Monocular Vision-Based Obstacle Detection for Unmanned Systems

by

Carlos Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Many potential indoor applications exist for autonomous vehicles, such as automated surveillance, inspection, and document delivery. A key requirement for autonomous operation is for the vehicles to be able to detect and map obstacles in order to avoid collisions. This work develops a comprehensive 3D scene reconstruction algorithm based on known vehicle motion and vision data that is specifically tailored to the indoor environment. Visible light cameras are one of the many sensors available for capturing information from the environment, and their key advantages over other sensors are that they are light weight, power efficient, cost effective, and provide abundant information about the scene. The emphasis on 3D indoor mapping enables the assumption that a large majority of the area to be mapped is comprised of planar surfaces such as floors, walls and ceilings, which can be exploited to simplify the complex task of dense reconstruction of the environment from monocular vision data.

In this thesis, the Planar Surface Reconstruction (PSR) algorithm is presented. It extracts surface information from images and combines it with 3D point estimates in order to generate a reliable and complete environment map. It was designed to be used for single cameras with the primary assumptions that the objects in the environment are flat, static and chromatically unique. The algorithm finds and tracks Scale Invariant Feature Transform (SIFT) features from a sequence of images to calculate 3D point estimates. The individual surface information is extracted using a combination of the Kuwahara filter and mean shift segmentation, which is then coupled with the 3D point estimates to fit these surfaces in the environment map. The resultant map consists of both surfaces and points that are assumed to represent obstacles in the scene. A ground vehicle platform was developed for the real-time implementation of the algorithm and experiments were done to assess the PSR algorithm. Both clean and cluttered scenarios were used to evaluate the quality of the surfaces generated from the algorithm. The clean scenario satisfies the primary assumptions underlying the PSR algorithm, and as a result produced accurate surface details of the scene, while the cluttered scenario generated lower quality, but still promising, results. The significance behind these findings is that it is shown that incorporating object surface recognition into dense 3D reconstruction can significantly improve the overall quality of the environment map.

Acknowledgements

This thesis would not have been possible without the hard work and dedication of the researchers and the open source community. I express my deepest appreciation for those who have worked and are working to improve the quality of life for others. It is always an inspiration to know that we are always looking forward to new and exciting innovations in order to benefit humanity as a whole. The motivation, insights, and support I had received from several people also played a vital role in my research effort, and their contributions deserve special mention. My sincere gratitude goes to all of them.

I would like to humbly thank my supervisor, Dr. Steven Waslander, for his excellent guidance, advice, trust, encouragement, and patience throughout my entire graduate experience. Throughout these two years, I had grown personally, professionally and academically with his careful and precise teachings and I owe him a great deal for making this experience both very enjoyable and memorable. I am very glad to have had him as my supervisor and enjoyed working with him.

I would like to thank Dr. Sanjeev Bedi and Mr. Rohan Jayasundera for being great mentors throughout my undergraduate experience. Their persistence, thoughtfulness, and valuable wisdom supported me during this time and help shaped me as a person, always pushing me to excel further in my life. I am very grateful and very indebted to them for their ongoing support.

I would like to thank Ryan, Peiyi, Yassir, Arun, P.J., Yan, Dr. Daly, and Mike for being wonderful colleagues. They have been great sources of help and motivation and also made my experience in the lab very entertaining. I would also like to thank Kandarp, Tarun, Sam, Rajnish, and Adel for their kind support. Special thanks go to Gerardo, for being my sidekick during our adventures in India and being an awesome friend who constantly pushes me to succeed.

I would like to thank my other friends: Ben, Anthony, Kai, Boyd, Kenneth, Simon, Chris, Uma, Fred, and my high school friends for their emotional support and for being great friends that I can always rely on.

Lastly, but most importantly, I express my deepest thanks to my family. I can never repay the support and love that my parents had given me throughout my life. Through my mother and father I learned that I should always put lots of effort and perseverance into everything I do. I also would like to express my deepest appreciation for my two older brothers, Bobby and Peter, for always looking after me and being wonderful role models and extraordinary friends. They have without a doubt influenced my life in many ways and helped me change for the better. This thesis would certainly not have existed without the constant encouragement, financial and moral support from my family.

Dedication

This is dedicated to my family, friends, supervisors, mentors, colleagues, the generous community of administrators, funders, researchers and programmers, people who helped shaped me into who I am today and those who worked and are working towards improving humanity as a whole.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Unmanned vehicles have continued to gain popularity over the past few decades. Their practical applications were first seriously explored in the military as combat and spy units, and later marketed as a toy for general hobbyists. With the proliferation of electronics and robotic systems, these remote controlled (RC) vehicles have become more common and less expensive. Efforts to develop a fully autonomous version of these vehicles have grown as research and intelligent systems improve over time.

RC ground and aerial vehicles are more prevalent in today's society while autonomous ground and aerial vehicles (UGVs and UAVs respectively) remain as research platforms for academia, military, and corporate institutions. Safe autonomous operation of the vehicle is essential for acceptance in the general populace and it is one of the main focuses of current research in the field. The vehicle must be able to safely navigate through known and unknown environments without the risk of damaging itself and its surroundings. Robust control systems, navigation systems (path planning, trajectory generation, localization, etc...), environment perception and mapping are all areas that help address this problem. If this goal is achieved, practical uses for UGVs and UAVs, including search and rescue missions, surveying dangerous and hazardous environments, inspection of power line faults or construction progress, and cargo delivery can be explored in the context of civilian applications. The research done is also applicable for space and underwater explorations, contributing to navigation technologies used for rovers and autonomous underwater vehicles (AUVs).

The main objective of this research is to develop an obstacle detection method for indoor surveillance for UGVs or UAVs. Vehicle limitations are considered as a constraint, as the available resources to perform obstacle detection are limited by the sensing and computing equipment of the vehicle. For UGVs, payload, size, power and cost restrictions can greatly affect sensor selection and computational resources. Dynamic constraints differ for different vehicles and should be taken into account when assessing time constraints for

detecting obstacles. Fast moving vehicles with limited steering (e.g. fixed-wing UAVs) would generally require faster obstacle detection as opposed to slow moving vehicles with a wide range of steering motion. Two kinds vehicles are considered for this work: wheeled motor vehicles and quadrotor helicopters [1] (also referred to as quadrotors), which work on a similar principle to helicopters and use four rotors to actuate vehicle motion. It is assumed that these vehicles are already capable of moving at low speeds and have moderate steering capabilities. Since this work focuses on indoor applications, the vehicles are constrained to be small and a simplification made for the scene is that it would have very few or no moving objects.

The diversity of objects found in indoor environments makes the task of defining the general features of an obstacle difficult. Many objects have different structural and material characteristics. For example, an indoor setting may contain a wide range of potential obstacles, including walls, ceilings, tables, trash bins, stairs cases, chairs, cabinets, and windows which all have different structural and reflective properties. The variety of objects makes it quite challenging to have an algorithm which encompasses all obstacles indoors. Object occlusion and clutter in the environment also increases the complexity of algorithms relying on object recognition.

The work done to address these issues is laid out in this thesis. Common sensing equipment, discussed in Section 1.1, used to perceive the environment were assessed and the camera was selected as the sensor to be used for this research. Relevant research is discussed in Section 1.2 and the research direction taken is explained in Section 1.3. The remaining chapters outline, in the following order, the background and theory necessary for the vision-based algorithm, the algorithm definition itself, the vehicle testbed developed to test the algorithm and finally experimental results and conclusions. It is assumed that the vehicle has good localization and is in environments with sufficient lighting conditions.

1.1 Sensors Used for Environment Perception

Sensors are the tool used for environmental perception by autonomous robots. A wide variety of sensors can be used, and can be categorized into two classes: active and passive. Active sensors emit some form of signal into the environment and measure the return signature. Examples include SOnic Detection And Ranging (SODAR), LIght Detection And Ranging (LIDAR), and conventional RAdio Detection And Ranging (RADAR) sensors. Passive sensors measure naturally occurring signals from the environment. Two common signals measured from the environment in the context of mobile robotic systems are heat and visible light signals, which are captured by infrared (IR) sensors and visible light cameras respectively.

Of the five sensors listed for mobile robotics applications, computer vision relying on

visible light cameras will be exclusively investigated in this work. Images of the environment captured by visible light cameras typically contain abundant information and can be used for a variety of tasks, such as object recognition and motion tracking, identifying the type of scene (inside rooms, on the staircase or in the hallways), and locating any other valuable information which helps identify vehicle threats. In addition, the video stream from the camera can be assessed for other tasks, such as facial recognition or package identification. Visible light cameras work in both indoor and outdoor settings with sufficient lighting conditions. They are also lightweight, energy efficient and inexpensive [2, 3], and are more robust against shock and noise when compared with laser scanners [4]. SODAR sensors are light weight, but they are typically sensitive to false echoes caused by specular reflections which makes these sensors difficult for multiple obstacle detection. LIDAR sensors and laser rangefinders may give both accurate and precise ranges but are comparatively more power, weight, and space consuming than passive sensors [5]. They are also planar or two-dimensional (2D) sensors, which require more time for them to obtain three-dimensional (3D) range maps [6]. The 3D range maps can also decrease in quality with increased vehicle motion. RADAR sensors can operate in most weather and can detect thin obstacles (severe , but their measurements are sparse with low scan rates. This makes them inappropriate in cluttered environments [7]. IR sensors perceive in the IR spectrum (i.e. sense heat information) and are at an advantage to visible light cameras when operating in night environments. However they are highly susceptible to varying degrees of temperature changes, therefore radiators, vents, and other heat sources and sinks can change the thermal information of the objects surrounding them. As a result, the heat images captured from the same environment can vary over time. Another disadvantage for IR sensors are that the relative IR emissivity between indoor objects at room temperature are less distinguishable than their visible light signatures.

Work done on using a combination of sensors (sensor fusion) for obstacle detection [8, 9, 10] attempts to benefit from the advantages of each individual sensor while reducing their disadvantages. While sensor fusion improves the accuracy and precision for obstacle detection comparatively with using individual sensors, the work done here seeks to minimize sensor payload and power requirements.

1.2 Related Work

Common vision-based methods can be grouped into the following categories: optical flow and flow field divergence, structure from motion, dense 3D reconstruction, and single-view and two-view methods. These methods are described in the following paragraphs.

Optical flow is defined as the distribution of apparent motion of brightness patterns (represented as motion or flow vectors) in an image, which can arise from the relative

motion of objects and the observer [11]. The distribution of flow vectors produced from optical flow can be used to detect obstacles by clustering similar motion vectors and finding discontinuities along the 2D vector field. The focus of expansion (FOE) is point which the set of flow vectors intersects and is indicative of the relative vehicle motion direction with respect to the environment. Analysing the FOE from images can be useful for determining if the vehicle is on a collision course or for returning a depth map estimate of the scene. Two popular techniques used for calculating optical flow vectors from two images (assumed to be captured at consecutive instances) are the Lucas-Kanade [12] and the Horn-Schunck [11] methods. Optical flow based methods generally face the aperture problem [13, 14, 11], where there are cases where the apparent motion of the image features or pixel brightness can not decipher to the actual motion of the object. For example if a diagonally striped paper is placed on a table and viewed with a circular mask, the stripes are seen as moving diagonally when the paper is moved horizontally and vertically along the table. Since optical flow methods rely heavily on the pixel brightness patterns, it means that images that are affected by motion blur, image noise, and local changes in intensity values from illumination affect the accuracy of the optical flow field calculated. Flow field divergence [15] extends from optical flow to combine translation and rotation information from the vehicle over time to find the presence of potential obstacles.

Structure from motion [16] is a procedure that examines motion details from images to estimate 3D structures of objects. The motion details are generally obtained from optical flow based methods or by finding and tracking image features (edges, corners, blobs, etc...) through other means. Popular edge detection methods include Canny edge detection [17] and Hough transform [18]. Some corner detection methods include [19, 20, 21], with the more popular one being the Harris corner detector [22]. Additionally, two widely used, and more elaborate, feature detection methods include Speeded Up Robust Features (SURF) [23] and Scale-Invariant Feature Transform (SIFT) [24]. Some of the common methods used for tracking these features include the Kanade-Lucas-Tomasi (KLT) feature tracker [25] and David Lowe's tracker [24] which is specific for SIFT features. Based on the apparent displacement of the features from one view to the next, calculations can be done to estimate their 3D locations [26, 27, 28]. Dense 3D reconstruction methods follows the same methodologies as the structure from motion procedure, with the idea of using many features to produce a dense 3D point cloud of the environment over time. Surface reconstruction methods [29, 30, 31, 32] can then be applied to the point cloud to produce a 3D model of the object or scene. The structure from motion procedure typically relies on accurate (and preferably large) motion details and camera pose estimates to produce an accurate 3D structure. Scenes that produce few features or little motion details would result in a poorly reconstructed 3D structure, since the quality of the 3D model produced relies on the number of feature points and their motion magnitudes. Also, a drawback for this procedure is the increased computation time associated with increasing features to track and use for 3D reconstruction. Tracking features can also be difficult in cluttered

or repeatedly textured environments, producing poorly tracked features and therefore an inaccurate map estimate.

Research is also based on the number of cameras used. Single-view (one camera or monocular vision) and two-view (two cameras or stereo vision) vision techniques are generally the main areas of focus when dealing with obstacle detection. The key difference between monocular and stereo vision is that monocular vision methods typically require relative motion between the observer and the objects while stereo vision methods do not. This is because the stereo baseline (separation distance between the two cameras) is made sufficient for triangulation of object distances. Monocular vision techniques include optical flow and structure from motion methods for finding obstacles, whereas stereo vision techniques performs stereo correspondence (matching of pixels or features) between two images from two different perspectives to form a disparity map. The disparity map can then be used for 3D reconstruction. The idea for stereoscopic analysis is to mimic human binocular vision to achieve depth perception. Stereo vision faces the correspondence problem [5, 33], which is the problem of correctly matching pixel or features between the two images, although some research was done to improve upon this problem [7, 33]. Monocular vision based methods also face this problem if the methods require features to be tracked, and it faces the problem of estimating its baseline for structure from motion based methods. A wide baseline is generally preferable in both cases for accurate 3D reconstruction.

It is noted that in static environments (environments with no moving objects), two perspective views obtained from a single camera can be treated as a stereo vision problem if the baseline of the two views is known. On the other hand, the two images obtained from stereo vision can be treated as a single camera moving from one end of the stereo baseline to the other. This connection is particularly useful for it allows monocular and stereo vision methods to be used interchangeably in static environments. Therefore many optical flow and structure from motion techniques can be used for stereopsis, and the stereo vision correspondence methods can also be used for the single camera case. The main advantage of using two cameras is that the baseline is a known priori for the stereo vision case, which removes the task of establishing a baseline (necessary for most structure from motion methods) and reducing the problem of 3D reconstruction to finding accurate correspondences between the images. A fixed and precisely measured baseline is also advantageous since poorly defined baselines negatively affect the reconstruction process. Therefore, as a result of a consistent and known baseline, the approximation of the 3D locations tend to be more accurate for stereo vision methods than their monocular vision counterparts [4]. The notable shortcomings for using stereo vision, however, are the increased weight, size, and power consumption [34].

Vision-based algorithms work on the images captured by the camera and usually depend highly on the quality of the images produced. Camera lens distortion can affect the overall quality of the images captured [28]. In general chromatic and spherical aberration from

the camera lens contribute to colour and spatial inaccuracies in the images. Warping and calibration methods [35, 36] can be used on the image to help reduce these inaccuracies. Image noise [37] is also a major factor to address, as two camera frames of the same scene captured at different instances can be subtly different when represented in an image matrix. Low-end cameras are also prone to issues pertaining to light sensitivity, low resolution and motion blur [38], which affect algorithms requiring sharp and consistent images.

The approaches used for determining obstacles indoors can be grouped into reactive, selective, or generalized. The methods used for the reactive approach places emphasis on whether the vehicle is on a collision course with the obstacles and generally provides indicative measures on a control response to avoid them. For the selective approach the focus is on finding known objects or using known environments in the scene to classify obstacles, while the generalized approach assumes there is no prior knowledge of the environment and or objects found in the environment. The following sections describe these approaches in detail and present examples of relevant research done. Although some of the research explained performs obstacle detection for outdoor scenarios, the basic principles behind these techniques can overlap to indoor situations as well.

1.2.1 Reactive Obstacle Detection

Methods primarily focused on indicating whether the vehicle is on a collision course are advantageous for vehicles travelling at high speeds (in particular, fixed-wing UAVs), where the available computation time for detecting potential collisions is very limited. As a result, the algorithm must be simple and efficient to quickly warn the vehicle of approaching obstacles so it can quickly react and avoid it. Optical flow based methods are generally favoured in this approach [4], because calculating the flow vectors are reasonably fast and can give an indication of an imminent collision with an object.

A combination of the FOE and optical flow vector patterns are used in [39, 40, 14] to infer the presence of obstacles and avoid potential collisions. Another similar approach in [41] is biologically inspired [42], where it takes the standard deviation (SD) sub-samples from a Nyquist image, resulting in a low density image which is less complex and simpler to process. These images are also mentioned to be independent of spatial and temporal variations in scene illuminations. Apart from the disadvantages explained for optical flow based methods, these methods all depend on the characteristics of the FOE flow patterns, which means that their performance can be affected in the presence of moving objects.

Feature detection and optical flow based tracking was used in [5, 43] to infer approaching obstacles from their feature expansion rates. In [3], the features (detected using the Harris corner detector) were clustered using an agglomerative clustering method to determine dangerous obstacles or areas. Another structure from motion method takes a probabilistic

approach by applying probability distributions to find the optical flow vectors on defining features [44], which reduce the affects of image blurring and noise at the cost of reduced precision of the flow vectors.

A more elaborate scheme was proposed by Hrabar et al. [45], which uses a pair of sideways-looking cameras and a front-facing stereo camera to detect and avoid obstacles while the vehicle navigates through urban canyons. The sideways-looking cameras uses optical flow to ensure the vehicle is centred, while the stereo camera finds potential obstacles in front for the vehicle to avoid. Other work introduced a new approach called expansion segmentation [46] to find regions where there is a possibility of vehicle collision and produced promising results from simulation experiments.

While these methods are designed to be efficient and fast at detecting potential obstacles or collisions, they usually suffer from decreased robustness and possible false alarms which can adversely place the vehicle in dangerous situations. The information returned from these methods are generally not detailed enough to be used for path planning algorithms, as their purpose is to avoid the most immediate threats (i.e. the very close obstacles in the way of the vehicle's current path). This may not be effective depending on the environment the vehicle is faced with, for example environments with dead ends or scattered obstacles are environments which requires more careful trajectory planning to guarantee the safety of the vehicle. The benefits of fast computation can outweigh the loss of precision and increased possibility of false alarms for certain applications.

1.2.2 Selective Obstacle Detection

In situations where the obstacles (the most abundant or dangerous obstacles) or the environment are a known priori, it is generally beneficial to develop algorithms that take advantage of this knowledge. By doing so, the research can be more focused, and produce algorithms that reliably finds obstacles based on the given information. Some additional outcomes from this situation are the consistent performance of the algorithm and reliable training data, on the condition that the operating environment does not change. For scenarios where there are several types of known obstacles, a multi-layered approach can be used to run several methods that detect each obstacle type in parallel. The result is a robust, fast and consistent algorithm. Some methods are presented here, where they utilize the properties of the known object or environment in order to find potential obstacles.

Visual cues are examined from images to detect specific obstacles in the scenario where they are a known priori. For example, detecting power lines or other thin objects, which are of importance for low-flying UAVs, were successfully detected using probabilistic occupancy mapping [47] and using edge detection [48]. However the use of 3D occupancy maps are memory intensive and edge detection does not perform well for highly cluttered images.

For detecting trees, one algorithm used a combination of edge detection and stereo ranging [49] while another used a supervised learning strategy to detect and estimate tree depths at high speeds [50]. The first algorithm relies on the vertical structure of trees for good performance while the other is dependent on the available training data. Various other algorithms for detecting specific types of obstacles were evaluated in [51] and by the Jet Propulsion Laboratory, which includes water, slopes, low-lying branches, ditches and small ground objects [49].

Obstacles can also be located or inferred if the environment is known in advance. Since UAVs and UGVs have different modes of navigation, the environments that they encounter are also different and are used in the methods to find obstacles. An example for UAVs uses sky segmentation [52] to differentiate between sky and non-sky regions, with the non-sky regions being potential obstacles. For low-flying UAVs, a proposed algorithm called sparse-edge reconstruction uses stereoscopic imaging to detect ground obstacles by piecing together incomplete pieces from sequentially capture images of the environment [34]. A high speed stereo camera is important for this algorithm as it requires little change between sequential images as well as minimal motion blur for proper edge detection. The ground plane is a useful characteristic of the environment for UGVs. One study uses odometry data of the vehicle to calculate an optical flow field of the ground plane and then compared with the image optical flow vectors to determine ground obstacles [53]. Other ground-based obstacle detection methods include [54], which introduces an approach called Appearance Based Object Detection, [2], which combines visual features and supervised learning for floor mapping, and [55], which uses fast RANdom SAmple Consensus (RANSAC) for ground plane extraction. For these methods to properly function, it is assumed that the desired feature of the scene (sky, flat ground, etc...) is distinguishable from the obstacles.

Selective approaches are often useful for environments where they are designed to work in, but would generally not perform well in different environments. This type of approach is also not preferable if the environment consists of a large range of different kinds of obstacles, for it would be computationally expensive to identify each of these obstacle types individually. The scene itself could be subject to change over time, which can be problematic for algorithms which rely on certain characteristics of the scene. Therefore it is essential to be aware of all possible scenarios, and whether these scenarios are likely to change over the operating duration of the vehicle, when considering on using this approach for certain applications.

1.2.3 General Obstacle Detection

A generalized approach is suitable for the case where the environment and obstacle are not known in advance and the vehicle is not travelling at high speeds. The time constraint is more relaxed in this scenario and allows for longer processing times, which results in

algorithms which can provide more detailed maps of the environment. Most methods that fall in this category involve some form of dense 3D reconstruction since these reconstruction methods can provide detailed 3D maps at the expense of high computational loads. Dense 3D reconstruction is a well established field in computer vision, with some research done in applying these methods for unmanned vehicles. It is generally more popular for ground-based vehicles as they typically have more computational resources available. Visual Simultaneous Localization And Mapping (VSLAM) algorithms are usually used in conjunction with the 3D reconstruction to localize the vehicle and globally construct the map of the entire environment for later use.

Dense 3D reconstruction methods for UAVs were explored in [56] using a MATCH-T modeller for top-down elevation mapping and in [57] which compared the use of features obtained from Harris corner detection and tracked with cross correlation with the use of optical flow in dense 3D reconstruction. For UGVs, [58] used a stereo set of catadioptric cameras to perform voxel colouring [59] for 3D reconstruction. The result was projected into a 2D occupancy map of the floor, which is then used to estimate a visibility map of non-visible and visible voxels by treating the visibility problem as a 2D case, and then assume that any voxel that is above a non-visible pixel is also non-visible. This is done to increase the speed of the algorithm at the expense of accuracy. Two state of the art stereo vision 3D reconstruction techniques are described in [31, 60]. In [31], plane detection was performed using a GPU-implementation of SIFT extraction and an extended KLT tracker to find robust 3D points. A GPU-implementation for the sweep stereo algorithm was implemented to return a depth map of the scene, where depth map fusion and triangular mesh was performed for surface reconstruction. In [60], the Instant Scene Modeler (iSM) was introduced for calibrating 3D models. It successfully generates a 3D scene using stereo correspondence and triangulation to produce 3D point estimates, and a voxel-based triangular meshing technique to place a surface to the set of points. Its major drawback is the long computation time to return a 3D scene estimate. Another dense 3D reconstruction method worth noting is described in [29]. It is a monocular vision based structure from motion method which builds a continuous surface map over time using optical flow, and refines it further upon gathering additional information of the scene. While this technique executes reasonably quickly and returns an accurate map estimate, it was only tested for small workspace environments.

Like the previous approaches, the use of the general approach is dependent on the application. Small platforms typically do not have the computational resources to compute detailed 3D maps of the environment, so they are typically limited to reactive or selective approaches for obstacle detection. As mentioned in Section 1.2, dense 3D reconstruction is highly dependent on the accuracy and precision of the ego-motion of the camera, the baseline, and the correspondences from the image(s). Environments that impose vehicle vibration (such as rough terrain for UGVs and high turbulence for UAVs) or illumination

changes that affects the performance of VSLAM are some examples that could negatively affect the accuracy of the 3D map estimate. These issues can be addressed with damping mechanisms and robust feature extraction and tracking methods.

1.3 Research Approach and Contribution

The following criteria are used to assess which approach is the most suitable for the research presented:

- The algorithm must work indoors.
- The algorithm must be flexible, that is useful for a range of vehicles including ground rovers and aerial platforms.
- The environment and obstacle types are not known in advance.
- The algorithm returns a rough map of the present obstacles to be used for path planning.
- The algorithm can execute while the vehicle is moving.
- The payload, power and cost requirement of the vehicle must be as small as possible.

Because the environment and obstacles are not known in advance, it rules out using the selective approach as it is desirable for the vehicle to be able to monitor any indoor environment. Quadrotors and four wheel ground robots are allowed to move at slow speeds so there is more computation time, allowing for the general approach to be considered. Path planning is also important as it reduces the risk of possible collisions. This criterion makes the reactive approach less plausible as it produces less accurate scene details when compared to the general approach, which is undesirable for proper path planning. The payload, power and cost can be reduced with using one camera instead of two, making the vehicle more affordable for civilian applications.

Therefore the direction taken for this research is to use a monocular vision-based general approach. Structure from motion and dense 3D reconstruction techniques from the general approach allows for a detailed map generation from a single camera. Although the work done in [31, 60] produced high quality results, their work is tailored for stereo cameras and platforms with higher payloads and power availability. Also the work done for [56] is useful for providing elevation maps of the scene which allows for path planning for UAVs, however the method is not portable for UGVs as the elevation maps require aerial snapshots of the floor. The dual catadioptric cameras apparatus used in [58] is both large and heavy, and

the method used for 3D reconstruction is specific for ground vehicles. [57] makes a trade-off between decreased computation time and a lower 3D reconstruction performance, but in this case more computation time is available for more complicated methods to be used to produce better environment maps. Lastly, [29] is designed specifically for small workspace environments, so it may not be useful for indoor map generation.

