full mesh environment, but are highlighted so that they are easily identifiable. The user can input a simple yes/no command to determine whether the surface should be included in the search space. If the automatic segmentation is poor, the user also has the option to crop the surface to eliminate low-quality perches from the surface. Figure 34 shows this interface, along with instances where user filtering may be beneficial.

Note that, depending on the selected search hyper-parameters, it can be worth disabling this node. Pausing for user input is inherently slow, compared to a fully autonomous approach. Additionally, reducing the number of distinct tasks asked of a user reduces the overall burden on the user. Even if poor surfaces are included in the search space, it is not guaranteed that they will be included in the proposed solution. If invalid surfaces do end up being selected in a proposed solution, the consequence is relatively small. The user is still given the opportunity to review perching locations in the *Perch Confirmation* node (discussed in the following section). Rejected perch locations are removed from the search space, and a new solution is found. In many search configurations, the optimization converges on new solutions quickly, and poor surfaces can efficiently be removed on an "as-needed" basis.

4.6.3 Perch Confirmation

The final node that requires user input is the *Perch Confirmation* node. After the camera placement optimization runs its course and a drone pose is specified, the prospective arrangement is displayed to the user as both a 3D rendering of the camera field of view in the environment, and as a 2D rendering of the simulated camera feed. The former allows the

user to assess the perching location in terms of feasibility, and gives a clear idea of the scene coverage gained by the camera placement. The latter allows the user to assess the quality of the location by giving a simulated preview of what the camera will actually capture.

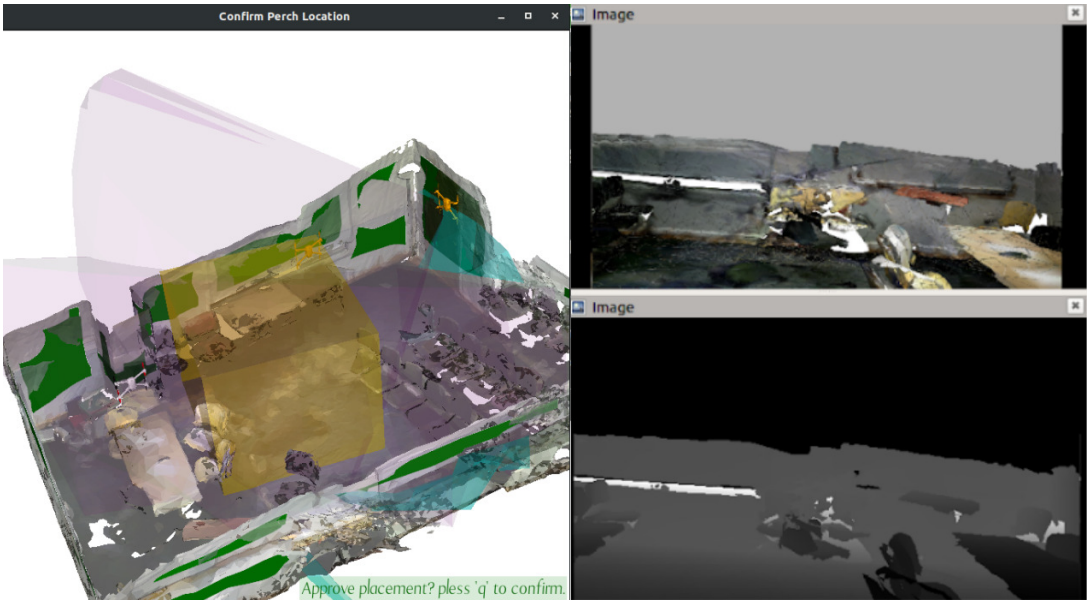


Figure 35: (left) Rendered 3D environment with simulated drone and camera frustum. Green shapes indicate the perch location search space. (right) Simulated camera view of the proposed placement shown on the left image.

After reviewing the proposed placement, the user is free to accept or reject the placement. If the placement is rejected, the location is removed from the search space and the optimization is re-run. If the placement is accepted, the location is published as a waypoint for the perching drones. Figure 35 shows an example of the user interface used for perch location confirmation.

4.7 ROS Integration and Gazebo Simulation

The proposed pipeline was developed to be compatible with future robotic systems. With this in mind, the pipeline was developed using the Robotic Operating System (ROS) [148] middleware library. The ROS Melodic implementation of the perch placement system mirrors the format shown in Figure 16, where each node can act as a service, being called with relevant input information, and subsequently publishing the output information.

Using the ROS framework also proved useful for simulating and displaying camera streams of suggested camera placements. The mesh environment and camera drones can be simulated in Gazebo [149], a powerful open source robotics simulation tool with ROS integrations.

When camera positions are published by the camera placement optimization node, simulated cameras can be moved within Gazebo to recreate the expected view, as shown in the right half of Figure 35. Simulating the view from a camera often offers the user more context than a 3rd person view of the environment and camera placement would. Gazebo is also able to render simulated depth maps and point clouds, which are published to easily accessed ROS topics. The simulated data can them be piped to other nodes, or visualized using the in-built RVIZ tool. This allows for rapid experimentation with different camera fusion techniques, and can facilitate testing and training on future object detection and grasp planning models. More detailed experimentation in the Gazebo simulated environment remains as future work.

4.8 Validation

This section will outline the test procedures used to validate the proposed perch placement pipeline. Results are included in Section 5

4.8.1 Hardware

All tests performed in this report are conducted using a laptop computer equipped with an Intel i5-9300H CPU, 8 GB DDR4 RAM, and a NVIDIA GeForce GTX 1650 GPU. Note that the GPU is only used to perform dense reconstruction, and is not presently used to accelerate any other nodes in the pipeline.

Scene reconstruction images are collected using an Intel RealSense D435 active stereo camera. The RealSense D435 specifications are included in Table 3 for reference.

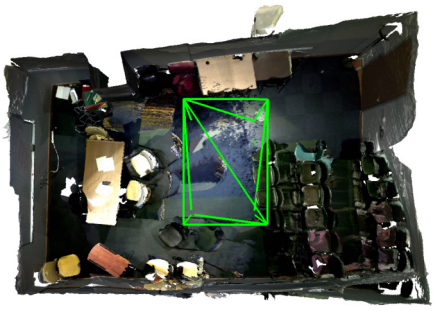
Table 3: Intel RealSense D435 Parameters

Camera Type		Active IR Stereo
RGB	FOV	69.4°(H) × 42.5°(V)
	Resolution	1920 × 1080 pixel
Depth	FOV	87°(H) × 58°(V)
	Resolution	1280 × 720 pixel
	Range	0.1 m - 10 m
Frame Rate		30 fps
Mass		72 g
Size		90 x 25 x 25 mm
Power Draw		3.5 W

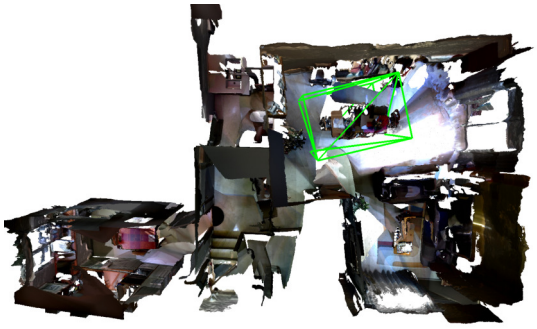
4.8.2 Data sets

The perch placement optimization pipeline requires an environment in which to operate. To avoid over-fitting or biasing the algorithm to specific scenes, or types of environment, results will be presented using reconstructions from open-source data sets. Additionally, to show the capability of the full pipeline from exploration to placement, a new indoor dataset was captured.

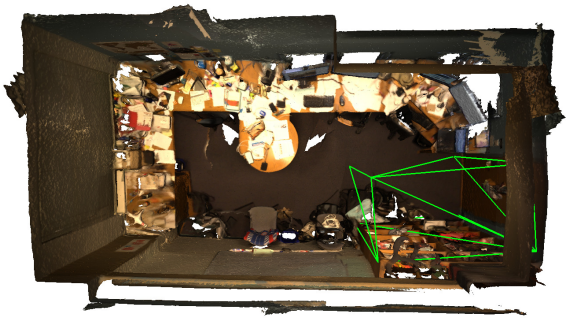
Figure 36 shows a preview of the different environments and the selected target volumes used for testing. Table 4 gives key details for each dataset.



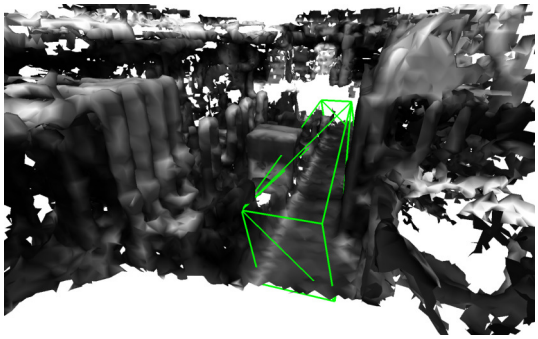
(a) MIT_32_D507



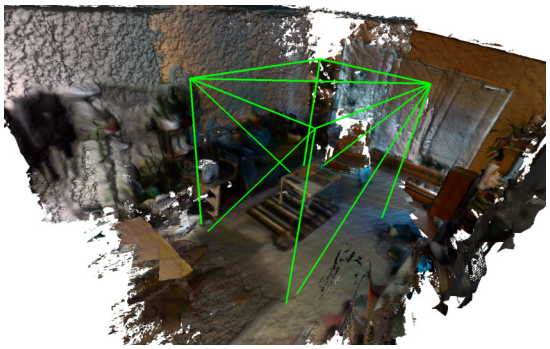
(b) Home_at



(c) Office3



(d) Machine Hall 01



(e) Apartment

Figure 36: Preview of mesh environments used in testing. Target volume edges are shown in green.

Table 4: Test Data Set Parameters

Environment	Data Set	Sensor	Reconstruction Method	Environment Size (m)	Target Location	Target Volume (m^3)
MIT_32_D507	SUN3D [150]	Asus XTion	BundleFusion [43]	7.5 x 4 x 2	Room Center	6.0
Home_at	SUN3D	Asus XTion	BundleFusion	12 x 6 x 2.3	Room Center	6.7
Office3	BundleFusion	Structure IO	BundleFusion	5 x 2.5 x 2.3	Doorway	3.3
MH01	EuRoC [151]	VI Sensor	MapLab [152]	11.5 x 10.5 x 5	Main Hallway	43.0
Apartment	This work	Intel D435	Open3D [128]	4.5 x 3.5 x 2.3	Room Center	5.7

These 5 environments each offer different levels of clutter, scene complexity, and reconstruction quality. The *MIT_32_D507* classroom reconstruction is a relatively open space, with little obstruction from the walls to the center of the room. The *Home_at* mesh is particularly non-convex, with connected rooms, doorways, and hallways. The *Office3* dataset has a fully reconstructed ceiling, allowing for the testing of ceiling perch selections. The *Machine Hall (MH)* dataset is included to challenge the system, as it is reconstructed from monochrome stereo images, is the largest environment, and the reconstruction quality is particularly low. Additionally, the sensor in this case is mounted aboard an aerial robot, resulting in more abrupt camera motions. As this dataset is monochrome, the color thresholding discussed in Section 4.3 is disabled while partitioning this model. The new *Apartment* dataset is meant to show the feasibility of using a D435 camera (recommended in Section 4.2) to explore and reconstruct new environments at a high enough quality for perch placement. The *Apartment* environment also offers a unique opportunity to validate the proposed camera poses with a real camera. Conformance between real and simulated camera views will be presented in Section 5.7.

4.8.3 Optimization Parameters

There are a large number of hyper-parameters which can have a large impact on the overall result of an optimization. To standardize between tests, the parameters in Table 5 will be used, unless otherwise stated. The camera parameters are based on the Intel D435 specifications. Other parameters are based on a best guess of what the future system limitations will be.

Table 5: Default Simulation Parameters

Parameter	Default Value
Number of Camera Drones	5
Number of Target Sampling Points	100
Camera Gimbal Range (Pan, Tilt)	(90°, 90°)
Camera Targeting Mode	Gimbal-Limited-Targeting
Target Deviation (Pan, Tilt)	(40°, 40°)
Max Camera Zoom	1x
Minimum Feature Size (World)	0.05 m
Minimum Feature Size (In Image)	10 px
Enable Perching On	Walls, Ceiling
Minimum Recovery Height	0.5 m
Minimum Perching Window	0.5 x 0.5 m

4.8.4 Evaluation Metrics

This section will discuss the many metrics used to evaluate the developed perch placement pipeline. First, the perching region segmentation and filtering nodes will be assessed for each dataset, to ensure that the optimization node has a suitable search space to work with. Then, the fitness function and search method will be validated.

Completeness of Perch Region Extraction

The best possible perching arrangement generated by the perch placement system is inherently limited to points within the search space. However, the perch placement pipeline has introduced several automated filters to narrow down the search space. While in principle, these filters only remove infeasible perches from the set, real world data may yield unexpected results. For this reason, it is desirable to validate the perch region filters by comparing the filtered set of surfaces to a manually segmented set.

Manual segmentation and area measurements will be performed on the original, pre-filtered reconstructions using *MeshLab* [130]. To ensure a fair comparison, the manual segmentation will abide by the criteria for perching discussed in Section 4.4. While many meshes in the selected dataset contain partially completed features, only visible regions will be considered when creating the ground truth segmentation, despite intuitive knowledge that surfaces will extend past the edge of the reconstruction. Additionally, the automatically filtered set of surfaces will be manually vetted to ensure that only valid perching areas are

considered. The area measures will be compared before and after manual verification. These results are demonstrated in Section 5.1.

Perch Selection Validity

A more qualitative metric that is important to assess is the validity of perching locations. The automatic filters applied to the search space prevent solutions which are geometrically infeasible from being proposed. However, model simplifications and segmentation errors can both result in invalid perches being proposed. While invalid perches are easily identified by the user using the GUI (see Section 4.6 for details), they are obviously non-ideal. When perch locations are rejected, a counter is incremented. Thus, the rate of invalid solutions can be assessed. This result will depend on the specific initialization of the optimizer, and the environment. To reduce uncertainty due to randomness, the failure rate will be evaluated over 30 camera placements for each test environment, and the average will be taken. For each camera placement, the optimizer will suggest positions until one is approved. Quantitative results will be displayed in Section 5.2.

Figure 37 shows a few characteristic cases which could be considered invalid.

Figure 37: Examples of invalid perching locations. (left) A drone placed at the interface between a door and a wall. The door jamb and sudden change in material make this an invalid perching location. (right) Drone placed on hanging blinds. While the blinds appear to the plane partitioning algorithm to be a solid surface, they would likely not be suitable for perching with the suggested grippers.

Next, the optimization node will be validated, starting with the fitness function.

Fitness Function Agreement

The fitness function defined in Equation 22 is used as the heuristic to assign a single numerical value corresponding to the quality of a given camera arrangement. It is crucial that this heuristic function actually corresponds well with the qualities that make a camera

arrangement optimal, to ensure that the optimization results are meaningful. The fitness function includes approximates of the coverage and uncertainty metrics discussed in Section 4.5.3, so some agreement is implied. However, the fitness function operates on a discrete subset of points, to increase computational efficiency. As a result, the agreement will not be perfect. Prior to using the fitness function in optimization, it is important to verify how well the fitness function correlates to coverage and tracking uncertainty. Predictive quality and computation time are both expected to be correlated with the number of sample points. An optimal sample size can be determined by comparing the agreement of the fitness function with the simulation time associated with the number of sampling points.

Camera coverage will be evaluated using a simulated camera frustum, discretized at 2° increments over the full field of view, to approximate the ground truth while remaining reasonably computationally efficient. Coverage evaluation will be discussed in greater in the discussion on ***Target Volume Coverage*** below. Tracking uncertainty will be evaluated over the target volume at a point density of 1000 pts/m^3. Results will be demonstrated in Section 5.3 using the *MIT_32_D507* environment, using the *Room Center* target. Additional results will be included in Appendix D for reference.

Search Convergence

To justify the hypothesis that PSO will provide an efficient method of obtaining near-optimal camera placements, it is worth comparing the solution qualities and speeds between different search approaches. To assess how quickly a solution will be found with the selected search method, the average fitness vs. time can be plotted. PSO, Grid Search, and Random Search will be compared. Speed will be compared against solution quality. Convergence speed will be assessed both in terms of the number of individual points assessed, and in terms of computation time. Both speed metrics will be compared, to determine whether the additional computational overhead associated with PSO is warranted by a faster convergence to a better solution. Solution quality will also be compared. The fitness evolution over time will be plotted for each search method. Ideally, a solution will converge quickly and will converge to a high fitness. As there is no direct way to solve for the optimal camera arrangement (hence the need for a meta-heuristic optimization), there is no way to generate a ground-truth optimal value against which the results can be compared. In lieu of this, the results of each method will be compared to the best result achieved by a brute force grid search which is allowed to optimize the camera placements overnight. The brute-force grid search will evaluate approximately 2,500,000 camera arrangements, evenly spaced over the perching surfaces, using greedy camera placement. By comparison, the other algorithms presented will evaluate approximately 10,000 arrangements.

As the success of these methods are all dependent on the initialization, N=30 trials with unique random seeds will be conducted. The mean and standard deviation of the considered metrics along the trials will be recorded for each method. Results for these tests are included in Section 5.4.

Target Volume Coverage

Perhaps the simplest metric to evaluate the efficacy of a camera arrangement is to assess the proportion of the target volume that is visible to at least one camera.

A camera's volume coverage can be approximated by ray-tracing along the path of each pixel, the distance of the nearest intersection point, and using this to generate a pyramidal volume for the pixel. The volume of each pixel can then be summed to give a camera volume coverage. A simplified version of this procedure is shown in Figure 38. The fitness function (Equation 22) used during the optimization approximates camera coverage volume by simply sampling over randomly generated discrete points in the volume. While this metric has reasonably high fidelity,

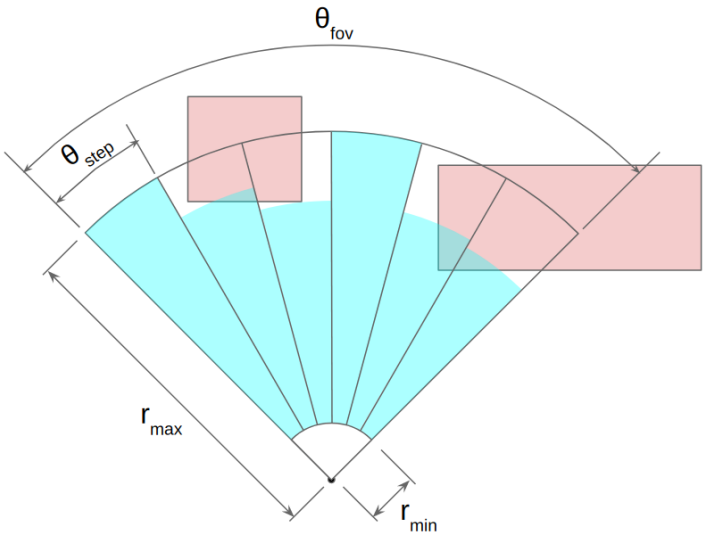


Figure 38: Simplified 2D view of ray-tracing as means of generating a coverage volume. r_{min} and r_{max} denote the minimum and maximum depth sensing range of the camera. θ_{fov} denotes the camera's field of view, in the plane of the diagram. θ_{step} denotes the discrete, angular step size used when evaluating a camera's volume coverage

To determine the coverage of a given volume, the computation becomes more complex. When coverage of a specific volume is desired, a boolean intersection of the camera volume and target volume is required. To efficiently compute the volumetric overlap, the `pymesh` library developed by Qingan Zhou [153] is used. `pymesh` supports mesh boolean operations,

allowing meshes to be combined, subtracted, and intersected. Using this library, a mesh representing a camera's field of view can be intersected with a any arbitrary target volume.

Camera meshes are generated by projecting a grid of sampling points on the image sensor, out from the camera's optical center. Points are projected to the minimum range of the depth camera to make the inner face of the mesh. The outer face of the mesh is taken as the minimum between the maximum range of the depth camera and the nearest obstruction point along that particular ray. For each point, the Brown-Conrady distortion model discussed in Section 2.2.3 is applied, to result in realistic view boundaries. The mesh is created by simply connecting the front and back surfaces with triangular faces.

Target volume meshes are generated using a simple oriented bounding box around the target surface.

While the camera mesh could be generated at the pixel level, in practice a much coarser mesh still yields high fidelity results and offers substantially reduced computation time. Using angular steps of 2° was found to have a nice balance between speed and accuracy. Figure 39 demonstrates the mesh intersection procedure with a 2° step.

Given a target mesh and an intersection mesh, a simple volume calculation of each mesh can yield a coverage metric for the placement.

$$\%Coverage = \frac{V(intersection)}{V(target)} \times 100 \tag{23}$$

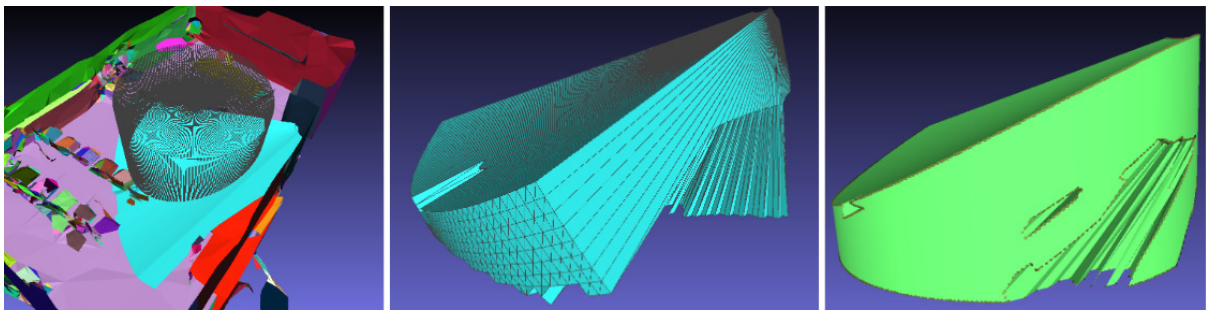


Figure 39: Camera-target mesh intersection procedure. (left) View of the partitioned mesh environment, along with a cylindrical target volume (black-mesh), and mesh representing the camera frustum (blue). (middle) Closeup of the camera frustum, demonstrating the missing volumes where the field of view is obstructed. (right) View of the target-camera intersection.

Tracking Uncertainty

Tracking uncertainty is evaluated using the point-wise ellipsoidal covariance representation, defined in Equations 20- 21. Because coverage is a highly discontinuous function, evaluating

tracking uncertainty using a continuous, analytic expression is non-trivial. Instead, the same discrete approach used in calculating fitness is taken, albeit with a higher density of sampling points. Figure 40 shows how points are generated over the target volume, and evaluated individually.