As a result, a new algorithm was developed to meet these criteria. The key contribution made in this work is the coupling of a generalized object extraction method (image segmentation) with the 3D point cloud generated from structure from motion techniques in order to extrapolate additional details of the environment. This is done to improve the accuracy of the scene map, and to fill in gaps in the 3D point cloud with information obtained from image segmentation. This technique was tested and assessed using real video data of an indoor environment, and the results proved promising for future investigation. Another contribution for this work was the development of a ground research platform for the real-time implementation of this method.

The reconstruction pipeline proposed in Chapter 3, with the necessary background and theory described in Chapter 2, attempts to facilitate the vehicle and payload constraints and provide a reliable map of the indoor environment to be used for path planning. The motivation for this is discussed in more detail in Section 3.1. A testbed, outlined in Chapter 4, was developed to evaluate this novel 3D reconstruction algorithm with the experimental results and conclusions discussed in Chapters 5 and 6.

Chapter 2

Background and Theory

This chapter presents the methods required to understand the proposed algorithm. The topics covered are laid out in sequence with the flow of the algorithm. Section 2.1 describes the background theory on Scale Invariant Feature Transform (SIFT), which is required to find distinct features in an image. Once these features are found, the method used to track them are described in Section 2.3. From these results the formulae to estimate the 3D locations of the tracked features to form a 3D point cloud is derived in Section 2.4. Further analysis can be done on the image using some image filtering and segmentation methods, detailed in Section 2.5, to determine regions that represent surfaces of objects. By correlating the regions to the tracked features, these surfaces are formed using the methods described in Section 2.6. Additional topics in epipolar geometry, which are necessary to understand Sections 2.3.2, 2.4, and 2.6, are discussed in Section 2.2. These techniques are combined in the next chapter to form the proposed algorithm.

2.1 Feature Extraction - Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) was first described by David Lowe [61]. It is used to find distinct features on an image that are location, scale, and rotation invariant. This allows the same features to be identified even if the image is offset (i.e. the locations of the features change), scaled or rotated from its original form. The features are also partially invariant to illumination and 3D viewpoint changes. The motivation for finding these features is that they can be used for object detection and motion tracking, and, for the purpose of this work, 3D reconstruction. A map of 3D points can be constructed by finding and tracking these features. A more recent version for the SIFT algorithm is found in [24], and its contents are summarized in the following sections.

The algorithm is divided into four steps: scale-space extrema detection, keypoint localization and rejection, orientation and keypoint descriptor assignment. The complete SIFT algorithm uses these four steps, to locate and describe keypoints (points which are distinct and are identifiable when viewed from different perspectives). The descriptors and locations for the keypoints found from the SIFT algorithm are stored for feature matching and tracking.

2.1.1 Scale-Space Extrema Detection

The first part of the algorithm searches for points (referred to as *candidate keypoints*) that are scale invariant. One way to achieve this is to work in the scale-space [62, 63], which is an image that represents a specific scale, to find candidate keypoints that are unique and insensitive to the effects of scaling.

The scale-space representation $L(x, y, \sigma)$ of an image is defined by:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.1}$$

where $G(x, y, \sigma)$ is a variable-scale Gaussian kernel and $I(x, y)$ is the image map. The two parameters, $x \in \mathbb{R}$ and $y \in \mathbb{R}$ are the pixel coordinates, and $\sigma \in \mathbb{R}^+$ is the variance for the Gaussian filter. Therefore, the scale-space of the image L is the convolution of the Gaussian kernel function G onto the image I. It is assumed that the pixel values are in the range $[0, 1]$.

The Gaussian function is defined as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(x^2+y^2\right)/2\sigma^2}. \tag{2.2}$$

A difference-of-Gaussians (DoG) function is used to detect stable extrema (i.e. detect local minima or maxima) in the scale-space. Let $D(x, y, \sigma)$ be the DoG function, which is the difference of two scale-space images:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma), \end{aligned} \tag{2.3}$$

where $k \in \mathbb{R}^+$ is a constant factor used to vary the two scales. It is noted that the DoG filter can be thought of as an approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$ [24] which is used for scale invariant blob detection. The difference between the two methods are by a factor $(k-1)$, but Lowe [24] explained that it has no impact on the stability of the extrema detection or localization for relatively high values of k.

The algorithm first pre-blurs the original image with an initial σ and then convolves the blurred image incrementally with Gaussian filters such that each scale-space image is separated in scale by a constant factor k. Each successive convolution results in an increase in σ, or the amount of blur, in an image. This blurring helps reveal larger scale features that are not apparent in the original image. Once the scale-space image doubles in σ (an octave) the image is downsampled by two (i.e. taking every second pixel in each row and column of the scale-space image to produce a downsampled image) and the process is repeated on the downsampled image. Resampling the image greatly reduces computation time without affecting the accuracy of the scale information relative to the start of the previous octave. A Difference of Gaussians is then applied to adjacent scale-space images within each octave. This results in a DoG pyramid which will be further analysed to find local extrema. An interval, $s \in \mathbb{N}_1$, of DoGs per octave to use for scale extrema detection can be set by setting $k = 2^{1/s}$. Each octave should have $s + 3$ scale-space images (one at the start and two after the scale-space image corresponding to the doubling of σ), so that during the extrema detection the number of DoGs examined per octave is s. Figure 2.1 shows an illustration of this procedure, producing a Gaussian pyramid along with its DoG counterpart. To increase the amount of stable extrema, it is suggested that the original image is initially blurred by $\sigma = 0.5$ to prevent aliasing and then upsampled by two using linear interpolation. This upsampled image will serve as the starting point for the DoG pyramid construction. The starting σ suggested in [24] is 1.6 for repeatability and efficiency, as well as $s = 2$ (producing five scale-space and four DoG images per octave) and the total number of octaves is four.

The local extrema detection (see Figure 2.2) is done by taking each sample point of the DoG images from each scale and comparing it to its 8 neighbours as well as the 9 neighbouring points on the scales above and below it. There are a total of 26 neighbours to check, and if the sample point is larger or smaller than all of its neighbours, it is selected as a candidate keypoint. This procedure is done for each octave.

An analysis on the effects of changing values for k and s is described in [24].

2.1.2 Keypoint Localization and Rejection

The candidate keypoints that have low contrast, relative to its neighbouring pixels, or lie on an edgel (pixels describing an edge) are rejected by performing a check on the nearby data for location, scale and ratio of principal curvatures. Points with low contrast tend to be sensitive to noise and points that lie on an edge tend to be poorly localized, making them difficult to track properly on different images taken from different perspectives.

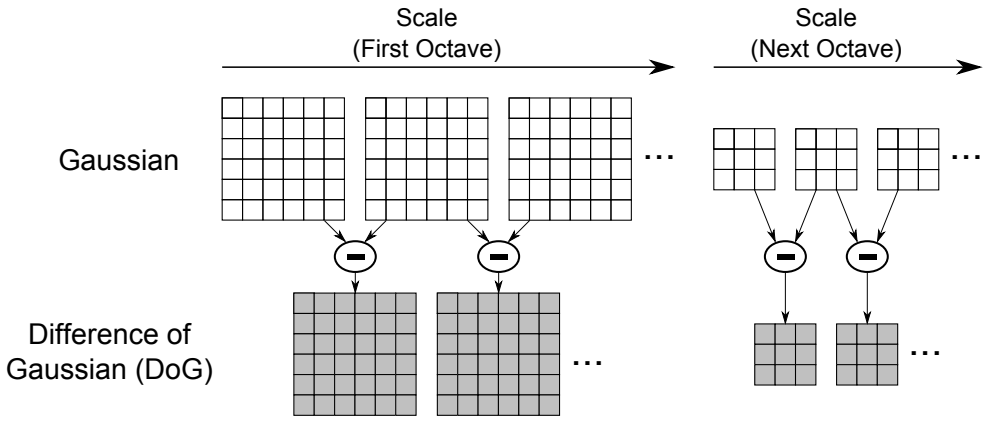


Figure 2.1: In the current octave, the initial image is incrementally convolved with Gaussian filters (resulting in scale-space images separated by a constant factor k). Once the set of scale-space images is found for the current octave, the Gaussian image is downsampled by two and the process is repeated again for the next octave, as shown at the top of the figure. The Difference of Gaussians is then applied to adjacent scale-space images in each octave, shown at the bottom of the figure.

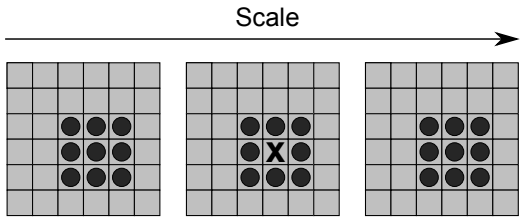


Figure 2.2: Diagram of three adjacent scale-spaces of DoG images used for extrema detection. The sample point (shown by the X) is compared with its 26 neighbours: 3x3 regions on the scale above and below and 8 in the scale of the sample point (shown by the circles).

Keypoints with Low Contrast

To find keypoints that have low contrast, the first step is to use a Taylor Series expansion (TSE) on the scale-space DoG function $D(x, y, \sigma)$ to approximate a 3D quadratic function with the candidate keypoints as the origin. The function is then used to find the location of the extremum. Let $D(\mathbf{x})$ be the TSE to second order, where $\mathbf{x} = (x, y, \sigma)^T$ is the offset from the origin, with D and its derivatives evaluated at the given candidate keypoint:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathbf{T}}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}. \tag{2.4}$$

Taking the derivative of $D(\mathbf{x})$ with respect to $\mathbf{x}$ and setting it to zero yields:

$$\hat{\mathbf{x}} = -\left(\frac{\partial^2 D}{\partial \mathbf{x}^2}\right)^{-1}\frac{\partial D}{\partial \mathbf{x}}, \tag{2.5}$$

where $\hat{\mathbf{x}}$ is the location of the extremum. The Hessian and derivative of $D(\mathbf{x})$, $\frac{\partial^2 D}{\partial \mathbf{x}^2}$ and $\frac{\partial D}{\partial \mathbf{x}}$ respectively, are approximated by using the differences of neighbouring sample points. If $\hat{\mathbf{x}}$ is greater than 0.5 in any dimension, then it means that the extremum is closer to another sample point. The location of the sample point is changed and the extremum is recalculated. The final offset $\hat{\mathbf{x}}$ is added to the current sample point in order to estimate the location of the extremum.

Substituting equation (2.5) into (2.4) simplifies to:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}, \tag{2.6}$$

where $D(\hat{\mathbf{x}})$ is the function value at the extremum. If $|D(\hat{\mathbf{x}})| < 0.03$ [24] then the corresponding keypoint has too low of a response from the DoG function, indicating that it is a poorly represented extrema, and is therefore discarded as a keypoint with low contrast.

Keypoints Localized on Edges

The DoG function has strong responses to edges, which is undesirable and can lead to unstable extrema. The keypoints along edges are poorly determined and therefore unstable to small amounts of noise.

The approach to find the candidate keypoints that lie on edges is to check its principal curvatures. The idea is that a poorly defined peak (e.g. an edge) in the DoG function will have a larger principal curvature across the edge than the principal curvature along it. A

2x2 Hessian matrix, $\mathbf{H}$, is computed to find the principal curvatures of the DoG function. The Hessian matrix is defined by:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}, \tag{2.7}$$

where the second order derivatives (D_{xx}, D_{xy}, and D_{yy}) are found by taking the differences of neighbouring sample points along the x and y directions, with the center being the location and scale of the keypoint. The eigenvalues of the Hessian matrix are proportional to the principal curvatures. It is shown in [24] that finding the eigenvalues is unnecessary because the ratio between the curvatures can be used instead to check whether or not the keypoint lies on an edge. Let $\alpha \in \mathbb{R}^+$ be the larger eigenvalue of $\mathbf{H}$ and $\beta \in \mathbb{R}^+$ be the smaller one, and $r = \alpha/\beta$ (i.e. the ratio between the two). The following relation from [24] is used:

$$\frac{\mathrm{Tr}\,(\mathbf{H})^2}{\mathrm{Det}\,(\mathbf{H})} = \frac{(r+1)^2}{r}, \tag{2.8}$$

where $\mathrm{Tr}\,(\mathbf{H})$ and $\mathrm{Det}\,(\mathbf{H})$ are the trace and determinant of the Hessian matrix, respectively. When the two eigenvalues are equal, the magnitude $(r+1)^2/r$ is at a minimum. It increases as r increases. Therefore the threshold criterion for candidate keypoints that are not localized to an edge is given by

$$\frac{\mathrm{Tr}\,(\mathbf{H})^2}{\mathrm{Det}\,(\mathbf{H})} < \frac{(r+1)^2}{r}. \tag{2.9}$$

Any keypoints that do not satisfy Eqn. (2.9) are discarded as being points that lie on edges. The value $r = 10$ is used in [24] for the experiments. It is noted that the decrease in value for r results in an increase in the number of keypoints rejected. This is because decreasing r means that the ratio between the two principal curvatures decreases, restricting the criterion from Eqn. (2.9) to cover a smaller range of values.

2.1.3 Orientation Assignment

The orientation of the keypoints is calculated to achieve invariance to image rotation. This is because the keypoint descriptor (described in Section 2.1.4) is calculated relative to this orientation and thereby ensures rotation invariance. To also ensure scale invariance, the computations are performed on the Gaussian smoothed image, $L(x, y)$, with the scale that is closest to the scale of the keypoint. For each sample point that is at the selected scale, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, is precomputed using pixel differences:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.10}$$

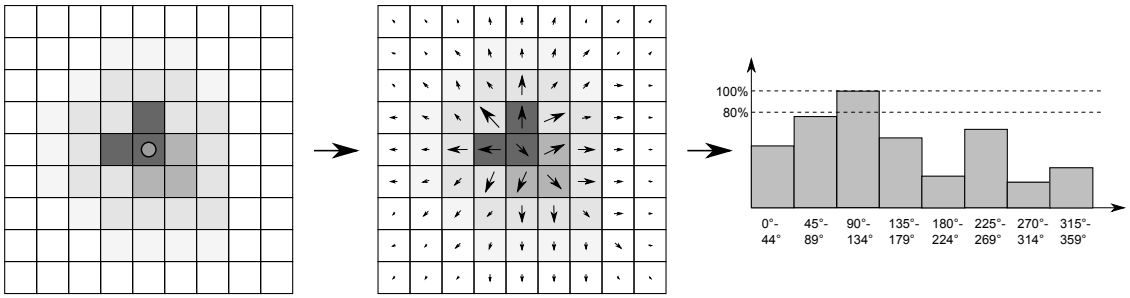


Figure 2.3: Diagram illustrating the orientation assignment method. The region around the keypoint (marked by the pixel with the grey circle) is shown on the left, the gradient magnitudes and directions for each pixel is shown in the middle (indicated by the arrow size and direction), and the resultant eight-bin orientation histogram is shown on the right. Since there are no peaks within 80% of the highest peak, only one orientation is assigned for the keypoint.

$$\theta(x,y) = \tan^{-1}\left(\left(L(x,y+1) - L(x,y-1)\right) / \left(L(x+1,y) - L(x-1,y)\right)\right). \qquad (2.11)$$

An orientation histogram is formed from the gradient orientations of sample points within the neighbourhood of the keypoint. The orientation histogram consists of 36 bins, each with a range of 10 degrees, covering 360 degrees in total. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. The highest peak, along with the peaks that are within 80% of the highest peak are chosen to be the orientation(s) for the keypoint. A parabola is fitted to the three histogram values that are near each peak to interpolate the peak position and improve its accuracy. Figure 2.3 shows a diagram illustrating the orientation assignment using an eight-bin histogram.

2.1.4 Local Image Descriptor

The last part of the algorithm, after finding the keypoints that are invariant to scale and rotation, is to define a distinct descriptor for each keypoint. The descriptor is designed to have some invariance to illumination and 3D viewpoint changes.

The descriptor is represented by a 128 element feature vector. To obtain the values for this vector, the gradient magnitudes and orientations are initially precomputed for the sample points in a region surrounding the keypoint and at the scale of the keypoint. The coordinates of the descriptor and gradient orientations are rotated relative to the keypoint orientation to achieve rotation invariance. To put more emphasis on the gradient magnitudes closer to the keypoint (i.e. center), a Gaussian weighting function with σ

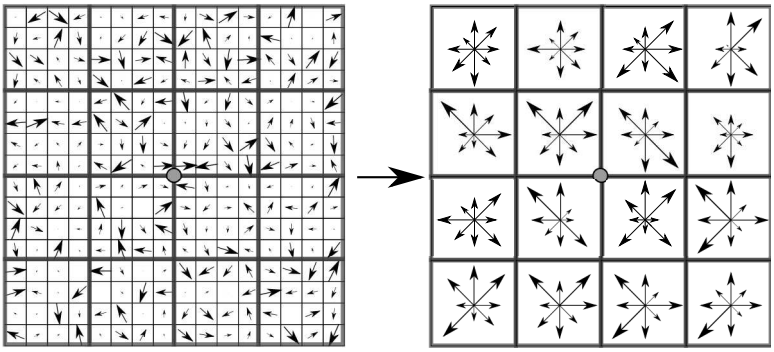


Figure 2.4: Shown on the left is the precomputed gradient magnitudes and orientations of a sample array (descriptor window). The keypoint is on the center of the descriptor window (marked with a grey circle). The sample array is divided into subregions. Shown on the right is are eight-bin orientation histograms formed for each subregion. Each bin are shown as arrows, with the orientation and magnitude indicated by the arrow's direction and size.

equal to one half the width of the descriptor window is used on the magnitudes of the gradients. The purpose of this function is to avoid sudden changes on the descriptor with small changes on the descriptor window position.

A region consists of 16x16 samples, with the keypoint in the center. The 16x16 samples is divided into 4x4 subregions, each subregion having 4x4 sample points. In each subregion, an 8-bin orientation histogram is formed from the gradient orientation and magnitudes. Each bin corresponds to a range of 45 degrees and one element in the feature vector. There are eight bins per subregion and 16 subregions, totalling 128 bins. Figure 2.4 shows an illustration of the keypoint descriptor regions and their respective bin formations.

The descriptor is made more robust through a series of enhancements. A trilinear interpolation is used on each gradient sample value, before the binning process, to distribute it into adjacent histogram bins. Each value added to a bin is weighted by $1 - d$ in each dimension, where d is the distance to the bin centers measured in units of histogram spacing. This allows for additional robustness to differences in the localization and orientation of the descriptor made from different image views, as it reduces abrupt changes to the descriptor caused by boundary effects from the samples changing from one bin to another. To reduce the effects of illumination change, the feature vector is first normalized to unit length and then thresholded such that all vector values are no greater than 0.2. The normalization of the vector is effective for affine, and relatively uniform, illumination changes, such as a change in image contrast (effectively multiplying the image brightness values by a constant) and brightness changes that have a constant brightness value added to each pixel. The

thresholding of values to be no greater than 0.2 is for reducing the effects from non-linear illumination changes, such as camera saturation or lighting changes due to 3D surfaces changing orientations.

Further details on the keypoint descriptor testing is discussed in [24].

2.2 Epipolar Geometry

This section serves as a background to the fundamental theories required for the understanding of the next sections. The geometry of using two camera views (stereo vision) is referred to as *epipolar geometry*, which is covered in depth in many computer vision textbooks [28, 27]. Only a limited scope that is relevant to the thesis is covered in the following subsections. Section 2.2.1 gives a brief description of homogeneous coordinates, which is used for the derivations of projecting a point in 3D Euclidean space (referred to as the *object point*) onto a point on the screen or image plane (referred to as the *image point*) (see Sections 2.2.2 to 2.2.4). Lastly, Section 2.2.5 extends this theory to address the problem of matching image points between camera views, by describing a geometric property known as the *epipolar constraint*.

The overview of the frame of reference notation used is explained here before proceeding further. A superscript prefixed to a point refers to the point's current frame of reference. For example, given a reference frame $\mathbf{B}$ and point $\mathbf{P}$, then ${}^{\mathbf{B}}\mathbf{P}$ is the point using $\mathbf{B}$ as its frame of reference. Another notation used is for the transformation matrices involved for changing reference frames on the point. Given the rotation matrix R and the translation vector $\mathbf{t}$ from one frame to another, the subscript and superscript prefixed to the matrix or vector variables are used for indicating the source and destination frames respectively. For example, ${}^{\mathbf{A}}_{\mathbf{B}}R$ and ${}^{\mathbf{A}}_{\mathbf{B}}\mathbf{t}$ denote the rotation matrix and translation vector used for transforming the point's frame of reference to be from $\mathbf{B}$ to $\mathbf{A}$. All reference frames in this thesis are assumed to obey the *right hand rule* and to be a basis in Euclidean space.

2.2.1 Homogeneous Coordinates

A brief description for homogeneous coordinate systems is described in this section. The homogeneous coordinate system is commonly used in projective geometry and computer graphics. It is similar to the Euclidean coordinate system with the exception that multiplying a point by a non-zero scalar in Euclidean coordinates represents the same point in homogeneous coordinates. In other words, given a point $\mathbf{P} = (X, Y, Z)^T \in \mathbb{R}^3$ and a non-zero scalar $\lambda \in \mathbb{R} \setminus \{0\}$, the point $\mathbf{P}/\lambda = (X/\lambda, Y/\lambda, Z/\lambda)^T$ in Euclidean space represents the same point in homogeneous coordinates. In order to represent a point

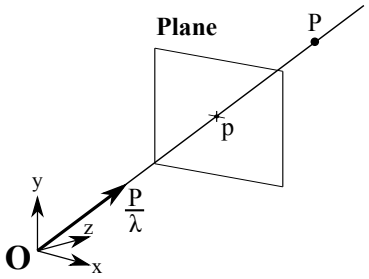


Figure 2.5: A perspective projection example of point $\mathbf{P}$ being projected onto the plane at point $\mathbf{p}$. Any point $\mathbf{P}/\lambda$ can be projected onto the same point $\mathbf{p}$ on the plane.

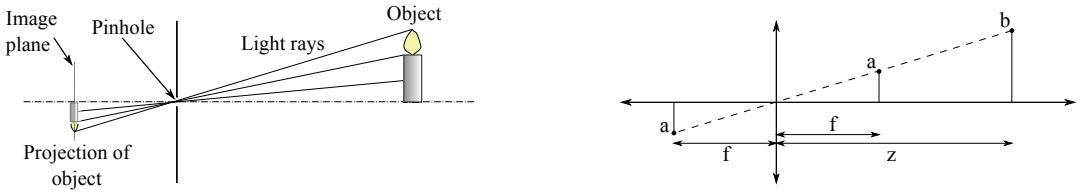
in homogeneous coordinates, an additional dimension is appended to the point with the scalar λ. For example, a 2D point $\mathbf{p} = (X, Y)^T$ in Euclidean coordinates is converted to $\mathbf{p} = (X/\lambda, Y/\lambda, 1)^T = (X, Y, \lambda)^T$ in homogeneous coordinates, and similarly $\mathbf{P} = (X/\lambda, Y/\lambda, Z/\lambda, 1)^T = (X, Y, Z, \lambda)^T$ is the homogeneous coordinates for a 3D point. Note that when $\lambda = 1$ the points are represented in Euclidean space, or equivalently, the homogeneous coordinates are *normalized*. This coordinate system is a useful representation for perspective projections, as shown in Figure 2.5. Any point $\mathbf{P}/\lambda$ can be projected onto the same point $\mathbf{p}$ on the plane, or in other words if the plane represents an image plane, any point along the ray traced by $\mathbf{P}/\lambda$ can be seen as the point $\mathbf{p}$ on the image plane. This implies that there is a scale ambiguity on the location of $\mathbf{P}$ when only given $\mathbf{p}$ on the plane, since it can lie anywhere along the line traced by $\mathbf{P}/\lambda$.

2.2.2 Pinhole Camera Model

The *pinhole camera model* is a model used for describing the geometric relations of light projected from an object onto the light sensor (image plane) of the camera. Figure 2.6 (a) shows a physical illustration of the pinhole camera model. The light rays emitted from the object passes through the pinhole (treated as a point), and projected onto an image plane. It is assumed that because the rays only pass through a single point. Figure 2.6 (b) shows a geometrical derivation on the projection of the point $\mathbf{b} \in \mathbb{R}$ (the object) to the point $\mathbf{a} \in \mathbb{R}$ (the projection of the object). Using similar triangles, the equation for $\mathbf{a}$ is derived to be:

$$\mathbf{a} = \frac{f}{\mathrm{z}}\mathbf{b}, \tag{2.12}$$

where $f \in \mathbb{R}^+$ is defined as the *focal length* of the camera and $\mathrm{z} \in \mathbb{R}^+$ is the distance from the object to the pinhole, along the horizontal axis (referred to as the *optical axis*). In actual cameras, the pinhole can be referred to as the *optical center* or *camera center*. It



(a) (b)

Figure 2.6: (a) An illustration of the pinhole camera model. (b) The geometric properties used to derive the model.

is assumed that the image plane can be in the front or behind the optical center without much difference other than the projected image being reversed. For this thesis the image plane is assumed to be in front of the camera for better understandability of the derivations involving object to camera image projection.

In reality, the pinhole size, camera lens, sensor used to capture the light (e.g. charge-coupled devices (CCD) vs. active-pixel sensors (APS)), and the light behaviour itself are not ideal and can affect the outcome described in Eqn. (2.12). Although this is the case, today's cameras are designed to minimize these errors, so the pinhole camera model is assumed to be a sufficient approximation and is widely used in many computer vision applications.

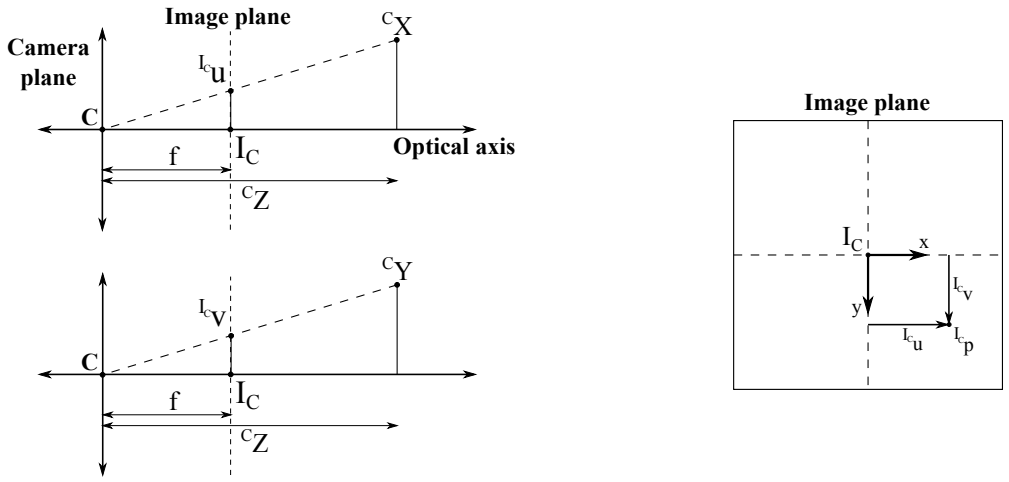
2.2.3 Camera Matrix

The *camera matrix* is a matrix used to convert an object point in 3D space to an image point on the screen. It is assumed that the frame of reference for the object point is the camera frame, and that the camera matrix is defined using the pinhole camera model described in Section 2.2.2. Figure 2.7 shows an illustration of point ${}^{\mathbf{C}}\mathbf{P} = \left({}^{\mathbf{C}}X, {}^{\mathbf{C}}Y, {}^{\mathbf{C}}Z\right)^T \in \mathbb{R}^3$ (using the camera frame $\mathbf{C}$ as a reference frame) being projected onto ${}^{\mathbf{I_C}}\mathbf{p} = \left({}^{\mathbf{I_C}}u, {}^{\mathbf{I_C}}v\right)^T \in \mathbb{R}^2$, where $\mathbf{I_C}$ is the image frame centered about the intersection of the optical axis and the image plane. Extending Eqn. (2.12) to the illustration gives:

$$ {}^{\mathbf{I_C}}u = \frac{f}{{}^{\mathbf{C}}Z} {}^{\mathbf{C}}X \tag{2.13} $$

$$ {}^{\mathbf{I_C}}v = \frac{f}{{}^{\mathbf{C}}Z} {}^{\mathbf{C}}Y. \tag{2.14} $$

Although ${}^{\mathbf{I_C}}\mathbf{p}$ lies on the image plane with each dimension having the same units as the dimensions of ${}^{\mathbf{C}}\mathbf{P}$, e.g. in meters, in reality the image is given in units of pixels



(a) (b)

Figure 2.7: (a) The projection of the object point onto the image plane for both x and y axes. (b) The front view of the image plane with the projected point.

and the image frame, denoted by $\mathbf{I}$, is situated on the top left corner of the image, as shown in Figure 2.8. The figure shows the x and y length for the pixel, $1/m \in \mathbb{R}^+$ and $1/n \in \mathbb{R}^+$ respectively, and the pixel offset from the frame $\mathbf{I}$ to $\mathbf{I_C}$ is $(u_0, v_0) \in \mathbb{N}_0^2$. From this information the transformation from the point ${}^{\mathbf{I_C}}\mathbf{p}$ to ${}^{\mathbf{I}}\mathbf{p}$ is determined to be:

$$ {}^{\mathbf{I}}\mathbf{p} = \begin{bmatrix} m & 0 \\ 0 & n \end{bmatrix} \left({}^{\mathbf{I_C}}\mathbf{p} \right) + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}, \tag{2.15} $$

and combining, then simplifying into matrix form, the relationships from Eqn. (2.13), Eqn. (2.14), and Eqn. (2.15) results in:

$$ {}^{\mathbf{I}}\mathbf{p} = \frac{1}{{}^{\mathbf{C}}Z} K \left({}^{\mathbf{C}}\mathbf{P} \right), \tag{2.16} $$

where the 3x3 matrix K is the camera matrix, which is defined to be:

$$ K = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.17} $$

where $\alpha = mf$ and $\beta = nf$. It should be noted that from Eqn. (2.16), the image point ${}^{\mathbf{I}}\mathbf{p} = \left({}^{\mathbf{I}}u \in \mathbb{N}_0, {}^{\mathbf{I}}v \in \mathbb{N}_0, 1 \right)^T$ is in the homogeneous coordinate representation described in Section 2.2.1. The camera matrix is modified further to account for manufacturing errors

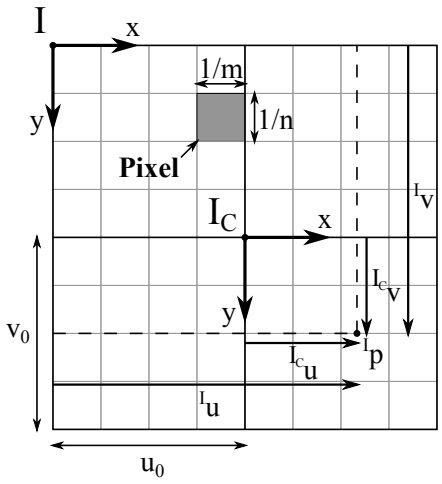


Figure 2.8: Diagram of the variables involved in the representation of the point **p** with respect to the image frame **I**.

on the camera as well as other defects [28], as shown below:

$$K = \begin{bmatrix} \alpha & -\alpha \cot\theta & u_0 \\ 0 & \frac{\beta}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.18}$$

where $\theta \in [0, \pi]$ is the angle between the two image axes. When $\theta = \pi/2$ the camera matrix from Eqn. (2.18) reverts back to the original camera matrix shown in Eqn. (2.17).

2.2.4 Projection Matrix

In Section 2.2.3, the object point **P**'s frame of reference was assumed to be **C**. This is generally not the case, as the frame of reference used can be any arbitrary frame, defined here as the world frame **W**. Also, for efficiency purposes, **P** is assumed to be represented in homogeneous coordinates, that is, $\mathbf{P} = (X, Y, Z, 1)^T$. To convert the point ${}^{\mathbf{W}}\mathbf{P}$ to ${}^{\mathbf{C}}\mathbf{P}$, or to convert the frame of reference for the object point from any arbitrary frame to the camera frame, the following transformation is used:

$${}^{\mathbf{C}}\mathbf{P} = \left({}^{\mathbf{C}}_{\mathbf{W}}R\right)\left[I_{3x3} \mid \left({}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t}\right)\right]\left({}^{\mathbf{W}}\mathbf{P}\right), \tag{2.19}$$

where I_{3x3} is the 3x3 identity matrix, ${}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t}$ is the 3x1 translation matrix to translate from the world frame origin to the camera frame origin, and ${}^{\mathbf{C}}_{\mathbf{W}}R$ is the 3x3 rotation matrix to

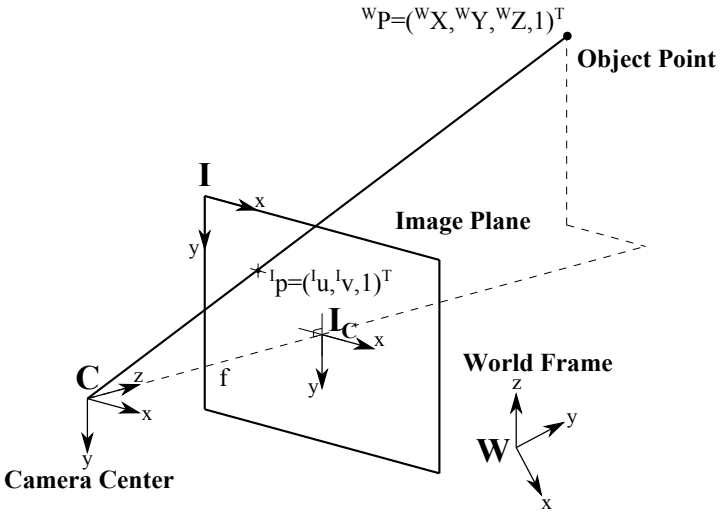


Figure 2.9: A diagram of the projection of the object point onto the image plane, with the four essential frames labelled.

rotate from the world frame axis to the camera frame axis. Substituting Eqn. (2.19) to Eqn. (2.16) results in:

$$\begin{aligned} {}^{\mathbf{I}}\mathbf{p} &= \frac{1}{{}^{\mathbf{C}}Z} K \left({}^{\mathbf{C}}_{\mathbf{W}}R\right) \left[I_{3x3} \,|\, \left({}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t}\right)\right] \left({}^{\mathbf{W}}\mathbf{P}\right) \\ &= \frac{1}{{}^{\mathbf{C}}Z} M \left({}^{\mathbf{W}}\mathbf{P}\right), \end{aligned} \tag{2.20}$$

where $M = K \left({}^{\mathbf{C}}_{\mathbf{W}}R\right) \left[I_{3x3} \,|\, \left({}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t}\right)\right]$ is a 3x4 matrix called the *projection matrix*. The projection from the object point to the image plane is depicted in Figure 2.9, with the four necessary frames shown.

2.2.5 Epipolar Constraint

It is possible to estimate a point in 3D space given two different camera views of the same point. The geometrical property that makes this possible is called the *epipolar constraint*, which is a geometrical relation between the two image points from the two camera views and the object point in 3D space. Figure 2.10 shows a diagram of the geometrical components used to define the epipolar constraint. The *epipolar plane* is the plane formed by the two camera centers and the object point, and the *baseline* is the line connecting the camera centers. The *epipolar line*, l, is defined as the line that intersects an image plane with the epipolar plane, while the *epipole*, $\mathbf{e}$, is defined as the intersection of the baseline with the image plane. From the diagram, the following geometric relationship is defined:

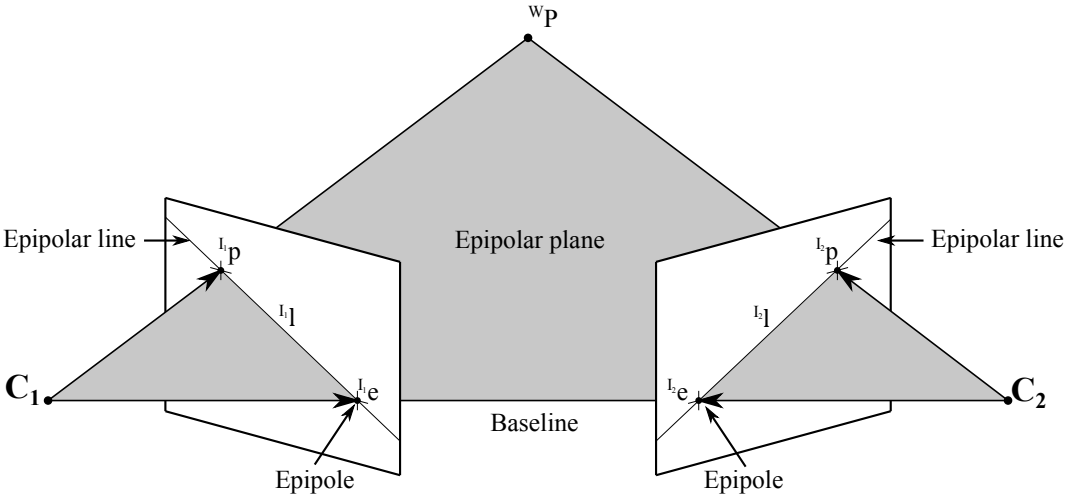


Figure 2.10: Diagram illustrating the epipolar geometry definitions.

$$ {}^{\mathbf{I_2}}\mathbf{p} \cdot \left({}^{\mathbf{I_2}}\mathbf{e} \times {}^{\mathbf{I_2}}\mathbf{p} \right) = 0, \tag{2.21} $$

where ${}^{\mathbf{I_2}}\mathbf{p}$ is the projected image point on the second camera view, and ${}^{\mathbf{I_2}}\mathbf{e}$ is the epipole on the second camera. The two image points can be treated as vectors originating from the second camera and extending to their respective points on the image plane (as shown in Figure 2.10). The cross product between the vectors ${}^{\mathbf{I_2}}\mathbf{p}$ and ${}^{\mathbf{I_2}}\mathbf{e}$ always produces a vector that is perpendicular to ${}^{\mathbf{I_2}}\mathbf{p}$, and therefore the dot product between this vector and ${}^{\mathbf{I_2}}\mathbf{p}$ will always be zero. Note that from Eqn. (2.20), ${}^{\mathbf{I_2}}\mathbf{e}$ can also be defined as:

$$ {}^{\mathbf{I_2}}\mathbf{e} = \gamma_2 M_2 \left({}^{\mathbf{W}}\mathbf{O}_{\mathbf{C_1}} \right), \tag{2.22} $$

where ${}^{\mathbf{W}}\mathbf{O}_{\mathbf{C_1}}$ is origin location of the first camera center with the world frame as its frame of reference, which is equal to the translation vector ${}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t} = -{}^{\mathbf{C_1}}_{\mathbf{W}}\mathbf{t}$. The projection matrix for the second camera is M_2 and $\gamma_2 \in \mathbb{R} \backslash \{0\}$ is a scalar that does not affect the relation defined in Eqn. (2.21). Applying Eqn. (2.20) for ${}^{\mathbf{I_1}}\mathbf{p}$ and ${}^{\mathbf{I_2}}\mathbf{p}$ yields:

$$ \begin{aligned} {}^{\mathbf{I_1}}\mathbf{p} &= \frac{1}{{}^{\mathbf{C_1}}Z} \left(M_1 \right) \left({}^{\mathbf{W}}\mathbf{P} \right) \\ {}^{\mathbf{W}}\mathbf{P} &= \left({}^{\mathbf{C_1}}Z \right) \left(M_1 \right)^{+} \left({}^{\mathbf{I_1}}\mathbf{p} \right), \end{aligned} \tag{2.23} $$

and

$$ {}^{\mathbf{I_2}}\mathbf{p} = \frac{1}{{}^{\mathbf{C_2}}Z} \left(M_2 \right) \left({}^{\mathbf{W}}\mathbf{P} \right), \tag{2.24} $$

where ${}^{\mathbf{I_1}}\mathbf{p}$ is the image point on the first camera view, $(M_1)^{+}$ is the pseudoinverse of the first camera projection matrix M_1, and ${}^{\mathbf{C_1}}Z$ and ${}^{\mathbf{C_2}}Z$ are the object point Z coordinates

described in the first and second camera frame of reference respectively. Since M_1 is a 3x4 (non-square) matrix, the pseudoinverse of the matrix is found in Eqn. (2.23) instead of its inverse, as the inverse of a matrix is only valid for square matrices. Substituting ${}^{\mathbf{W}}\mathbf{P}$ from Eqn. (2.23) to Eqn. (2.24) yields:

$$ {}^{\mathbf{I_2}}\mathbf{p} = \gamma_1 \left(M_2\right) \left(M_1\right)^{+} \left({}^{\mathbf{I_1}}\mathbf{p}\right), \tag{2.25} $$

where $\gamma_1 = {}^{\mathbf{C_1}}Z / {}^{\mathbf{C_2}}Z$ is another scalar which does not affect the relation defined in Eqn. (2.21). Combining Eqn. (2.21), Eqn. (2.22), and Eqn. (2.25) and simplifying results in:

$$ \begin{aligned} 0 &= {}^{\mathbf{I_2}}\mathbf{p} \cdot \left(\gamma_2 M_2 \left({}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t}\right) \times \gamma_1 \left(M_2\right) \left(M_1\right)^{+} \left({}^{\mathbf{I_1}}\mathbf{p}\right)\right) \\ 0 &= {}^{\mathbf{I_2}}\mathbf{p}^T \left[M_2 \left({}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t}\right)\right]_{\times} \left(M_2\right) \left(M_1\right)^{+} \left({}^{\mathbf{I_1}}\mathbf{p}\right) \\ 0 &= {}^{\mathbf{I_2}}\mathbf{p}^T \left(\mathbf{E}\right) {}^{\mathbf{I_1}}\mathbf{p}, \end{aligned} \tag{2.26} $$

where $\mathbf{E} = \left[M_2 \left({}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t}\right)\right]_{\times} \left(M_2\right) \left(M_1\right)^{+}$ is a 3x3 matrix defined as the *essential matrix* and $\left[M_2 \left({}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t}\right)\right]_{\times}$ is the matrix representation of $M_2 \left({}^{\mathbf{W}}_{\mathbf{C_1}}\mathbf{t}\right)$ to convert the vector cross product term from Eqn. (2.21) into a matrix multiplication. The relationship between the cross product and a vector product of two vectors $\mathbf{x} = \left(x_1, x_2, x_3\right)^T$ and $\mathbf{y} = \left(y_1, y_2, y_3\right)^T$ is as follows:

$$ \mathbf{x} \times \mathbf{y} = \left[\mathbf{x}\right]_{\times} \mathbf{y}, \text{ where } \left[\mathbf{x}\right]_{\times} \overset{\text{def}}{=} \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{2.27} $$

In other words, $\left[\mathbf{x}\right]_{\times}$ is the skew symmetric matrix of $\mathbf{x}$. In addition, the *fundamental matrix* is defined analogously to the essential matrix, but without the use of the camera matrix conversion, i.e. $K = I_{3x3}$. This means that the fundamental matrix relates the image points in terms of the units associated with the object point, rather than in pixels.

The main purpose of Eqn. (2.26) is to address the *correspondence problem*, i.e. finding which image points from one camera view correspond to which image points from another camera view. Figure 2.11 shows that given the image point ${}^{\mathbf{I_1}}\mathbf{p}$ and the two camera frames $\mathbf{C_1}$ and $\mathbf{C_2}$, the location of ${}^{\mathbf{I_2}}\mathbf{p}$ can be anywhere on the epipolar line ${}^{\mathbf{I_2}}l$. This is because ${}^{\mathbf{C}}Z$ is not known in Eqn. (2.20), so the location of ${}^{\mathbf{W}}\mathbf{P}$ can only be defined up to a scale ambiguity. The epipolar constraint helps reduce the search space for ${}^{\mathbf{I_2}}\mathbf{p}$ to be one dimensional (the epipolar line) instead of two dimensional (the entire image), and therefore reduces the search time and narrows the set of possible choices. The epipolar line ${}^{\mathbf{I_2}}l$ can be found by analysing Eqn. (2.26). Let ${}^{\mathbf{I_2}}l = \left({}^{\mathbf{I_2}}l_1, {}^{\mathbf{I_2}}l_2, {}^{\mathbf{I_2}}l_3\right)^T \in \mathbb{R}^3$, where it is defined to be:

$$ {}^{\mathbf{I_2}}l_1 = \left(\mathbf{E}\right) {}^{\mathbf{I_1}}\mathbf{p}. \tag{2.28} $$

From this definition, Eqn. (2.26) can be simplified to:

$$ \begin{aligned} \left({}^{\mathbf{I_2}}\mathbf{p}\right)^T {}^{\mathbf{I_2}}l &= 0 \\ {}^{\mathbf{I_2}}l_1 {}^{\mathbf{I_2}}u + {}^{\mathbf{I_2}}l_2 {}^{\mathbf{I_2}}v + {}^{\mathbf{I_2}}l_3 &= 0, \end{aligned} \tag{2.29} $$

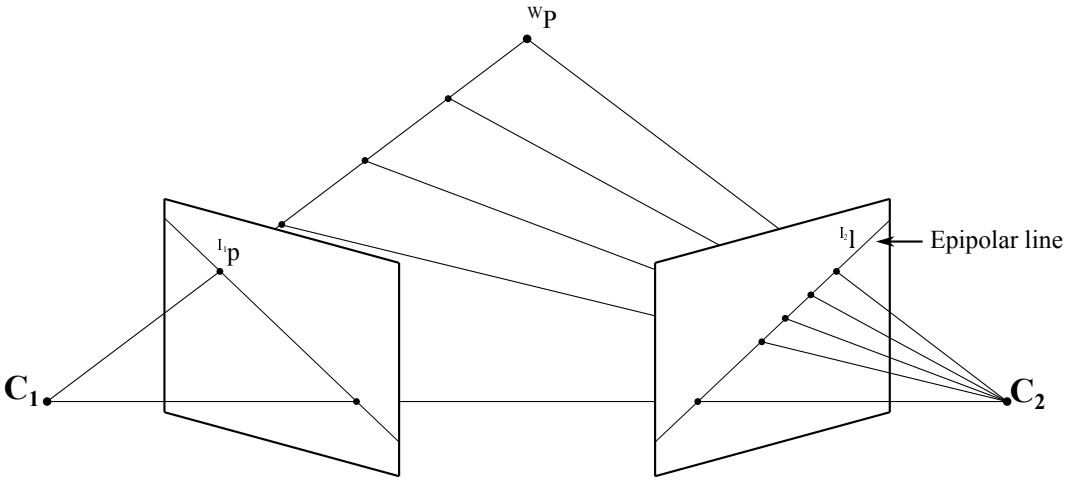


Figure 2.11: Diagram showing how the epipolar line on the second view is derived as a consequence of knowing only the location of the image point on the first camera view and not knowing the location of the object point.

which is an equation of a line. Therefore ${}^{\mathbf{I_2}}l = (\mathbf{E})\,{}^{\mathbf{I_1}}\mathbf{p}$ defines the constants for the epipolar line. It can be used for the correspondence problem in constraining the location of ${}^{\mathbf{I_2}}\mathbf{p}$ to the line defined by ${}^{\mathbf{I_2}}l$, which can greatly improve the performance of correspondences between two image views.

2.3 Feature Tracking

The main purpose for the SIFT features calculated from Section 2.1 are that they can be repeatedly identified in successive images. The features (keypoints) can be tracked from one image to the next through a matching procedure described in Section 2.3.1. An extension improving the matching procedure by utilizing available camera information is described in Section 2.3.2. Once the keypoints are matched between each successive image, the matches can be used for the 3D feature mapping described in Section 2.4, which maps the feature points in 3D Euclidean space, resulting in a 3D point cloud of the environment.

2.3.1 Keypoint Matching

The method suggested in [24] to match SIFT keypoints between two images focuses on comparing the Euclidean distances of their descriptors. The Euclidean distance metric is

defined as:

$$d_E(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T (\mathbf{x}_1 - \mathbf{x}_2)}, \tag{2.30}$$

where $d_E(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^+ \cup \{0\}$ is the distance between the two feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Let $X^{\mathbf{d1}} = \left\{\mathbf{x}_{i_1}^{\mathbf{d1}} | i_1 = 1, \ldots, n_1\right\}$ and $X^{\mathbf{d2}} = \left\{\mathbf{x}_{i_2}^{\mathbf{d2}} | i_2 = 1, \ldots, n_2\right\}$ be the sets of descriptors for two images, where $\mathbf{x}_{i_1}^{\mathbf{d1}} \in \mathbb{R}^{128}$ and $\mathbf{x}_{i_2}^{\mathbf{d2}} \in \mathbb{R}^{128}$ are the descriptor vectors, i_1 and i_2 are the keypoint indices, and $n1$ and $n2$ are the total number of descriptors for the first and second image respectively. The following steps are used to determine the keypoint matches between the two images:

1. Calculate the Euclidean distances from the first keypoint descriptor (the current *descriptor of interest*) from the first image to all of the descriptors from the second image using Eqn. (2.30), i.e. calculate $d_E\left(\mathbf{x}_1^{\mathbf{d1}}, \mathbf{x}_{i_2}^{\mathbf{d2}}\right), i_2 = 1, \ldots, n_2$.

2. Store the smallest and second smallest distance as well as the descriptor index corresponding to the smallest distance, i.e. $i_s = \arg\min_{i_2} d_E\left(\mathbf{x}_1^{\mathbf{d1}}, \mathbf{x}_{i_2}^{\mathbf{d2}}\right)$.

3. If the ratio between the smallest and second smallest distance is less than a threshold, $\epsilon_r \in \mathbb{R}^+$, then a match exists between the first keypoint from the first image and the ${i_s}^{\text{th}}$ keypoint from the second image. Otherwise there is no match found for the first keypoint from the first image to any of the keypoints in the second image.

4. Repeat steps 1 to 3 for the rest of the keypoints ($i_1 = 2, \ldots, n_1$) from the first image to find the rest of the matches.

The idea behind comparing the ratio of the smallest and second smallest distance is to reduce incorrect matches. If the ratio is too large (close to one) then that means that there are at least two descriptors from the second image that are closely related to the descriptor of interest from the first image. The smaller the ratio the more distinct (less ambiguous) the match. It is noted that ϵ_r should not be set too small for it would return very few matches that are highly distinctive, so a balance between the number of matches versus the distinction of those matches are required. The value for ϵ_r suggested by [24] is 0.8. Additional analysis for this matching procedure is discussed in [24].

2.3.2 Epipolar Constraint Criterion

An extension to the keypoint matching algorithm discussed in Section 2.3.1 is to include an *epipolar constraint* (see Section 2.2.5) to limit the number of keypoints from the second image that are being compared with the keypoint of interest from the first image. The motivation for this constraint is to improve the speed and the accuracy of the matching process by utilizing camera information, which is assumed to be available for this research.

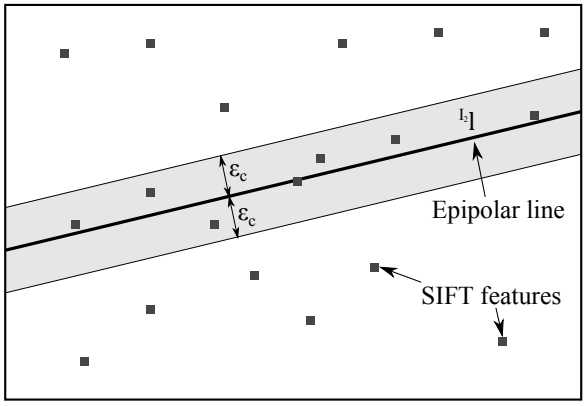


Figure 2.12: Diagram illustrating the epipolar constraint criterion used to find which keypoints (SIFT features) to include in the matching algorithm. The points that are within the perpendicular distance ϵ_c away from the epipolar line $^{\mathbf{I_2}}l$ (shown as the shaded area) is considered for the matching algorithm.

For each keypoint on the first image, the epipolar line $^{\mathbf{I_2}}l$ can be calculated for the second image using Eqn. (2.28). It is assumed that a keypoint from the second image, which is a potential match with the keypoint from the first image, should be somewhere on the epipolar line $^{\mathbf{I_2}}l$. But due to possible errors in the camera information as well as the keypoint location estimates, a more flexible approach is introduced where the keypoint can lie anywhere within a perpendicular distance, $\epsilon_c \in \mathbb{R}^+$, away from $^{\mathbf{I_2}}l$. This is illustrated in Figure 2.12, where the shaded area is the area that contains all possible points which are ϵ_c distance away from the epipolar line $^{\mathbf{I_2}}l$. Any keypoint (SIFT feature) from the second image that is inside the boundary surrounding the shaded area is considered for the matching algorithm described in Section 2.3.1.