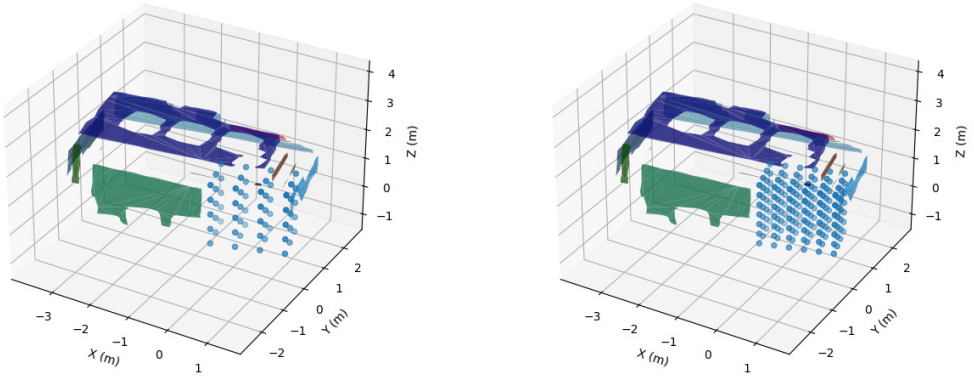


Figure 40: The segmented Office environment, shown with a sampling density of 20 pts/m^3 (left) and 70 pts/m^3 (right)

The average tracking uncertainty over the entire visible volume can then by assessed by considering the matrix norm of all non-NaN point-wise uncertainties. The coverage distribution over the space can also be assessed. The RMS uncertainty at each point can be expressed as the norm of the point-wise uncertainty matrix. The mean uncertainty over the total volume is then:

$$C_n = \frac{1}{N}\sum_{i=1}^{N} ||C_i|| \tag{24}$$

Tracking uncertainty results for the MIT environment will be demonstrated in Section 5.5. Results for other environments will be summarized in Section 5.5.

Processing Time

As there are many independent nodes in the proposed pipeline, it is worth understanding how long is spent in each node. This will help identify bottlenecks and will help better understand the system limitations for real world applications. Mean processing time for each node will be compared. The processing time is expected to be highly dependent on the environment selected and the reconstruction complexity, therefore results will be displayed for each individual environment. As the *MIT_32*, *House*, *Machine Hall 01*, and *Office* data sets consist of previously reconstructed meshes, mesh reconstruction time is not included. 3D reconstruction is performed on the *Apartment* dataset. Therefore, the processing time data

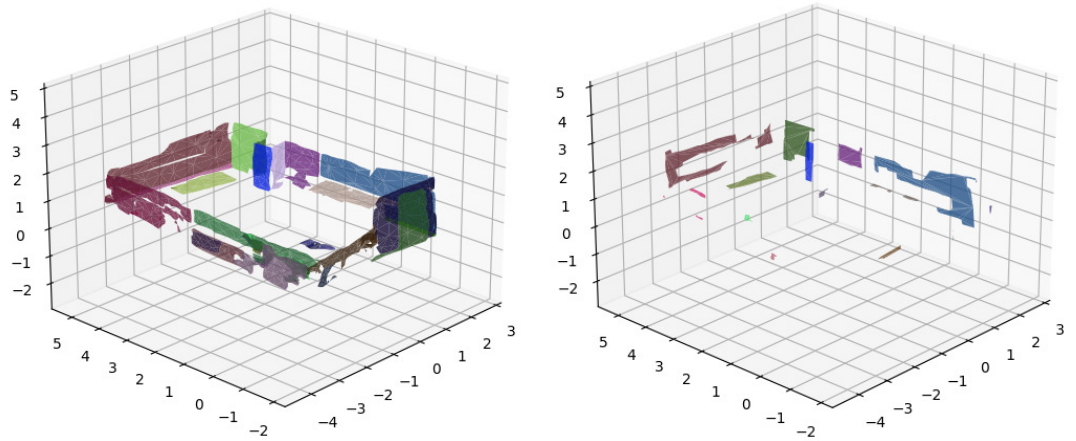
for this environment will indicate the processing time which can be expected when running the entire pipeline. Results are demonstrated in Section 5.6.

5 RESULTS

This section will show key results from each verification test. Due to the large number of generated figures, only the graphical results from the MIT_32_D507 will be shown in this section. Graphical results from the other environments will be included Appendix D for reference. Key results from each environment will still be tabulated and presented in this section.

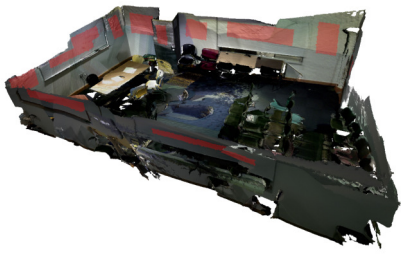
5.1 Completeness of Perch Region Extraction

Perch regions were automatically extracted, and manually extracted from the five environments. Figure 41 demonstrates the automatically generated perching regions (top) and the manually segmented perching regions (bottom).

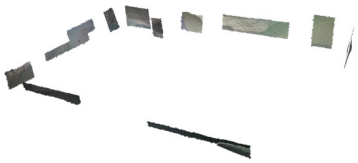


(a) Automatically Segmented Clusters

(b) Automatically Filtered Clusters



(c) Original Mesh



(d) Manually Segmented Regions

Figure 41: Comparison between manual and automatic filtering in the MIT environment

The resultant area of each segmentation technique is compared in Figure 42.

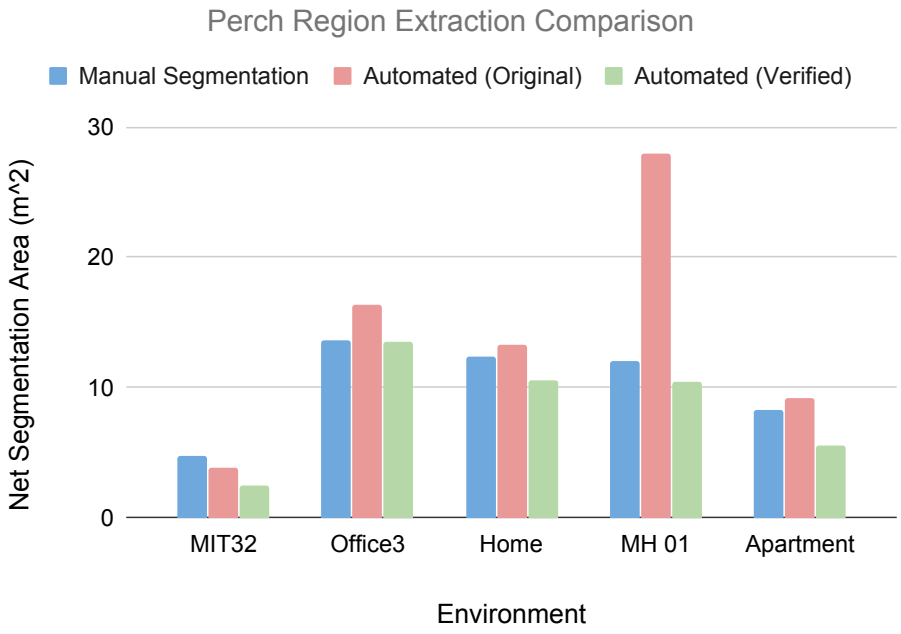


Figure 42: Comparison of extracted perch regions, by area, between manual segmentation, automatic segmentation and filtering, and automatic segmentation with manual verification and filtering.

Manual segmentation of each environment takes a trained user several minutes to complete, with the larger, more complex environments such as *MH 01* and *Home* taking somewhat longer.

5.2 Perch Selection Validity

Table 6 shows the success rate of proposed perching locations generated using greedy PSO, as determined by the system user. The reason for each rejection was noted. Figure 43 indicates the relative frequency of different causes for perch location rejection.

Table 6: Proportion of valid perching location proposals for different environment

Environment	Perch Location Approval Rate					
	1 Camera	2 Cameras	3 Cameras	4 Cameras	5 Cameras	**Average**
MIT32	100.00%	90.00%	66.67%	90.00%	86.67%	86.67%
Office3	100.00%	100.00%	90.00%	66.67%	70.00%	85.33%
Home	90.00%	90.00%	100.00%	90.00%	70.00%	88.00%
MH 01	90.00%	80.00%	80.00%	86.67%	76.67%	82.67%
Apartment	100.00%	90.00%	76.67%	80.00%	80.00%	85.33%
Average	96.00%	90.00%	82.67%	82.67%	76.67%	85.60%

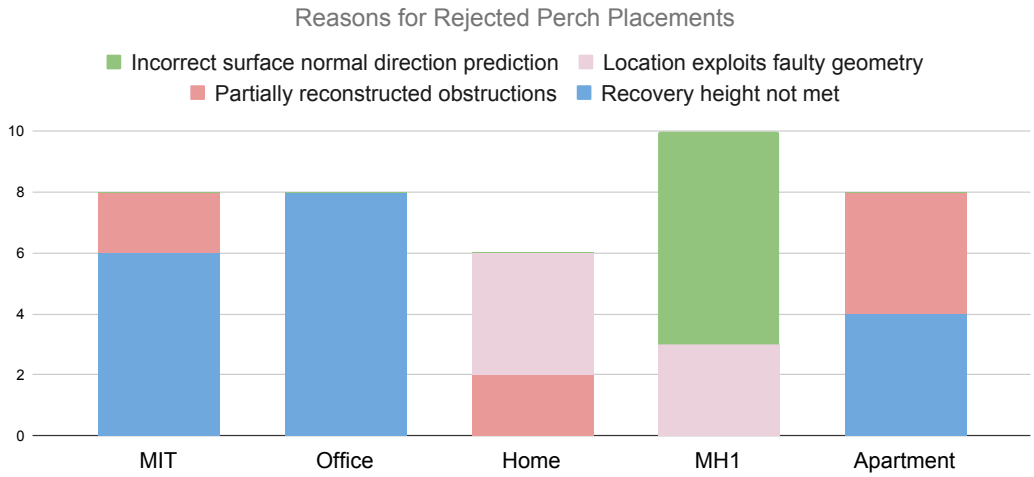


Figure 43: Reasons for perch location rejection

Each of the categories are described in more detail below.

- *Recovery height not met*: the perch location on a wall was rejected as it was too close to a nearby surface, such as a table or chair, and therefore did not leave enough vertical space to perform a recovery maneuver.
- *Partially reconstructed obstructions*: the perch location was rejected due to its proximity to geometry which was not fully or correctly reconstructed, but which can be identified by the operator as problematic.
- *Location exploits faulty geometry*: the perch location was rejected due to its exploitation of geometric faults, such as holes in the mesh. This error was more common in non-

convex environments, where holes in the wall would provide unique perspectives on the target.

- *Incorrect surface normal prediction*: the automatically predicted surface normal for a surface is incorrect, resulting in a camera that is in an unreachable area.

5.3 Fitness Function Verification

The fitness function was compared to baseline volumetric coverage and tracking uncertainty metrics. As the fitness function evaluates coverage and uncertainty metrics over discrete sampling points in the target volume, it was desired to determine the optimal sampling density. Fitness (z-axis) is plotted as a function of volumetric coverage (x-axis) and tracking uncertainty (y-axis) below. The agreement of the fitness function is evaluated by generating a best fit quadric surface of the form $z = c_0 + c_1 x + c_2 y + c_3 xy$, and quantifying the RMSE between the surface and the actual points. A lower RMSE indicates a better degree of agreement with the desired parameters. Several different sampling densities are tested to better understand the relationship between sampling density, computation time, and

Here, least-square best fit surfaces of the form are plotted, to better visualize the trend of the data points. The RMSE error between the reconstructed surface and the original fitness values is computed, and shown along with the graphs. Figure 44 shows the results for 10, 100, and 1000 sampling points.

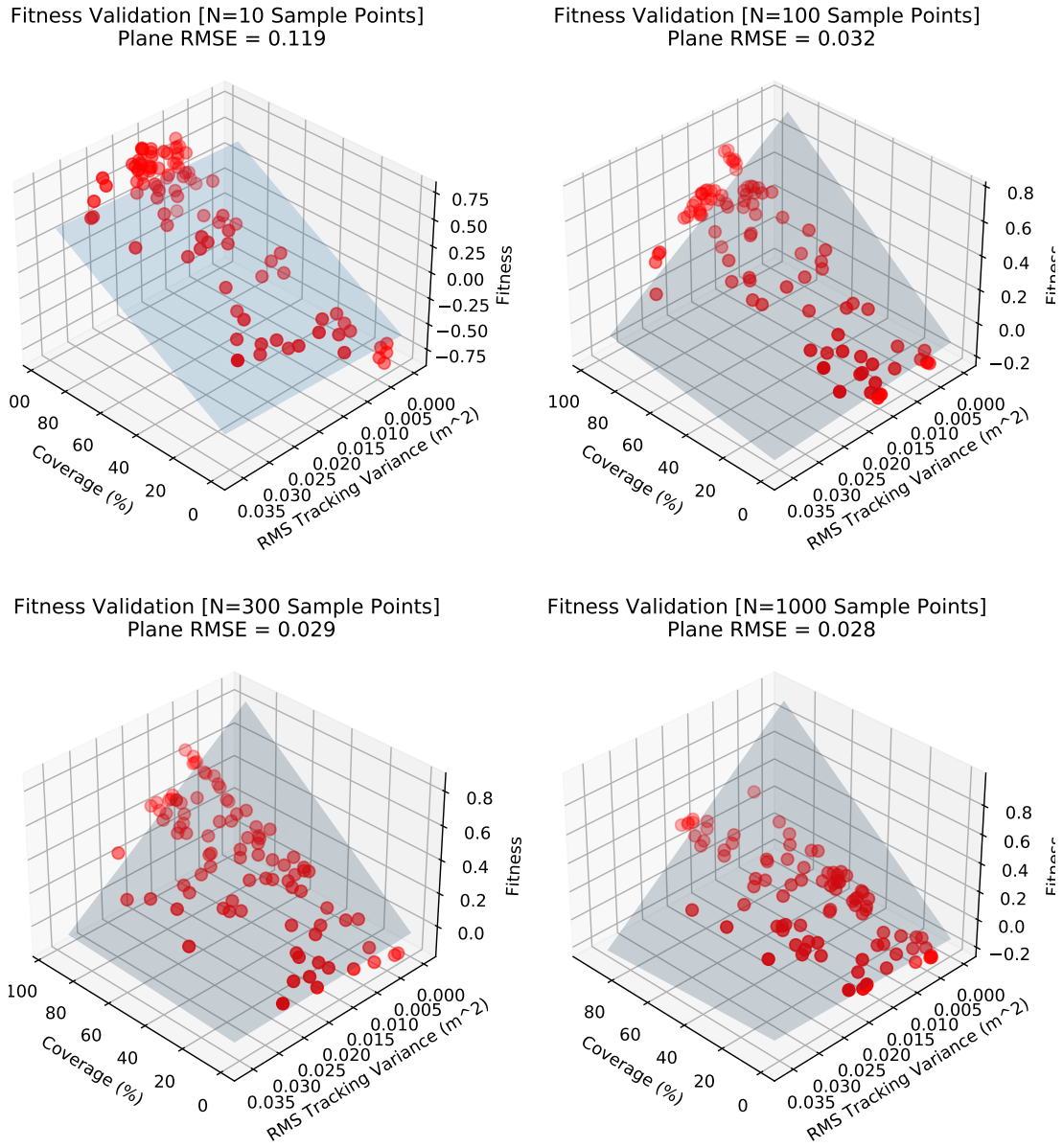


Figure 44: Fitness function agreement with coverage and tracking uncertainty metrics, for different sample sizes.

As the fitness function's predictive quality and computation time are expected to be conversely related, these values were plotted as well. Figure 45 shows these results.

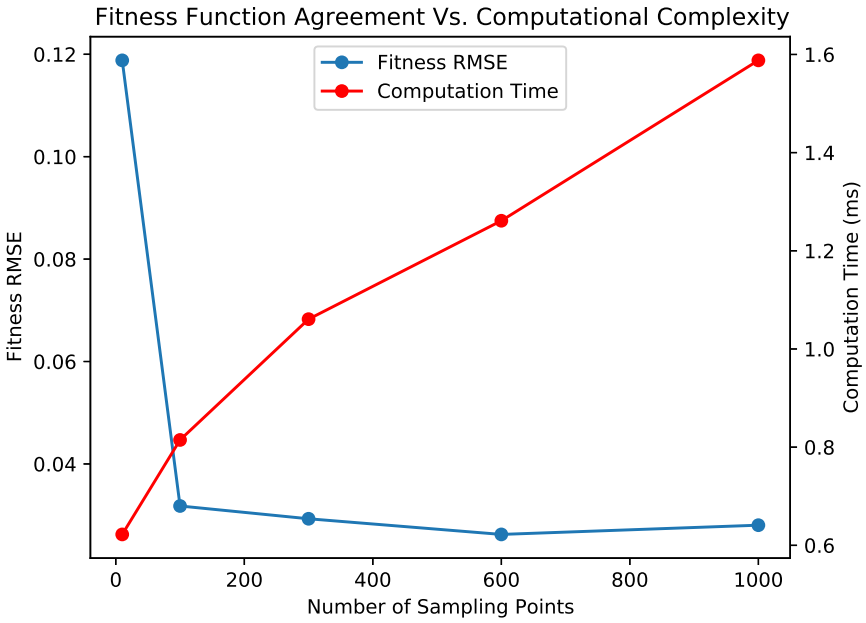


Figure 45: Predictive quality of fitness and computation time compared to the number of sample points. Predictive quality is quantified using the RMSE between the fitness and a best fit plane. Deviation from this plane indicates

5.4 Convergence Time

This section presents results comparing the convergence time and maximum fitness achieved by the different search algorithms tested. First the different algorithms were compared using greedy search. Cameras were incrementally added to the environment and placed optimally, according to the proposed fitness function. Figure 46 shows the results for the MIT environment. PSO, grid search, and random search are compared. A brute force "best" solution is also plotted to give a sense of the limit for the given environment and number of cameras. It is worth noting that the brute force "best" solution is not the global optimum, as it still relies on discrete sampling.

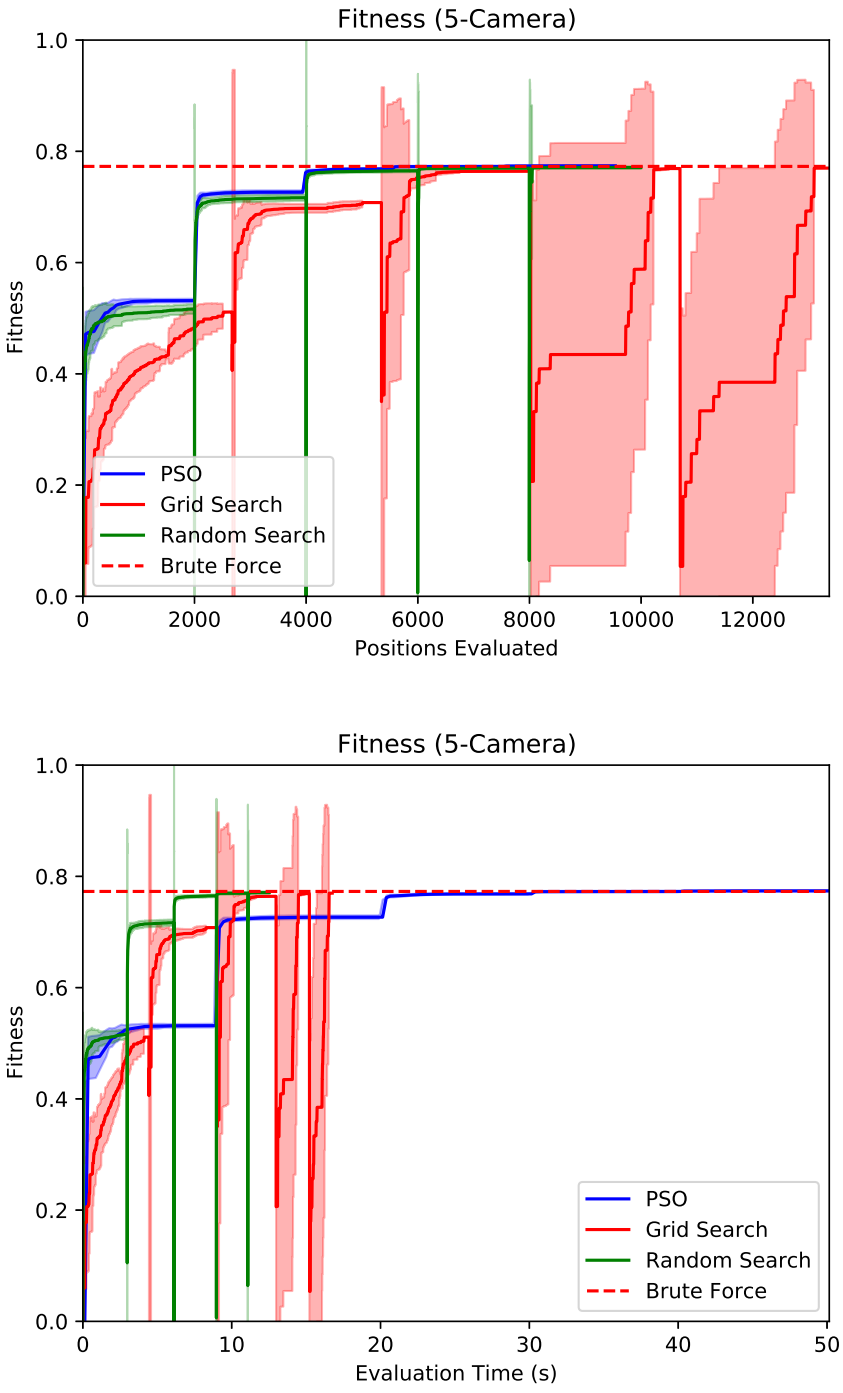


Figure 46: MIT Environment single camera search convergence comparison, comparing fitness vs points searched (top) and search time (bottom). The lines represent the mean fitness over 30 trials, and the filled region represents the $\pm 1\sigma$ standard deviation bounds. The dashed red line represents the brute-force "gold standard" achieved using a fine grid search over an 8 hour search period.

The steps seen in the graph correspond with new cameras being added using the greedy search paradigm. Note that the average fitness for Grid Search and Random Search drop after each step. This is because invalid proposals are penalized with 0 fitness, and each camera, the maximum fitness for each search is reset.

Figure 47 plots the same comparison as in Figure 46, however, when collecting the data for the PSO optimization, search convergence is automatically detected and is used as an event to stop the search early (see Section 4.5.3 for more detail). In Figure 47, the convergence flag is set if the fitness does not grow more than 0.001 over 5 generations.