Given an arbitrary image point $\mathbf{p}_0 = (u_0, v_0, 1)^T$ and the line $l = (l_1, l_2, l_3)^T$, the perpendicular distance, $d_P(\mathbf{p}_0, l) \in \mathbb{R}^+ \cup \{0\}$, from the point to the line can be determined using scalar projection. Let $\mathbf{p} = (u, v, 1)^T$ be an image point on the line l, $\mathbf{v} = (u - u_0, v - v_0)^T$ be a vector from the point $\mathbf{p}_0$ to $\mathbf{p}$, and $\mathbf{n} = (l_1, l_2)^T$ be the normal of the line l. The line equation is given to be of the form:

$$\begin{aligned} l_1 u + l_2 v + l_3 &= 0 \\ -l_3 &= l_1 u + l_2 v, \end{aligned} \tag{2.31}$$

Figure 2.13 illustrates that the magnitude of $d_P(\mathbf{p}_0, l)$ can be found by a scalar projection of the vector $\mathbf{v}$ onto the line normal $\mathbf{n}$:

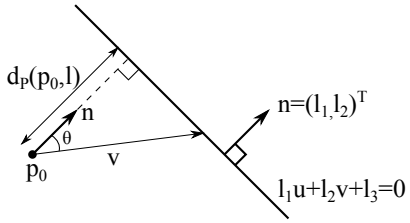


Figure 2.13: Diagram illustrating the derivation of the perpendicular distance from a 2D point to a 2D line.

$$\begin{aligned} d_P(\mathbf{p}_0, l) &= ||\text{proj}_{\mathbf{n}}\mathbf{v}|| \\ &= \frac{||\mathbf{v}\cdot\mathbf{n}||}{||\mathbf{n}||} \\ &= \frac{||l_1u + l_2v - l_1u_0 - l_2v_0||}{\sqrt{l_1^2+l_2^2}} \\ &= \frac{||l_1u_0 + l_2v_0 + l_3||}{\sqrt{l_1^2+l_2^2}} \\ &= \frac{||l\cdot\mathbf{p}_0||}{\sqrt{l_1^2+l_2^2}}. \end{aligned} \tag{2.32}$$

From Eqn. (2.32), the criterion for keypoints lying within a distance ϵ_c to the line l is defined to be:

$$d_P(\mathbf{p}_0, l) \leq \epsilon_c. \tag{2.33}$$

With this criterion the keypoints that are not sufficiently close to the epipolar line can be excluded from the matching process. The keypoint matching procedure from Section 2.3.1 is modified such that in step 1, instead of finding the Euclidean distance from the descriptor of interest to all descriptors in the second image, first determine the epipolar line $^{\mathbf{I_2}}l$ in the second image based on the keypoint of interest and then only find the distance to the descriptors who's corresponding keypoint locations satisfies the criterion in Eqn. (2.33).

2.4 3D Feature Mapping

This section describes a method to use camera information and the tracked feature points to locate their position in 3D Euclidean space. The result is a 3D point cloud that can reveal potential obstacles in the scene, where each feature point on the 3D map can be treated as a point pertaining to an obstacle. The geometric principles for how a feature point (object

point) in 3D space can be projected onto an image is described in Section 2.2. Here, the reverse is done to take the projected image points and find the location of the object point. At least two image points in two different views are required for approximating the location of the object point (see Section 2.2.5). The approximation method described can use more than two views of the same feature point to obtain a better approximation of where that feature point is in 3D space.

Let the image axes for each camera be denoted by $\mathbf{I}_k, k = 1, \ldots, m_c$ and M_k be the camera's respective projection matrix, where k is the camera index and m_c is the number of cameras. Also let the set of feature points in 3D space and in an image be $\mathbf{P}_i, i = 1, \ldots, n_p$ and $\mathbf{p}_i$ respectively, where i is the feature point index and n_p is the number of feature points. Using these definitions, ${}^{\mathbf{I}_k}\mathbf{p}_i$ is denoted as the i^{th} feature point on the k^{th} image. The projection of a 4x1 object point ${}^{\mathbf{W}}\mathbf{P}$ to its corresponding 3x1 image point ${}^{\mathbf{I}}\mathbf{p}$ is shown in Eqn. (2.20), which can be expanded and rewritten as:

$$\begin{aligned} {}^{\mathbf{I}}u &= \frac{M^1}{M^3}{}^{\mathbf{W}}\mathbf{P} \\ {}^{\mathbf{I}}v &= \frac{M^2}{M^3}{}^{\mathbf{W}}\mathbf{P}, \end{aligned} \tag{2.34}$$

where M^1, M^2, and M^3 denote the three rows of M, and u and v are the image point's coordinates. Eqn. (2.34) can be rearranged in matrix form to be:

$$\begin{pmatrix} {}^{\mathbf{I}}uM^3 - M^1 \\ {}^{\mathbf{I}}vM^3 - M^2 \end{pmatrix} {}^{\mathbf{W}}\mathbf{P} = 0. \tag{2.35}$$

Applying Eqn. (2.35) on the k^{th} camera and the i^{th} feature point yields:

$$\begin{pmatrix} {}^{\mathbf{I}_k}u_iM_k^3 - M_k^1 \\ {}^{\mathbf{I}_k}v_iM_k^3 - M_k^2 \end{pmatrix} {}^{\mathbf{W}}\mathbf{P}_i = 0. \tag{2.36}$$

Therefore extending Eqn. (2.36) to focus on the i^{th} feature point and all camera views, $k = 1, \ldots, m_c$, results in:

$$Q{}^{\mathbf{W}}\mathbf{P}_i = 0, \text{ where } Q = \begin{pmatrix} {}^{\mathbf{I}_1}u_iM_1^3 - M_1^1 \\ {}^{\mathbf{I}_1}v_iM_1^3 - M_1^2 \\ {}^{\mathbf{I}_2}u_iM_2^3 - M_2^1 \\ {}^{\mathbf{I}_2}v_iM_2^3 - M_2^2 \\ \vdots \\ {}^{\mathbf{I}_{m_c}}u_iM_{m_c}^3 - M_{m_c}^1 \\ {}^{\mathbf{I}_{m_c}}v_iM_{m_c}^3 - M_{m_c}^2 \end{pmatrix}. \tag{2.37}$$

Eqn. (2.37) is of the homogeneous form $A\mathbf{x} = 0$, which can be solved using Singular Value Decomposition (SVD). Finding the SVD of A returns the eigenvectors as well as the

singular values pertaining to the eigenvectors. The eigenvector with the smallest singular value (i.e. the smallest eigenvalue) is chosen as the solution for $\hat{\mathbf{x}}$ that minimizes $A\mathbf{x}$. It is also noted that this solution is for $|\mathbf{x}| = 1$, in other words the solution always returns a unit vector. Therefore finding the solution of $Q^{\mathbf{W}}\mathbf{P}_i = 0$ using SVD returns ${}^{\mathbf{W}}\hat{\mathbf{P}}_i^u$, where $\left|{}^{\mathbf{W}}\hat{\mathbf{P}}_i^u\right| = 1$ and $\hat{\mathbf{P}}$ denotes an estimate to $\mathbf{P}$. Recall from Section 2.2.1 that homogeneous coordinates can be normalized to be represented in Euclidean space. In other words, the estimate ${}^{\mathbf{W}}\hat{\mathbf{P}}_i$, is found by normalizing ${}^{\mathbf{W}}\hat{\mathbf{P}}_i^u$:

$$ {}^{\mathbf{W}}\hat{\mathbf{P}}_i = \frac{1}{\hat{\lambda}}{}^{\mathbf{W}}\hat{\mathbf{P}}_i^u = \frac{1}{\hat{\lambda}}\begin{bmatrix} \hat{\lambda}{}^{\mathbf{W}}\hat{X}_i \\ \hat{\lambda}{}^{\mathbf{W}}\hat{Y}_i \\ \hat{\lambda}{}^{\mathbf{W}}\hat{Z}_i \\ \hat{\lambda} \end{bmatrix} = \begin{bmatrix} {}^{\mathbf{W}}\hat{X}_i \\ {}^{\mathbf{W}}\hat{Y}_i \\ {}^{\mathbf{W}}\hat{Z}_i \\ 1 \end{bmatrix}, \tag{2.38} $$

where $\hat{\lambda}$ is the fourth element of ${}^{\mathbf{W}}\hat{\mathbf{P}}_i^u$.

The procedure for estimating the location of the feature points is described in Algorithm 1. It is assumed that the projection matrix (Section 2.2.4) for each camera view is calculated in advance. The algorithm attempts to find ${}^{\mathbf{W}}\hat{\mathbf{P}}_i$ such that it minimizes variance of the

Algorithm 1 3D Point Estimation from Multiple Camera Views

for $i = 1, \ldots, n_p$ **do**
  Assign Q with data from ${}^{\mathbf{I}_k}\mathbf{p}_i$ and M_k, $k = 1, \ldots, m_c$ (Eqn. (2.37)).
  Solve $Q^{\mathbf{W}}\mathbf{P}_i = 0$ using SVD and set ${}^{\mathbf{W}}\hat{\mathbf{P}}_i^u$.
  ${}^{\mathbf{W}}\hat{\mathbf{P}}_i = \frac{1}{\hat{\lambda}}{}^{\mathbf{W}}\hat{\mathbf{P}}_i^u$.
end for

coordinate distances, i.e. the u and v distances, between the projection of ${}^{\mathbf{W}}\hat{\mathbf{P}}_i$ and the i^{th} feature point in all camera views.

2.5 Image Segmentation

Image segmentation refers to the process of segmenting the pixels from an image into regions. The idea behind this approach is to simplify or represent the image in a way that reveals relevant information for solving a problem. An image can be segmented into regions in order to find objects or boundaries of the objects. It can also be segmented for the purpose of artistically enhancing the image. For example, an image of a road intersection can be segmented into regions that represent the road and number of vehicles that are on the road, for surveying purposes. Another example involves modifying the image by first partitioning and then removing textural information from the image to give

it an artistic, water-painted look [64]. Although image segmentation is often used as an intermediate step in an algorithm, it is a vital tool used to interpret and represent an image in a way that makes it easier to solve an application.

The basis for using image segmentation in the proposed algorithm is to determine regions representative of stationary objects. Since the objects in the environment are not known a priori, some generalizing assumptions are made on the features that define the majority of the possible objects in the scene. The first assumption is that various objects tend to have grainy or textural details and the second assumption is that pieces of an object are uniform, or similar, in colour in the broader sense (e.g. a house built with red bricks appears red overall). Using these assumptions, the segmentation section for the algorithm is divided into two steps. The first step is to remove textural information in the image and the second step is to find regions that are chromatically (colour) and spatially similar. These two steps are achieved first by applying the Kuwahara filter [65] to the image and then using mean shift segmentation [66] on the filtered image, which in turn returns a map of regions that are similar in colour. The regions found are assumed to be representative of an object or a part of an object. The surfaces for these objects are interpolated in Section 2.6 using the 3D point cloud calculated from Section 2.4, providing a more elaborate and informative map of the environment. The Kuwahara filter and the mean shift segmentation technique are described in the following sections.

2.5.1 Kuwahara Filter

The Kuwahara filter [65] is used prior to segmenting the image in order to improve the segmentation process. It is an edge-preserving, non-linear smoothing image filter. Edge-preserving image filters are filters which maintain the edgels on an image, while a non-linear image filter has an output that is not a linear function of its input (i.e. not a filter that is convolved with the image to produce a filtered image). A smoothing filter is a filter that is used to blur the image (to blend the colour information of neighbouring pixels in an image). The filter's purpose is to first reduce texture details, while maintaining the edge information, to decrease the likelihood of *oversegmentation* (i.e. producing too many segments from an image) from mean shift segmentation (Section 2.5.2).

Some additional mathematical notation for images will be defined first prior to the overview of the filter. Let $I(i,j) = \left\{ \mathbf{x}^r_{i,j} \in \mathbb{R}^p \,|\, i = 1, \ldots, m; j = 1, \ldots, n \right\}$ be a general image map, where the index variables i and j are the row and column indices of the image map respectively. The image row and column size are denoted by m and n. An image is comprised of $m \times n$ pixels, where the variable that stores the value for the pixel in the i^{th} row and j^{th} column is denoted by $\mathbf{x}^r_{i,j}$. The two-dimensional space spanned by the pixel coordinates for the image is known as the *spatial domain* and the pixel value itself, $\mathbf{x}^r$, spans a p-dimensional space known as the *range domain*. Each dimension of $\mathbf{x}^r$

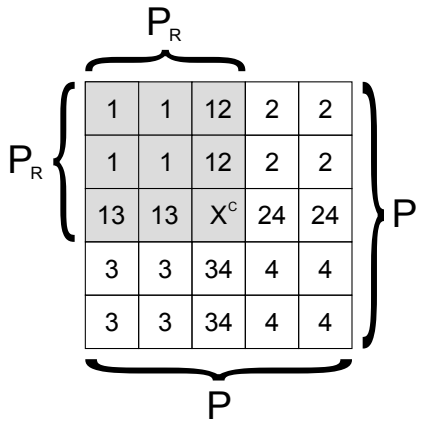


Figure 2.14: Diagram of a 5x5 Kuwahara filter window. The numbers 1, 2, 3 and 4 denote the regions, where overlapping regions are denoted by multiple numbers in a given pixel. Region 1 is shaded in grey. $\mathbf{x}^c$ denotes the sample point and P and P_R denotes the kernel and region widths respectively. The sample point also contains all four regions, though not depicted in the figure.

constitutes for each distinct component (channel) of the colour space represented by the image. Therefore, p is dependent on the colour space of the image. For example, the value of p is one for greyscale images and three for images in the HSV (Hue, Saturation, and Value) colour space. A superscript following the variable I denotes the colour space of the image while a subscript denotes the type of image, with the exception that the variable I without a subscript is assumed to denote the input image. For example, I_f^{HSV} is a filtered image that is in HSV colour space (the pixel values have three components: H, S, and V) and I^{g} denotes an input image that is in greyscale. The Kuwahara filter described here analyses the image in the RGB and HSV colour spaces to produce the filtered image.

While other versions of the Kuwahara filter exist [67], the traditional Kuwahara filter is presented here. The filter uses a square kernel (i.e. a square window by which the filtering takes place, that operates over the entire image) as shown in Figure 2.14. The kernel is divided into four overlapping regions (1, 2, 3 and 4, denoted by R_1, R_2, R_3 and R_4 respectively) with the center, $\mathbf{x}^c$, being the sample point (or the pixel of interest) on the image. The pixels which the regions overlap are shown as pixels having more than one number, indicative which regions the pixel contains. Region 1, or R_1, is shown in Figure 2.14 shaded in grey. It is noted that $\mathbf{x}^c$ contains all four regions, which is not depicted in the figure due to space constraints. There are always four overlapping regions for the traditional kernel, each covering the four corners of the kernel. The width of the kernel is denoted by $P \in \mathbb{N}_1$ while the width of the region is denoted by $P_R \in \mathbb{N}_1$.

The width of the kernel and region is given by

$$P = 4L + 1 \tag{2.39}$$

and

$$P_R = 2L + 1, \tag{2.40}$$

where $L \in \mathbb{N}_1$ is a parameter for controlling the kernel size. The area, A_R, of each region is given by $A_R = (P_R)^2$.

The Kuwahara filter requires that the RGB component of the mean vector, $\mu_k^{\mathrm{RGB}}(R_k) : \mathbb{R}^{p+2} \to \mathbb{R}^p$, and the V component of the variance, $\mathrm{var}_k^{\mathrm{V}}(R_k) : \mathbb{R}^3 \to \mathbb{R}$, be calculated for each region, where $k = 1, \ldots, 4$ is the region number. The equations for finding μ_k^{RGB} and $\mathrm{var}_k^{\mathrm{V}}$ are

$$\mu_k^{\mathrm{RGB}}(R_k) = \frac{1}{A_R} \sum_{\mathbf{x}^{r,\mathrm{RGB}} \in R_k} \mathbf{x}^{r,\mathrm{RGB}} \tag{2.41}$$

and

$$\mathrm{var}_k^{\mathrm{V}}(R_k) = \frac{1}{A_R - 1} \sum_{\mathbf{x}^{r,\mathrm{V}} \in R_k} \left(\mathbf{x}^{r,\mathrm{V}}\right)^2 - \frac{1}{A_R^2 - A_R} \left(\sum_{\mathbf{x}^{r,\mathrm{V}} \in R_k} \mathbf{x}^{r,\mathrm{V}} \right)^2, \tag{2.42}$$

where $\mathbf{x}^{r,\mathrm{RGB}}$ and $\mathbf{x}^{r,\mathrm{V}}$ is the RGB and V component, respectively, of $\mathbf{x}^r$.

The *Kuwahara filter procedure* is described in Algorithm 2. It is assumed that the input image is in the RGB colour space and can be converted into the HSV colour space.

Algorithm 2 Kuwahara Filter

```
for i = 1, ..., m do
  for j = 1, ..., n do
    Set x^c = I^RGB(i, j)
    for k = 1, ..., 4 do
      Compute μ_k^RGB(R_k) from Eqn. (2.41)
      Compute var_k^V(R_k) from Eqn. (2.42)
    end for
    l = arg min_k var_k^V
    Set I_f^RGB(i, j) = μ_l^RGB
  end for
end for
```

The procedure looks at each pixel in the input image, and selects the neighbouring region with the lowest variance in the Value channel as the region that has the least contrast. This is indicative that the region is least likely to lie on an edge or corner. The

pixel value in the filtered image is chosen to be the mean of the selected region in order to add the smoothing effect. The end result from applying this procedure is a smoothing effect, while preserving the edge information, of an image.

2.5.2 Mean Shift Segmentation

Mean shift segmentation [66] is a popular image segmentation technique, known for its robust performance and few tunable parameters. The algorithm is derived from mean shift clustering [68] and performs the clustering based on both colour and spatial data from the image. The mean shift clustering technique differs from traditional feature space based clustering techniques in that it does not impose a general shape for the clusters, such as an elliptical shape resulting from modelling multivariate normal distributions to the datasets [69]. A brief outline on the derivation for the algorithm is described in the following subsections.

Mean Shift Clustering

Mean shift clustering is a nonparametric clustering algorithm (i.e. a method which does not rely on a particular distribution profile to form groups in a dataset of points) introduced by Fukunaga et al. [68] and later utilized by Comanicu et al. [66] in the context of image segmentation. The main idea of the method is to find local maxima, or modes, of densely populated regions in a given feature space. A feature space is the space spanned by a feature vector, and a feature vector is a vector whose dimensions are features that describe a piece of information which is used to solve certain applications. An example of a feature vector is a vector formed in the RGB (Red, Green, and Blue) colour space of an image, where the dimensions of the vector denotes the numerical value of the R, G and B channels from a given pixel.

The density of a point in the feature space is estimated with a probability density estimator function, denoted by $\hat{f}(\mathbf{x})$. A gradient ascent approach (an approach which is used in optimization problems in which a local maxima is found by recursively shifting the current solution in the direction of maximum increase of the cost function until convergence) is used to find the modes, and any point that is connected to the modes is treated as being part of the same cluster (group). A brief overview of the algorithm is described below.

Given a set of n features, $X = \left\{\mathbf{x}_i \in \mathbb{R}^d | i = 1, \ldots, n\right\}$, in a d-dimensional feature space, the kernel density estimator, $\hat{f}\left(\mathbf{x}\right) : \mathbb{R}^d \to \mathbb{R}$, at an arbitrary point $\mathbf{x} \in \mathbb{R}^d$ is:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \tag{2.43}$$

where $h \in \mathbb{R}^+$ is the bandwidth parameter and $K(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ is the d-variate kernel (window) centered about $\mathbf{x}$. The purpose for the kernel is to define the neighbourhood around the point $\mathbf{x}$ in which operations are performed on points within that neighbourhood.

A radially symmetric kernel (i.e. a circular window) is defined as a class of functions of the form:

$$K(\mathbf{x}) = c_{k,d} k\left(||\mathbf{x}||^2\right), \tag{2.44}$$

where $c_{k,d} \in \mathbb{R}^+$ is a normalization constant that makes $K(\mathbf{x})$ integrate to one and $k(x)$ is the *profile* for the kernel. The normalization constant, $c_{k,d}$, is dependent on k and d. Note that for radially symmetric kernels, h can also be defined as the radius of the kernel.

While other kernel profiles can be used for the density estimation, the Epanechnikov ($k_E(x) : \mathbb{R}^+ \cup \{0\} \to [0,1]$) and normal ($k_N(x) : \mathbb{R}^+ \cup \{0\} \to [0,1]$) profiles are commonly used [66]. They are defined as:

$$k_E(x) = \begin{cases} 1 - x & 0 \leq x \leq 1 \\ 0 & x > 1 \end{cases} \tag{2.45}$$

$$k_N(x) = \exp\left(-\frac{1}{2}x\right) \qquad x \geq 0 \tag{2.46}$$

where $x \in \mathbb{R}^+ \cup \{0\}$.

Rewriting Eqn. (2.43) with the kernel definition from Eqn. (2.44) yields:

$$\hat{f}_K(\mathbf{x}) = \frac{c_{k,d}}{nh^d} \sum_{i=1}^{n} k\left(\left|\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right|^2\right), \tag{2.47}$$

where $\hat{f}_K$ is the density estimator function which adopts the radially symmetric kernel definition.

Let $g(x) = -\frac{dk(x)}{dx}$ and $G(\mathbf{x}) = c_{g,d} g\left(||\mathbf{x}||^2\right)$, where $c_{g,d}$ is the normalization constant that makes $G(\mathbf{x})$ integrate to one. Therefore the radially symmetric kernel G uses the negative derivative of k as its profile. The gradient of $\hat{f}_K(\mathbf{x})$ can be written as

$$\begin{aligned} \nabla \hat{f}_K(\mathbf{x}) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{x}) g\left(\left|\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right|^2\right) \\ &= \underbrace{\frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|\right|^2\right)\right]}_{\text{Term 1}} \underbrace{\left[\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right)}{\sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right)} - \mathbf{x}\right]}_{\text{Term 2}}, \end{aligned} \tag{2.48}$$

and replacing the kernel G for K in Eqn. (2.43) results in:

$$\hat{f}_G(\mathbf{x}) = \frac{c_{g,d}}{nh^d} \sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right). \tag{2.49}$$

Note that $\hat{f}_G$ is similar to $\hat{f}_K$, but instead uses the kernel profile $g(x)$ which is the negative derivative of the profile $k(x)$. It is assumed that there exists a derivative for $k(x)$ for all $x \in [0, \inf)$.

The first term of Eqn. (2.48) is proportional to the density estimate at $\mathbf{x}$ using the kernel G, and $\sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right)$ is assumed to be a positive number. The second term is the mean shift vector ($\mathbf{m}_G(\mathbf{x}) \in \mathbb{R}^d$):

$$\mathbf{m}_G(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right)}{\sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right)} - \mathbf{x}, \tag{2.50}$$

which is the difference between the weighted mean (using the kernel G for weights) and $\mathbf{x}$. Substituting Eqn. (2.49) and Eqn. (2.50) to Eqn. (2.48) and simplifying with respect to $\mathbf{m}_G(\mathbf{x})$ gives:

$$\mathbf{m}_G(\mathbf{x}) = \frac{1}{2} h^2 c \frac{\nabla \hat{f}_K(\mathbf{x})}{\hat{f}_G(\mathbf{x})}, \tag{2.51}$$

where $c = \frac{c_{g,d}}{c_{k,d}}$. Therefore from Eqn. (2.51), the mean shift vector always points at the gradient direction (i.e. direction of maximum increase in density), with $\hat{f}_G(\mathbf{x})$ serving to help normalize the step size [66]. The magnitude of the step size is kept within the size of the kernel.

The *mean shift procedure* for a given point $\mathbf{x}$ and a dataset X is defined in the following steps:

1. Compute the mean shift vector $\mathbf{m}_G$ at the current point $\mathbf{x}$.
2. Translate the point $\mathbf{x}$ (or the window center) by the mean shift vector: $\mathbf{x} = \mathbf{x} + \mathbf{m}_G$
3. Repeat steps 1 and 2 until convergence (i.e. $\nabla \hat{f}_{h,K}(\mathbf{x}) = 0$)

The mean shift vector always points towards the direction of maximum increase in density [66]. By constantly shifting the window by the mean shift vector, the window will eventually converge to a local stationary point. Figure 2.15 illustrates the first three iterations of the mean shift procedure. The kernel (circular window) shifts towards the

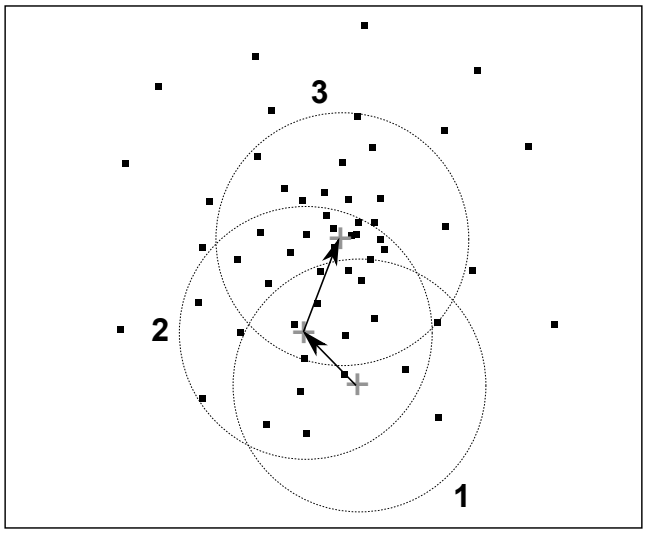


Figure 2.15: Mean shift procedure illustration for a two-dimensional case. The numbers denote the iteration number, the arrows are the mean shift vectors. The kernel (circular window) is shown as the dotted circle with the center point, $\mathbf{x}$, marked with a plus sign. The kernel shifts towards the densest region on each iteration.

densest region on each iteration. The set of points that converges to the same mode is called the *basin of attraction* of that mode.

Steps 1 and 2 from the mean shift procedure can be reduced to one step. Let $\{\mathbf{x}_t\}, t = 1, 2, \ldots$ be the sequence of successive kernel center locations, with t being the iteration number. Taking Eqn. (2.50) and adding $\mathbf{x}_t$ to obtain the next location, $\mathbf{x}_{t+1}$, yields:

$$\mathbf{x}_{t+1} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left|\left|\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right|\right|^2\right)}{\sum_{i=1}^{n} g\left(\left|\left|\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right|\right|^2\right)} \quad t = 1, 2, \ldots \tag{2.52}$$

Not all stationary points found from the mean shift procedure above point to a definitive local maxima (mode). A practical procedure for mode detection was detailed in [66] and briefly described below:

1. Use the *mean shift procedure* to find all stationary points ($\nabla \hat{f}_K(\mathbf{x}) = 0$).

2. Prune these points by finding the local maxima. This is done by shifting each stationary point by a random vector with a small magnitude, re-applying the mean shift procedure to each point again and letting it converge. If the point of convergence

remains unchanged (up to a tolerance) then the point is a local maximum, otherwise repeat the process until all of the points reach a local maximum.

Additional details about sufficient conditions for convergence, smooth trajectory property, and bandwidth selection for the procedure are explained in [66].

Mean Shift Filter

An image feature space is divided into two components: the spatial (pixel location) and range (colour space) domains. The *mean shift filter* works in the joint spatial-range domains, for both gray scale and colour images. Since the range domain is conceptually different than the spatial domain, a separate Euclidean metric is used for each domain. The L*u*v* colour space [70] is used in [66] for the filtering process because it is approximately uniform with a linear mapping property. Therefore, a Euclidean metric is assumed when using this colour space.

A multivariate kernel notation is defined to compensate for normalizing both domain spaces:

$$K_{h_s,h_r}(\mathbf{x}) = \frac{C}{h_s^2 h_r^p} k\left(\left|\left|\frac{\mathbf{x}^s}{h_s}\right|\right|^2\right) k\left(\left|\left|\frac{\mathbf{x}^r}{h_r}\right|\right|^2\right), \tag{2.53}$$

where $h_s \in \mathbb{R}^+$ and $h_r \in \mathbb{R}^+$ are the kernel bandwidths for the spatial and range spaces respectively. The corresponding spatial and range portion in the feature vector $\mathbf{x} = (\mathbf{x}^s, \mathbf{x}^r)^T$ are $\mathbf{x}^s \in \mathbb{R}^2$ and $\mathbf{x}^r \in \mathbb{R}^p$. The dimension of the range domain is denoted by p. $C \in \mathbb{R}^+$ is the normalization constant and $k(x)$ is the common kernel profile used for both spatial and range spaces.

Let $\{\mathbf{x}_i \in \mathbb{R}^{p+2} \,|i = 1, \ldots, n\}$ be the i^{th} input image pixel, and n is the number of pixels. The point of convergence for $\mathbf{x}_i$ is denoted by $\mathbf{x}_{i,c} = \left(\mathbf{x}_{i,c}^s, \mathbf{x}_{i,c}^r\right)^T$, where $\mathbf{x}_{i,c}^s$ and $\mathbf{x}_{i,c}^r$ are the spatial and range components for $\mathbf{x}_{i,c}$. Let $\mathbf{x}_{i,t}$ be the t^{th} iteration of pixel $\mathbf{x}_i$ governed by Eqn. (2.52), so the symbol c represents the iteration at which the pixel $\mathbf{x}_i$ converges. Also, let $\{\mathbf{z}_i \in \mathbb{R}^{p+2} \,|i = 1, \ldots, n\}$ be the i^{th} filtered image pixel.

The *mean shift filter procedure* is described in Algorithm 3.

The filter performs the *mean shift procedure* on every pixel in the image and assigns each pixel the colour of its mode. The value for ϵ should be chosen to be small for sufficient convergence.

Mean Shift Segmentation

The final step is to determine homogeneous regions based on the filter result. The homogeneous regions are defined here as being distinctly outlined (delineated) regions with pixels

Algorithm 3 Mean Shift Filter Procedure

for $i = 1, \ldots, n$ **do**
  Initialize $t = 1$, $\mathbf{x}_{i,1} = \mathbf{x}_i$, $\mathbf{x}_i \in I$
  repeat
    $t = t + 1$
    Compute $\mathbf{x}_{i,t}$ from Eqn. (2.52)
  until $||\mathbf{x}_{i,t} - \mathbf{x}_{i,t-1}|| < \epsilon$, $\epsilon \in \mathbb{R}^+$
  $\mathbf{x}_{i,c} = \mathbf{x}_{i,t}$
  Assign $\mathbf{z}_i = \left(\mathbf{x}_i^s, \mathbf{x}_{i,c}^r\right)^T$, $\mathbf{z}_i \in I_f$
end for

which are similar in colour. The mode $\mathbf{x}_{i,c}$ of each pixel $\mathbf{x}_i$ in the input image I is determined using the mean shift filter procedure. But instead of assigning the pixel colour with that of its mode's, the pixel is assigned a label, $L_i = \{k | \mathbf{x}_{i,c} \in \mathbf{C}_k\}, i = 1, \ldots, n$, where i is the pixel index and n is the number of pixels. $\{\mathbf{C}_k\}$ is the set of clusters (groups) determined from the input image, where $k = 1, \ldots, l$ is the cluster index and l is the total number of clusters. The clusters are found by grouping the modes together based on their distance to one another. If they are within the distance of their bandwidths, then they are classified as being in the same cluster. This can be done by using the *single linkage clustering method*, also known as the *nearest neighbour method*, which is described in [71].

Let the minimum distance between two clusters, $D_{\mathbf{C}}(\mathbf{C}_1, \mathbf{C}_2)$, be defined by:

$$D_{\mathbf{C}}(\mathbf{C}_1, \mathbf{C}_2) = \min_{\mathbf{x}_1 \in \mathbf{C}_1, \mathbf{x}_2 \in \mathbf{C}_2} d_E(\mathbf{x}_1, \mathbf{x}_2), \tag{2.54}$$

where $\mathbf{C}_1$ and $\mathbf{C}_2$ are two different clusters, and $\mathbf{x}_1$ and $\mathbf{x}_2$ are the features in clusters $\mathbf{C}_1$ and $\mathbf{C}_2$, respectively. The Euclidean distance metric, d_E, is from Eqn. (2.30). Extending from Eqn. (2.54), the minimum spatial and range distance between clusters are defined as

$$D_{\mathbf{C}}^s(\mathbf{C}_1, \mathbf{C}_2) = \min_{\mathbf{x}_1 \in \mathbf{C}_1, \mathbf{x}_2 \in \mathbf{C}_2} d_E(\mathbf{x}_1^s, \mathbf{x}_2^s) \tag{2.55}$$

and

$$D_{\mathbf{C}}^r(\mathbf{C}_1, \mathbf{C}_2) = \min_{\mathbf{x}_1 \in \mathbf{C}_1, \mathbf{x}_2 \in \mathbf{C}_2} d_E(\mathbf{x}_1^r, \mathbf{x}_2^r), \tag{2.56}$$

respectively.

The *mean shift segmentation procedure* is described in Algorithm 4.

The procedure merges the modes, or merging the basin of attractions, that are adequately close to each other forming l clusters. Figure 2.16 illustrates a clustering example. Each pixel is labelled with their corresponding cluster index. An additional (optional) step to the procedure is to eliminate spatial regions containing less than M pixels, removing

Algorithm 4 Mean Shift Segmentation Procedure

for $i = 1, \ldots, n$ **do**
  Initialize $t = 1$, $\mathbf{x}_{i,1} = \mathbf{x}_i$, $\mathbf{x}_i \in I$
  repeat
    $t = t + 1$
    Compute $\mathbf{x}_{i,t}$ from Eqn. (2.52)
  until $||\mathbf{x}_{i,t} - \mathbf{x}_{i,t-1}|| < \epsilon$, $\epsilon \in \mathbb{R}^+$
  $\mathbf{x}_{i,c} = \mathbf{x}_{i,t}$
end for
Initialize $\mathbf{x}_{i,c} \in \mathbf{C}_i, \forall i$
repeat
  for each $\mathbf{C}_i \in \{\mathbf{C}\}$ **do**
    for each $\mathbf{C}_j \in \{\mathbf{C}\}, i \neq j$ **do**
      Compute $d^s = D^s_{\mathbf{C}}(\mathbf{C}_i, \mathbf{C}_j)$ from Eqn. (2.55)
      Compute $d^r = D^r_{\mathbf{C}}(\mathbf{C}_i, \mathbf{C}_j)$ from Eqn. (2.56)
      if $d^s \leq h_s$ and $d^r \leq h_r$ **then**
        Merge $\mathbf{C}_i$ and $\mathbf{C}_j$ and update $\{\mathbf{C}\}$
      end if
    end for
  end for
until $D^s_{\mathbf{C}}(\mathbf{C}_i, \mathbf{C}_j) > h_s$ and $D^r_{\mathbf{C}}(\mathbf{C}_i, \mathbf{C}_j) > h_r, \forall \mathbf{C}_i \in \{\mathbf{C}\}, \forall \mathbf{C}_j \in \{\mathbf{C}\}, i \neq j$
for $i = 1, \ldots, n$ **do**
  $L_i = \{k | \mathbf{x}_{i,c} \in \mathbf{C}_k\}$
end for

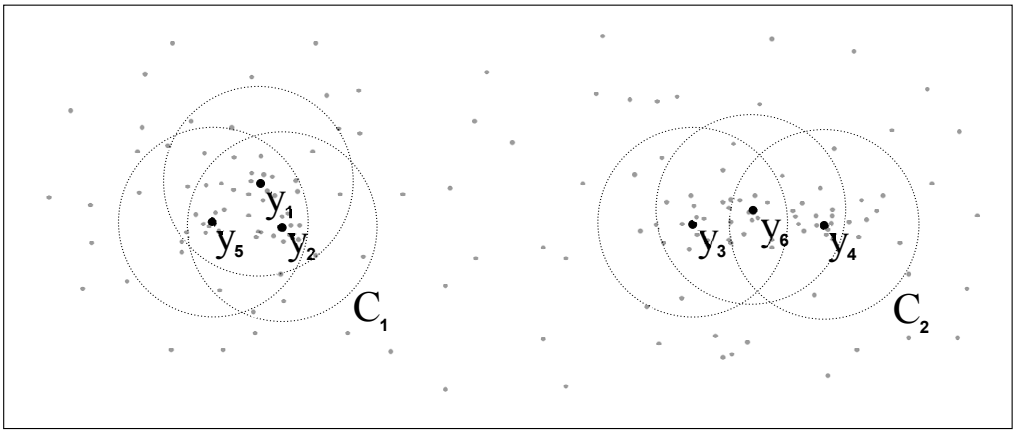


Figure 2.16: Clustering illustration for the mean shift segmentation procedure. The modes $\mathbf{y}_1$, $\mathbf{y}_2$, and $\mathbf{y}_5$ are grouped to the cluster $\mathbf{C}_1$, while the modes $\mathbf{y}_3$, $\mathbf{y}_4$, and $\mathbf{y}_6$ are grouped to the cluster $\mathbf{C}_2$. The kernel is shown in the illustration as a dotted circle, with the radius of the circle being h, the bandwidth.

small segments that could be by-products of image noise. The spatial and range bandwidths (h_s and h_r), as well as the minimum region pixel size (M), are used as parameters for the segmentation algorithm.

This method is an extension to the mean shift filtering procedure, which merges the set of modes based on their proximity to each other, forming larger regions that have more distinct boundaries. In other words, the fusion technique combines regions that are similar in colour and close to one another. Refer to [66] for further details and experimental results.

2.6 Surface Fitting

The 3D point cloud generated in Section 2.4 may consist of points belonging to several obstacles. A possible method to identify these obstacles is to use the region information gathered from image segmentation (see Section 2.5) to label feature points that are associated with each region. This method looks at the last segmented image (the last camera "snapshot") and determines which tracked feature points from the image lie in which region. The points in the 3D point cloud that correspond to those feature points are labelled to their respective regions. The points with the same label are treated as being a complete surface or part of a surface for an object. Figure 2.17 illustrates this labelling process. In the proposed algorithm, in order to fit surfaces from these points an assumption is made that these surfaces are flat in order to simplify the problem and speed up the algorithm.

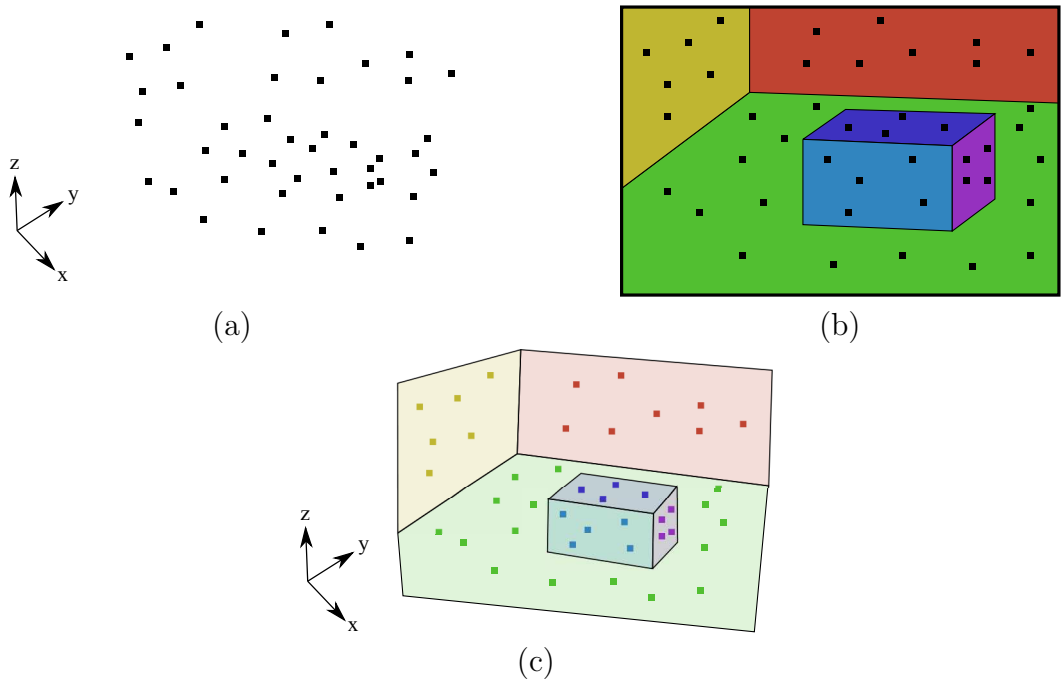


Figure 2.17: Illustration of the 3D point cloud labelling process based on the last segmented image and its corresponding tracked feature points. (a) The 3D point cloud. (b) The last segmented image with SIFT features (black squares) overlaid. (c) The 3D points associated with the features are labelled to the region which these features are part of. The faint visuals of the room and box are added for clarity.

More complicated surfaces can be fitted to the set of points, such as B-Spline or cubic-spline surfaces, however these surface interpolation methods will be a subject of future investigation to further improve the algorithm.

This section outlines the two steps of the surface fitting method given a set of points in 3D space, and its corresponding region in the last segmented image, that represents an obstacle surface. The first step is to obtain the plane that best fits these points, using an orthogonal distance regression plane method described in Section 2.6.1, and the second step is to project the boundary of the region from the image map onto the plane of best fit, which is discussed in Section 2.6.2. These steps are done for all segments in the last segmented image that have tracked feature points (at least three feature points are required for planar fitting) which are matched to them, resulting in a more detailed environment map of potential obstacles.

2.6.1 Orthogonal Distance Regression Plane

The orthogonal distance regression plane is a plane which minimizes the perpendicular distances from a set of 3D points to that plane. A brief derivation is outlined in this section. Let $X = \{\mathbf{x}_i \in \mathbb{R}^3 \,|i = 1, \ldots, n\}$ be a set of 3D points and $\mathbf{L} = (a, b, c, d)^T \in \mathbb{R}^4$ be a plane that is fitted to those points, which is defined by the following equation:

$$ax_{\mathbf{L}} + by_{\mathbf{L}} + cz_{\mathbf{L}} + d = 0, \tag{2.57}$$

where the point $\mathbf{x}_{\mathbf{L}} = (x_{\mathbf{L}}, y_{\mathbf{L}}, z_{\mathbf{L}})^T$ denotes any point that lies on the plane $\mathbf{L}$. The objective is to find the parameters of $\mathbf{L}$ (a, b, c, and d) such that the summation of the shortest distances from $\{X\}$ to the plane $\mathbf{L}$ is minimized.

Finding the shortest distance between a 3D point and a plane is derived using vector manipulation, similar to the 2D point to line distance derivation to obtain Eqn. (2.32). Let $\mathbf{n}_{\mathbf{L}} = (a, b, c)^T$ be the normal vector of the plane and $\mathbf{x}_0 = (x_0, y_0, z_0)^T$ be an arbitrary point in 3D space. The vector from the plane $\mathbf{L}$ to $\mathbf{x}_0$ is defined as $\mathbf{w} = (x_0 - x_{\mathbf{L}}, y_0 - y_{\mathbf{L}}, z_0 - z_{\mathbf{L}})^T$. The shortest distance, $d_{\mathbf{L}}(\mathbf{x}_0, \mathbf{L})$, from the point $\mathbf{x}_0$ to the plane $\mathbf{L}$ can be found by a scalar projection of $\mathbf{w}$ onto $\mathbf{n}_{\mathbf{L}}$ and further simplified by the substitution of Eqn. (2.57):

$$\begin{aligned}
d_{\mathbf{L}}(\mathbf{x}_0, \mathbf{L}) &= \left|\left|\text{proj}_{\mathbf{n}_{\mathbf{L}}}\mathbf{w}\right|\right| \\
&= \frac{||\mathbf{n}_{\mathbf{L}} \cdot \mathbf{w}||}{||\mathbf{n}_{\mathbf{L}}||} \\
&= \frac{||(a, b, c) \cdot (x_0 - x_{\mathbf{L}}, y_0 - y_{\mathbf{L}}, z_0 - z_{\mathbf{L}})||}{||(a, b, c)||} \\
&= \frac{||a(x_0 - x_{\mathbf{L}}) + b(y_0 - y_{\mathbf{L}}) + c(z_0 - z_{\mathbf{L}})||}{\sqrt{a^2 + b^2 + c^2}} \\
&= \frac{||ax_0 + by_0 + cz_0 - ax_{\mathbf{L}} - by_{\mathbf{L}} - cz_{\mathbf{L}}||}{\sqrt{a^2 + b^2 + c^2}} \\
&= \frac{||ax_0 + by_0 + cz_0 + d||}{\sqrt{a^2 + b^2 + c^2}}.
\end{aligned} \tag{2.58}$$

A minimization of the total squared distance between the set of points, $\{X\}$, and the plane $\mathbf{L}$ is used to find the plane that best fits those points. Using the distance formula from Eqn. (2.58), the summation of the squared distances to a plane is given by the following equation:

$$E(\{X\}, \mathbf{L}) = \sum_{i=1}^{n} \left(\frac{ax_i + by_i + cz_i + d}{\sqrt{a^2 + b^2 + c^2}}\right)^2, \tag{2.59}$$

where $E(\{X\}, \mathbf{L})$ is defined as the total squared distance error from $\{X\}$ to $\mathbf{L}$. If $E(\{X\}, \mathbf{L}) = 0$, that means the plane lies perfectly on the set of points (all of the points

have a distance of zero to $\mathbf{L}$. Therefore finding the plane of best fit means solving the following the minimization problem:

$$\min_{\mathbf{L}} E\left(\{X\}, \mathbf{L}\right). \tag{2.60}$$

Taking the partial derivative of E with respect to d, setting it to zero, and then solving for d yields:

$$\begin{aligned}
\frac{\partial E}{\partial d} &= \left(\frac{2}{a^2+b^2+c^2}\right)\sum_{i=1}^{n}\left(ax_i+by_i+cz_i+d\right) \\
&= 0 \\
d &= \frac{\sum_{i=1}^{n}\left(-ax_i-by_i-cz_i\right)}{n} \\
&= -\left(ax_\mu+by_\mu+cz_\mu\right) \\
&= -\left(\mathbf{n}\cdot\mathbf{x}_\mu\right)
\end{aligned} \tag{2.61}$$

where $\mathbf{x}_\mu = (x_\mu, y_\mu, z_\mu) \in \mathbb{R}^3$ is the centroid of the dataset, $\{X\}$. The substitution of d, from Eqn. (2.61), into Eqn. (2.59) results in:

$$E\left(\{X\}, \mathbf{L}\right) = \sum_{i=1}^{n}\left(\frac{a\left(x_i-x_\mu\right)+b\left(y_i-y_\mu\right)+c\left(z_i-z_\mu\right)}{\sqrt{a^2+b^2+c^2}}\right)^2. \tag{2.62}$$

Let

$$A = \begin{bmatrix} x_1-x_\mu & y_1-y_\mu & z_1-z_\mu \\ x_2-x_\mu & y_2-y_\mu & z_2-z_\mu \\ \vdots & \vdots & \vdots \\ x_n-x_\mu & y_n-y_\mu & z_n-z_\mu \end{bmatrix}. \tag{2.63}$$

Rewriting Eqn. (2.62) into matrix notation results in:

$$E\left(\{X\}, \mathbf{L}\right) = \frac{\left(\mathbf{n_L}\right)^T\left(A\right)^T A\mathbf{n_L}}{\left(\mathbf{n_L}\right)^T\mathbf{n_L}}. \tag{2.64}$$

Eqn. (2.64) is in the form known as the Rayleigh Quotient or Rayleigh-Ritz ratio [72]. It can be minimized by finding the eigenvector of A with the minimum eigenvalue, which can be determined through Singular Value Decomposition (SVD). The solution is assigned to $\mathbf{n_L}$ and d can be determined by substituting the values of $\mathbf{n_L}$ into Eqn. (2.61).

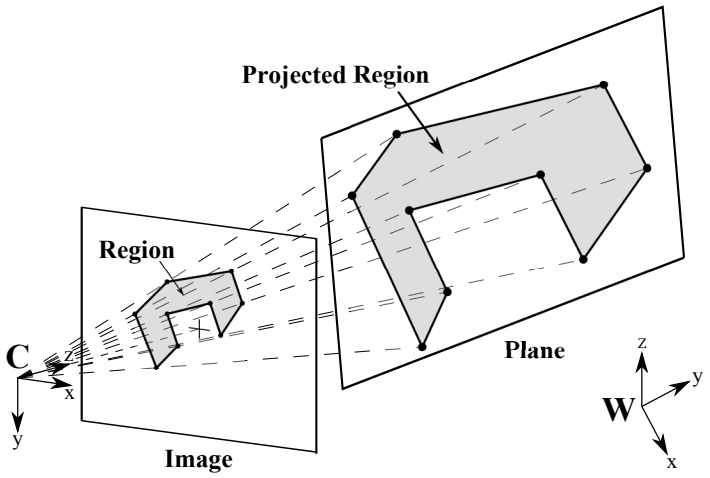


Figure 2.18: Illustration of a region boundary (defined by the points surrounding the region) in an image being projected onto a plane in 3D space. The camera and world frames are denoted by **C** and **W**.

2.6.2 Projection of Image Point to Plane

Using the plane **L** that is fitted to the set of points, described in Section 2.6.1, and the region associated with the set of points, a reverse projection is done to project the boundary of that region to the plane of best fit (see Figure 2.18). The boundary of a region is defined by a set of image points surrounding the region, and can be found using the Moore-Neighbor tracing algorithm modified by Jacob's stopping criteria [73]. From Eqn. (2.20) it is noted that $M\left({}^{\mathbf{W}}\mathbf{P}\right)$ is a ray originating from the camera center pointing towards the object point ${}^{\mathbf{W}}\mathbf{P}$. In the case of a reverse projection of an image point ${}^{\mathbf{I}}\mathbf{p}$ to a 3D point on the plane **L**, a modified version of Eqn. (2.20) is introduced:

$$
\begin{aligned}
{}^{\mathbf{I}}\mathbf{p} &= \frac{1}{\lambda_P} K \left({}^{\mathbf{C}}_{\mathbf{W}}R\right) \left[I_{3x3} \mid \left({}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t}\right)\right] \left({}^{\mathbf{W}}\mathbf{P}\right) \\
\begin{bmatrix} {}^{\mathbf{W}}X \\ {}^{\mathbf{W}}Y \\ {}^{\mathbf{W}}Z \end{bmatrix} &= \lambda_P \left({}^{\mathbf{C}}_{\mathbf{W}}R\right)^T (K)^{-1} \left({}^{\mathbf{I}}\mathbf{p}\right) - {}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t},
\end{aligned} \tag{2.65}
$$

where ${}^{\mathbf{W}}X$, ${}^{\mathbf{W}}Y$ and ${}^{\mathbf{W}}Z$ are the coordinates of ${}^{\mathbf{W}}\mathbf{P}$, and λ_P is a scalar value which scales the ray $M\left({}^{\mathbf{W}}\mathbf{P}\right)$. Therefore the value of λ_P is found such that the ray touches the plane's surface. This introduces another constraint, which is that the point ${}^{\mathbf{W}}\mathbf{P}$ must lie on the

plane $\mathbf{L}$, or in other words:

$$\mathbf{L}^T \left({}^{\mathbf{W}}\mathbf{P} \right) = 0$$

$$\left(\mathbf{n_L} \right)^T \begin{bmatrix} {}^{\mathbf{W}}X \\ {}^{\mathbf{W}}Y \\ {}^{\mathbf{W}}Z \end{bmatrix} + d = 0, \tag{2.66}$$

where $\mathbf{n_L}$ is the normal vector of the plane $\mathbf{L}$. Solving for λ_P with the constraint in Eqn. (2.66) yields:

$$\lambda_P = \frac{-d + \left(\mathbf{n_L} \right)^T \left({}^{\mathbf{C}}_{\mathbf{W}}\mathbf{t} \right)}{\left(\mathbf{n_L} \right)^T \left({}^{\mathbf{C}}_{\mathbf{W}}R \right)^T (K)^{-1} \left({}^{\mathbf{I}}\mathbf{p} \right)}. \tag{2.67}$$

The solution for λ_P is substituted back into Eqn. (2.65) to find the projected point ${}^{\mathbf{W}}\mathbf{P}$ that lies on the plane $\mathbf{L}$. This method is repeated for all of the image points defining the boundary of the region.

Chapter 3

Planar Surface Reconstruction Algorithm

3.1 Motivation

The motivation for the proposed algorithm, called the Planar Surface Reconstruction (PSR) algorithm, is to take advantage of the abundant information that is inherent in an image. It does so by identifying and tracking individual persistent features, but also by using colour segmentation to define contiguous regions. As outlined in the introduction, the main drawback for pure feature tracking algorithms is that they require the detection of many features to return a meaningful 3D environment map estimate. Some surface fitting techniques have been used on the estimated feature locations to enhance the map visually, however it is still reliant on the number of features tracked in the images. Images where few features are found result in a less detailed and reliable map reconstruction. Humans, on the other hand, generally look for more than just distinct features (i.e. corners and edges) when determining distances and surface structures of the environment. Research was done on the process of how humans perceive and organize visual elements into groups, where objects (also object surfaces) can be perceived from these groups [74]. Once they perceive these objects, they can infer the objects' surface structures and sizes before determining the objects' distances based on their distinct features. Therefore the surface structure, and size, for an object and its perceived depth are interlinked.