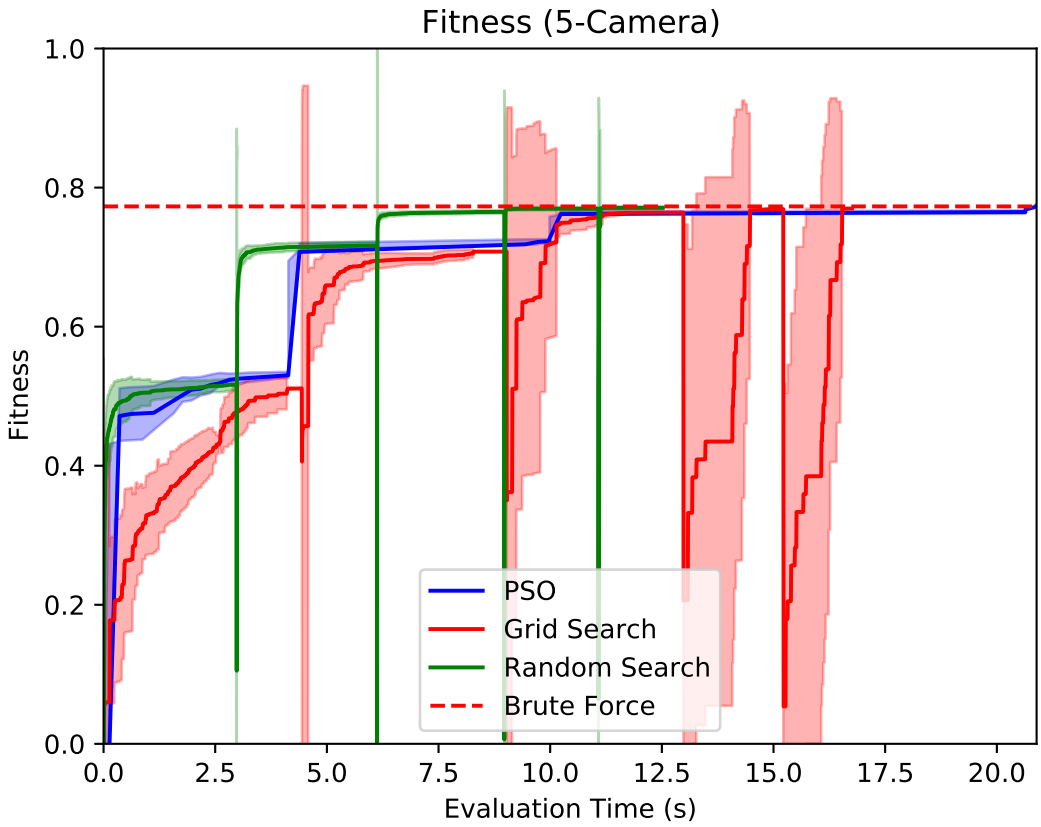


Figure 47: MIT Environment greedy camera search convergence comparison of fitness vs. time. The PSO search has convergence detection enabled and terminates the search for each camera early, if the fitness variance is too low.

Please refer to Appendix D.1 for the convergence plots of the remaining four environments. The key findings from each plot are summarized in Table 7 for reference.

Table 7: Convergence time (t_{conv}) and solution quality fitness (f_{max}) between Grid Search, Random Search, and PSO across the 5 test environments. Values are the mean of 30 trials. An algorithm's largest value f_{max} include the time to outperform the 2nd highest scoring option t_2 in parentheses. The best fitness and fastest convergence time for each trial is bolded. Solutions that outperform Brute Force are yellow.

Environment	**Algorithm**	1 Camera		2 Cameras		3 Cameras		4 Cameras		5 Cameras	
		f_{max}	t_{conv} (t_2)	f_{max}	t_{conv} (t_2)	f_{max}	t_{conv} (t_2)	f_{max}	t_{conv} (t_2)	f_{max}	t_{conv} (t_2)
MIT32	**PSO**	**0.325**	**3.9 (2.1)**	**0.606**	**1.3 (0.4)**	**0.667**	0.4 (0.3)	**0.673**	**0.1 (0.0)**	**0.674**	**0.0 (0.0)**
	Grid Search	0.296	4.1	0.579	3.6	0.66	2.1	0.667	1.5	0.669	1.3
	Random Search	0.304	2.4	0.592	0.7	0.662	**0.2**	0.668	0.0	0.67	0.0
	Brute Force	0.319	416.8	0.608	438.3	0.665	394.8	0.672	243.5	0.673	126.4
Office3	**PSO**	**0.383**	**7.5 (0.1)**	**0.588**	**6.9 (0.1)**	**0.658**	**2.0 (0.2)**	**0.686**	**0.6 (0.1)**	**0.699**	**0.2 (0.1)**
	Grid Search	0.365	6.9	0.572	6.1	0.647	4.7	0.676	2.7	0.689	1.0
	Random Search	0.323	4.2	0.547	2.7	0.637	1.2	0.674	0.5	0.69	0.1
	Brute Force	0.377	582.9	0.614	619.0	0.671	620.0	0.691	458.1	0.706	467.3
Home	**PSO**	**0.449**	**2.8 (0.3)**	0.647	1.9	**0.743**	**1.1 (0.1)**	**0.796**	**0.7 (0.0)**	**0.817**	**0.2 (0.0)**
	Grid Search	0.372	3.3	**0.655**	2.2 (2.0)	0.709	2.3	0.734	0.7	0.752	0.8
	Random Search	0.398	1.9	0.622	**0.9**	0.72	1.0	0.768	0.7	0.794	0.2
	Brute Force	0.428	251.1	0.659	273.6	0.771	271.4	0.8	253.2	0.817	245.0
MH 01	**PSO**	**0.091**	**2.0 (0.3)**	0.17	1.7	**0.239**	**1.8 (0.1)**	**0.296**	**1.5 (0.0)**	**0.342**	**1.7 (0.0)**
	Grid Search	0.08	1.5	**0.172**	1.6 (1.6)	0.216	0.9	0.256	1.0	0.293	1.3
	Random Search	0.085	1.3	0.162	**1.1**	0.228	1.1	0.286	1.1	0.333	1.0
	Brute Force	0.098	250.4	0.196	289.9	0.292	320.2	0.351	334.4	0.405	359.0
Apartment	**PSO**	**0.387**	**5.0 (0.6)**	**0.647**	**2.2 (0.1)**	**0.695**	**0.4 (0.1)**	**0.713**	**0.1 (0.0)**	**0.724**	**0.1 (0.0)**
	Grid Search	0.349	1.9	0.633	1.4	0.677	0.8	0.702	0.5	0.718	0.5
	Random Search	0.37	2.0	0.627	0.7	0.687	0.2	0.708	0.0	0.72	0.0
	Brute Force	0.393	528.7	0.641	536.2	0.7	448.3	0.718	338.1	0.725	339.1

It was also interesting to compare a greedy search approach, where cameras are incrementally added and fixed in place at their local optima, to a more naive approach where all camera positions are optimized simultaneously. PSO was used to compare the two methods. An equal number of point evaluations was allocated to each method, so that the only difference was search approach. Figure 48 shows the results for the MIT environment.

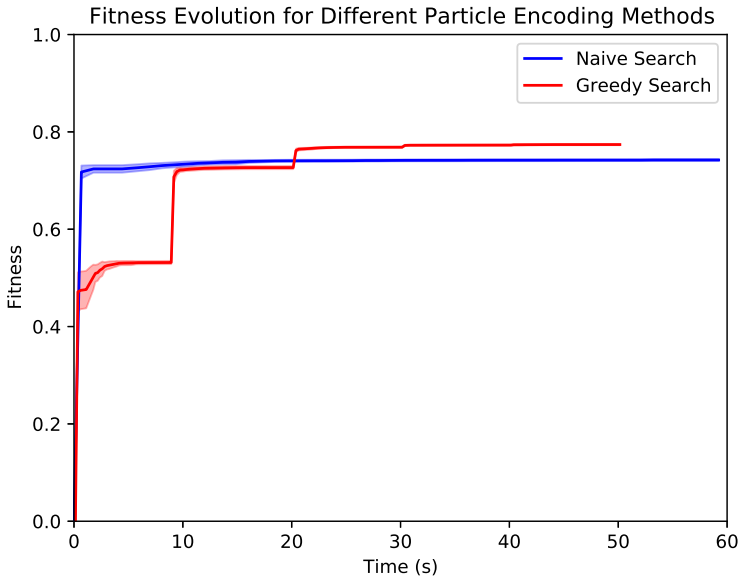


Figure 48: MIT Environment five camera search - comparison of greedy search, where cameras are incrementally placed at the best location, and naive search, where all cameras are simultaneously optimized.

Figure 49 compares the particle encoding methods suggested in Section 4.5.3. Mesh Encoding places particles in the center mesh faces, while Plane Encoding maps particles to a planar representation of the surface and searches in continuous space.

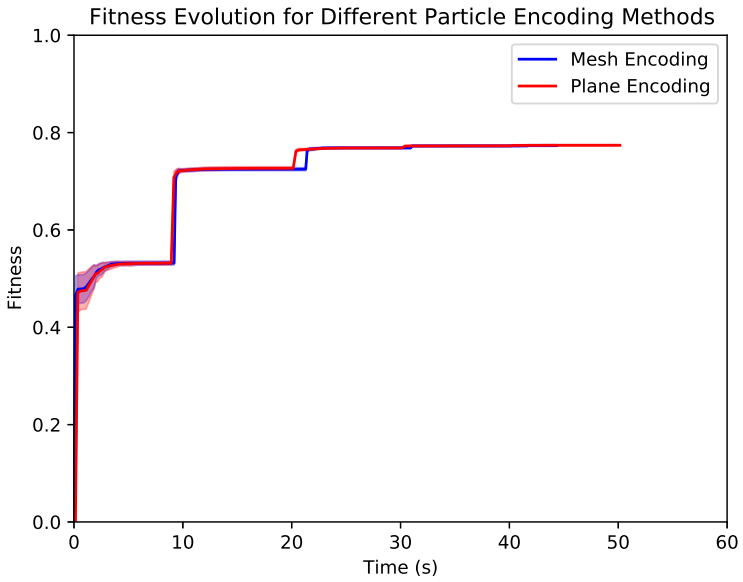


Figure 49: MIT Environment five camera search - comparison of Face encoding and Mesh encoding techniques.

5.5 Target Volume Coverage and Tracking Uncertainty

For each environment, a suggested camera placement was generated using greedy PSO. Tracking uncertainty and volumetric coverage will be compared for each environment, with one to five placed cameras. Figure 50 summarizes these results. Figure 51 demonstrates the different camera locations proposed by the PSO optimizer for the MIT environment. Note that times shown are for the default optimization parameters. In particular, the search time can be reduced significantly by enabling convergence detection.

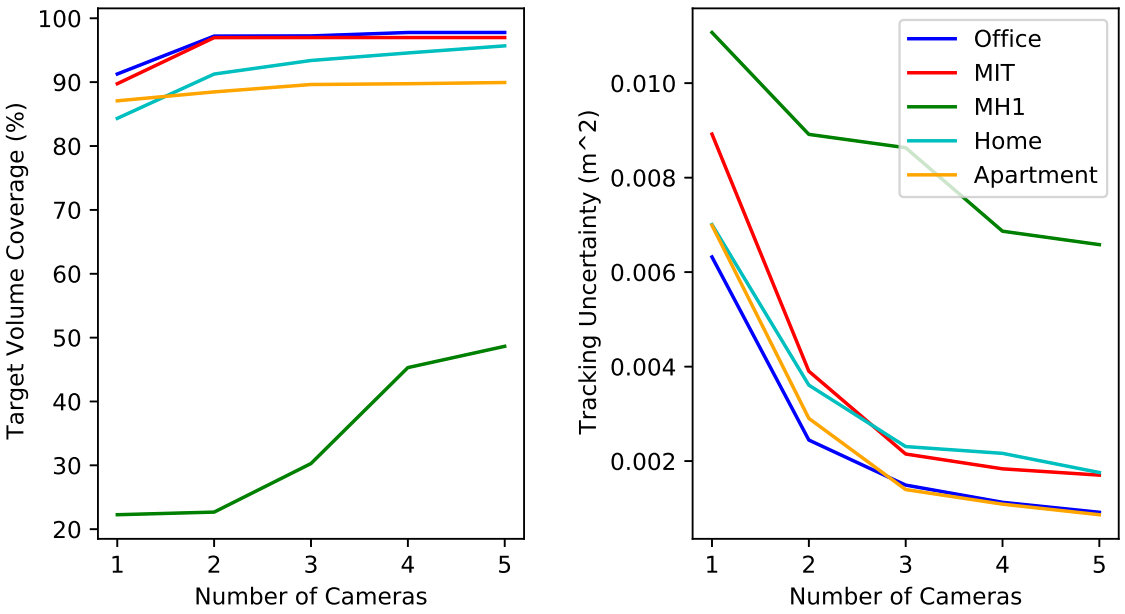
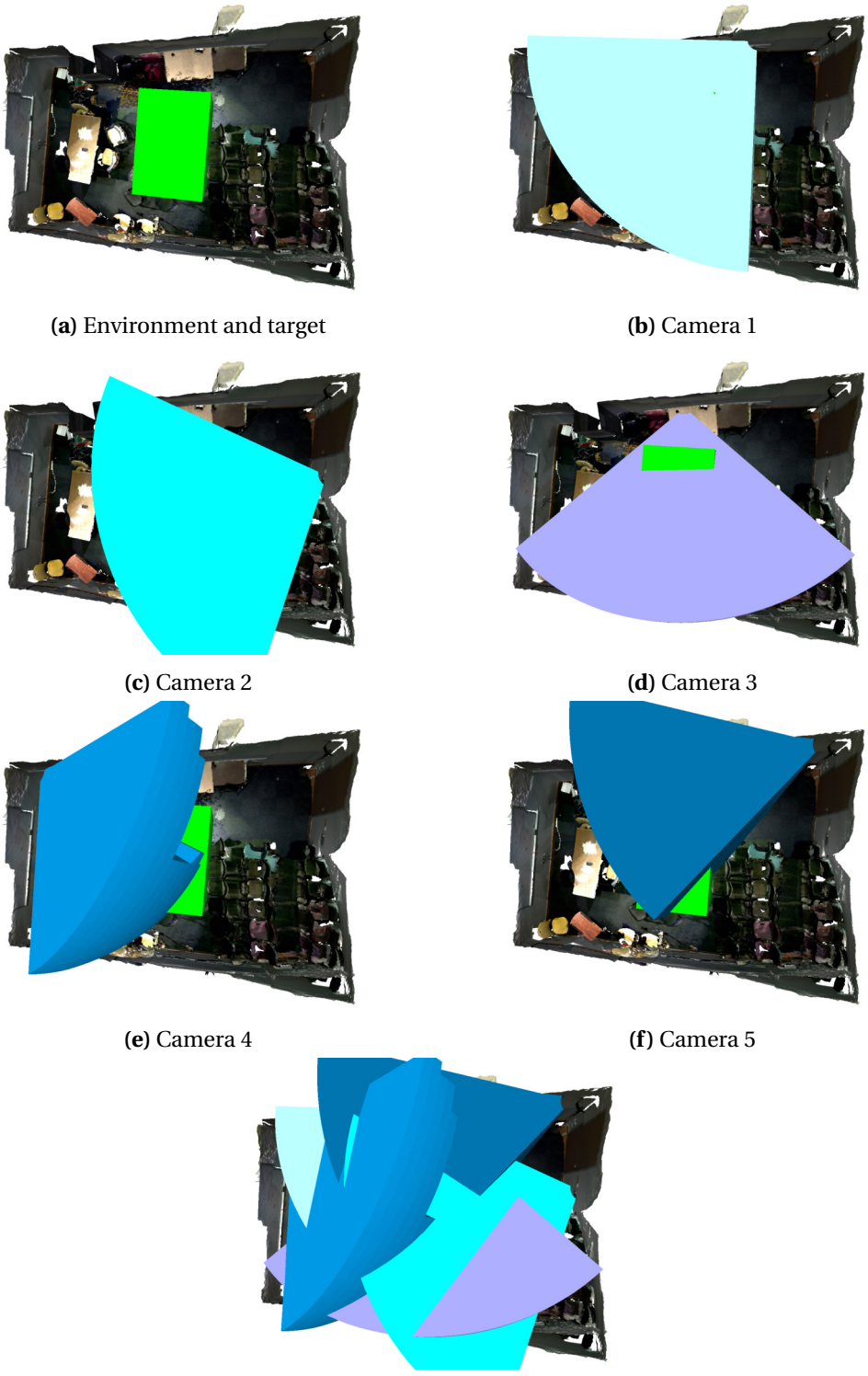


Figure 50: Comparison of camera coverage and tracking uncertainty, with increasing numbers of cameras.



(a) Environment and target

(b) Camera 1

(c) Camera 2

(d) Camera 3

(e) Camera 4

(f) Camera 5

(g) All Cameras

Figure 51: Camera placement results using greedy PSO in the MIT environment with a central box shaped target volume (green). Camera fields of view are shown in different colors and are added incrementally to demonstrate the variety of perspectives achieved.

Results for the remaining environments can be found in Appendix D.2.

5.6 Processing Time

Using the hardware discussed in Section 4.8, the computation speed for each step of the pipeline could be bench-marked. Figure 52 shows the mean processing time for each non-user interface node. Processing time varies with environment complexity, as one might expect.

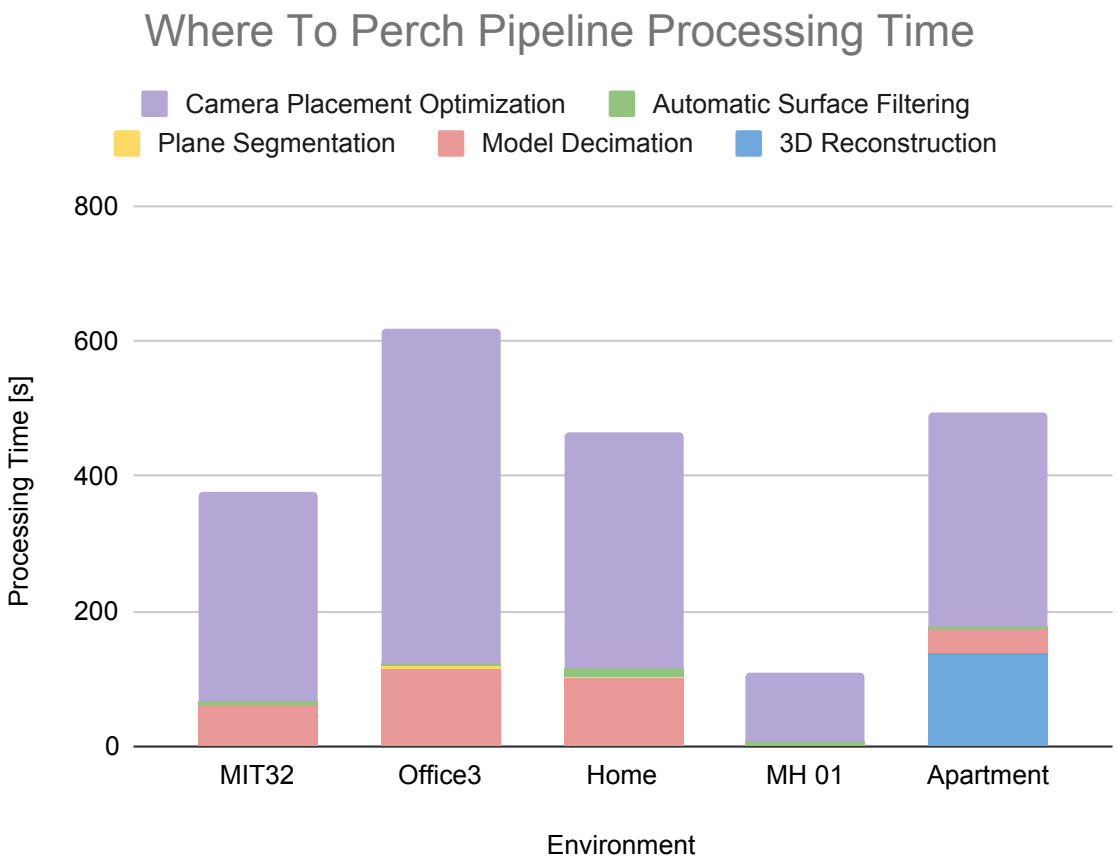


Figure 52: Mean node processing time for all non-GUI nodes. GUI nodes do little processing and are time limited by the user, not the computations. Optimization time shown is for a 5-camera placement problem, solved using Greedy PSO without convergence detection. Note that dense reconstruction is only performed in the Apartment dataset to test the capabilities of the D435 camera.

Processing time is shown in Table 8, to clearly quantify the faster nodes, such as the Plane Segmentation node.

Table 8: Node Processing Time

	Mean (Std. Dev.) Processing Time [s]				
Node	**MIT 32**	**Office3**	**Home**	**MH 01**	**Apartment**
Reconstruction	N/A	N/A	N/A	N/A	138.65 (0.41)
Mesh Filtering	58.70 (0.38)	115.86 (0.62)	101.24 (1.58)	0.59 (0.02)	35.43 (0.71)
Segmentation	1.73 (0.06)	1.81 (0.05)	1.59 (0.08)	1.63 (0.06)	1.92 (0.02)
Surface Filtering	4.86 (0.06)	4.54 (0.04)	11.66 (0.20)	6.37 (0.08)	3.88 (0.02)
Camera Placement	311.10 (4.38)	495.47 (37.24)	351.40 (1.13)	102.41 (1.75)	314.27 (1.51)

The Mesh Filtering node has considerably variance between the environments. To investigate the impact of input mesh size, processing time was plotted against mesh file size, as shown in Figure 53.

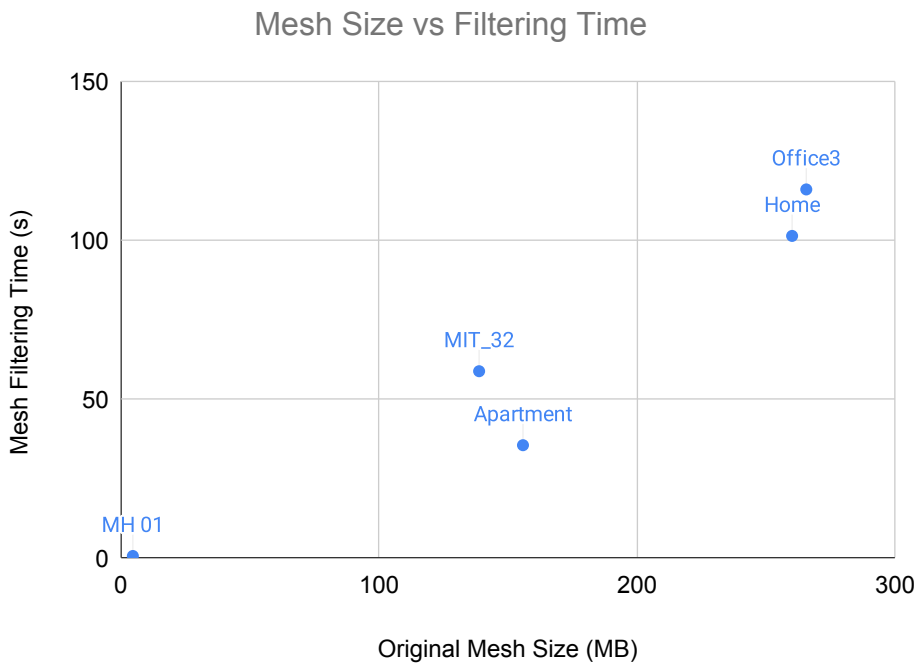
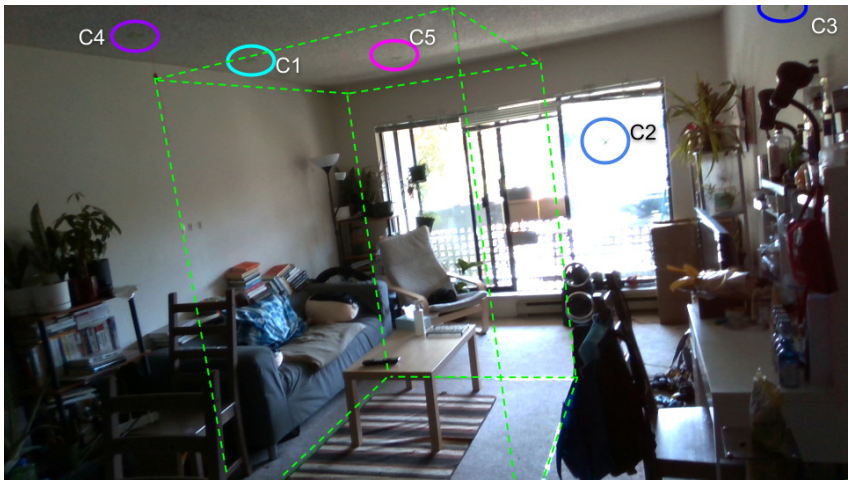


Figure 53: Mean mesh processing time compared to initial environment mesh file size.