Meaningful surfaces can be extrapolated even when given few distinct features and the surface structure linked to those features. Although this is the case, distinct features still must be identified to estimate the depth and scale information of the surface. For example, it is difficult to perceive the distance and size of a smooth wall if the four corners (distinct features) are not detectable, as shown in Figure 3.1. Similarly, only detecting and

(a) (b)

Figure 3.1: Example showing the link between object recognition and distinct features for surface and depth perception. (a) A wall (surface recognition) without any of its corners (distinct features) revealed. (b) The entire wall with the four corners.

tracking the corners of the wall says nothing about the wall itself. It is up to the human to make the connection that the four corners are linked to the wall, and extrapolate the visual information to generate a surface and range for the wall. Extending this idea to the problem of surface fitting to a 3D point cloud, additional information can be extracted from the image and correlated to the feature points. Once this link is established, various gaps between feature points can be filled more accurately and confidently, resulting in a more informative map of object surfaces rather than object points.

While determining object surfaces based on visual information is a complicated task, a simplification made in this work is to only assess the chromatic (colour) properties of the image and assume that object surfaces are similar in colour. A large majority of the area to be mapped for indoor environments is comprised of planar surfaces such as floors, walls, and ceilings, so another simplification made are that most of the surfaces are assumed to be flat. The feature points that correspond to these object surfaces are determined, and the surface fitting technique combines the feature points and object surfaces to place flat surfaces in a 3D map. A description of the PSR algorithm is defined in the following section.

Figure 3.2: Three consecutive images of a box (captured at a horizontal displacement from one another) used for the illustration of the steps in the PSR algorithm.

3.2 Algorithm Definition

The reconstruction pipeline is divided into four main steps: feature extraction, feature tracking, 3D mapping and surface fitting. The feature extraction step (Section 3.2.1) finds distinct and identifiable features from an image and the feature tracking step (Section 3.2.2) matches these features from several images, producing a list of tracked features. The tracked features are then used, in conjunction with camera information (i.e. position, orientation, and calibration parameters), to determine where they lie in a 3D map (Section 3.2.3), which is known as the 3D mapping step. The last step (Section 3.2.4) attempts to identify parts of objects based on colour information and then fit surfaces to these parts using the 3D points estimated from the previous step. The last section (Section 3.2.5) is summary of the algorithm with all of the steps integrated. The algorithm requires at least two consecutive images to work. Three example images of a box (Figure 3.2), captured a horizontal distance apart from one another, are used to illustrate the concepts for each step of the algorithm. The box from the figure used has approximate dimensions of 34cm x 43cm x 54cm.

It is assumed that the camera information is known in advance for each image. The scene is also assumed to be static (no moving objects). This assumption allows multiple view geometry techniques to be incorporated into the algorithm in order to estimate the 3D location of the feature points.

The algorithm can also be executed for video data, where a number of sequential frames, denoted by $n_{\text{frame}} \in \mathbb{N} \geq 2$, with their corresponding camera information are fed to the algorithm to return a surface map. This is done repeatedly for the following frames, and the new map information returned is added to the same map. Each frame can also be separated with the next by a constant $n_{\text{fsep}} \in \mathbb{N}$ to improve the quality of the baseline and to reduce processing loads. The fusion process of the surfaces is not evaluated in this work, although it is an area of future work.

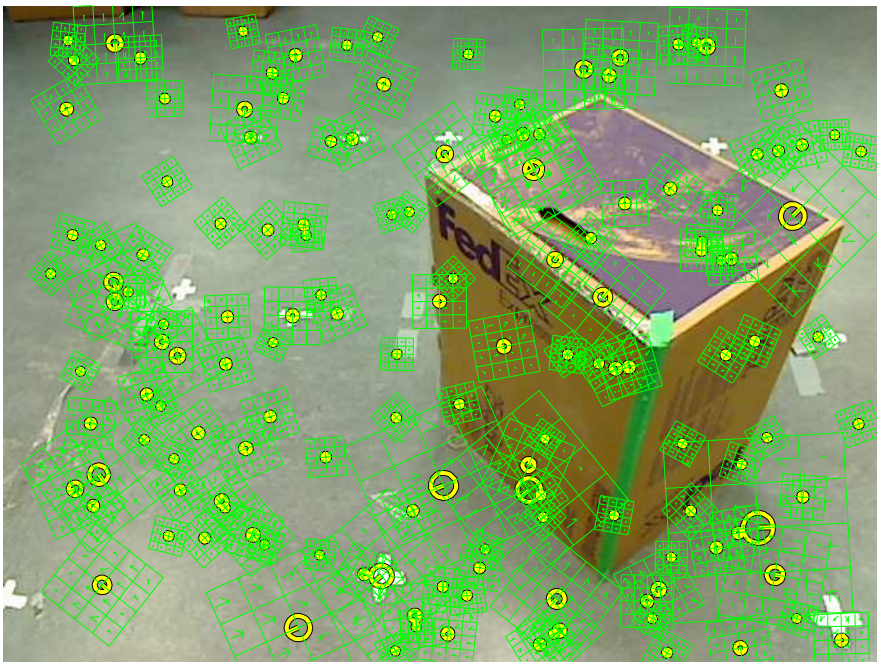


Figure 3.3: SIFT implementation example in MATLAB using VLFeat and the third image from Figure 3.2.

3.2.1 Feature Extraction

The first step for the algorithm is to extract SIFT features (see Section 2.1) from the images (converted to greyscale). It is assumed that the features found in each image are indexed and stored for future use. Figure 3.3 shows an example of SIFT features extracted from an image (a small number of those features are shown in the figure) in MATLAB using an open source library called VLFeat [75]. The centers of the yellow circles indicate the feature locations and the yellow lines connecting the circles to their centers indicate the feature orientations. The green grid with arrows is a visualization of the descriptors of those features.

3.2.2 Feature Tracking

The SIFT features extracted from the images (see Section 3.2.1) are tracked using David Lowe's keypoint matching method described in Section 2.3.1 bounded by the epipolar constraint criterion (Section 2.3.2). The tunable parameters are the thresholds ϵ_r and d_P. ϵ_r is the maximum value allowed for the ratio between the smallest and the second smallest distance of two SIFT feature descriptors for a match to occur. d_P is the largest distance

value allowed between a keypoint in the second image and the epipolar line corresponding to the keypoint of interest in the first image. If the distance is less than d_P then the keypoint in the second image is included in the matching procedure with the keypoint of interest. The following matching procedure is used for the case of three or more images where features are to be tracked from.

1. Features are tracked between the first two images using the keypoint matching method and the epipolar constraint criterion, using the parameters ϵ_r and d_P. The tracked features' corresponding indices and links (i.e., a way of binding a feature from the first image to the same feature in the second image) are stored in a list.

2. The tracked features from the second image are then compared with the features in the third image (using the same procedures from step 1). The results are updated in the list and the unmatched features in the second image along with their corresponding matched features to the previous images are removed from the list.

3. Step 2 is repeated for successive images until all images are covered, or if there are no matches found in the latest image. If there are no matches found, then the algorithm stops at the last image where matches occurred.

The end result of this procedure is a list of distinct features that are matched in all images, with their indices and links stored. The procedure can also be reversed (optional), i.e. starting from the last image where matches occurred and matching towards the first image. The results from the reverse case are combined with the results from the forward case to produce more correspondences. Figure 3.4 shows an illustration of the procedure, illustrating both the forward and reverse case. The row of numbers are the feature indices corresponding to the image the features reside in. The dashed box encloses two rows (images) which the matching algorithm is applied to. The rows are updated with the matched feature indices, where each link is indicated by the thick line connecting the matched indices. The top of the figure shows the procedure being run starting from the first image to the last image, and the bottom of the figure is the reverse case. The results from both cases are then combined (with the duplicates removed) to produce the final result. For example, the feature index 7, 5, and 2 from image 1, 2, and 3 all correspond to the same (matched) feature, as shown in the combined results of Figure 3.4. Figure 3.5 shows an example of the matching procedure applied to three images with SIFT features. The yellow markers show the matched feature locations on the images, and the blue dashed lines show the links between the features on the images. The links are all relatively horizontal, since the three images are captured at positions offset horizontally from one another.

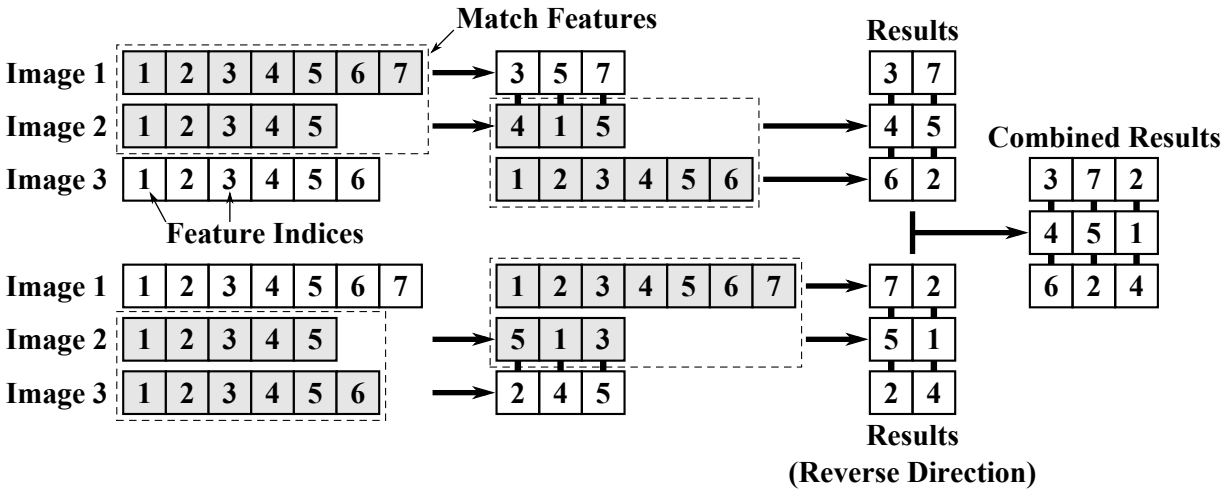


Figure 3.4: Illustration of the feature matching procedure. The top portion shows the matching done in the forward sequence and the bottom portion shows the matching done in the reverse sequence. The combined result is shown at the right of the figure.

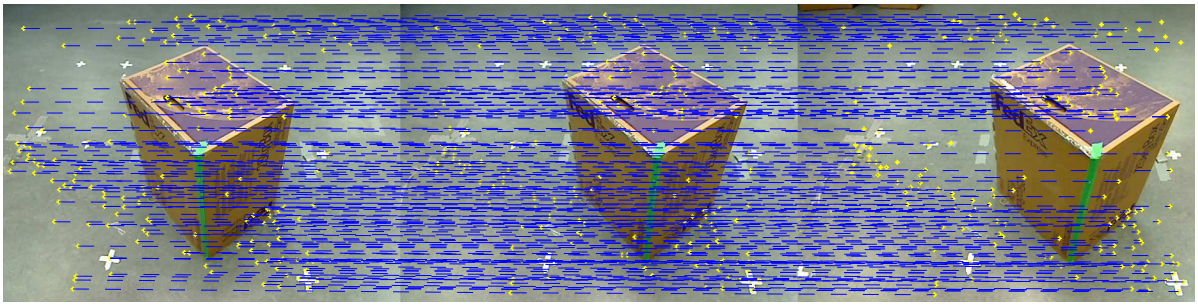


Figure 3.5: Feature matching procedure example for the three consecutive images from Figure 3.2.

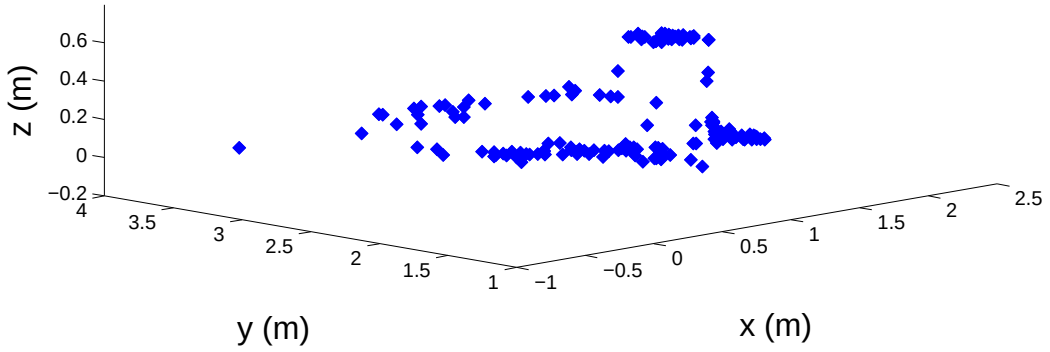


Figure 3.6: 3D feature point mapping example using the feature correspondences from Figure 3.5.

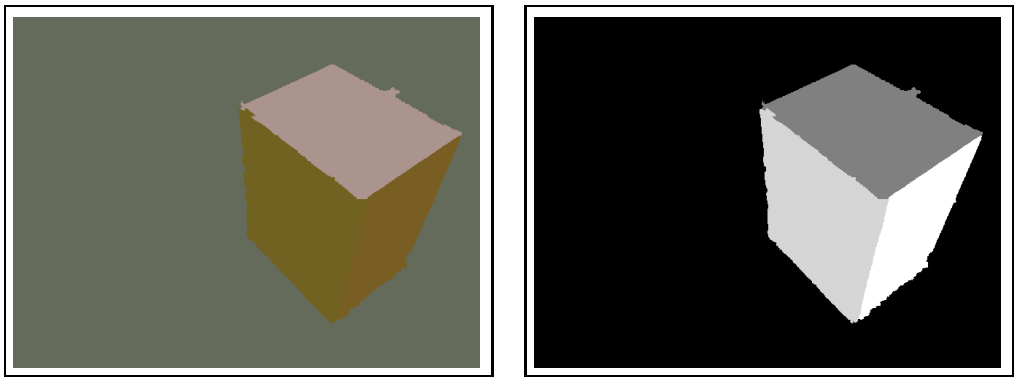
3.2.3 3D Feature Mapping

A 3D point cloud is computed using the feature correspondences (from the first image to the image where correspondences occurred) and the 3D point estimation from multiple views method (Section 2.4, Algorithm 1). Figure 3.6 shows a 3D point cloud (each point shown as blue markers) generated from the matched features for the images shown in Figure 3.5. The results are stored for later use.

3.2.4 Surface Fitting

The most recent image with feature correspondences (in this case, the third image) is used for surface fitting. In order to do so, the image is first segmented (see Section 2.5) into the regions which represent object surfaces. The procedure for segmenting the image is as follows:

1. Downsample (shrink) the image by a factor $r_d \in (0, 1]$ (multiply r_d with the image size to obtain the downsampled image size).
2. Apply the Kuwahara filter (Section 2.5.1, Algorithm 2) on the down sampled image $n_d \in \mathbb{N}_1$ times to produce a filtered image.
3. Apply the mean shift segmentation procedure (Section 2.5.2, Algorithm 4) on the filtered image to produce a segmented image.
4. Upsample (grow) the segmented image by a factor of $\frac{1}{r_d}$ to return it to its original size.



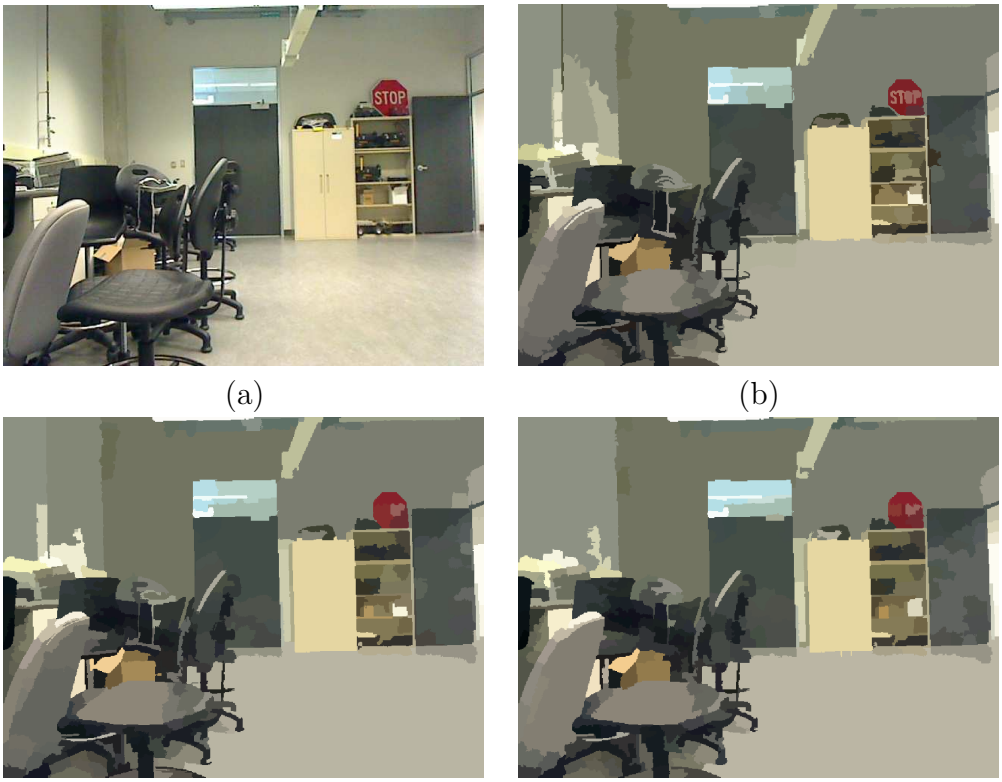
(a) (b)

Figure 3.7: Example of segmentation procedure applied to the third image from Figure 3.5. (a) The segmented result from mean shift segmentation. (b) The labelled regions.

The parameters r_d and n_d, the size of the Kuwahara filter kernel (L), and the spatial (h_s) and range (h_r) bandwidths, as well as the minimum region pixel size (M) for mean shift segmentation are all tunable parameters. Figure 3.7 shows this procedure being applied to the third image, from Figure 3.5. The mean shift segmentation is applied using a MATLAB wrapper [76] for the Edge Detection and Image SegmentatiON (EDISON) system [69].

As mentioned in Section 2.5, the main purpose of applying the Kuwahara filter the image before the segmentation process is to reduce textural information, which reduces the number of segments that are by-products of surface textures. Figure 3.8 shows an example of this. Figure 3.8 (a) is the original image and Figure 3.8 (b) is the segmented image with only the mean shift segmentation applied to it. Figure 3.8 (c) and Figure 3.8 (d) shows one and two Kuwahara filter iterations, respectively, applied to the image before the segmentation process. The coloured regions in a segmented image correspond to a segment. Some specific examples of reduced effects of textural information with the application of the Kuwahara filter include the stop sign and the seat portion of the black chair on the front left of the image, resulting in less colour segments. The extra segments due to lighting on the wall on the left side of the image are also reduced because of the smoothing property of the Kuwahara filter. It can be seen with careful observation that, although the segmented image from Figure 3.8 (d) has less segments that are caused by textures, the filter itself is not scale invariant, which means that objects that are seen far away are filtered differently than objects that are close. As a result, edge-preserving characteristics of far away objects may be reduced due to the nature of the filter, as can be seen from the shelf on the far right side of the image. This drawback can be compensated by capturing the images at a higher resolution.

Downsampling the image prior to the segmentation process is done to reduce its computation time. Figure 3.9 shows results of the segmentation process with the image scaled



(a) (b)

(c) (d)

Figure 3.8: Effects of including the Kuwahara filter iterations in the segmentation process. (a) The original image. (b) The segmented image with only mean shift segmentation applied to it. (c) One and (d) two Kuwahara filter iterations applied to the image prior to mean shift segmentation.

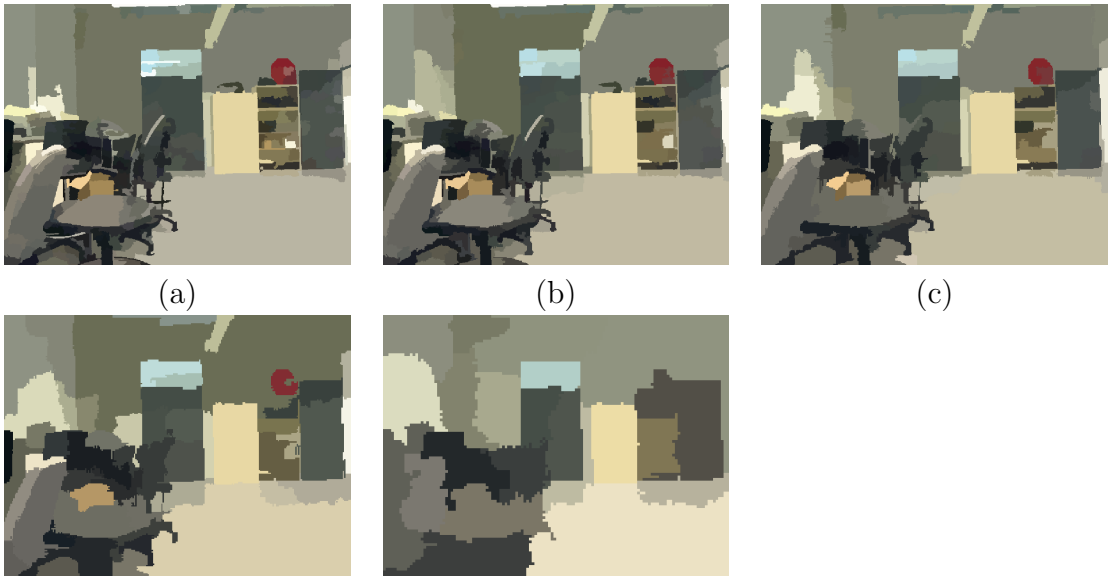


Figure 3.9: The resulting segmented image from setting r_d for the segmentation procedure. (a) $r_d = 1$ (no downsampling). (b) $r_d = 0.8$. (c) $r_d = 0.6$. (d) $r_d = 0.4$. (e) $r_d = 0.2$.

down to different sizes. From Figure 3.9 (a) to Figure 3.9 (e) the image sizes before segmentation are: 100% (of the original size), 80%, 60%, 40%, and 20% respectively. It is empirically observed from these figures that a decreased image size leads to poorer segmentation quality and reduced number of segments. This is reasonable because by shrinking the image, some pixel and edge information is lost. Shrinking the image is also effectively scaling down the scene, which reduces the filter performance since the Kuwahara filter is not scale invariant. This in turn leads to a reduction in the segments found from the segmentation process. Therefore, a balance should be assessed between computation time and the quality and quantity of segments when tuning the parameter r_d.

The matched feature points in the third image are labelled using the segmented image, with each label corresponding to an object region (refer to Figure 2.17). An approximate planar surface is fitted to each set of labelled points by fitting an orthogonal distance regression plane (solving Eqn. 2.64 to the points and projecting the respective segment boundary onto the plane (solving Eqn. 2.65 for each point on the boundary). Surface reconstruction will only be applied to regions with at least $n_{\text{pts}} \in \mathbb{N} \geq 3$ points associated with it. Infeasible surfaces are surfaces that are too far away or close (in 3D space) to any camera view are rejected. This is achieved by checking each projected point that makes the boundary to the surface and determine if that point is greater than $d_{\text{pmax}} \in \mathbb{R}^+ \cup \{0\}$ or less than $d_{\text{pmin}} \in \mathbb{R}^+ \cup \{0\}$ from any camera view (along the optical axis). The following procedure is used to reject infeasible surfaces:

1. For each projected point on the boundary of the first generated surface, transform the point from the world frame to the frame of the first camera view using Eqn. 2.19.

2. If $z < d_{\text{pmin}}$ or $z > d_{\text{pmax}}$, where the z value of the transformed point is the position along the optical-axis of the first camera, then the surface represented by the point is removed from the map.

3. Repeat steps 1 and 2 for each camera view until all camera views are checked or until the surface is removed from the map.

4. Repeat steps 1 to 3 for each generated surface in the map.

Figure 3.10 shows an example of the final map produced by the algorithm. The colour represents a different region (object surface). The coloured circles show the estimated feature points in 3D space and the surface boundaries are shown as the coloured outlines. The figure shows four different views of the map for better inspection.

3.2.5 Algorithm Summary

The summary of the PSR algorithm is illustrated in Figure 3.11. The algorithm uses the processes explained from Sections 3.2.1 to 3.2.4. Its input is a set of sequential images with their corresponding camera data and the output is a surface map in 3D space, with respect to a global reference frame. For video data, n_{frame} frames separated by n_{fsep} intermediate frames are the input parameters for the method. This is done repeatedly for the following frames, also separated by n_{fsep} number of frames, with the resultant map being appended to the current environment map. The following steps summarize the PSR algorithm.

1. Extract SIFT features (Section 2.1) from each input image and store the feature locations and descriptors in a list.

2. Apply the SIFT tracking method (Section 3.2.2) for each consecutive image, using the thresholds ϵ_r and d_P, until all images have correspondences or until there are no matches. The SIFT feature lists are updated to include the correspondence information.

3. Apply the multi-view 3D point estimation method for each matched feature using Algorithm 1 from Section 2.4, and store the resultant 3D point cloud information.

4. Shrink the most recent image with correspondences by a factor r_d and apply the Kuwahara filter (Section 2.5.1, Algorithm 2) to the image for n_d iterations. The size of the filter kernel is adjusted with L.