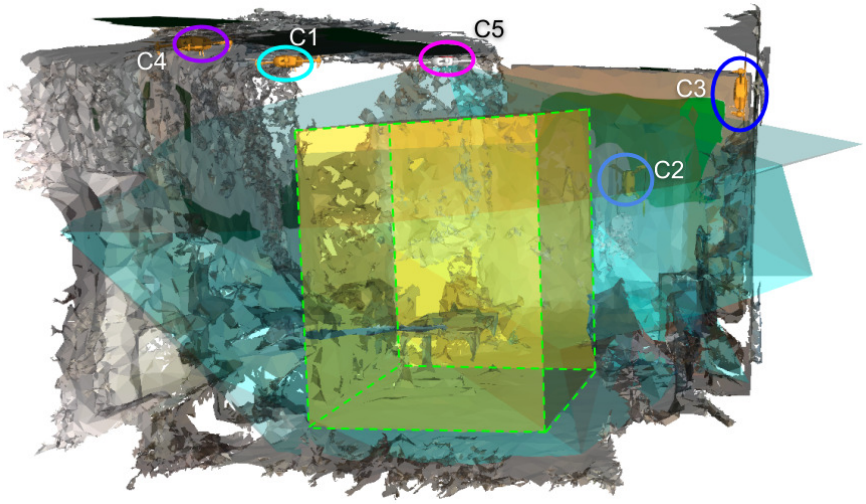
5.7 Camera Placement in the Real World

This section demonstrates the real-world camera views proposed by the camera placement optimization for the *Apartment* environment. Frames are captured manually using the Intel RealSense D435 camera. The camera is manually placed in the orientation suggested by the optimizer, and the target volume (in green) is overlaid to improve clarity.

Figure 54 gives an overview of the generated camera positions in the real-world and reconstructed environments.



(a) Real-world



(b) Simulated environment

Figure 54: Optimized camera positions in the *Apartment* environment, proposed by greedy PSO. Labels indicate the numerical order in which cameras 1 through 5 were placed.

Figures 55 - 59 compare simulated frames from each camera placement with frames

captured with the D435 in the real-world environment.

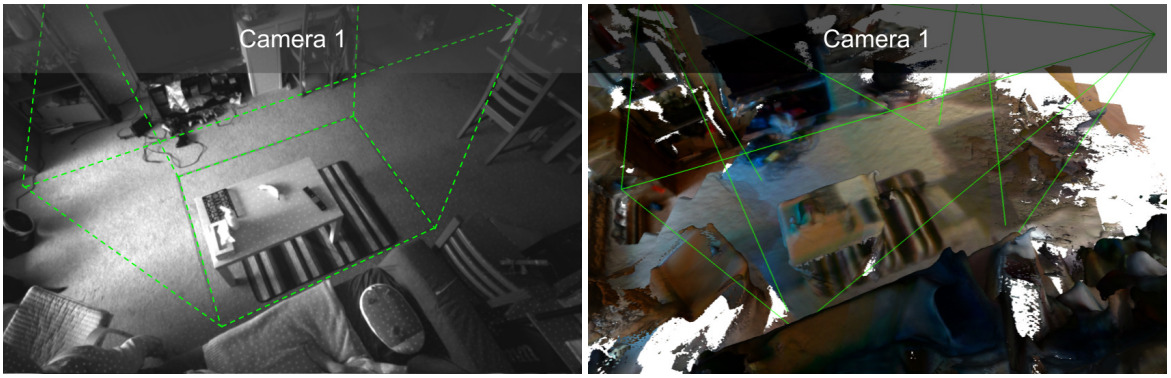


Figure 55: Camera 1 real-world (left) and simulated (right) view of the Apartment environment. The target volume is represented by the green edges.

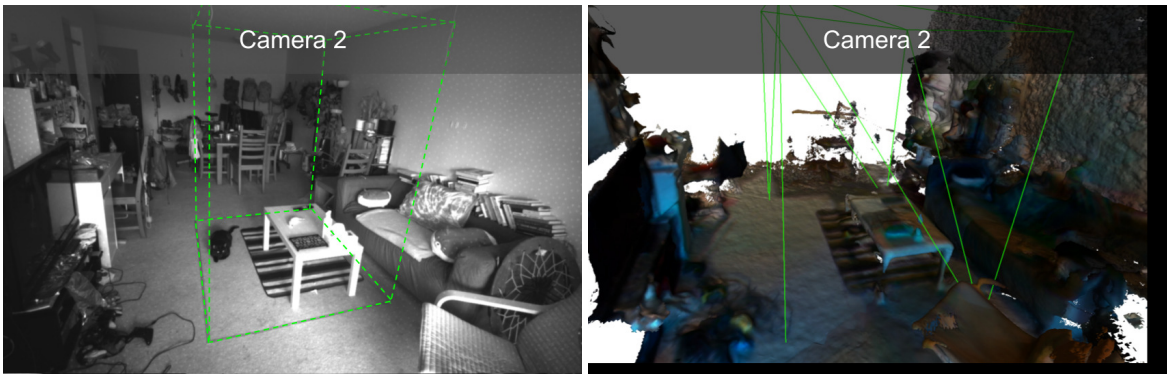


Figure 56: Camera 2 real-world (left) and simulated (right) view of the Apartment environment. The target volume is represented by the green edges.

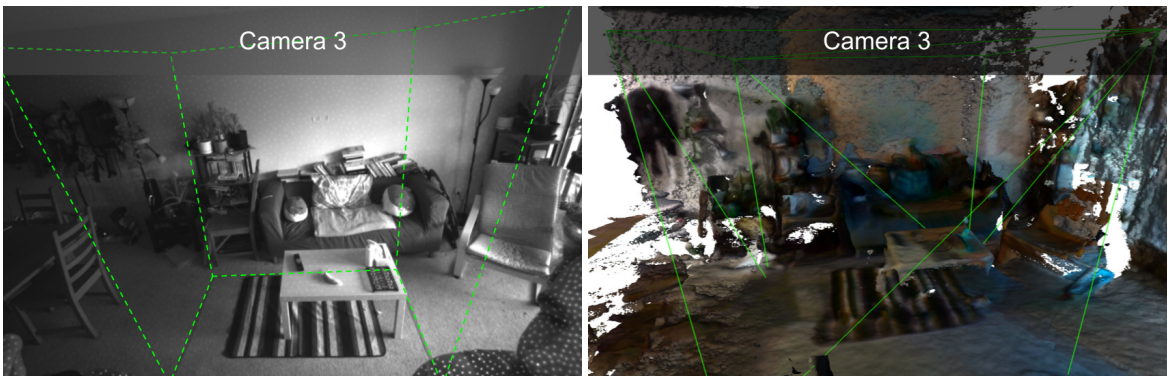


Figure 57: Camera 3 real-world (left) and simulated (right) view of the Apartment environment. The target volume is represented by the green edges.

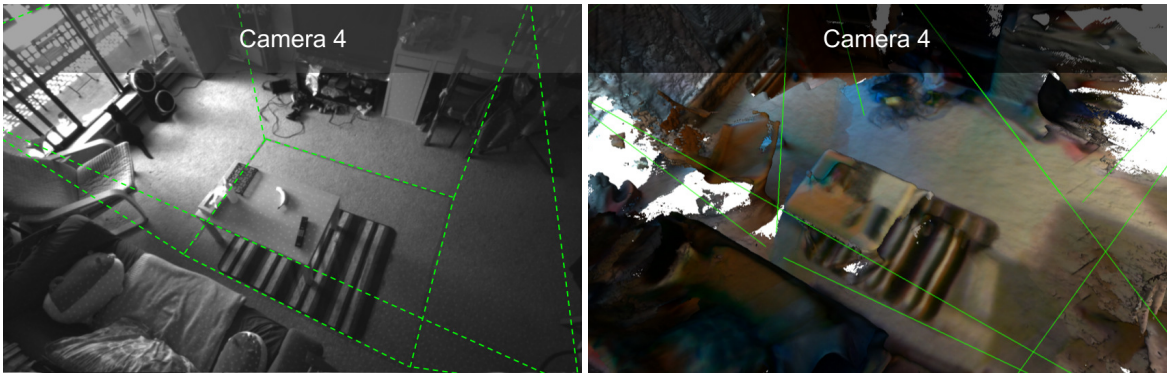


Figure 58: Camera 4 real-world (left) and simulated (right) view of the Apartment environment. The target volume is represented by the green edges.

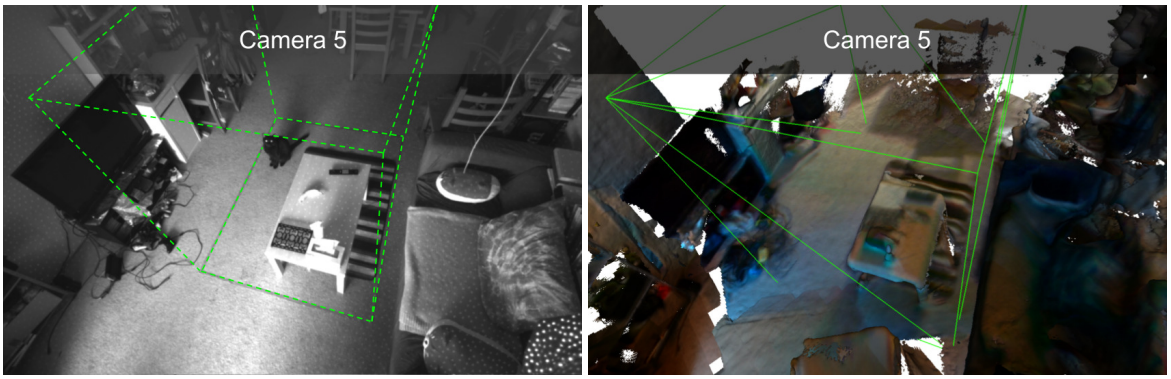


Figure 59: Camera 5 real-world (left) and simulated (right) view of the Apartment environment. The target volume is represented by the green edges.

6 DISCUSSION

This work's goal was to develop a full pipeline capable of identifying optimal perching locations to facilitate the deployment of an in-situ, drone-based motion capture system. The results shown demonstrate that this goal was largely met, and the proposed system meets the design requirements originally set (see Section 3 for reference).

6.1 Perching Surface Identification

The first system objective was to identify potential perching locations in reconstructions of a target environment suitable for dry adhesive grippers. To address this problem, a plane segmentation algorithm was developed, which reduces the mesh to discrete, nearly-planar regions. As shown in Table 8, this node runs in under 2 seconds, on average. When combined with the automated mesh filtering step, as a pre-process, the net run-time comes to between 1 and 2 minutes. When compared to manual segmentation, which averaged 5 minutes in testing environments, the automated node presents a significantly faster alternative. The time difference can become even larger if the original reconstruction file size is reduced. Figure 53 shows that mesh filtering time is roughly linearly related to mesh file size. Additionally, the mesh filtering step is required only due to memory limitations on the test system and limitations of the mesh reconstruction algorithm, and could potentially be removed altogether in future works.

But is the increased speed warranted? The answer is complicated. Figure 42 shows that there are large discrepancies between the fully automated filtering and manual segmentation results. However, these results were never meant to be in perfect agreement. Qualitative comparison of the segmented regions shown in Figure 69 and in Appendix D.3, show that the extracted regions in the mesh are largely the same, despite having different shapes. The user can only approximately follow the set perching guidelines, and was only given tools to make coarse, rectangular selections when identifying perching regions in the mesh. In contrast, the automated filtering can precisely extract non-obvious potential perching locations, regardless of shape. This leads to occasional over-segmentation, resulting in rejected perch locations, but more often than not results in a search space with a larger selection of surfaces. Indeed, in every environment except *MIT32* the automated filtering produced a larger set of perching surfaces than the manual segmentation. As the optimization search restricts the perching space as it progresses, a larger search space is not inherently bad.

In tests, manual verification of the automatically segmented surfaces does identify a significant number of invalid regions. Excluding the *Machine Hall* environment as an obvious

and intended outlier, on average 30% of the automatically segmented set was identified by the user to be invalid and is removed. There are a number of reasons for which these judgements are made. The most common reasons, counter-intuitively, are both under-segmentation and over-segmentation. Under-segmented regions are common when surface boundaries exist, but there is little geometric information to indicate the boundaries presence. Examples include the whiteboard-wall interface in the MIT and Office environments (e.g. Figure 4.3). In reality, these two surfaces should be separate, and a perching window limitation should be applied to each region. When they are combined as one, the accrued area is artificially high. Over-segmentation results from complex features being extracted as planes. In the *Apartment* dataset, the shelves in the room were reconstructed poorly, and ultimately segmented as planar surfaces, despite having very little frontal surface area.

The *Machine Hall* environment can be discussed independently, as it posed unique challenges to the system; the effect of this is noticed, as it required nearly twice that number to be removed in review. This is due to both the shear size of the environment, at nearly 4 times the volume of the next largest dataset (see Table 4), and the poor quality of the original mesh. As a result, many segmented surfaces were in far corners of the room and were removed by the user as they had no line-of-sight to the target. Cropping the environment mesh as a pre-processing step would help avoid these cases. Other surfaces were rejected due to there poor reconstruction quality. As the environment is large, so to are its features. Thus, a large number of poor quality, non-planar surfaces exceeded the minimum area requirement, which typically works well to filter out small or noisy features.

This leads to several important takeaways. First, the quality of the automated segmentation depends highly on the quality of the original mesh reconstruction. Environments with well defined planar surfaces with few reconstruction artefacts, like *Office3* and *Home* have comparably lower discrepancies between the manual and automated segmentations. Scenes with conflicting geometry and loop closure issues, including *MIT32*, and *Apartment* fared worse. The shear size of the *MH01* mesh causes the discrepancy to be exaggerated. User cropping of the environment prior to segmentation is warranted for larger environments, and would result in a large number of unusable surfaces being removed. Additionally, in all environments, except *MIT32*, the automatic segmentation resulted in larger sets of perching areas. This is in part due to the segmentation algorithm not being able to distinguish subtle geometric differences, but also because automated segmentation identifies potential perching areas that are missed during manual segmentation.

This sentiment is confirmed when considering the perch selection validity results demonstrated in Section 5.2. Even though the automatic segmentation and filtering nodes produce

sets of perching regions which are on average 30% invalid, Table 6 shows that only 15% of proposed perching locations are rejected. This discrepancy is largely attributed to the observation that many invalid surfaces are simply not useful as potential camera placement locations. Optimization techniques, such as PSO, naturally avoid regions in the search space which offer low reward. As a result, many of the surfaces that could have been filtered manually are implicitly filtered by the optimizer.

Of the 15% of proposed perching locations which are deemed invalid by the human operator invalid, it is interesting to analyze the reasoning and the various reasons for rejection, and their relative occurrences. As shown in Figure 43, the most common reason that perching regions were rejected was that the minimum recovery height requirement did not appear to be met. This was typically due to the proximity of objects such as chairs or desks, which were near the wall, but not contacting it. In the case of a failed perch, some rebound is expected and this volume must be left free. Unfortunately, the naive approach to minimum recovery height filter (discussed in Section 4.4.2) does not capture these edge cases. File the proximity filter does identify these obstructions, the region removal process projects obstructions along the surface normal. This has no effect when the obstructions are below the surface in question. In retrospect, a simple modification to the minimum perch height filter could be implemented to eliminate this failure mode. One possible solution is to add a simple secondary check to the proximity filter, which checks for neighboring obstructions that are below the mesh within a radius equal to the minimum perching height.

The remaining rejections were largely due to issues with the environment reconstruction. Failures due to incorrect surface normal prediction. These were only noticed in the Machine Hall environment, but could be expected to appear in other large, non-convex areas. The correct surface normal is predicted to be the direction which points towards the center of the environment. This assumption is invalid in non-convex spaces, such as in multi-room environments, where the wall of a side room will not necessarily point towards the center of the building. Normally, line-of-sight filtering (see Section 4.4.2) is normally sufficient to remove these surfaces. However, when the models contain holes or missing patches, surfaces can survive filtering and result in cameras being placed on the wrong sides of walls. Exploitation of faulty geometry stems from a very similar issue, but its manifestation was slightly different. In these cases, cameras were placed on valid perching surfaces, but in different rooms altogether. Holes in the mesh near the ceiling provided unique vantage points of the target in simulation, but would of course be useless in the real world.

Despite the room for improvement, it should be reiterated that the system does still function as intended the majority of the time. Additionally, with user in the loop to review

prior to drone dispatch, the severity of selecting from the set of invalid perching regions that may be in the search space is very low.

6.2 Camera Placement Optimization

The next two objectives were to maximize coverage over a target volume and to minimize target tracking uncertainty over the volume. Metaheuristic search using Particle Swarm Optimization was selected to address this challenge. The heuristic fitness function for the optimizer (defined in Equation 22) uses discrete approximations of both coverage and tracking uncertainty to predict the performance of a camera arrangement. As a result, the optimization of the fitness function should also lead to the optimization of the desired metrics. As the fitness function is perhaps the most important element of the optimization process, it is worth taking a moment to discuss how it was validated.

6.2.1 Fitness Validation

The fitness function was validated by generating random camera arrangements, evaluating their fitness, and then precisely evaluating the coverage and tracking uncertainty metrics, as defined in Section 4.8.4. By plotting the data collected in 3D and generating a best fit quadric surface, as shown in Figure 44, the fitness function is shown to agree well with the coverage and uncertainty metrics as measured by the low RMSE error between the surface and the points. Low RMSE to a continuous and monotonic surface implies both that there is little unexplained variance, and that there is high predictive quality of the fitness function. This is implied of course, as discrete approximations are expected to converge to their continuous counterparts. However, precision tends to come at a computational cost.

The secondary goal of the fitness function verification was to optimize the sampling density. Figure 44 shows the best fit plots for different numbers of sampling points, between to the best fit surface is calculated for each case, and is plotted against the computation time associated with evaluating that number of points. The computation time scales linearly with the number of points, as shown in Figure 45. Conversely, the predictive error is expected to scale with the inverse of the sampling density. The number of points is linearly related to the sampling density and target volume. So for a fixed volume, an optimal sampling density will exist that balances the trade-off between complexity and accuracy. Indeed, a clear inflection point was found at 100 points, and provided suitable justification to use this sampling density in all subsequent tests.

6.2.2 Search Convergence

With reasonable confidence in the fitness function, the next step was to test the optimization algorithm. The Camera Placement Optimization Node was built in such a way that most optimizers should be compatible. Camera placements are simply encoded using either continuous or discrete state variables (as discussed in Section 4.5.3). PSO was selected to take advantage of local and global inter-particle interactions to fine-tune solutions; however, PSO and other meta-heuristic optimizers add overhead to the problem, associated with updating individuals between iterations. When using greedy search, the problem's dimensionality is quite low, and could potentially be solved using naive or brute force search methods. To justify the choice of PSO, it was compared with Grid Search and Random Search.

Figure 46 and 47 demonstrate the mean and standard deviation performance of the three search methods for the MIT environment. Refer to Table 7 and Appendix D.1 for the results of other environments. Each algorithm is tasked with finding the best 5-camera arrangement for the 5 data sets. First, the methods are compared over a given number of iterations (Figure 46, top), and PSO is found to consistently yield solutions with higher fitness than grid search and random search. Plotting fitness vs. evaluation time shows that the PSO algorithm implemented does generate significant overhead, and more than doubles the optimization time. Random search completes the placement of 5 cameras in approximately 15s, while PSO takes nearly 50s to evaluate the same number of positions.

An augmentation the the PSO search was to add convergence detection, to preemptively terminate a search once progress has stopped. This feature is discussed in more detail in Section 4.5.3. Enabling convergence detection offers a considerable reduction in the overall computation time for PSO, and allows it to take advantage of its faster convergence speeds. Figure 47 shows that a convergence limited PSO search completes a 5 camera placement in approximately 60% less time. This reduction was possible using a fitness threshold of 0.001 over 5 generations. A further increase in speed appears to be possible, by simply increasing this threshold. This of course comes at the risk of prematurely terminating a search.. In the tests performed, the impact on the fitness achieved was negligible.

To determine whether PSO is worth the added evaluation time, one must consider two points. The first is that the success of random search in the early stages of optimization is largely due to the smart search principles discussed in Section 4.5.3. The search space for new camera arrangements is limited to a reasonable window around the target. As a result, generated camera placements will, on average, be pointing towards the target. For the first few cameras, volumetric coverage will dominate the fitness function, and there is little difference in fitness quality between Random Search and PSO. The distinction comes

when the placed cameras have already covered most of the target volume. This is where tracking uncertainty distinguishes between the best solutions. Indeed, Table 7 shows that PSO outperforms Grid Search and Random Search in nearly every case. In particular, it performs significantly better than Grid Search and Random search, both in terms of speed and quality in all of the 4th and 5th camera placements.

Comparing PSO to the Brute Force Grid Search is also encouraging. The brute force solution was allowed, on average, 100 times the computation time to take incremental steps over the search space and attempt to find an optimal solution. In the majority of cases, this methodical approach yielded the best possible fitness. This is to be expected given the processing power it was allocated. However, in 7/25 tests (shown in yellow in Table 7), PSO was found to outperform the brute force search, despite spending a small fraction of the time searching. This indicates the value of metaheuristic optimization compared to naive search methods.

Figure 49 compares the continuous Plane Encoding and the discrete Mesh Encoding strategies discussed in Section 4.5.3. While both methods result in nearly the same final fitness, there are two key findings. The first is that Plane Encoding is generally slower, by about 10% when compared to Mesh Encoding. This is to be expected, as Plane Encoding requires a 6-dimensional particle, while Mesh Encoding only requires 5-dimensions. The second finding is that the Plane Encoding converges faster, but the Mesh Encoding achieves slightly higher fitnesses, in all environments except the Machine Hall. This could of course be due to hyper parameter tuning, and does not necessarily indicate that one method is superior to the other. However, the discrete position representation does align with what prior groups [71, 75] have done.

The final test regarding search convergence was to evaluate the effectiveness of the greedy search strategy. Camera coverage and tracking uncertainty are both sub-modular functions (as discussed in Section 2.2.1) and can in theory be effectively optimized in a greedy fashion. To verify this idea, greedy and naive (or simultaneous) search techniques were compared. Figure 48 shows that, despite an early lead by the Naive Search, Greedy search consistently converges to a better final solution in the given amount of time. The early lead by the Naive Search can be attributed entirely to having more cameras looking at the target from the beginning. This allows it to achieve early results which are impossible with a single camera. However, the high dimensionality of the search space makes it challenging to assign credit to particles and move in the direction predict the fitness landscape. As a result, the Naive Search is largely stagnant.