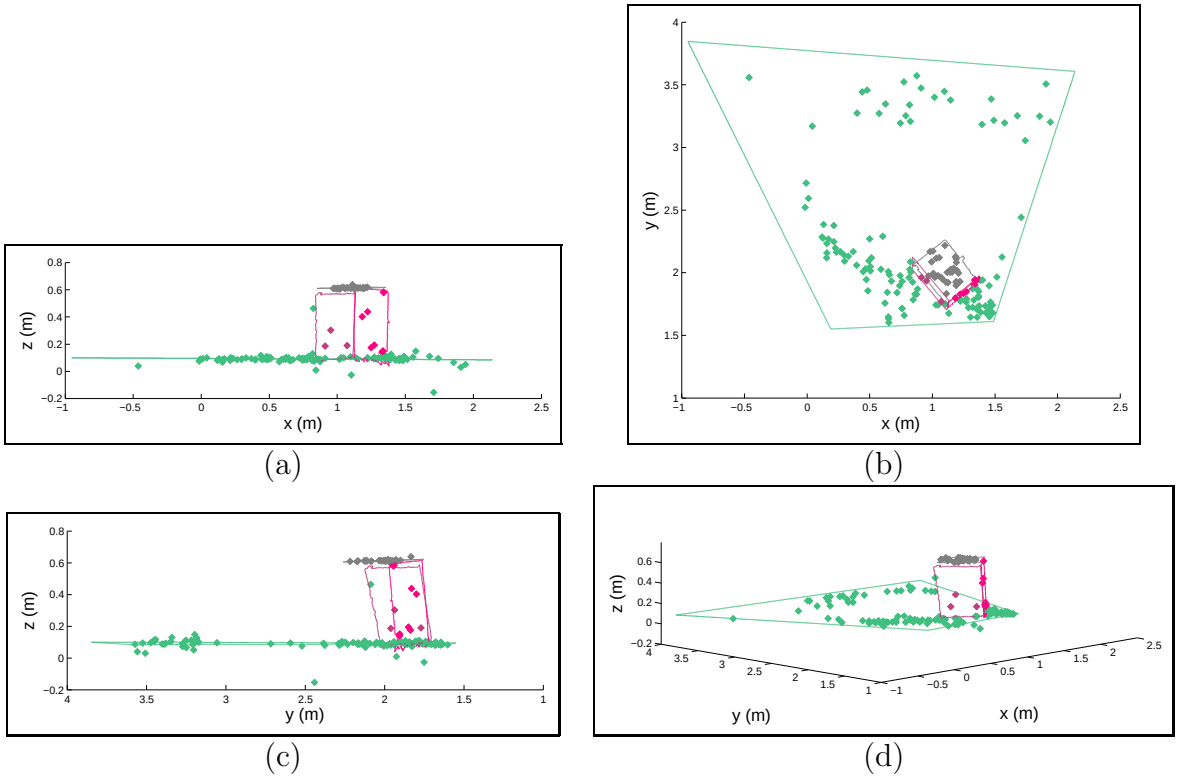


Figure 3.10: Example of the surface map produced by the PSR algorithm using the images in Figure 3.2. The surfaces are shown as the coloured outlines and the feature points are shown as the coloured circles, each where each region is of a different colour. (a) Front view. (b) Top view. (c) Side view. (d) Angled view.

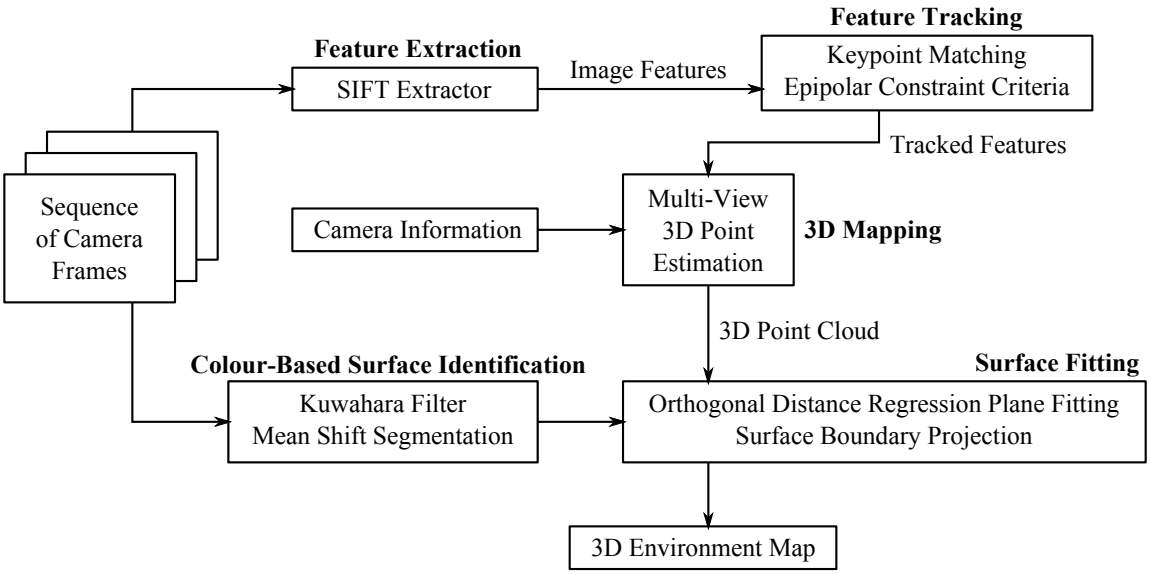


Figure 3.11: Diagram summarizing the main components of the PSR algorithm.

5. Apply the mean shift segmentation procedure (Section 2.5.2, Algorithm 4) to the filtered image using the parameters h_s, h_r, M. Grow the segmented image by a factor of r_d to return it to its original size.

6. Each matched feature point from the most recent image with correspondences is labelled to the region from the segmented image which it resides in.

7. For each set of labelled points, if there are at least n_{pts} points then a plane is fit to the set of points using the orthogonal distance regression plane method (Eqn. 2.64). A 3D surface is generated by projecting the pixel boundary of the segment (associated to the set of points) onto the plane of fit (Eqn. 2.65). Surfaces that are not between $z < d_{\text{pmin}}$ and $z > d_{\text{pmax}}$ away from any camera optical axis are removed.

The output is a map of 3D points and surfaces. This can be used for path planning or object assessment algorithms depending on the application. Chapter 5 presents the results and observations of applying the PSR algorithm to video stream data recorded from a ground vehicle test platform.

Chapter 4

Autonomous Vehicle Platform

An autonomous vehicle platform was developed at the University of Waterloo (UW) as a testbed for various path planning, controls, and vision algorithms. Its initial purpose was to enter the International Autonomous Robot Racing Challenge (IARRC), though it was decided that after the competition it will be used for research and other related activities. In the context of this research, the testbed is used to validate the PSR algorithm. The first iteration of the platform underwent an irreparable collision and currently the second iteration is being built for the real-time implementation of the algorithm.

Details on the physical and software aspects of the vehicle are described in this chapter. Arun Das, Gerardo Salas Bolanos, Prasenjit Mukherjee were also part of the group working on the platform, under the team name University of Waterloo Autonomous Racing (UWAR).

4.1 Mechanical Design

The Traxxas E-MAXX chassis was selected as the base vehicle for retrofitting electrical and control systems in order to make it autonomous. The chassis itself was chosen for some key features: wide wheel track, four wheel drive, and customizability. The wide wheel track increases stability for steering since it can accommodate for higher horizontal shifts in the center of gravity when performing aggressive turns. The four wheel drive increases vehicle traction and reduces slippage in off-road conditions. The build of the chassis was modified in numerous ways as listed in Table 4.1 and electrical and control systems components were added.

The springs and gearbox were purchased from other sources and installed. The wheel mounts for the encoders (as shown in Figure 4.1) were designed using Computer Aided

Table 4.1: Mechanical modifications for the autonomous vehicle chassis.

Modification	Purpose
Stiffer Suspension	Account for increased weight of sensors, electrical and control components.
Addition of two-speed gearbox	Low gearing for better speed control at low speeds while high gearing to achieve greater speeds.
Wheel-mounts for encoders	Measurement of position on both wheels can better estimate speeds on turns.
Aluminium frame mount with steel casing	For mounting and partial protection of laptop, sensors, and electronics.
Carbon-fibre body for chassis	Helps prevent damages to the vehicle from minor collisions.

Design (CAD) and built to be attached to the wheel axles inside the rear wheels. The mount assembly has a spring mechanism which pushes two rubber O-rings against the inside of the wheel. The O-rings rotate as the wheel rotates, which in turn causes the encoder shaft to rotate.

An aluminium frame with supports was designed and manufactured for the electronics (i.e. controllers, breakout boards, power management modules, and motor drivers), sensors and laptop. A steel cage was made to guard the Hokuyo UTM-30LX Light Detection And Ranging (LIDAR) sensor against collision damage. The battery harnesses allow for quick-swap capabilities.

The carbon fibre body was designed to encompass the entire vehicle, only exposing the bottom area and the necessary sensors. This ensures proper operation and provides some protection from rain and debris that can potentially damage the electronics or interfere with the mechanical joints. The mould for the body was created using CAD and machined using tool path planning algorithms and Computer Numerical Control (CNC) methodologies [77]. The carbon fibre was reinforced with a steel wire mesh to increase its rigidity and robustness to collisions.

The entire vehicle assembly, including both designed and stock parts, was modelled using Solidworks 2009. The full rendered model is shown in Figure 4.2.

Two emergency stops (e-stops) were fitted to the vehicle to ensure safe operation. When triggered, the motor controller brake command is activated and the vehicle motor is locked from rotating (analogous to an engine brake) until the e-stop is reset. The first e-stop can be mechanically triggered (i.e. a button) and mounted on the back of the vehicle. The second e-stop can be triggered remotely through a wireless controller, with its operational range being a 200 meter radius.

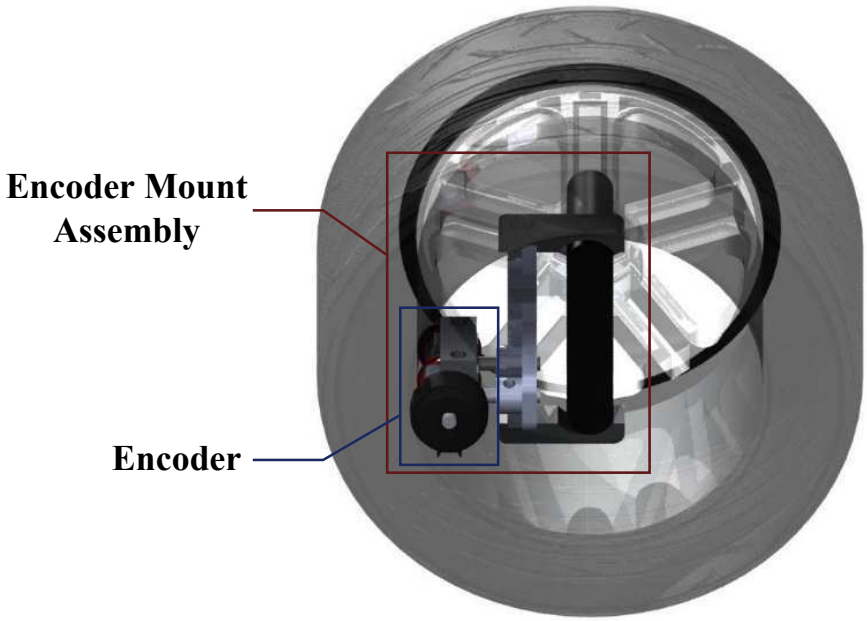


Figure 4.1: Rendered image of the CAD model of the wheel with the mount assembly and encoder attached.

Figure 4.2: Rendered image of the CAD model of the autonomous vehicle platform.

(a) (b)

Figure 4.3: (a) The earlier stages and (b) completed version of the autonomous vehicle platform.

The platform construction results are shown in Figures 4.3 and 4.4. Figure 4.3 (a) shows the vehicle in its earlier stages of development, before the fabrication of the camera mounts and carbon fibre body, and Figure 4.3 (b) shows the completed model of the testbed with the carbon fibre body mounted. The resultant robot has a maximum forward speed of about 50km/hr and a steering angle (δ from Figure 4.9) of about ± 26 degrees. Figure 4.4 shows the encoder mount assembly attached to one of the rear wheel axles of the vehicle.

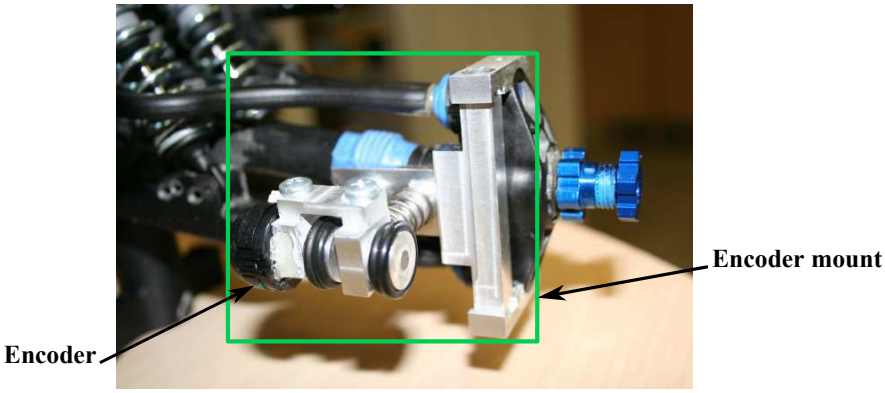


Figure 4.4: Encoder mount assembly for the autonomous vehicle platform.

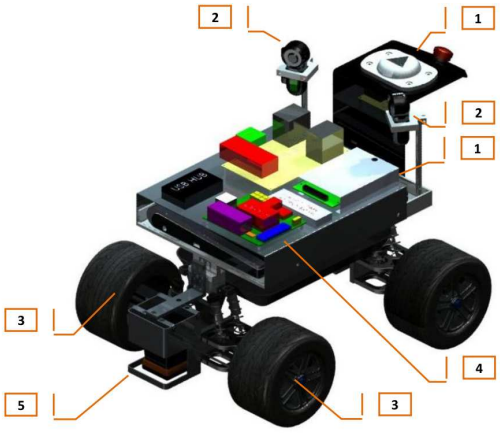


Figure 4.5: Sensor network for the autonomous vehicle platform. Each sensor is labelled with a number and is cross-referred in Table 4.2, which describes the sensors in more detail.

Table 4.2: Sensor network for the autonomous vehicle platform.

#	Sensor	Type	Function
1	Novatel OEM-V3G GPS Receiver and Antenna	GPS	Global localization
2	Logitech WebCam Pro 9000	Camera	Computer vision
3	USD-E4P-OEM	Optical encoder	Position, speed estimation and localization
4	IMU MCU	INS	Acceleration, orientation, heading estimation and localization
5	Hokuyo UTM-30LX	LIDAR	Obstacle range sensing and localization

4.2 Electrical Design

4.2.1 Sensors and Instrumentation

The sensor network outline for the system is shown in Figure 4.5 with the functions briefly described in Table 4.2.

The Novatel OEM-V3G Global Position System (GPS), Inertial Measurement Unit (IMU) MicroController Unit (MCU) Inertial Navigation System (INS), and wheel-mounted USD-E4P-OEM optical encoders are used for pose estimation, localization, and global localization for the vehicle. The Hokuyo UTM-30LX LIDAR is a planar laser range finder that returns frontal-planar distances of objects ranging up to 30 meters. It also has the capability of outdoor operations. There are two Logitech WebCam Pro 9000 cameras

Table 4.3: Battery power distribution for the electrical systems of the vehicle testbed.

Battery	Electrical systems to power
2 x 5000mAH LiPo in series (16.8V)	The Neu-Castle 2200KV electric motor
1 x 2100mAH LiPo (7.4V)	IMU MCU
2 x 3300mAH NiMH in series (14.4V)	MCU board, LIDAR, and GPS receiver
Laptop battery	Webcams (through usb ports) and laptop

mounted on each side of the platform, which can be used for monocular or stereo computer vision techniques. The cameras were chosen as they are the most cost-effective solution and are commonly used in computer vision research.

4.2.2 Power Distribution

Two 5000mAH Lithium Polymer (LiPo) batteries are responsible for powering the 2200KV electric motor used to drive the vehicle. The brushless motor has a maximum current rating of 120A peak while the batteries are capable of providing 175A of current at a 35C discharge rate, which is sufficient for when the motor is undergoing heavy load from high torque and acceleration demands.

The IMU MCU has its own power management on-board and is powered by a 2100mAH LiPo battery. The two 3300mAH Nickel-Metal Hydride (NiMH) batteries power a custom power management module, which uses switching power regulators to supply 5V and 12V power. The 5V supply powers the MCU board and GPS receiver while the 12V supply powers the LIDAR. The two Logitech webcams are powered through a usb hub connected to the laptop.

Based on various tests made on the operational life for the vehicle, the whole system is expected to run for 30 to 60 minutes using fully charged batteries.

Table 4.3 summarizes the batteries used to power the various electric systems.

4.2.3 Hardware Architecture

The processing is distributed amongst two microcontroller units (MCU), the primary and the IMU MCU, and the main processor (i.e. a Samsung SENS Q45 laptop, running with a 2.1GHz dual core processor). Figure 4.6 shows a simplified version of the system. The primary MCU is responsible for quadrature decoding and the actuation of the steering and throttle for the platform. The quadrature decoding is offloaded to LS7166 24 bit chips, while the steering and throttle commands for the vehicle are initiated using an LPC2148 microcontroller that features an ARM7 core. These chips are integrated into the

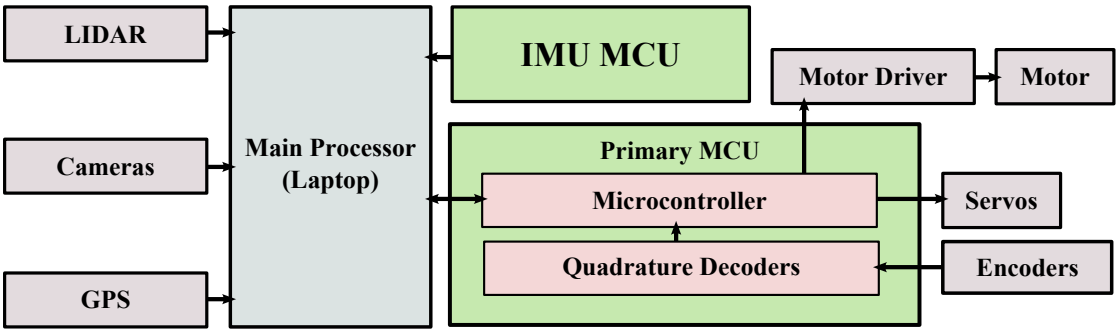


Figure 4.6: Simplified illustration of the electrical hardware architecture. The arrows indicate the general direction of data flow. Only the main processor and microcontroller in the primary mcu have bidirectional data transfer.

primary MCU. Heading and inertial measurements are offloaded to the IMU MCU, and then fed back to the primary processor. The main processor performs all of the higher-level processing, such as object detection, state estimation, localization and trajectory planning and control. It also receives GPS, camera, and LIDAR sensor data as well as sending commands and receiving feedback from the primary MCU.

4.3 Software Design

4.3.1 Software Architecture

The system architecture described in [78] is used as a base model for the platform's software architecture. The model is divided into low-level and high-level nodes. The low-level nodes are responsible for obtaining sensor data and interfacing with the microcontroller while the high-level nodes are responsible for the higher level object detection, state estimation and path planning. Message passing between the two groups is done through a message broker using a publish-subscribe paradigm. Additional information that the high-level nodes needs are stored in a blackboard mechanism [78]. The architecture is shown in Figure 4.7.

The general flow outline of the system architecture is as follows:

1. Odometry and environment information is retrieved from the sensors and microcontroller using the low-level nodes and then is published to the message broker.

2. The message broker passes the information to the subscribing high-level nodes.

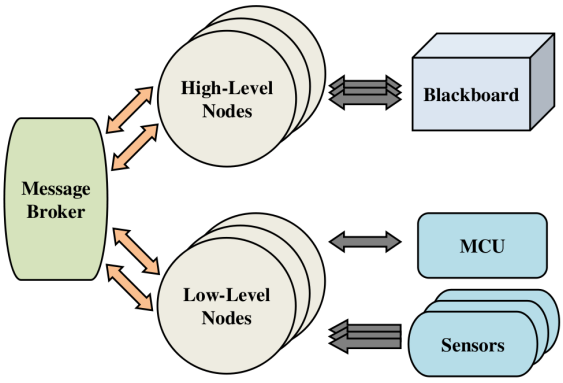


Figure 4.7: Software architecture for the autonomous vehicle platform.

3. The high-level nodes use the subscribed information while interacting with the blackboard to perform state estimation, localization, mapping, path and trajectory planning.

4. The navigation node uses the trajectory to publish commands to the low-level node which then sends the command to the microcontroller to control the platform.

The main advantages for using a publish-subscribe paradigm are that the nodes are encapsulated and will send and receive data when it is available. A disadvantage is that the message passing is asynchronous and can affect performance of the high-level nodes. For example, the state estimation node update rate will only be as high as the rate of the slowest sensor that it subscribed to, whereas it can be more beneficial to update the state at a faster rate using a sufficient amount of sensor data and continue using old data until it is updated in the next iteration. It is reasoned that the blackboard mechanism is useful for the high-level nodes since it can be used to provide data to the processes when queried rather than waiting for the next update to be published. This ensures a synchronous execution of high-level nodes and increased overall update rate, therefore providing a smoother execution and data flow. The downside are the additional errors caused by using previous data, though the errors are small due to the high update rates of the nodes and sensors.

Robot Operating System(ROS) [79] is used to implement the system architecture. ROS is a meta-operating system designed specifically for robotics applications and thus is a useful base tool for programming the vehicle platform. In the context of the architecture, ROS provides abstraction for the message creation and passing, service queries and updates, node construction and linking, and sensor communication. This allows the primary focus for the software implementation to be on the higher-level processes.

4.3.2 Low-Level Communication

Communication between the laptop and the primary MCU is done via serial UART. A custom 7 byte protocol was designed for sending commands to the MCU and receiving data. The first byte is the command byte, for identifying which command is being sent or received, while the next four bytes contain the data corresponding to the command. The last two bytes contain a 16 bit CRC checksum to improve the robustness of data transfer. The data transfer is done in little-endian byte order.

4.3.3 Control Systems

The primary MCU is also responsible for the closed-loop velocity controller. Velocity feedback is provided from two optical encoders mounted to the hubs of the two rear wheels. Each encoder is connected to a LS7166 24 bit quadrature decoder, which is interfaced via an 8-bit parallel bus to the MCU. The quadrature decoding was offloaded to the LS7166 chips to allow for high speed operation, and to free up computational resources in the MCU.

The low-level systems use robust and computationally inexpensive algorithms to fuse some of the available data and provide effective control over the throttle of the vehicle. The steering actuators of the platform are directly controlled by the high level trajectory systems.

Velocity Controller

A proportional and integral (PI) controller is implemented for the robot for controlling velocity. Velocity control is necessary for maintaining the proper velocity in changing terrain or vehicle conditions (e.g. hills, grass, ice, turns, etc...), therefore increasing vehicle performance in trajectory tracking. Derivative control can be included to the controller, but it was determined that the PI controller is sufficient. The proportional control is necessary for response time and integral control is necessary for zero steady state error.

The Laplace transform of a PI controller is as follows:

$$\frac{U(s)}{E(s)} = k_p + \frac{k_i}{s}, \tag{4.1}$$

where $s \in \mathbb{R}$, $U(s) \in \mathbb{R}$ is the velocity input, $E(s) \in \mathbb{R}$ is the velocity error, $k_p \in \mathbb{R}$ and $k_i \in \mathbb{R}$ are the proportional and integral gains. Applying a z-transform (Tustin

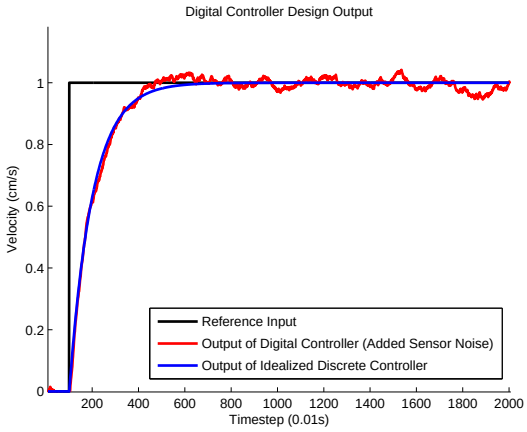


Figure 4.8: Simulation of the discrete PI velocity controller using MATLAB.

transformation) to the Eqn. (4.1) yields:

$$\begin{aligned} U(z) &= E(z)\left(k_p + k_i \frac{T_s}{2}\frac{z+1}{z-1}\right) \\ U(z)(2z-2) &= E(z)(2zk_p - 2k_p + T_s z k_i + T_s k_i) \\ U(z)\left(2-2z^{-1}\right) &= E(z)\left(2k_p - 2z^{-1}k_p + T_s k_i + T_s z^{-1} k_i\right), \end{aligned} \tag{4.2}$$

where T_s is the sample time of the system. Converting Eqn. (4.2) into a difference equation results in:

$$u[k] = u[k-1] + \frac{e[k](T_s k_i + 2k_p) + e[k-1](T_s k_i - 2k_p)}{2}, \tag{4.3}$$

where k is the discrete time.

Figure 4.8 shows a simulation for the discrete PI controller, using Eqn. (4.3) for its implementation. The simulation shows the response of the controller for a unit step reference input, with and without simulated sensor noise.

Steering Controller

The steering controller considered is non-linear and was first applied by the Stanford University entry into the DARPA Grand Challenge [80]. The control law is given as:

$$\delta(t) = \psi(t) + \tan^{-1}\left(\frac{k_c x(t)}{v(t)}\right), \tag{4.4}$$

where $\delta(t) \in \mathbb{R}$ is the control steering angle, $\psi(t) \in \mathbb{R}$ is the heading error, $x(t) \in \mathbb{R}$ is the cross track error, $v(t) \in \mathbb{R}$ is the robot velocity, and $k_c \in \mathbb{R}$ is the controller gain. Figure 4.9 illustrates the parameters of the steering control system.

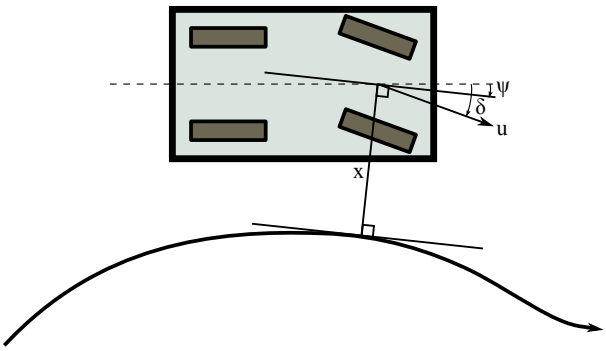


Figure 4.9: Non-linear steering controller diagram.