6.2.3 Target Volume Coverage and Tracking Uncertainty

So far, the optimizations have been evaluated based on the achieved fitness. And while the fitness function itself was validated, it is still worth asking what an "optimized" arrangement even means. Figure 50 shows the target volume coverage with respect to the number of placed cameras. It should be no surprise that the volumetric coverage increases and approaches 100%, while the tracking uncertainty decreases, approaching 0 m^2. This acts as additional validation that both tracking uncertainty and coverage can be treated as sub-modular functions. As a brief aside, 100% volumetric coverage should not be expected, as an artefact of how the target volumes are placed, on the floor of each environment. In many cases, such as the Apartment and Home data sets where tables are featured in the center of the targe region, the obstructions take up a considerable portion of the volume. As such, part of the volume remains obstructed from all possible perspectives. Coverage is visualized for the MIT environment in Figure 51, showing the diversity of views proposed by the optimizer. Similar plots are also available in Appendix D.2.

6.2.4 Processing Time

As a final quantitative means of assessing the system, the mean processing time was computed for each of the automated nodes, using each environment. These results are demonstrated in Figure 52. Unsurprisingly, the majority of time is dedicated towards camera placement optimization. All prior steps are just precursors to the ultimate goal of generating an effective camera network. Note that the data shown is for the default PSO algorithm, without enhancements such as automatic convergence detection. As previously discussed in Section 5.4, the time spent in optimization could be reduced by approximately 60%, roughly halving the mean process time from 412s to 220s. In case more speed is needed, search time can further be reduced by enabling parallel processing. Parallel processing is implemented in the PySwarms library [142] used to manage the PSO optimizations. However, it was not implemented in the Random Search and Grid Search optimizers, and thus was not used when generating results to ensure for fair comparisons. Further optimization is possible by porting the search algorithms from Python into a faster compiled language, such as C++.

The apartment dataset goes one step further than the others, and demonstrates dense reconstruction using commercially available hardware and open source software. Going from start to finish, including dense reconstruction, averages just over 8 minutes. While there are several improvements that will be discussed in Section 7, 8 minutes would likely fit comfortably in the time it would take to prepare a small team of drones for flight, and fits

well within the constraints of the proposed disaster relief scenario.

6.3 Real World Camera Placement

Section 5.7 shows the camera placement system in action, working with a real world environment and a real camera. Figure 54a provides an image of the apartment, captured with the D435 camera's rgb sensor. The selected target volume and camera positions are overlaid on the image. Below, in Figure 54b, the simulated drone positions are shown in the reconstructed environment. The green regions in this figure represent the perch surface space being searched. Once optimized camera placements were generated, the RealSense camera was manually moved to each location, matched to the predicted field of view.

It is important to note that each camera is placed such that it has a substantially different perspective than the previous. The first camera is placed on the ceiling, near the left wall. The next camera's view is at nearly 90° to the first, with a lower perspective from the back wall. The 3rd camera makes the jump to the right wall, and the 4th and 5th cameras attempt to spread themselves in the remaining space on the ceiling.

7 RECOMMENDATIONS AND FUTURE WORK

The proposed system has demonstrated the feasibility of generating a near-optimal camera arrangement using perching drones. Within the constraints of the proposed disaster response use-case, the system is able to quickly generate a solution proposal and submit it for operator review. Despite the success of this proof-of-concept system, there are many potential areas to extend performance and explore new ideas. This section will briefly discuss some potential areas of improvement and future areas of study.

First and foremost is deployment in hardware. A perching drone-based motion capture system is certainly a tantalizing idea, and would open the door to several new applications in distributed robotics and sensing. However, there are several potential challenges that must be addressed prior to achieving this dream. Of primary concern is the prevalence of uncertainty, especially in autonomous tasks. This work did not investigate the impact of positional uncertainty on the resultant camera network quality, as small variations can be largely accounted for using the camera's gimbal. However, uncertainty could be accounted for through a minor change to the camera placement optimization node. By simply adding artificial noise to the camera position defined by the particle, and employing a noise resistant variant of PSO, such as PSO Optimal Computing Budget Allocation (OCBA) [154], the solution would be more likely to converge on solutions that are near-optimal, even with the increased level of uncertainty. Beyond perching uncertainty, camera noise will also have an impact. Verification that a multi-camera motion capture system can be produced using the RealSense D435 camera, or similar is recommended. If a more accurate camera is required, it will have implications on the surfaces which can support perching, as the drone's payload weight will likely be a limiting factor. A major area for future work that would be required prior to the full automation of the pipeline is to develop a system which can automatically detect perching surfaces. The proposed system works well within the context of having operator oversight, but a fully autonomous system would need the ability to make more reliable distinctions between surfaces. To that end, it could be interesting to attempt to train a deep learning model capable of directly evaluating surfaces based on perching suitability. If successful, this would be a considerable improvement from the current gold standard, which is operator intuition.

Beyond testing the system components in hardware it could be interesting to investigate different frameworks for the perch selection and drone control. The proposed system employs a centralized framework, where a single processing node assigns positions to each drone. This fits the framework of human-robot collaboration nicely, and takes advantage of

the computing hardware already required to generate a dense reconstruction of the environment. However, there is no reason that the methods presented could not be reworked into a distributed framework where drones optimize their own positions. One could envision the greedy search becoming a leader-follower arrangement where the leader selects the optimal location and the follower react accordingly. Alternately, a divide and conquer approach could be employed, where drones search different areas of the environment until a suitable perch is found, while broadcasting their position and search success to other drones.

A final interesting topic that merits more thought is the notion of human-robot interaction and collaboration. Indeed, one of key goals of the FlyCroTug project was to explore human robot collaboration, and how autonomous or semi-automomous robotic systems can help first responders. Human interaction was considered briefly in this project, and it impacted the development of the GUI. However, no robust, quantitative measures were performed to assess operator burden, either with or without the system. Future user studies comparing camera placement quality, speed, and difficulty both with and without the proposed pipeline could yield interesting revelations. Previous works [71, 72] have mentioned that optimal camera placement is deceptively non-intuitive. Thus, user studies may reaffirm the need for computational methods.

8 CONCLUSION

Remote, impromptu motion capture systems could provide a low-cost and effective way to capture state information and enable teams of aerial robots to interact with novel environments in new and exciting ways. New perching mechanisms, depth sensors, and miniaturized computers brings this idea ever closer to reality. One of the questions that arises when considering this possibility is, "where to perch?" This question has received little attention in prior works and has interesting implications. The work presented here attempts to answer this question with an integrated perch placement system. This system solves two key problems on the road towards DroCap. The first is perch location identification. This was solved using a fast plane clustering algorithm and subsequent geometric filters to automatically generate a set of feasible perching surfaces. The second challenge that was addressed is how to optimally select perching locations to maximize sensing coverage and quality. Using Particle Swarm Optimization with novel search space modifications was shown to be an effective way to quickly generate these positions in unstructured environments. The system presented is realistic in its assumptions, and is ready for integration with hardware systems. New data was presented which shows that the system is capable of taking exploratory video and outputting near-optimal camera arrangement in under ten minutes. This work represents the first system of its kind and a step towards functional DroCap.

REFERENCES

[1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones", en, *Nature*, vol. 521, no. 7553, pp. 460–466, May 2015, Number: 7553 Publisher: Nature Publishing Group.

[2] S. Mintchev and D. Floreano, "A pocket sized foldable quadcopter for situational awareness and reconnaissance", in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2016, pp. 396–401.

[3] O. Fernandes, R. Murphy, D. Merrick, J. Adams, L. Hart, and J. Broder, "Quantitative Data Analysis: Small Unmanned Aerial Systems at Hurricane Michael", Sep. 2019, pp. 116–117.

[4] R. R. Murphy, "International Cooperation in Deploying Robots for Disasters: Lessons for the Future from the Great East Japan Earthquake", *Journal of the Robotics Society of Japan*, vol. 32, no. 2, pp. 104–109, 2014.

[5] M. Półka, S. Ptak, and Ł. Kuziora, "The Use of UAV's for Search and Rescue Operations", en, *Procedia Engineering*, 12th international scientific conference of young scientists on sustainable, modern and safe transport, vol. 192, pp. 748–752, Jan. 2017.

[6] M. Silvagni, A. Tonoli, E. Zenerino, and M. Chiaberge, "Multipurpose UAV for search and rescue operations in mountain avalanche events", *Geomatics, Natural Hazards and Risk*, vol. 8, no. 1, pp. 18–33, Jan. 2017.

[7] Z. Zaheer, A. Usmani, E. Khan, and M. A. Qadeer, "Aerial surveillance system using UAV", in *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, ISSN: 2151-7703, Jul. 2016, pp. 1–7.

[8] J. H. Marden, "Maximum Lift Production During Takeoff in Flying Animals", en, *Journal of Experimental Biology*, vol. 130, no. 1, pp. 235–258, Jul. 1987, Publisher: The Company of Biologists Ltd Section: Journal Articles.

[9] K. Karydis and V. Kumar, "Energetics in robotic flight at small scales", *Interface Focus*, vol. 7, no. 1, p. 20 160 088, Feb. 2017, Publisher: Royal Society.

[10] Y. Mulgaonkar, M. Whitzer, B. Morgan, C. M. Kroninger, A. M. Harrington, and V. Kumar, "Power and weight considerations in small, agile quadrotors", in *Micro- and Nanotechnology Sensors, Systems, and Applications VI*, vol. 9083, International Society for Optics and Photonics, Jun. 2014, 90831Q.

[11] M. T. Pope, C. W. Kimes, H. Jiang, E. W. Hawkes, M. A. Estrada, C. F. Kerst, W. R. T. Roderick, A. K. Han, D. L. Christensen, and M. R. Cutkosky, "A Multimodal Robot for Perching and Climbing on Vertical Outdoor Surfaces", *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 38–48, Feb. 2017, Conference Name: IEEE Transactions on Robotics.

[12] H. Tsukagoshi, M. Watanabe, T. Hamada, D. Ashlih, and R. Iizuka, "Aerial manipulator with perching and door-opening capability", in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 1050-4729, May 2015, pp. 4663–4668.

[13] M. A. Estrada, S. Mintchev, D. L. Christensen, M. R. Cutkosky, and D. Floreano, "Forceful manipulation with micro air vehicles", en, *Science Robotics*, vol. 3, no. 23, Oct. 2018, Publisher: Science Robotics Section: Research Article.

[14] S. Wang, "Traversing Highly-Varied Terrain: Enanced Contacts for Human-Scale Robot Locomotion", Doctoral Thesis, Stanford University Department of Mechanical Engineering, Aug. 2017.

[15] D. L. Christensen, S. A. Suresh, K. Hahm, and M. R. Cutkosky, "Let's All Pull Together: Principles for Sharing Large Loads in Microrobot Teams", *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 1089–1096, Jul. 2016, Conference Name: IEEE Robotics and Automation Letters.

[16] D. Vasisht, S. Kumar, and D. Katabi, "Decimeter-Level Localization with a Single WiFi Access Point", en, 2016, pp. 165–178.

[17] M. Kamel, S. Verling, O. Elkhatib, C. Sprecher, P. Wulkop, Z. Taylor, R. Siegwart, and I. Gilitschenski, "The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle", *IEEE Robotics & Automation Magazine*, vol. 25, no. 4, pp. 34–44, 2018.

[18] X. Ding, P. Guo, K. Xu, and Y. Yu, "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems", en, *Chinese Journal of Aeronautics*, vol. 32, no. 1, pp. 200–214, Jan. 2019.

[19] M. Vrba and M. Saska, "Marker-less micro aerial vehicle detection and localization using convolutional neural networks", *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020.

[20] F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano, "Learning vision-based flight in drone swarms by imitation", *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4523–4530, 2019.

[21] J. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots", en, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD: IEEE, May 2018, pp. 2502–2509.

[22] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera", en, in *Robotics Research : The 15th International Symposium ISRR*, ser. Springer Tracts in Advanced Robotics, H. I. Christensen and O. Khatib, Eds., Cham: Springer International Publishing, 2017, pp. 235–252.

[23] X.-D. Ke, H. W. Schreier, M. A. Sutton, and Y. Q. Wang, "Error Assessment in Stereo-based Deformation Measurements: Part II: Experimental Validation of Uncertainty and Bias Estimates", en, *Experimental Mechanics*, vol. 51, no. 4, pp. 423–441, Apr. 2011.

[24] F. Schiano, A. Franchi, D. Zelazo, and P. Robuffo Giordano, "A rigidity-based decentralized bearing formation controller for groups of quadrotor UAVs", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 5099–5106.

[25] F. Schiano and P. Robuffo Giordano, "Bearing rigidity maintenance for formations of quadrotor UAVs", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1467–1474.

[26] F. Schiano and R. Tron, "The dynamic bearing observability matrix nonlinear observability and estimation for multi-agent systems", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1–8.

[27] N. Saini, E. Price, R. Tallamraju, R. Enficiaud, R. Ludwig, I. Martinovic, A. Ahmad, and M. J. Black, "Markerless outdoor human motion capture using multiple autonomous micro aerial vehicles", in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 823–832.

[28] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm", in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, May 2001, pp. 145–152.

[29] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics", en, *Intelligent Industrial Systems*, vol. 1, no. 4, pp. 289–311, Dec. 2015.

[30] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments", *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, 2011.

[31] K. M. Jatavallabhula, G. Iyer, and L. Paull, "gradSLAM: Dense SLAM meets Automatic Differentiation", Montreal, Canada, Oct. 2019.

[32] C. Zhao, L. Sun, P. Purkait, T. Duckett, and R. Stolkin, "Dense RGB-D Semantic Mapping with Pixel-Voxel Neural Network", en, *Sensors*, vol. 18, no. 9, p. 3099, Sep. 2018, Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[33] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]", *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[34] C. Chang, H. Zhu, M. Li, and S. You, "A Review of Visual-Inertial Simultaneous Localization and Mapping from Filtering-Based and Optimization-Based Perspectives", *Robotics*, vol. 7, p. 45, Aug. 2018.

[35] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016", *IPSJ Transactions on Computer Vision and Applications*, vol. 9, Dec. 2017.

[36] *Comet Labs Research Team, "Teaching Robots Presence: What You Need to Know About SLAM", CometLabs.io.* en, Aug. 2017.

[37] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", en, *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[38] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras", *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, arXiv: 1610.06475.

[39] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An Open-Source Library for Real-Time Metric-Semantic Localization and Mapping", *arXiv:1910.02490 [cs]*, Mar. 2020, arXiv: 1910.02490.

[40] R. Mur-Artal and J. D. Tardós, "Visual-Inertial Monocular SLAM With Map Reuse", *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, Apr. 2017, Conference Name: IEEE Robotics and Automation Letters.

[41] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[42] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM", en, in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 834–849.

[43] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration", *arXiv:1604.01093 [cs]*, Feb. 2017, arXiv: 1604.01093.

[44] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D Mapping With an RGB-D Camera", en, *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, Feb. 2014.

[45] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph", en, in *Robotics: Science and Systems XI*, Robotics: Science and Systems Foundation, Jul. 2015.

[46] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction", en, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI: IEEE, Jul. 2017, pp. 6565–6574.

[47] S. Bell, P. Upchurch, N. Snavely, and K. Bala, "Material recognition in the wild with the Materials in Context Database", en, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA: IEEE, Jun. 2015, pp. 3479–3487.

[48] C. Zhao, L. Sun, and R. Stolkin, "A fully end-to-end deep learning approach for real-time simultaneous 3D reconstruction and material recognition", *2017 18th International Conference on Advanced Robotics (ICAR)*, pp. 75–82, Jul. 2017, arXiv: 1703.04699.

[49] S. Pillai and J. Leonard, "Monocular SLAM Supported Object Recognition", in *Robotics: Science and Systems XI*, Rome, Italy, Jun. 2015.

[50] D. L. Butler, "Surface Roughness Measurement", en, in *Encyclopedia of Microfluidics and Nanofluidics*, D. Li, Ed., Boston, MA: Springer US, 2014, pp. 1–7.

[51] P. Giguere and G. Dudek, "A Simple Tactile Probe for Surface Identification by Mobile Robots", *IEEE Transactions on Robotics*, vol. 27, no. 3, pp. 534–544, Jun. 2011, Conference Name: IEEE Transactions on Robotics.

[52] C. Liu, L. Sharan, E. H. Adelson, and R. Rosenholtz, "Exploring features in a Bayesian framework for material recognition", en, in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, USA: IEEE, Jun. 2010, pp. 239–246.

[53] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing Textures in the Wild", *arXiv:1311.3618 [cs]*, Nov. 2013, arXiv: 1311.3618.

[54] J. Xue, H. Zhang, K. Dana, and K. Nishino, "Differential Angular Imaging for Material Recognition", 2017, pp. 764–773.

[55] C. Kampouris, S. Zafeiriou, A. Ghosh, and S. Malassiotis, "Fine-Grained Material Classification Using Micro-geometry and Reflectance", en, in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 778–792.

[56] K. Matheus and A. M. Dollar, "Benchmarking grasping and manipulation: Properties of the Objects of Daily Living", in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, ISSN: 2153-0866, Oct. 2010, pp. 5020–5027.

[57] A. Dame, V. A. Prisacariu, C. Y. Ren, and I. Reid, "Dense Reconstruction Using 3D Object Shape Priors", en, in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA: IEEE, Jun. 2013, pp. 1288–1295.

[58] L. Hu, W. Xu, K. Huang, and L. Kneip, "Deep-SLAM++: Object-level RGBD SLAM based on class-specific deep shape priors", *arXiv:1907.09691 [cs, eess]*, Dec. 2019, arXiv: 1907.09691.

[59] S. Yang and S. Scherer, "CubeSLAM: Monocular 3-D Object SLAM", en, *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, Aug. 2019.

[60] J. McCormac, R. Clark, M. Bloesch, A. J. Davison, and S. Leutenegger, "Fusion++: Volumetric Object-Level SLAM", *arXiv:1808.08378 [cs]*, Aug. 2018, arXiv: 1808.08378.

[61] C. Wang and X. Guo, "Plane-Based Optimization of Geometry and Texture for RGB-D Reconstruction of Indoor Scenes", en, in *2018 International Conference on 3D Vision (3DV)*, Verona: IEEE, Sep. 2018, pp. 533–541.

[62] S. Yang and S. Scherer, "Monocular Object and Plane SLAM in Structured Environments", *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3145–3152, Oct. 2019, arXiv: 1809.03415.

[63] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", SRI International, Tech. Rep., May 1980, p. 42.

[64] O. Chum and J. Matas, "Optimal Randomized RANSAC", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 8, pp. 1472–1482, Aug. 2008, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[65] M. Y. Yang and W. Forstner, “Plane Detection in Point Cloud Data”, en, Department of Photogrammetry Institute of Geodesy and Geoinformation, University of Bonn, Tech. Rep. 1, 2010, p. 16.

[66] R. Schnabel, R. Wahl, and R. Klein, “Efficient RANSAC for Point-Cloud Shape Detection”, en, *Computer Graphics Forum*, vol. 26, no. 2, pp. 214–226, 2007, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01016.x.

[67] A. Monszpart, N. Mellado, G. J. Brostow, and N. J. Mitra, “RAPter: Rebuilding man-made scenes with regular arrangements of planes”, en, *ACM Transactions on Graphics*, vol. 34, no. 4, pp. 1–12, Jul. 2015.

[68] The CGAL Project, *CGAL User and Reference Manual*, 5.0.3. CGAL Editorial Board, 2020.

[69] A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto, “Octree-based region growing for point cloud segmentation”, en, *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 104, pp. 88–100, Jun. 2015.

[70] Y. Cai, X. Guo, Y. Liu, W. Wang, W. Mao, and Z. Zhong, “Surface Approximation via Asymptotic Optimal Geometric Partition”, en, *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 12, pp. 2613–2626, Dec. 2017.

[71] X. Chen and J. Davis, “Camera Placement Considering Occlusion for Robust Motion Capture”, en, Stanford University, Tech. Rep. CS-TR-2000-07, Jul. 2000, p. 8.

[72] R. Chabra, A. Ilie, N. Rewkowski, Y.-W. Cha, and H. Fuchs, “Optimizing placement of commodity depth cameras for known 3D dynamic scene capture”, in *2017 IEEE Virtual Reality (VR)*, ISSN: 2375-5334, Mar. 2017, pp. 157–166.

[73] M. Schwager, B. J. Julian, M. Angermann, and D. Rus, “Eyes in the Sky: Decentralized Control for the Deployment of Robotic Camera Networks”, *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1541–1561, Sep. 2011, Conference Name: Proceedings of the IEEE.

[74] J. Zhao, D. Haws, R. Yoshida, and S.-c. Cheung, “Approximate Techniques in Solving Optimal Camera Placement Problems”, vol. 2013, Nov. 2011, pp. 1705–1712.

[75] P. Rahimian and J. K. Kearney, “Optimal Camera Placement for Motion Capture Systems”, *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 3, pp. 1209–1221, Mar. 2017, Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[76] E. Hörster and R. Lienhart, “On the optimal placement of multiple visual sensors”, in *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, ser. VSSN ’06, Santa Barbara, California, USA: Association for Computing Machinery, Oct. 2006, pp. 111–120.

[77] J. Marzal, “The three-dimensional art gallery problem and its solutions”, en-gb, 2012.

[78] E. Yildiz, K. Akkaya, E. Sisikoglu, and M. Y. Sir, “Optimal Camera Placement for Providing Angular Coverage in Wireless Video Sensor Networks”, *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1812–1825, Jul. 2014, Conference Name: IEEE Transactions on Computers.

[79] Y. Morsly, N. Aouf, M. S. Djouadi, and M. Richardson, “Particle Swarm Optimization Inspired Probability Algorithm for Optimal Camera Network Placement”, *IEEE Sensors Journal*, vol. 12, no. 5, pp. 1402–1412, May 2012, Conference Name: IEEE Sensors Journal.

[80] L. C. A. Pimenta, V. Kumar, R. C. Mesquita, and G. A. S. Pereira, “Sensing and coverage for a network of heterogeneous robots”, in *2008 47th IEEE Conference on Decision and Control*, ISSN: 0191-2216, Dec. 2008, pp. 3947–3952.

[81] A. Agogino, K. Tumer, and R. Miikkulainen, “Efficient Credit Assignment through Evaluation Function Decomposition”, in *Genetic and Evolutionary Computation Conference*, Washington DC, USA, Jun. 2005.

[82] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, “An analysis of approximations for maximizing submodular set functions—I”, en, *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, Dec. 1978.

[83] A. Krause and C. Guestrin, “Near-optimal Observation Selection using Submodular Functions”, en, *American Association for Artificial Intelligence*, vol. 2, pp. 1650–1654, 2007.

[84] J. Vondrak, “Polyhedral Techniques in Combinatorial Optimization”, Stanford University, Tech. Rep., Nov. 2010.

[85] *"Camera Calibration and 3D Reconstruction", OpenCV Documentation*, Dec. 2019.

[86] D. Brown, “Decentering Distortion of Lenses”, *Photogrammetric Engineering*, pp. 444–462, May 1966.

[87] Z. Zhang, “A flexible new technique for camera calibration”, en, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.

[88] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[89] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A Toolbox for Easily Calibrating Omnidirectional Cameras", en, in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China: IEEE, Oct. 2006, pp. 5695–5701.

[90] A. Buades, B. Coll, and J. M. Morel, "A Review of Image Denoising Algorithms, with a New One", *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 490–530, Jan. 2005, Publisher: Society for Industrial and Applied Mathematics.

[91] A. K. Boyat and B. K. Joshi, "A Review Paper: Noise Models in Digital Image Processing", *arXiv:1505.03489 [cs]*, May 2015, arXiv: 1505.03489.

[92] C. Zhu, X. Shao, C. Liu, and X. He, "Accuracy analysis of an orthogonally arranged four-camera 3D digital image correlation system", *Applied Optics*, vol. 58, Aug. 2019.

[93] A. Grunnet-Jepsen, J. Sweetser, and J. Woodfill, *Tuning depth cameras for best performance*, en, Library Catalog: dev.intelrealsense.com, Apr. 2020.

[94] G. Olague and R. Mohr, "Optimal camera placement for accurate reconstruction", en, *Pattern Recognition*, vol. 35, no. 4, pp. 927–944, Apr. 2002.

[95] P. Re, *Blond-crested Woodpecker - eBird*, en, 2019.

[96] C. E. Doyle, J. J. Bird, T. A. Isom, J. C. Kallman, D. F. Bareiss, D. J. Dunlop, R. J. King, J. J. Abbott, and M. A. Minor, "An Avian-Inspired Passive Mechanism for Quadrotor Perching", *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 2, pp. 506–517, Apr. 2013, Conference Name: IEEE/ASME Transactions on Mechatronics.

[97] A. Kalantari, K. Mahajan, D. Ruffatto, and M. Spenko, "Autonomous perching and take-off on vertical walls for a quadrotor micro air vehicle", in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 1050-4729, May 2015, pp. 4669–4674.

[98] M. Lee, "Drones get a grip", en, *Nature Electronics*, vol. 2, no. 4, pp. 139–139, Apr. 2019, Number: 4 Publisher: Nature Publishing Group.

[99] M. T. Pope and M. R. Cutkosky, "Thrust-Assisted Perching and Climbing for a Bioinspired UAV", en, in *Biomimetic and Biohybrid Systems*, N. F. Lepora, A. Mura, M. Mangan, P. F. Verschure, M. Desmulliez, and T. J. Prescott, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 288–296.

[100] K. Zhang, P. Chermprayong, T. M. Alhinai, R. Siddall, and M. Kovac, "SpiderMAV: Perching and stabilizing micro aerial vehicles with bio-inspired tensile anchoring

systems", in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Sep. 2017, pp. 6849–6854.

[101] J. Thomas, M. Pope, G. Loianno, E. W. Hawkes, M. A. Estrada, H. Jiang, M. R. Cutkosky, and V. Kumar, "Aggressive Flight With Quadrotors for Perching on Inclined Surfaces", en, *Journal of Mechanisms and Robotics*, vol. 8, no. 5, Oct. 2016, Publisher: American Society of Mechanical Engineers Digital Collection.

[102] F. J. Garcia-Rubiales, P. Ramon-Soria, B. Arrue, and A. Ollero, "Magnetic detaching system for Modular UAVs with perching capabilities in industrial environments", in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, Nov. 2019, pp. 172–176.

[103] E. Culler, G. Thomas, and C. Lee, "A Perching Landing Gear for a Quadcopter", en, Honolulu, Hawaii: American Institute of Aeronautics and Astronautics, Apr. 2012.

[104] H. Zhang, J. Sun, and J. Zhao, "Compliant Bistable Gripper for Aerial Perching and Grasping", in *2019 International Conference on Robotics and Automation (ICRA)*, ISSN: 2577-087X, May 2019, pp. 1248–1253.

[105] M. Tieu, D. M. Michael, J. B. Pflueger, M. S. Sethi, K. N. Shimazu, T. M. Anthony, and C. L. Lee, "Demonstrations of bio-inspired perching landing gear for UAVs", in *Bioinspiration, Biomimetics, and Bioreplication 2016*, vol. 9797, International Society for Optics and Photonics, Apr. 2016, p. 97970X.

[106] E. W. Hawkes, E. V. Eason, D. L. Christensen, and M. R. Cutkosky, "Human climbing with efficiently scaled gecko-inspired dry adhesives", *Journal of the Royal Society Interface*, vol. 12, no. 102, Jan. 2015.

[107] A. T. Asbeck and M. R. Cutkosky, "Designing Compliant Spine Mechanisms for Climbing", en, *Journal of Mechanisms and Robotics*, vol. 4, no. 3, Aug. 2012, Publisher: American Society of Mechanical Engineers Digital Collection.

[108] J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, "Visual Servoing of Quadrotors for Perching by Hanging From Cylindrical Objects", *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 57–64, Jan. 2016, Conference Name: IEEE Robotics and Automation Letters.

[109] D. Ruffatto, A. Parness, and M. Spenko, "Improving controllable adhesion on both rough and smooth surfaces with a hybrid electrostatic/gecko-like adhesive", *Journal of The Royal Society Interface*, vol. 11, no. 93, p. 20 131 089, Apr. 2014, Publisher: Royal Society.

[110] M. A. Graule, P. Chirarattananon, S. B. Fuller, N. T. Jafferis, K. Y. Ma, M. Spenko, R. Kornbluh, and R. J. Wood, "Perching and takeoff of a robotic insect on overhangs using switchable electrostatic adhesion", en, *Science*, vol. 352, no. 6288, pp. 978–982, May 2016, Publisher: American Association for the Advancement of Science Section: Report.

[111] B. Aksak, K. Sahin, and M. Sitti, "The optimal shape of elastomer mushroom-like fibers for high and robust adhesion", en, *Beilstein Journal of Nanotechnology*, vol. 5, no. 1, pp. 630–638, May 2014, Publisher: Beilstein-Institut.

[112] A. Parness, D. Soto, N. Esparza, N. Gravish, M. Wilkinson, K. Autumn, and M. Cutkosky, "A microfabricated wedge-shaped adhesive array displaying gecko-like dynamic adhesion, directionality and long lifetime", *Journal of The Royal Society Interface*, vol. 6, no. 41, pp. 1223–1232, Dec. 2009, Publisher: Royal Society.

[113] B. Yurdumakan, N. R. Raravikar, P. M. Ajayan, and A. Dhinojwala, "Synthetic gecko foot-hairs from multiwalled carbon nanotubes", en, *Chemical Communications*, vol. 0, no. 30, pp. 3799–3801, 2005, Publisher: Royal Society of Chemistry.

[114] K. Autumn, A. Dittmore, D. Santos, M. Spenko, and M. Cutkosky, "Frictional adhesion: A new angle on gecko attachment", en, *Journal of Experimental Biology*, vol. 209, no. 18, pp. 3569–3579, Sep. 2006, Publisher: The Company of Biologists Ltd Section: Research Article.

[115] P. Day, E. Eason, N. Esparza, D. Christensen, and M. Cutkosky, "Microwedge machining for the manufacture of directional dry adhesives", *Journal of Micro and Nano-Manufacturing*, vol. 1, p. 011 001, Mar. 2013.

[116] G. Monkman, "An Analysis of Astrictive Prehension", *The International Journal of Robotics Research*, vol. 16, no. 1, pp. 1–10, Feb. 1997, Publisher: SAGE Publications Ltd STM.

[117] D. Mellinger, M. Shomin, and V. Kumar, "Control of Quadrotors for Robust Perching and Landing", en, Philadelphia, PA: American Helicopter Society International, Oct. 2010, p. 8.

[118] A. Fukushima and Y. Kawaguchi, "Insect leg inspired friction attachment for miniature quadcopter", in *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, ser. ACE '15, Iskandar, Malaysia: Association for Computing Machinery, Nov. 2015, pp. 1–4.

[119] M. Martone, C. Pavlov, A. Zeloof, V. Bahl, and A. M. Johnson, "Enhancing the Vertical Mobility of a Robot Hexapod Using Microspines", *arXiv:1906.04811 [cs]*, Sep. 2019, arXiv: 1906.04811.

[120] A. Parness, T. Evans, W. Raff, J. King, K. Carpenter, A. Willig, J. Grimes-York, A. Berg, E. Fouad, and N. Wiltsie, "Maturing Microspine Grippers for Space Applications through Test Campaigns", en, in *AIAA SPACE and Astronautics Forum and Exposition*, Orlando, FL: American Institute of Aeronautics and Astronautics, Sep. 2017.

[121] K. Hang, X. Lyu, H. Song, J. A. Stork, A. M. Dollar, D. Kragic, and F. Zhang, "Perching and resting—A paradigm for UAV maneuvering with modularized landing gears", en, *Science Robotics*, vol. 4, no. 28, Mar. 2019, Publisher: Science Robotics Section: Research Article.

[122] M. Kovač, J. Germann, C. Hürzeler, R. Y. Siegwart, and D. Floreano, "A perching mechanism for micro aerial vehicles", en, *Journal of Micro-Nano Mechatronics*, vol. 5, no. 3, pp. 77–91, Dec. 2009.

[123] H. W. Wopereis, T. D. van der Molen, T. H. Post, S. Stramigioli, and M. Fumagalli, "Mechanism for perching on smooth surfaces using aerial impacts", in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2016, pp. 154–159.

[124] T. Lee, M. Leok, and N. H. McClamroch, "Control of Complex Maneuvers for a Quadrotor UAV using Geometric Methods on SE(3)", *arXiv:1003.2005 [cs, math]*, Mar. 2010, arXiv: 1003.2005 version: 1.

[125] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear Quadrocopter Attitude Control", en, *ETH Zurich Research Collection*, p. 21, 2013.

[126] H. Jiang, M. T. Pope, E. W. Hawkes, D. L. Christensen, M. A. Estrada, A. Parlier, R. Tran, and M. R. Cutkosky, "Modeling the dynamics of perching with opposed-grip mechanisms", in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 1050-4729, May 2014, pp. 3102–3108.

[127] H. Jiang, "Surface Grasping With Bio-Inspired Opposed Adhesion", Ph.D. Dissertation, Stanford University, Dec. 2017.

[128] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing", en, *arXiv:1801.09847*, p. 6, 2018.

[129] W. Dong, J. Park, Y. Yang, and M. Kaess, "GPU Accelerated Robust Scene Reconstruction", en, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China: IEEE, Nov. 2019, pp. 7863–7870.

[130] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: An Open-Source Mesh Processing Tool", en, Salerno, Italy, 2008, p. 8.

[131] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics", en, in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, Not Known: ACM Press, 1997, pp. 209–216.

[132] M. Ben-Chen, *CS468 Lector 08 - Simplification*, Course Lecture, Stanford University, 2010.

[133] D. Knuth, "The Art of Computer Programming: Sorting and Searching", in *The art of computer programming, volume 3 | Guide books*, 2nd Edition, vol. 3, Redwood City, Ca, USA: Addison Wesley Longman Publishing Co., Inc., Jan. 1998.

[134] H. Park, S. Lim, J. Trinder, and R. Turner, "Voxel-based volume modelling of individual trees using terrestrial laser scanners", Jan. 2010.

[135] R. Brown, "Building a Balanced k-d Tree in O(kn log n) Time (JCGT)", *The Journal of Computer Graphics Techniques*, vol. 4, no. 1, pp. 50–68, 2015.

[136] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection", en, in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, Not Known: ACM Press, 1996, pp. 171–180.

[137] H. König and T. Strothotte, "Fast Collision Detection for Haptic Displays Using Polygonal Models.", Jan. 2002, pp. 289–300.

[138] C.-H. Teh and R. Chin, "On the detection of dominant points on digital curves", en, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 859–872, Aug. 1989.

[139] J. Kennedy and R. Eberhart, "Particle swarm optimization", in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Nov. 1995, 1942–1948 vol.4.

[140] J. Toner and Y. Tu, "Flocks, herds, and schools: A quantitative theory of flocking", *Physical Review E*, vol. 58, no. 4, pp. 4828–4858, Oct. 1998, Publisher: American Physical Society.

[141] R. Mendes, "Population topologies and their influence . . .", Doctoral Thesis, Universidade do Minho, 2004.

[142] L. J. Miranda, "PySwarms: A research toolkit for Particle Swarm Optimization in Python", en, *Journal of Open Source Software*, vol. 3, no. 21, p. 433, Jan. 2018.

[143] A. B. Röhler and S. Chen, "An Analysis of Sub-swarms in Multi-swarm Systems", en, in *AI 2011: Advances in Artificial Intelligence*, D. Wang and M. Reynolds, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2011, pp. 271–280.

[144] G. I. Evers and M. Ben Ghalia, "Regrouping particle swarm optimization: A new global optimization algorithm with improved performance consistency across benchmarks", en, in *2009 IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, TX, USA: IEEE, Oct. 2009, pp. 3901–3908.

[145] D. Bratton and T. Blackwell, "A Simplified Recombinant PSO", en, *Journal of Artificial Evolution and Applications*, vol. 2008, pp. 1–10, 2008.

[146] M. Taherkhani and R. Safabakhsh, "A novel stability-based adaptive inertia weight for particle swarm optimization", en, *Applied Soft Computing*, vol. 38, pp. 281–295, Jan. 2016.

[147] M. Musy, G. Dalmasso, and B. Sullivan, *Vedo, a python module for scientific visualization and analysis of 3D objects and point clouds based on VTK (Visualization Toolkit)*, Feb. 2019.

[148] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: An open-source Robot Operating System", en, in *ICRA*, 2009.

[149] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator", in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, 2149–2154 vol.3.

[150] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels", en, in *2013 IEEE International Conference on Computer Vision*, Sydney, Australia: IEEE, Dec. 2013, pp. 1625–1632.

[151] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets", *The International Journal of Robotics Research*, vol. 35, Jan. 2016.

[152] T. Schneider, M. Dymczyk, M. Fehr, K. Egger, S. Lynen, and I. Gilitschen, “Maplab: An Open Framework for Research in Visual-inertial Mapping and Localization”, *IEEE Robotics and Automation Letters Letters*, 2018, original-date: 2017-11-27T21:58:23Z.

[153] Q. Zhou, “Pushing the Limits of Additive Fabrication Technologies”, en, PhD thesis, New York University, May 2016.

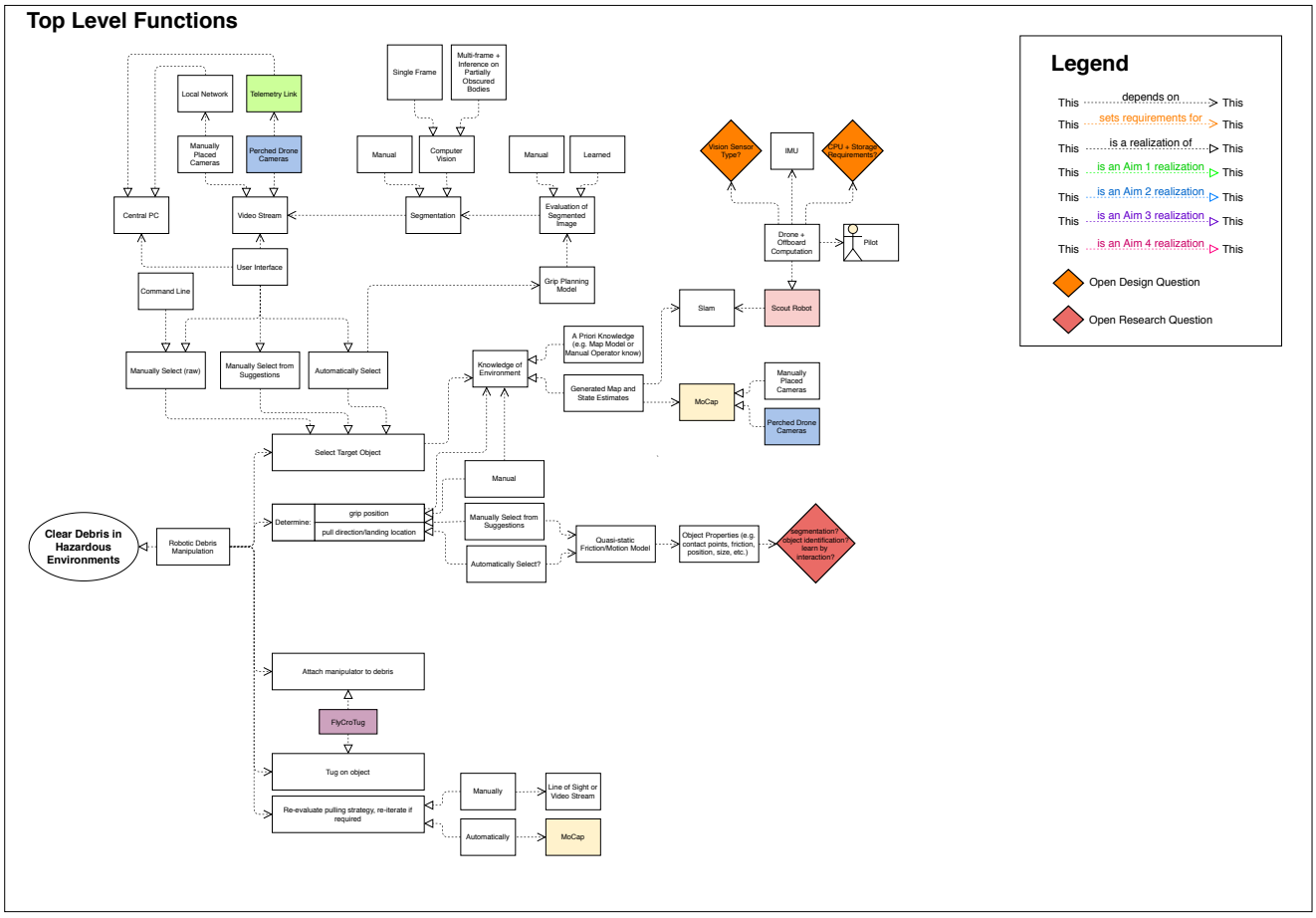
[154] E. Di Mario, I. Navarro, and A. Martinoli, “A distributed noise-resistant Particle Swarm Optimization algorithm for high-dimensional multi-robot learning”, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 1050-4729, May 2015, pp. 5970–5976.

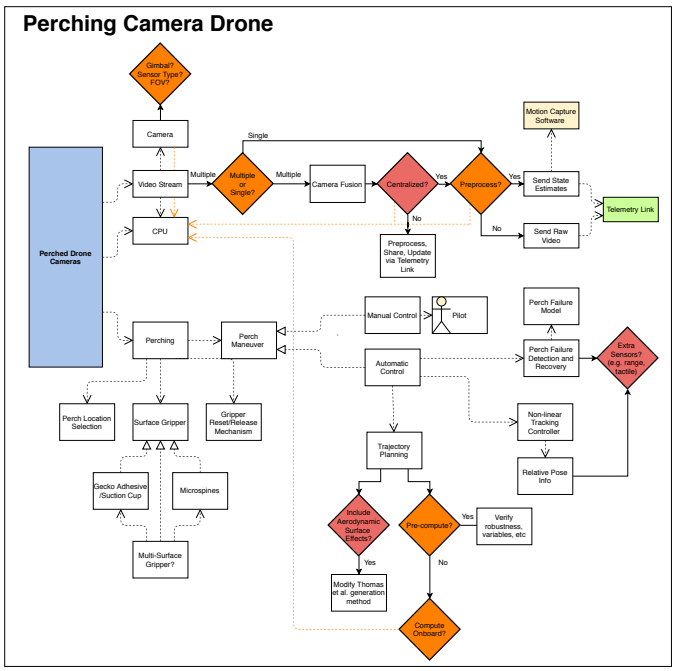
APPENDIX

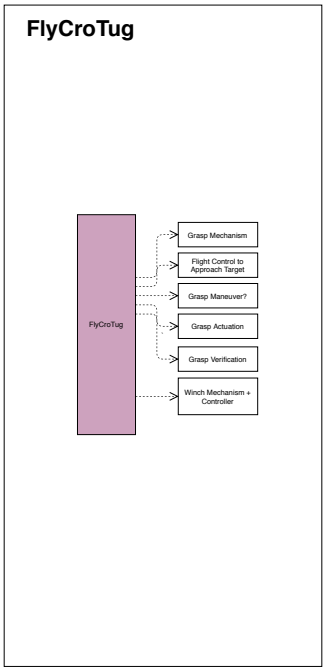
9 SYSTEM REPOSITORY

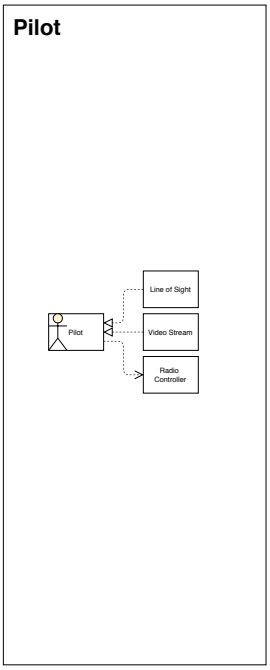
This work would certainly not be possible without the hard work and generosity of several groups and individuals who made their own works open source. This work will likewise be released as open source. Source code for all modules discussed in this project will be available at: `https://github.com/shonigmann/whereToPerch` following the presentation of this work.

A FlyCroTug Functional System Diagram









						RGB/Mono			Depth					Size (mm)					
Manufacturer	Model	Price	Camera Type	Range (m)	Active Illumination?	FOV (°)	Resolution	# of pixels (MP)	FOV (°)	Resolution	# of pixels (MP)	Frame Rate (fps)	Mass (g)	Length (fwd-bck)	Width (left-right)	Height (up-down)	Power Supply	Power	Interface
CamBoard	pico Flexx	$350	Time of Flight (depth only)	0.1 - 4 m	IR				62○(H) × 45○(V)	320 x 240		45	8	7.35	68	17			USB 3
E-Con Systems	Tara USB Camera		Stereo	.5 m - 3 m		752 x 480				752 x 480		60	28.5					1.15 W	USB 3
duo3d	DUO MC	$612	Stereo	0.23 - 2.5m	no	165	752 × 480 pixel	0.36				45 (up to 320)	30	14.7	57	30.5	USB	2.5W	MicroUSB
Occipital Inc	Structure Core	$399	Stereo-IR	0.4m - 5 m	yes	85(D)	640 × 480 pixel	0.31	59○(H) × 46○(V)	1280 × 960 pixel	1.23	54	52.5	24	109	18	USB C	3.1 W	USB 3 (Type C)
Stereolabs	Mini	$396	Stereo	0.1m - 15m		90(H) x 60(V) x 100 (D)	1920 × 1080 pixel	2.07				30 (up to 100fps)	62.9	30.5	124.5	26.5	USB C	1.9 W	USB 3 (Type C)
Occipital Inc	Structure IO Mk II	$335	Stereo	0.4m - 5 m					59○(H) × 46○(V)	1280 × 960 pixel	1.23	54	65	24	109	18			USB 2
Intel	D435	$179	Active IR Stereo	0.1 m - 10 m	IR	69.4○(H) × 42.5○(V)) × 77○(D)	1920 × 1080 pixel	2.1	87○(H) × 58○ (V)) × 95○(D)	1280 × 720 pixel (down to 640x480)	0.9	30, 60, 90 fps; 30fps for RGB	72	25	90	25	USB C	3.5W	
Intel	D435i	$199	Active IR Stereo + IMU	0.1 m - 10 m	IR	69.4○(H) × 42.5○(V)) × 77○(D)	1920 × 1080 pixel	2.1	87○(H) × 58○ (V)) × 95○(D)	1280 × 720 pixel (down to 640x480)	0.9	30, 60, 90 fps; 30fps for RGB	72	25	90	25	USB C	3.5W	
MYNT	MYNT EYE	$260	Stereo	0.5 - 20m		120(H) x 75(V)	752 × 480 pixel	0.36				60	85	31	165	30	USB	3.5W	USB3
Intel	L515	$349	LIDAR + RGB-D	0.25m - 9 m @ 95% reflectivity, 0.25 - 3.9 m @ 15% reflectivity	For depth, not color	Presumably also 70○ (H) × 55○(V)	1920 × 1080 pixel	2.07	70○(H) × 55○(V)	1024x768 or 640x480	0.79	30.0	95	26	61	61	USB 3.1	3.3W max	USB 3.1
LeddarTech	Vu8		LIDAR	185m (down to 6m for wide angle)	active depth	from 16 x 2 to 99 x 3	8 segments, no listed angular precision					100 Hz	120	44	36	65	12V	2W	SPI, USB
CamBoard	pico monstar	$1,700	Time of Flight (depth only)	0.5 - 6 m	IR				100○(H) × 85○ (V)	352 x 287		60	142						USB 3
Stereolabs	ZED 2		Stereo	0.3m - 20m		110(H), 70(V), 120(D) (max)	1920 × 1080 pixel	2.07				30 (up to 100fps)	166	30	175	33	USB 3		
FRAMOS	D435e		Stereo	0.2m - 10m		86(H) x 57(V)	1920 × 1080 pixel	2.07	86(H) x 57(V)	1280 × 720 pixel	0.92	30	250	38	100	47	12-24V		Ethernet
ifm	O3X100		Global Shutter TOF	.05 m - 3.0 m	active depth	60○(H) × 45○(V)	224 × 172 pixels	0.04	60○(H) × 45○(V)			20	266	21	43.5	80	20-28V	3.7 W	Ethernet
Intel	SR300		Coded Light	0.3 m–2 m	IR	41.5○(H) × 68○(V) × 75.2○(D)	1920 × 1080 pixel	2.1	55○(H) × 71.55○ (V) × 88○(D)	640 × 480 pixel	0.3	30, 60 fps	300		110	12.6	USB 3.0	1.8 W	
Orbbec	Astra S	$149	Structured Light	0.4 m–2 m	active depth	60○(H) × 49.5○(V) × 73○(D)	640 × 480 pixel	0.3	60○(H) × 49.5○ (V) × 73○(D)	640 × 480 pixel	0.3	30 fps	300	40	165	30	USB 2.0	2.4 W	
Microsoft	Azure Kinect DK	$399	RGB-D	0.5-3.8 m (narrow) or 0.25-2.2 m (wide)	IR	90○(H) × 59○(V)	3840 × 2160 (down to 1280 x 720)	8.3	75°x65° (Narrow) or 120°x120° (Wide)	1024 × 1024 (down to 320 x288)	1.0	0, 5, 15, 30 fps	400	103	126	39	USB-C	5.9W	USB-C
Basier	ToF640	$2,340	GS + D	0 - 13m	active depth	57(H) x 43(V)	640 × 480 pixel	0.31	57(H) x 43(V)	640 × 480 pixel	0.31	20	400	61	142	68	24V	15W	GigE
Intel	ZR300		Active Stereo	0.55 m - 2.8 m		41.5○(H) × 68○(V) × 75○(D)	1920 × 1080 pixel	2.1	59○(H) × 46○(V) × 70○(D)	480 × 360 pixel	0.2	30, 60 fps	410	6.4	118	10.5	USB 3.0	3.8 W	
Hitachi	HLS-LFOM	$1,200	LIDAR						76○(H) × 60○(V)			30	540	69	138	69			
Velodyne	Puck Lite	$4,000	LIDAR	0-100m		360(H) x 30(V)	3600 x 15					Rotation rate: 5 - 20 rps	590	103	103	72	9-18V	8W	Ethernet
Basler	Blaze		Global Shutter TOF	0 - 10m		60(H) x 45(V)	640 × 480 pixel	0.31	60(H) x 45(V)	640 × 480 pixel	0.31	30	650				24V	24W	
Livox	Mid 40	$599	LIDAR	>90m		38.4 deg (circular)	~340 x 340					100,000 pts/s	760	69	88	76	10-16V	10W	Ethernet
Livox	Horizon	$1,499	LIDAR	>90m		81.7(H) x 25.1(V)	~1600 x 500					240,000 pts/s	760	115	77	84	10-15V	12W	Ethernet
Velodyne	Velabit	$100	LIDAR	100m		60(H) x 10(V)						240,000 pts/s	760	61	61	35	10-15V	12W	Ethernet
ifm	O3D303	$1,100	Global Shutter TOF	0.3 m - 8 m		60○(H) × 45○(V)	352 × 264 (down to 176 x 132)	0.1	60○(H) × 45○(V)	352 × 264 (down to 176 x 132)	0.1	25 fps	810	95	65	72	20-28V	10 W	Ethernet
Roboception	rc_visard	$2,700	Stereo	0.5m - 3m			1280 × 960 pixel	1.23				0.8 (up to 25 Hz at 214x160 px)	850				18-30V		
Microsoft	Kinect II	Used, ~$50	RGB-D	0.5 m–4.5 m		70.6○(H) × 60○(V)	1920 × 1080 pixel	2.1	70.6○(H) × 60○ (V)	512 × 424 pixel	0.2	30 fps	966	66	255	67	power supply + USB 3.0	15 W	USB 3.0
Carnegie Robotics	MultiSense S7	$12,000	Stereo	0.4 - inf*		80(H) x 45(V)	2048 × 1088 pixel	2.23				15	1200						
Intel	R200		RGB-D	0.5 m - 3.5 m					59○(H) × 46○(V) × 70○(D)	480 × 360 pixel	0.2	60		3.8	101.6	9.6	USB 3.0	1.7 W	
Intel	F200	Discontinued	Structured Light	0.5 m - 3.5 m					73○(H) × 59○(V) × 90○(D)			30		3.8	110	12.5	USB 3.0		
Ensenso	N35	$20,000	Structured Light	0.47m - 1.1 m						1280 × 1024 pixel	1.31								Ethernet
Nerian Vision Technologies	Karmin 2	$1,500	Stereo	0.7m -		1600 x 1200				1600 x 1200		10							Ethernet
Argos 3D	P100		Time of Flight (depth only)						90○(H)	160 x 120		160							

C Quadric Edge Collapse Decimation

As a reference, this section briefly describes the quadric edge collapse decimation algorithm. For more information, please see the original publication by Garland et al. [131].

Quadric edge collapse decimation is a mesh simplification algorithm where edges are iteratively removed from a mesh (and points merged), in such a way that the quadric error of each simplification is minimized. Error quadrics are approximated for each vertex in the mesh using the following method.

First, each face connected to a given vertex is expressed using a plane equation of the form $ax + by + cz + d = 0$. This plane equation is expressed as the vector $\mathbf{p} = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$. The error associated with a given vertex, $\mathbf{v}$, can then be expressed as the sum of the squared distances to each plane.

$$\Delta(\mathbf{v}) = \sum_{\mathbf{p} \in \text{planes}(v)} (\mathbf{p}^T \mathbf{v})^2 \tag{25}$$

which can be re-expressed in quadratic form as:

$$\Delta(\mathbf{v}) = \sum_{\mathbf{p} \in \text{planes}(v)} (\mathbf{v}^T \mathbf{p})(\mathbf{p}^T \mathbf{v}) \tag{26}$$

$$\Delta(\mathbf{v}) = \sum_{\mathbf{p} \in \text{planes}(v)} \mathbf{v}^T (\mathbf{p}\mathbf{p}^T) \mathbf{p} \tag{27}$$

$$\Delta(\mathbf{v}) = \mathbf{v}^T \Big(\sum_{\mathbf{p} \in \text{planes}(v)} \mathbf{p}\mathbf{p}^T \Big) \mathbf{v} \tag{28}$$

The quadric error for a point, $\mathbf{v}$, can then be denoted by:

$$\mathbf{Q} = \sum_{\mathbf{p} \in \text{planes}(v)} (\mathbf{K}_p) = \sum_{\mathbf{p} \in \text{planes}(v)} (\mathbf{p}\mathbf{p}^T) = \sum_{\mathbf{p} \in \text{planes}(v)} \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{29}$$

Note that $\mathbf{K}_p$ is symmetric, and can thus be compactly represented in memory.

Originally, Q is a zero matrix for each point, as the original points intersect their adjacent planes perfectly. As edges are collapsed and vertices are merged, the error becomes non-zero.

Because $\mathbf{Q}_i$ is defined as a summation, the quadric error of the two vertices associated

with an edge contraction can also be defined using addition to be:

$$\bar{\mathbf{Q}} = \mathbf{Q}_1 + \mathbf{Q}_2 \tag{30}$$

The optimal vertex location, $\bar{\mathbf{v}}$ for a given edge contraction is defined as the vertex which minimizes $\Delta(\bar{\mathbf{v}})$; as Equation 25 is quadratic, $\bar{\mathbf{v}}$ can be solved explicitly by setting the partial derivatives of $\bar{\mathbf{Q}}$ to 0.

The minimum quadric error associated with every edge is computed and stored in a heap. The quadric decimation algorithm then iteratively selects and applies the edge collapse with the minimum $\Delta(\bar{\mathbf{v}})$, and updates the quadric error of the affected vertices. The heap is updated with the new edge errors and the process is repeated until the target number of edges or faces is met.

D Additional Results

Additional results which could not be fit into the main body of the report are included here.

D.1 Convergence Rate

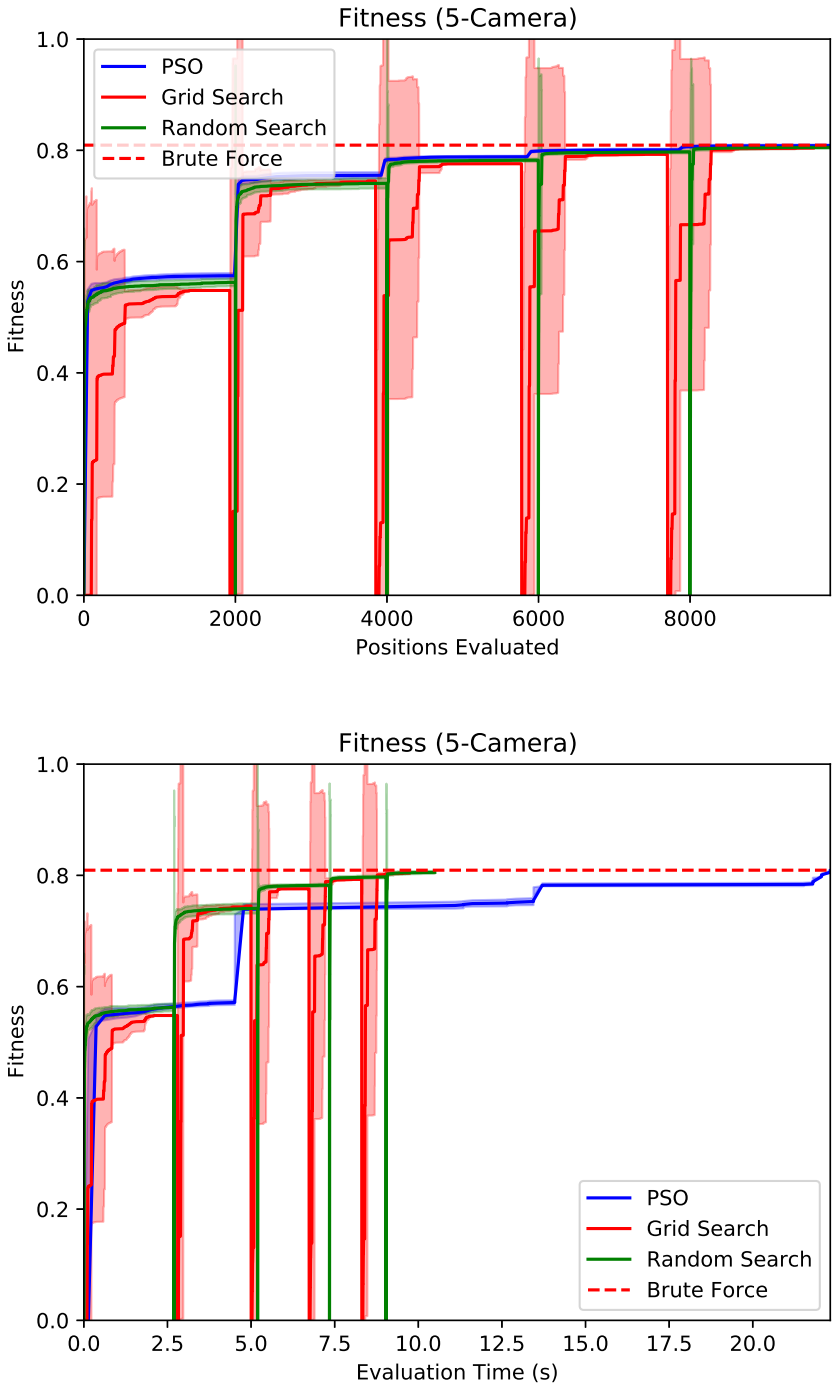


Figure 60: Apartment environment single camera search convergence comparison, comparing fitness vs points searched (top) and search time with automatic convergence detection enabled (bottom). The lines represent the mean fitness over 30 trials, and the filled region represents the $\pm\sigma$ bounds.

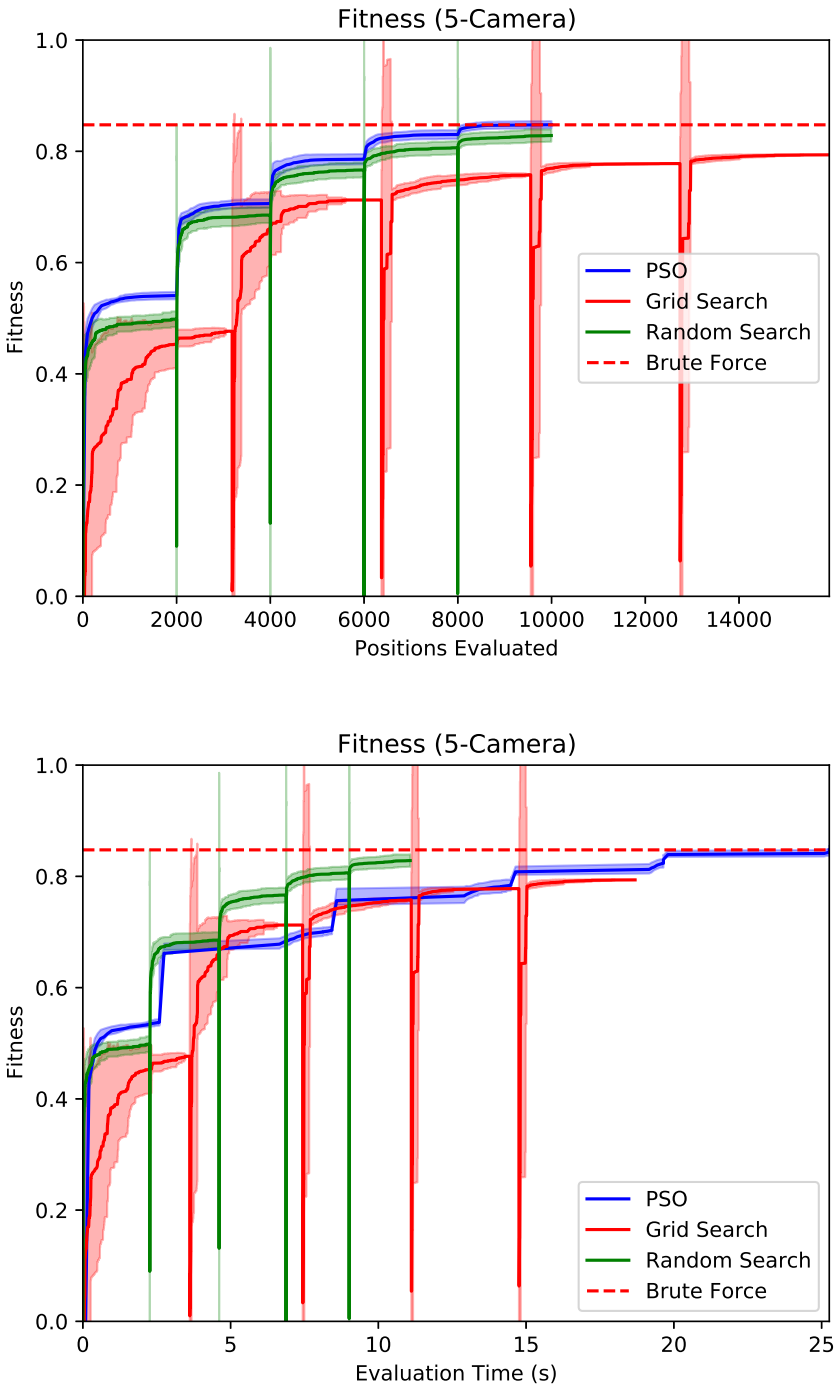


Figure 61: Home environment single camera search convergence comparison, comparing fitness vs points searched (top) and search time with automatic convergence detection enabled (bottom). The lines represent the mean fitness over 30 trials, and the filled region represents the $\pm\sigma$ bounds.

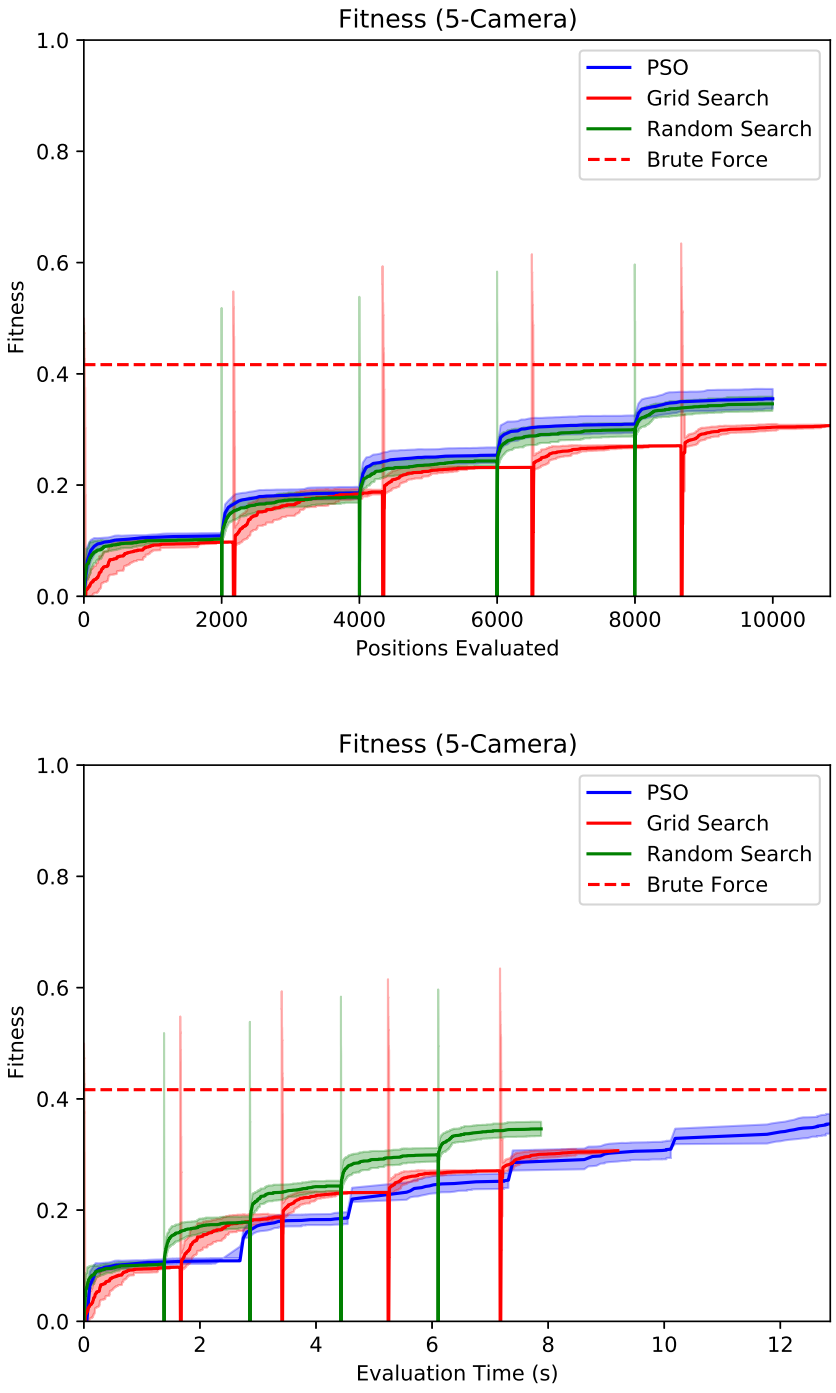


Figure 62: Machine Hall environment single camera search convergence comparison, comparing fitness vs points searched (top) and search time with automatic convergence detection enabled (bottom). The lines represent the mean fitness over 30 trials, and the filled region represents the $\pm\sigma$ bounds.

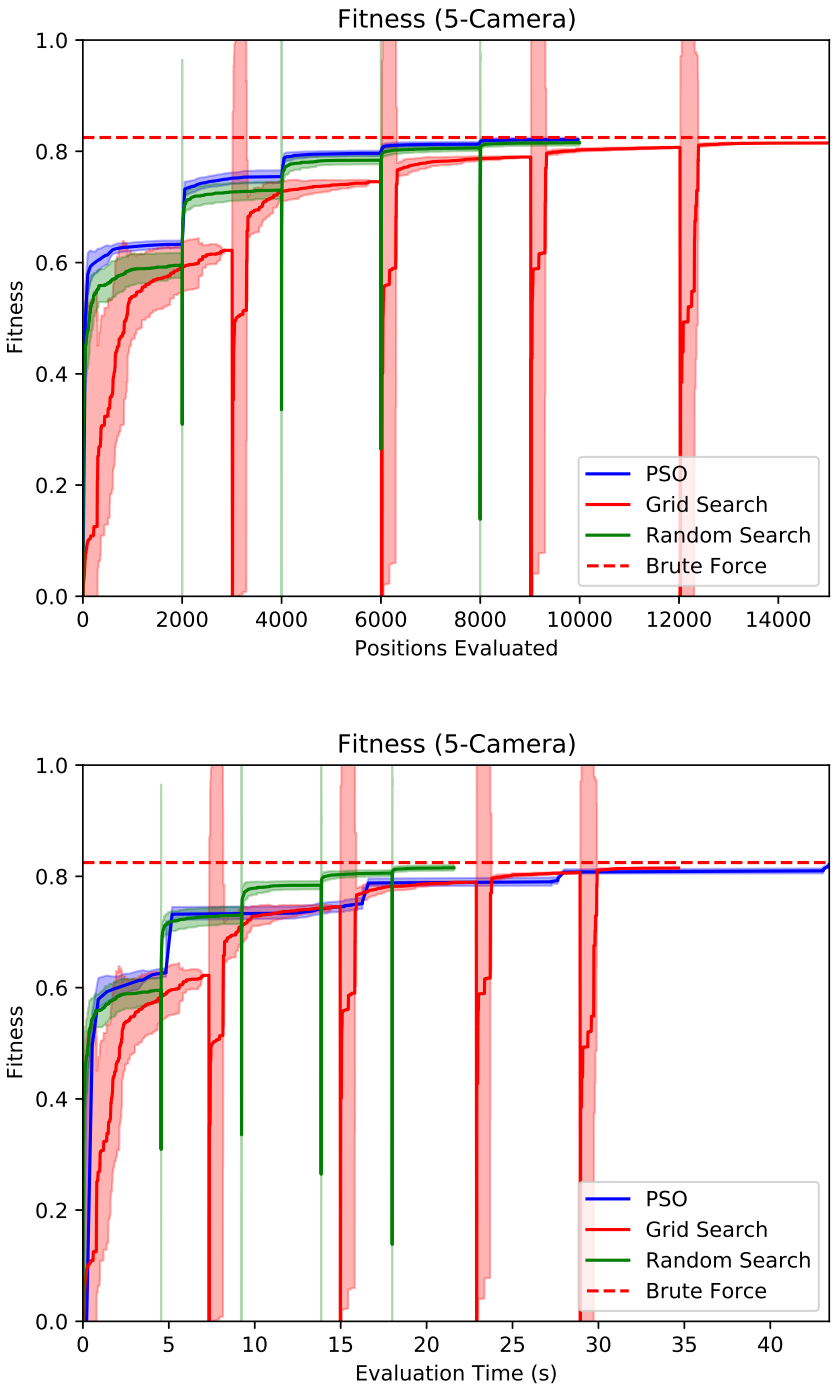
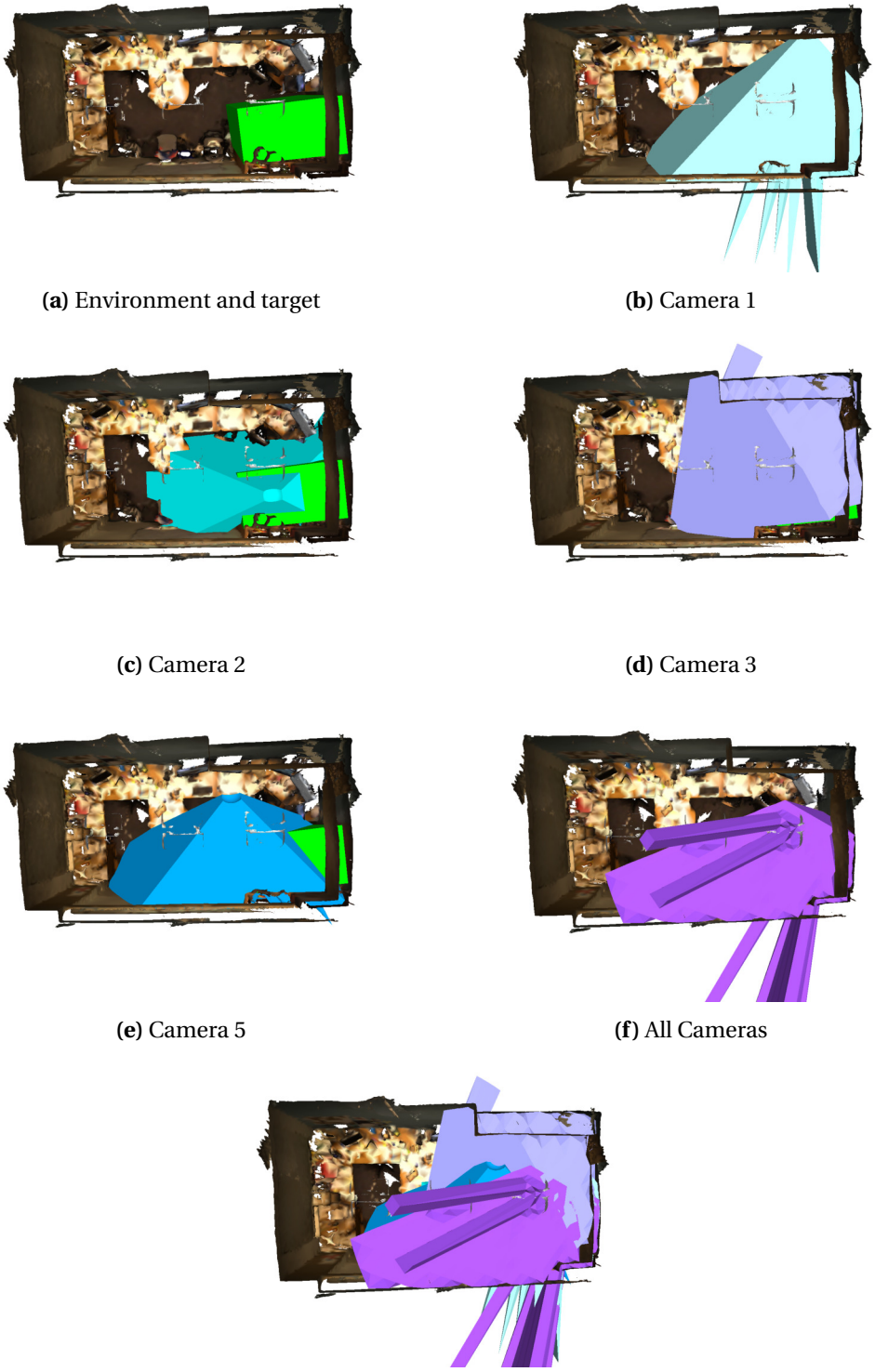


Figure 63: Office environment single camera search convergence comparison, comparing fitness vs points searched (top) and search time with automatic convergence detection enabled (bottom). The lines represent the mean fitness over 30 trials, and the filled region represents the $\pm\sigma$ bounds.

D.2 Camera Placement Visualizations



(a) Environment and target

(b) Camera 1

(c) Camera 2

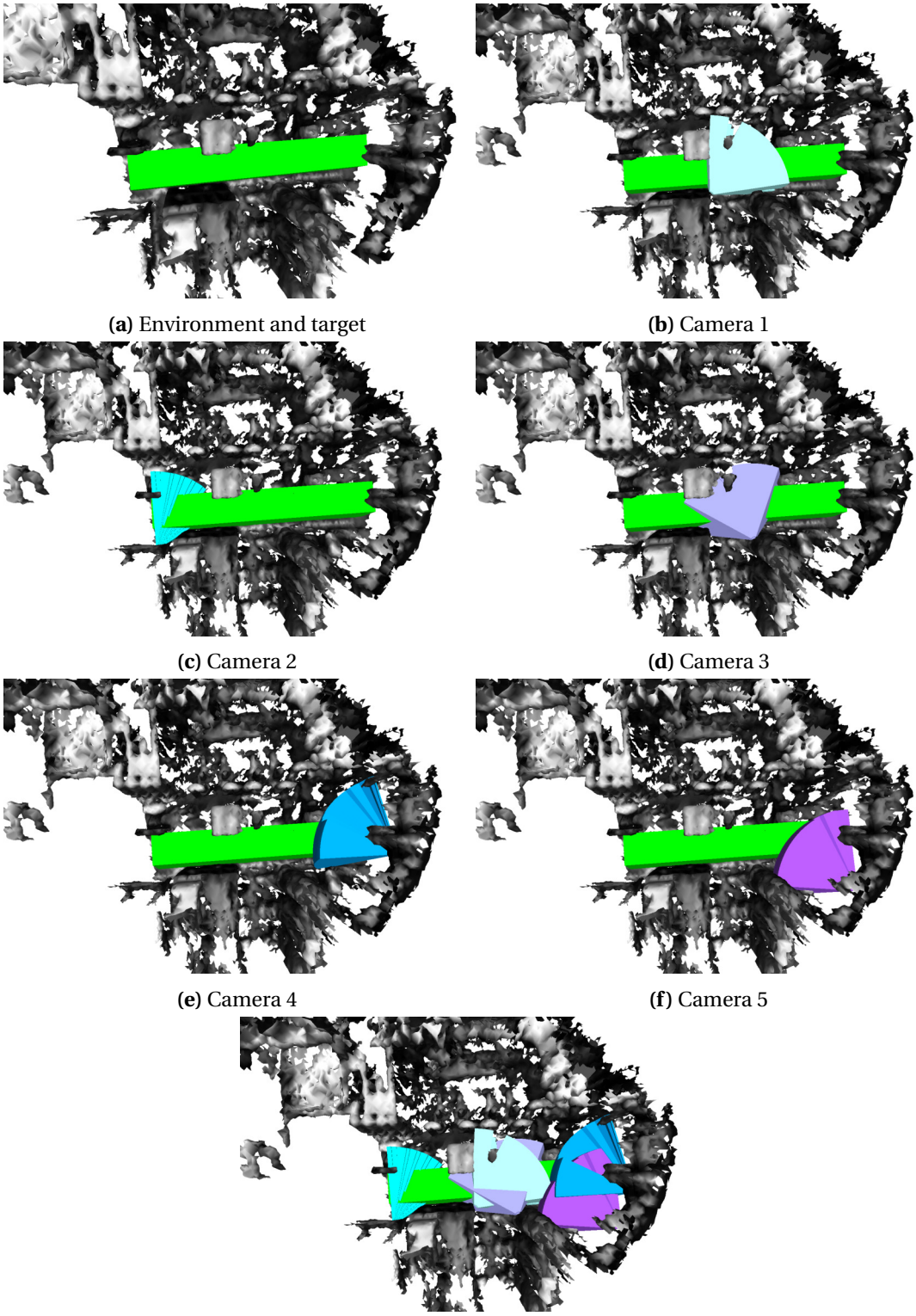
(d) Camera 3

(e) Camera 5

(f) All Cameras

(g) Camera 4

Figure 64: Camera placement results using greedy PSO in the Office environment with a central box shaped target volume (green). Camera fields of view are shown in different colors and are added incrementally to demonstrate the variety of perspectives achieved.



(a) Environment and target

(b) Camera 1

(c) Camera 2

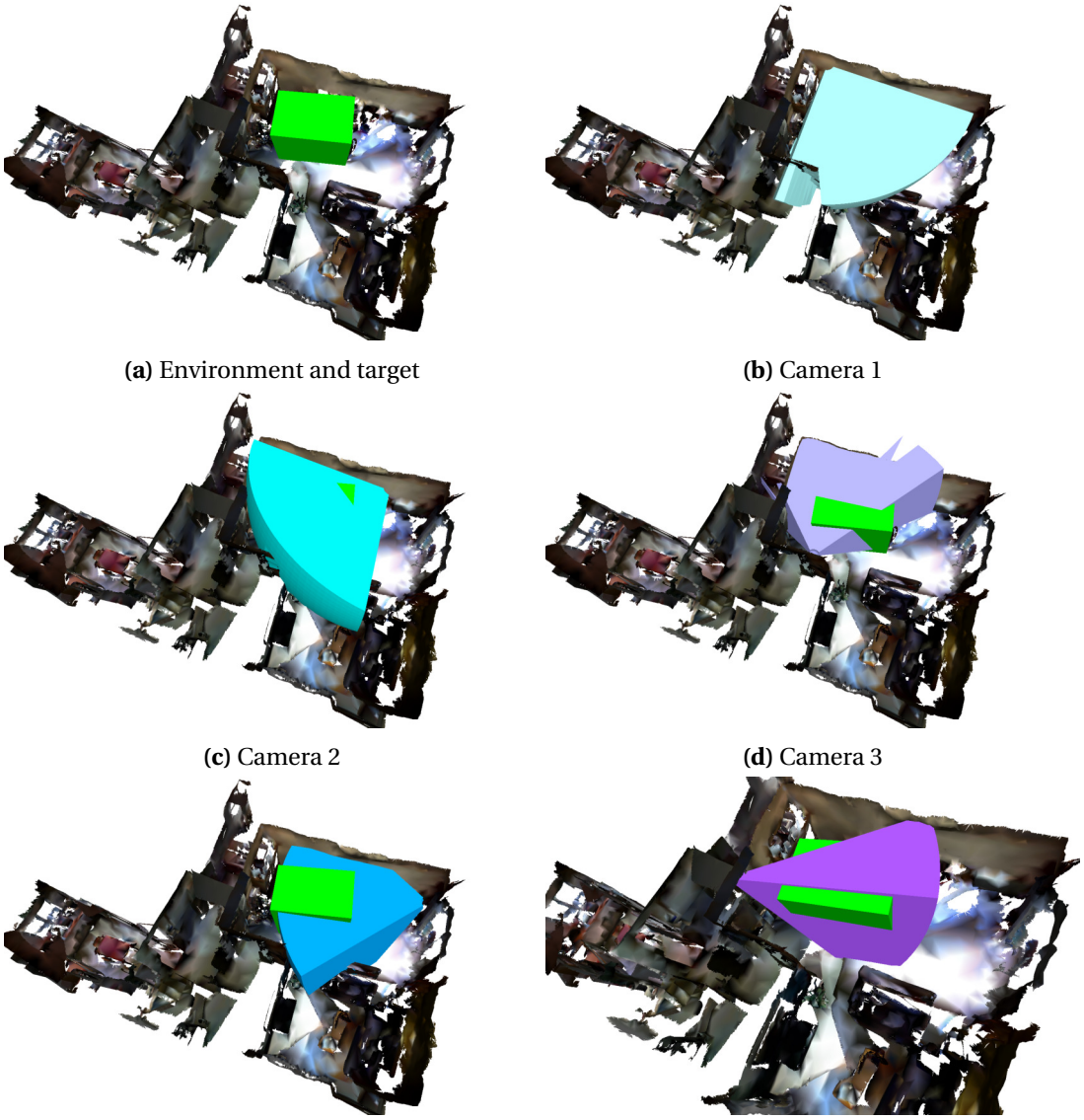
(d) Camera 3

(e) Camera 4

(f) Camera 5

(g) All Cameras

Figure 65: Camera placement results using greedy PSO in the Machine Hall environment with a central box shaped target volume (green). Camera fields of view are shown in different colors and are added incrementally to demonstrate the variety of perspectives achieved.



(a) Environment and target

(b) Camera 1

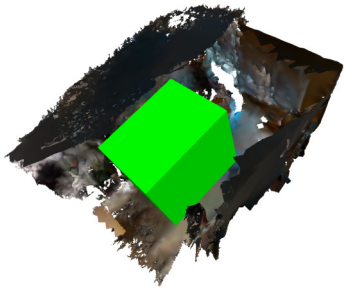
(c) Camera 2

(d) Camera 3

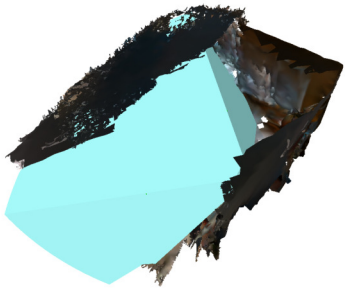
(e) Camera 4

(f) All Cameras

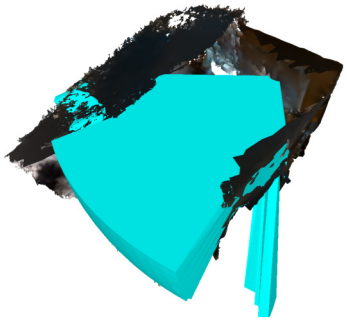
Figure 66: Camera placement results using greedy PSO in the Home environment with a central box shaped target volume (green). Camera fields of view are shown in different colors and are added incrementally to demonstrate the variety of perspectives achieved.



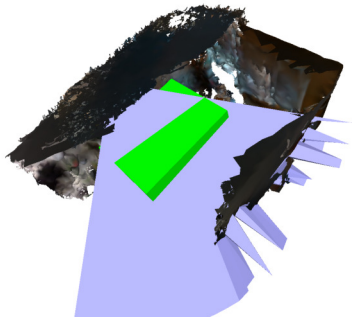
(a) Environment and target



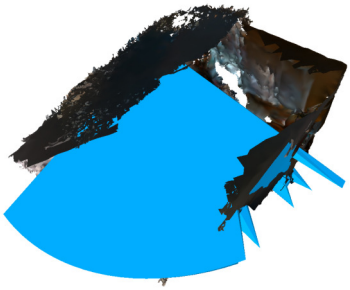
(b) Camera 1



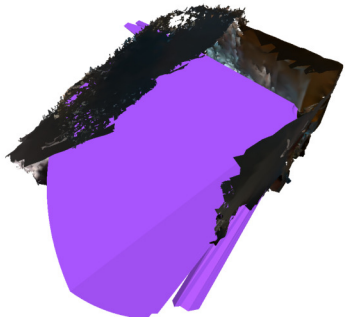
(c) Camera 2



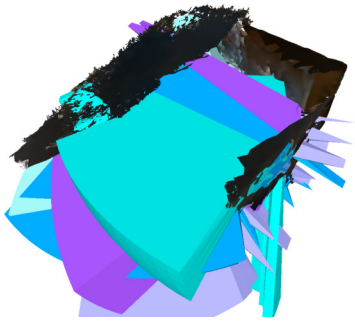
(d) Camera 3



(e) Camera 4



(f) Camera 5



(g) All Cameras

Figure 67: Camera placement results using greedy PSO in the Apartment environment with a central box shaped target volume (green). Camera fields of view are shown in different colors and are added incrementally to demonstrate the variety of perspectives achieved.

D.3 Perch Region Extraction and Filtering

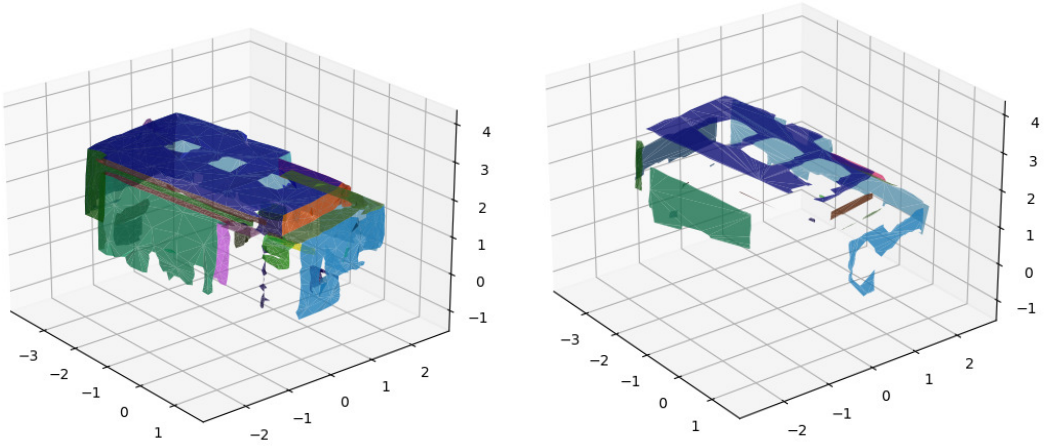


Figure 68: Office environment after segmentation (left) and after filtering (right)

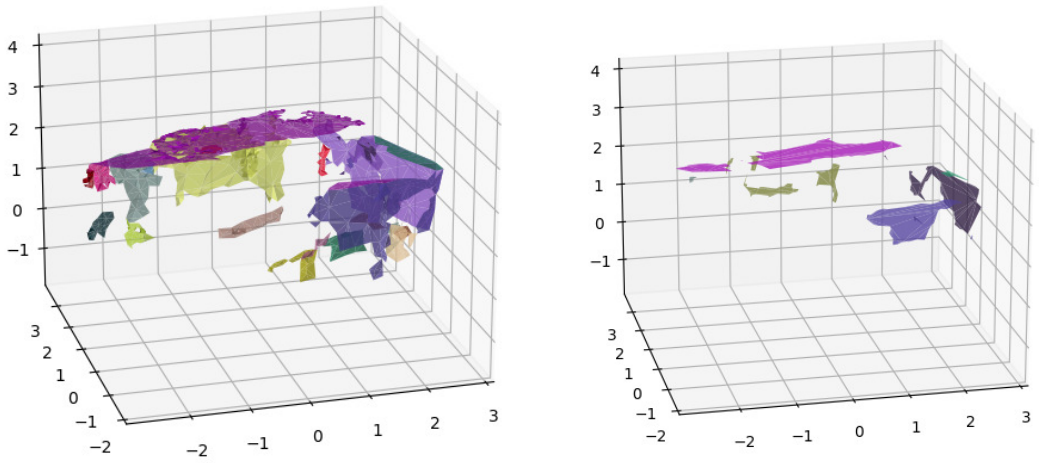


Figure 69: Apartment environment after segmentation (left) and after filtering (right)

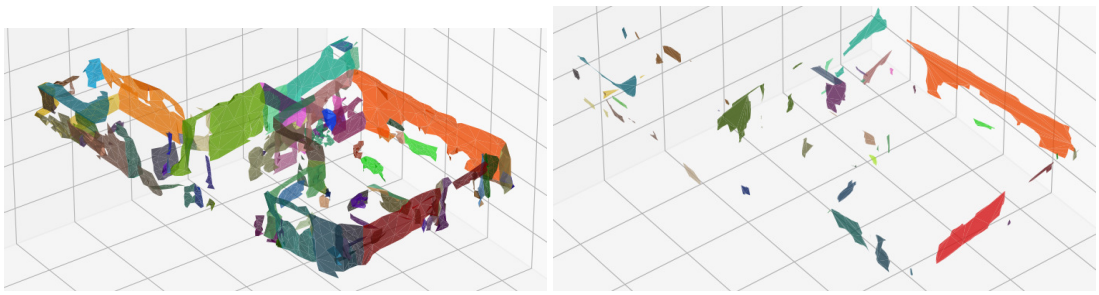


Figure 70: House environment after segmentation (left) and after filtering (right)

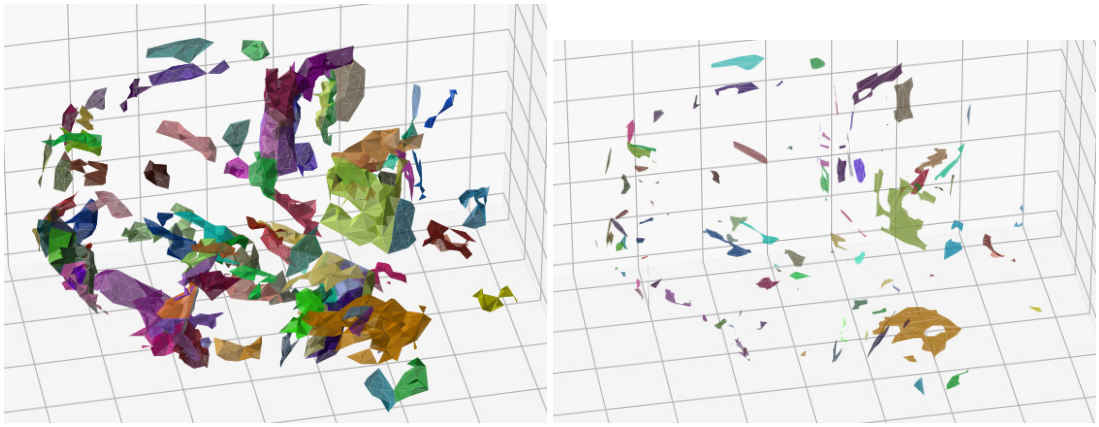


Figure 71: Machine Hall environment after segmentation (left) and after filtering (right)