The first term of Eqn. (4.4) is used for correcting the heading error while the second term corrects for cross track error. The second term ensures that the vehicle's trajectory converges with the desired trajectory, at an exponential convergence rate [80]. An additional damping gain for the heading error is implemented in the control system to account for smoothing noisy heading errors, thereby improving the performance of the steering control. The resultant controls equation is as follows:

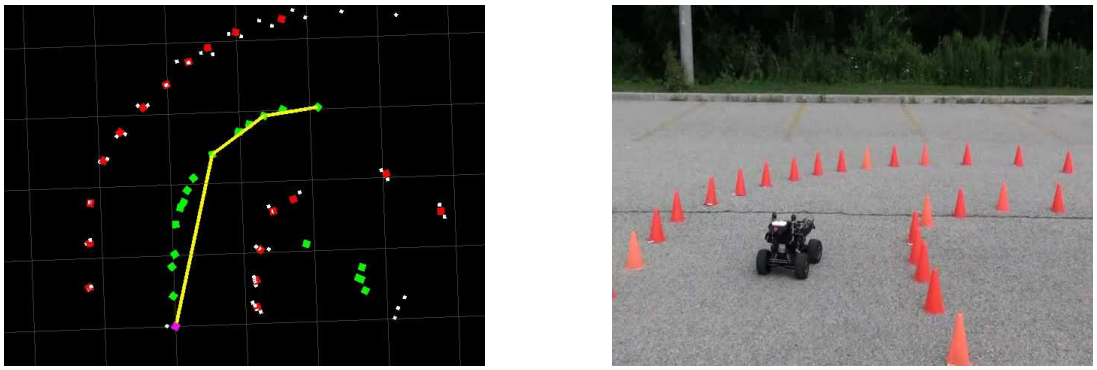
$$\delta(t) = \frac{\psi(t)}{k_h v(t)} + \tan^{-1}\left(\frac{k_c x(t)}{v(t)}\right), \tag{4.5}$$

where $k_h \in \mathbb{R}$ is the damping gain for heading error. The control law from Eqn. (4.5) was implemented for the vehicle platform.

4.4 Path Planning and Vision Processing

The platform was tested with various path planning and vision algorithms for the IARRC entry as soon as it became fully operational. While many path planning algorithms were considered, Figure 4.10 shows the preferred online path planning algorithm successfully generating trajectories (shown as the yellow line) for the robot to traverse through a test circuit enveloped by pylons. Figure 4.11 (a) and Figure 4.11 (b) shows the results of the vision methods implemented for one of the platform's cameras, where it successfully detects a stop sign and a customised traffic light.

The platform was shown to successfully execute vision-based algorithms and therefore can be used to validate the real-time performance of the PSR algorithm. The first version of the platform underwent an irreparable collision and the second version is currently being built. Experiments were done to test the algorithm offline using video data captured from an alternate vehicle platform, with the results and observations discussed in Chapter 5.



(a) (b)

Figure 4.10: (a) The markers (white for filtered LIDAR points, red for pylons and green for path points) used in the path planning algorithm to form a trajectory (yellow line) for the vehicle. (b) The vehicle's current environment at which the path from (a) is planned from.

(a) (b)

Figure 4.11: Vision algorithms tested using a camera on the autonomous vehicle platform. (a) Custom traffic light detection. (b) Stop sign detection.

Chapter 5

Experimental Results and Observations

Experiments were performed using an alternate vehicle platform known as the Chameleon R100 from Clearpath Robotics. The Chameleon R100 is an educational mobile robot platform capable of achieving speeds of up to 1.5 m/s and is designed for Ackermann steering. Its dimensions are 42.6cm x 31.6cm x 18.6cm and weigh about 10kg with a maximum payload of about 5kg. It has an operating time of about 2 hours and uses two 12V, 4.6 Ah NiCd batteries. The platform also comes with an Acer netbook, Garmin GPS, Hokuyo LIDAR, Logitech Quickcam webcam, and three IR and SODAR range finders.

Video and odometry data were collected and the used as inputs for the MATLAB 2008b implementation of the PSR algorithm for offline evaluation. The video data was captured at 640x480 pixel resolution and the odometry of the vehicle testbed was estimated using an OptiTrack positioning system in the lab. Two test cases were evaluated by the algorithm. The first test case (Section 5.1) involves the robot driving along a clean environment with a box placed on the relatively planar scene. The second case (Section 5.2) is the evaluation of the algorithm for cluttered indoor environments. It involves the vehicle moving across the lab space and reconstructing a side of the room full of clutter. Table 5.1 lists the parameters used for the two cases. The following sections analyse the results from two test cases, showing the frames used, map generated, segmentation and feature matching results. Section 5.3 assesses the computational performance (processing times) of the main algorithm procedures for the two cases.

Table 5.1: Parameters used for the two experimental test cases.

Test Case	1	2
Number of sequential frames per iteration (n_{frame})	3	4
Frame separation constant (n_{fsep})	5	10
SIFT matching ratio threshold (ϵ_r)	4	4
Epipolar constraint criterion distance (d_P)	50px	50px
Reverse matching for matching procedure	Yes	Yes
Segmentation image scale factor (r_d)	1	0.75
Kuwahara filter kernel bandwidth modifier (L)	1	1
Kuwahara filter iteration number (n_d)	2	2
Mean shift segmentation spatial kernel bandwidth (h_s)	7	7
Mean shift segmentation range kernel bandwidth (h_r)	6.5	6.5
Mean shift segmentation minimum region area (M)	1500px^2	2000px^2
Minimum number of points for planar fitting (n_{pts})	8	10
Surface feasibility minimum camera distance (d_{pmin})	0m	0m
Surface feasibility maximum camera distance (d_{pmax})	8m	8m

5.1 First Test Case: Clean Environment

The first test evaluates a clean environment to examine the validity of the assumptions made for the design of the algorithm. These assumptions include the environment consisting of rigid, non-moving planar surfaces, with each surface having an overall similar colour (when ignoring texture information). In this scenario the vehicle moves along a flat surface with a white background, and a box (with dimensions of 55.5cm length, 43.5cm width (purple side), and 34.2cm height) can be seen on the ground. Figure 5.1 shows the frames used from the video stream. Since there are eight frames and three frames per iteration used ($n_{\text{frame}} = 3$), the total number of iterations applied to the PSR algorithm is six (the first iteration uses frames 1-3, the second iteration uses frames 2-4, etc.). Figure 5.2 shows the resultant surface map of the combined surface generation results of the six iterations. It can be seen that there is an overlap of many surfaces because there is currently no surface fusion method implemented in the algorithm. It can also be seen from the surface map that the two sides of the box, as well as the portion of the ground plane has been reconstructed with relatively good accuracy. This implies that the box location with respect to the vehicle is also accurate as the distance from the box to the vehicle is proportional to the generated size of the box (both factors are scalar dependent).

The segmented image and SIFT matching results for each iteration are shown in Figure 5.3 and Figure 5.4, respectively. Although the segmentation process did remove some textural information to produce chromatically similar segments, the segmented results of

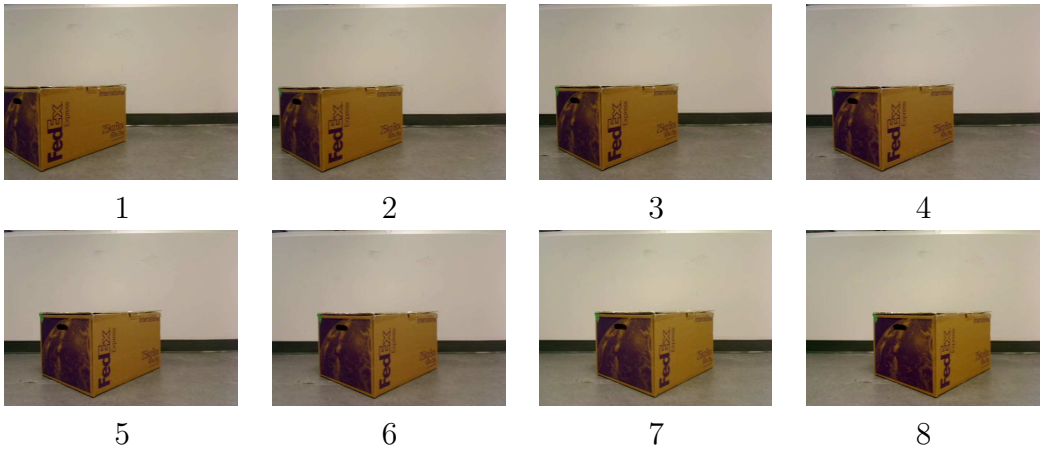


Figure 5.1: Image frames used for the first test case. The number below each frame corresponds to the frame's index in the sequence.

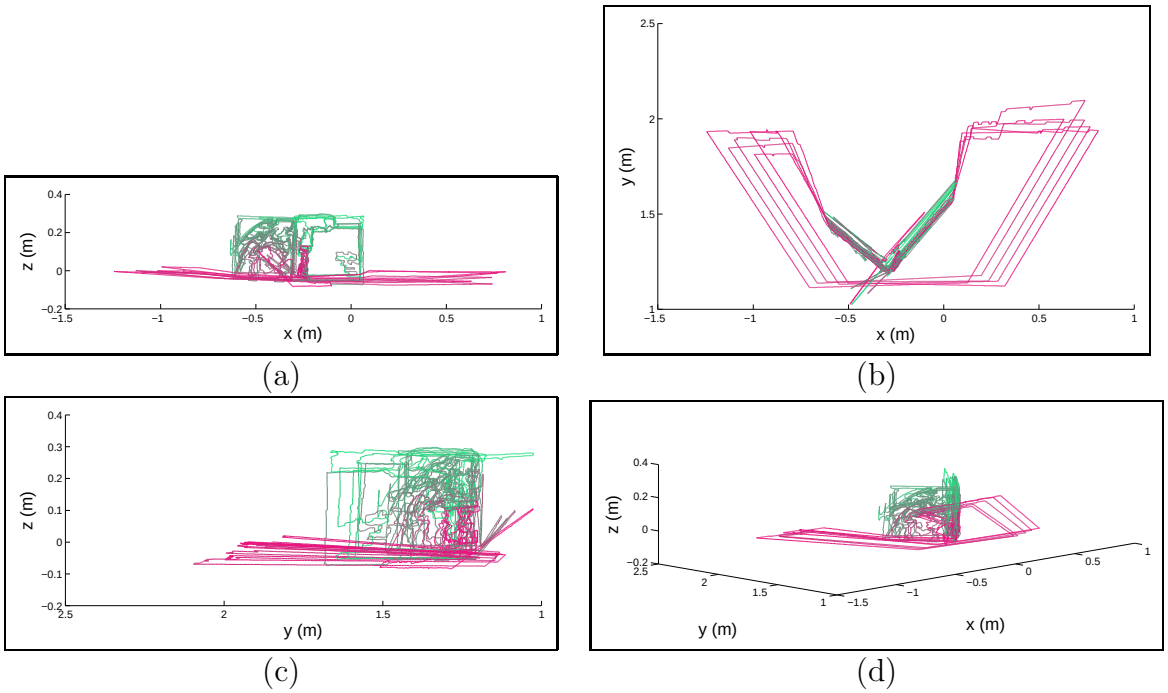


Figure 5.2: Surface mapping result for the first test case. The surfaces are shown as the coloured outlines. (a) Front view. (b) Top view. (c) Side view. (d) Angled view.

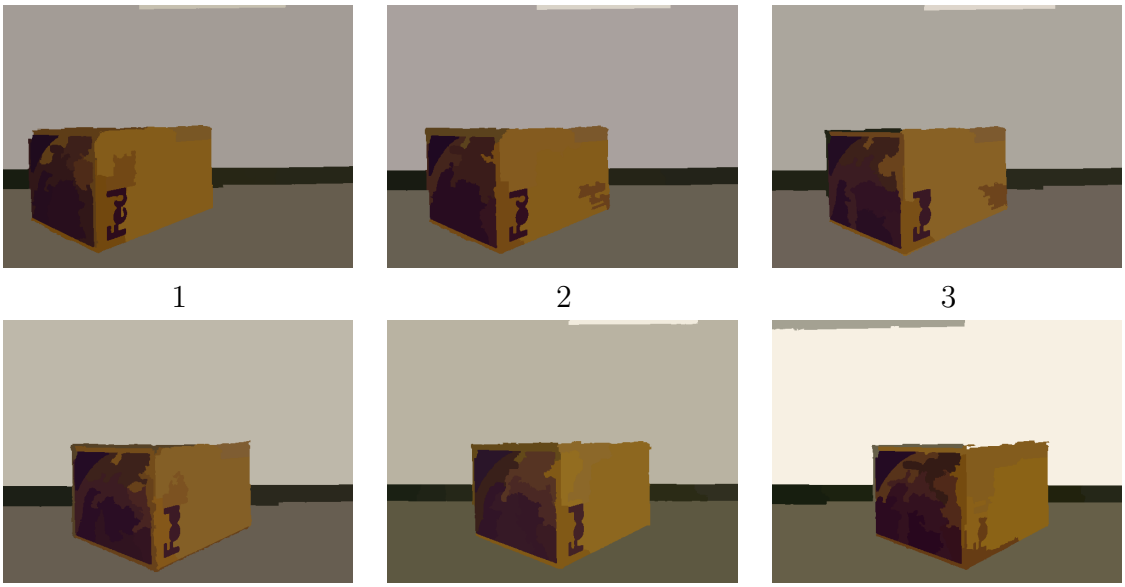


Figure 5.3: Segmentation results for the first test case. The number below each segmented image represents the number of iterations of the PSR algorithm.

each iteration is inconsistent due to lighting affects, camera view changes, and image noise. The segmentation process is also heavily reliant on the input parameters. Parameters selected to favour few and large segments do not work well when viewing objects that are far away or small, as they typically get clumped together with neighbouring objects into a larger segment and is treated as a single surface as opposed to many surfaces. The parameters favouring many small segments produces little or no surfaces because there are not be enough feature matches to be coupled with the segments for surface fitting. Since the accuracy of the planar estimation is reliant on the accuracy of the feature matches, ϵ_r and d_P are chosen to be more focused on finding good feature matches at the expense of a decreased number of matches. This makes it difficult to reconstruct the entire scene if the scene does not have enough distinguishable features on each object surface.

Overall the algorithm proves to work reasonably well when the environment fits the assumptions made. It also shows that distinct surfaces are able to be detected and reconstructed if the object surfaces are chromatically similar and have enough distinguishing features. Extending this notion, two boxes placed side by side should be able to be identified and reconstructed as separate entities. More tests of different scenarios are required to determine which parameter values work well for most cases.

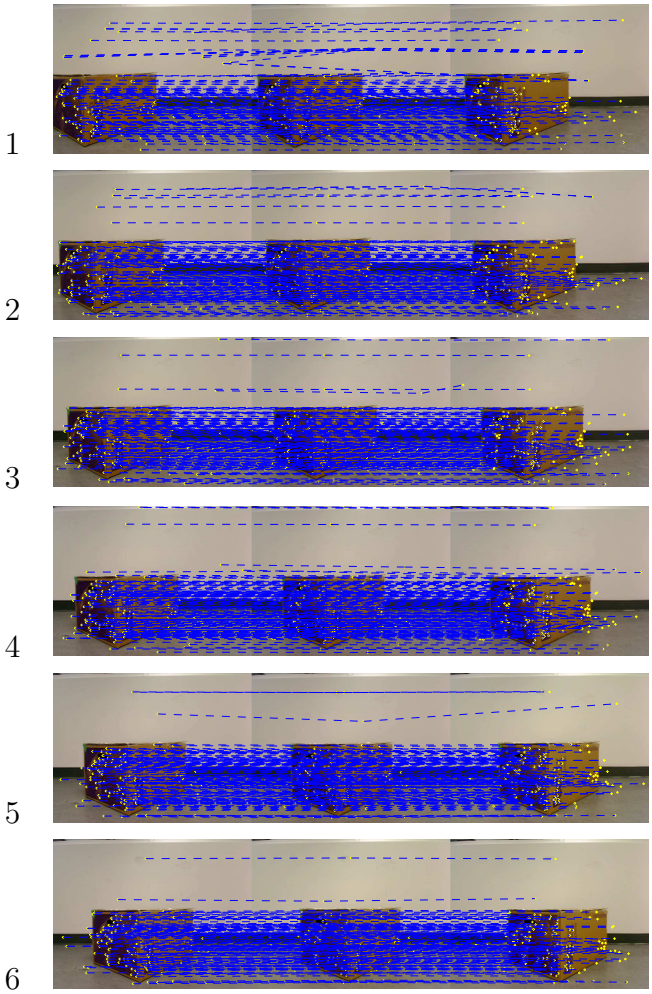


Figure 5.4: SIFT matching results for the first test case. The number to the left of each image represents the number of iterations of the PSR algorithm.

5.2 Second Test Case: Cluttered Environment

The second test case evaluates the algorithm in a cluttered indoor environment. The vehicle is travelling across the lab space and attempts to reconstruct a portion of the lab. Figure 5.5 shows the frames used in the video data, and since $n_{\text{frame}} = 4$, the total number of iterations for the algorithm is 11 (the first iteration uses frames 1-4, the second iteration uses frames 2-5, etc.). Figure 5.6 shows the resultant surface map. Because the assumptions made for the algorithm are not valid in this case, the quality of the environment map is also directly impacted as a result. The quality of some surfaces produced are relatively poor but the overall structure defined by the set of surfaces is indicative of the obstacles ahead. The height of the table is 90cm, and the surfaces defined by the algorithm also have an overall height of 90cm. The clutter of objects on the left and right side of the white drawers section in the middle also affects the quality of the surfaces produced based on the segmentation and feature matched results of those objects.

Figure 5.7 and Figure 5.8 shows the segmented image and feature matches for each iteration. It can be seen that the white drawers section is relatively flat overall and was segmented properly, so therefore properly reconstructed in Figure 5.6. The far away objects on the right and left sides of the drawers produced few segments due to the scale invariant nature of the segmentation process (many smaller objects are clumped together to form a larger segment). This, coupled with the matched features pertaining to objects of various distances (resulting in biased planar estimates), produced the skewed surfaces on the right and left sides of the drawers. The surface projection is sensitive to the plane estimates, as illustrated in Figure 5.9. It can be seen that a plane that is parallel to the optical axis (Figure 5.9(a)) has projected points which are unevenly distanced apart. Extrapolating the results shows that the projected points from 4 and onward become farther and farther away, meaning that any pixel error from the segmentation process can significantly increase the surface size away from the camera. Comparatively, a plane that is perpendicular to the optical axis results in projected points which are evenly spaced, and errors in the segment sizes are only significant for plane estimates which are far away from the camera, as the farther the plane is away the larger the spacing.

Overall the quality of the map produced from the algorithm is moderate because the conditions for the algorithm to perform well were invalid in this scenario. The algorithm also showed high sensitivity to the chosen parameters, where changing the parameters can significantly change the resultant surface map. The white drawers in the middle of the frames was reconstructed properly because it fulfilled the necessary assumptions.

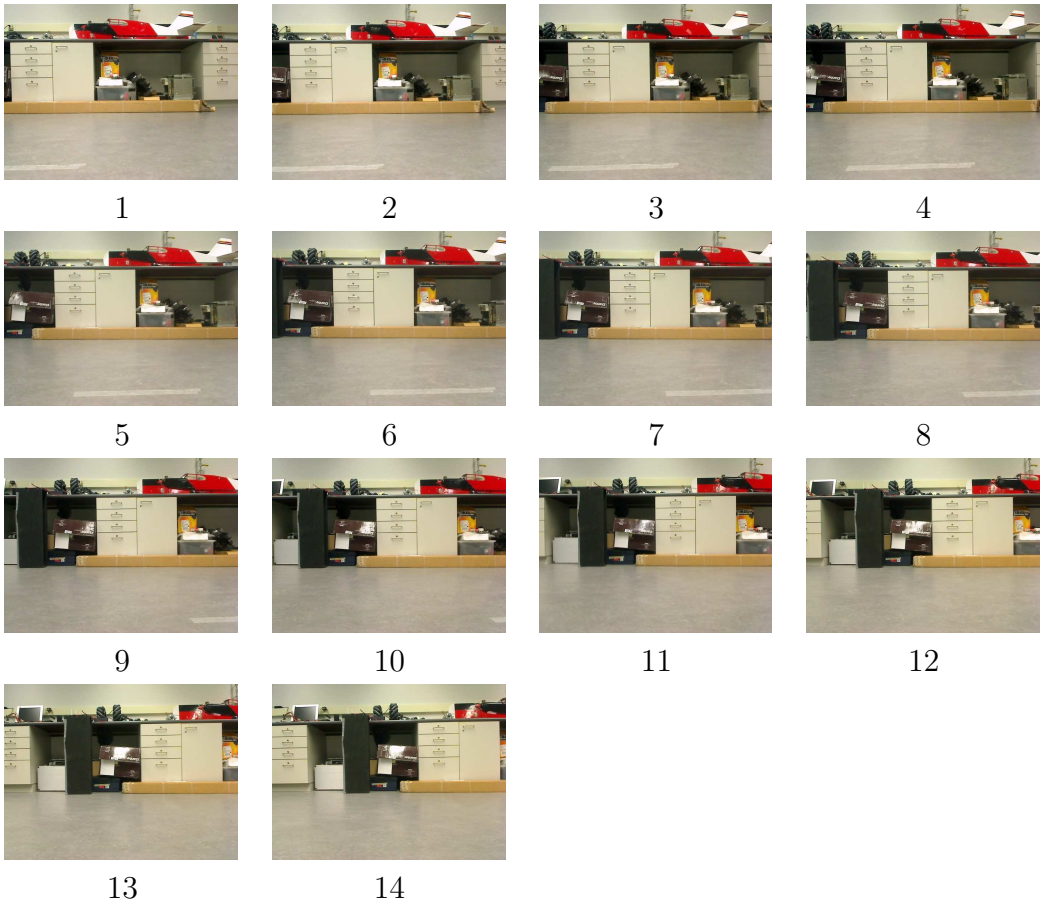


Figure 5.5: Image frames used for the second test case. The number below each frame corresponds to the frame's index in the sequence.

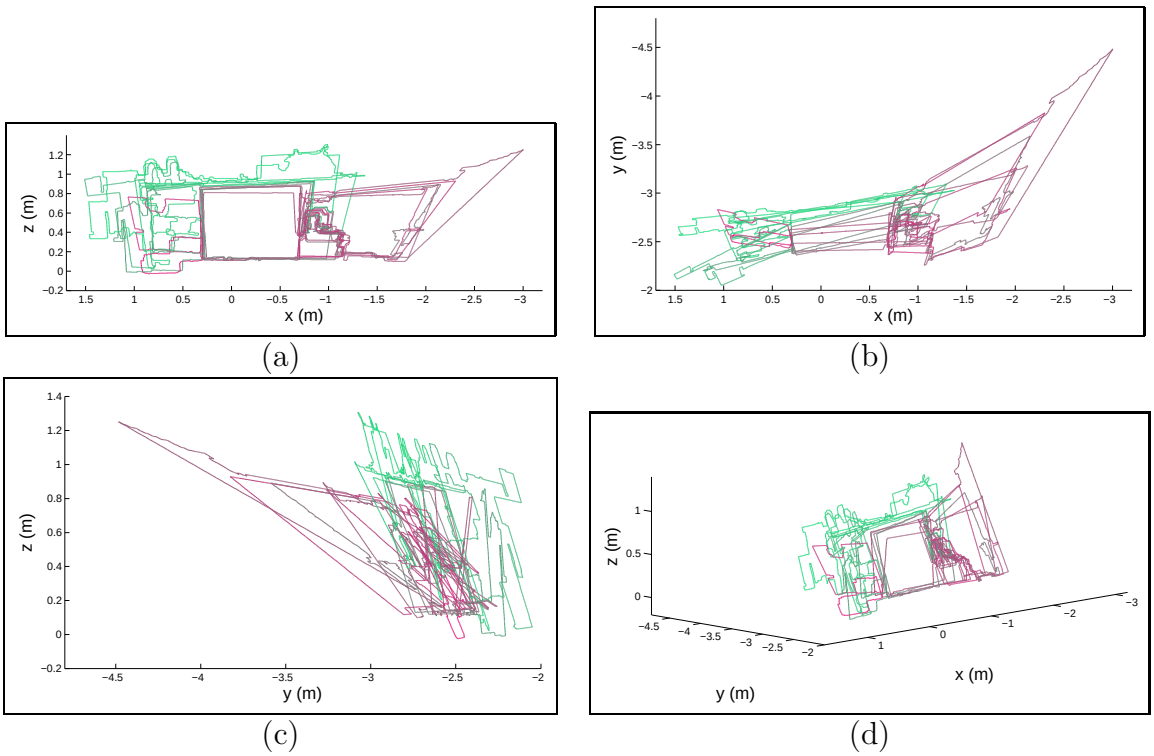


Figure 5.6: Surface mapping result for the second test case. The surfaces are shown as the coloured outlines, where each region is of a different colour. (a) Front view. (b) Top view. (c) Side view. (d) Angled view.

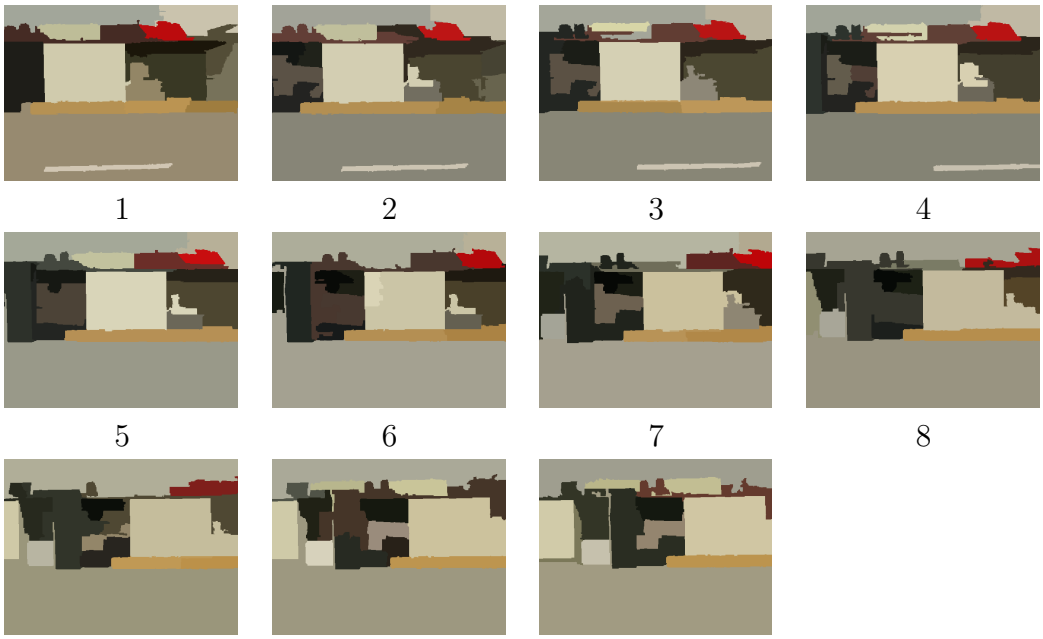


Figure 5.7: Segmentation results for the second test case. The number below each segmented image represents the number of iterations of the PSR algorithm.

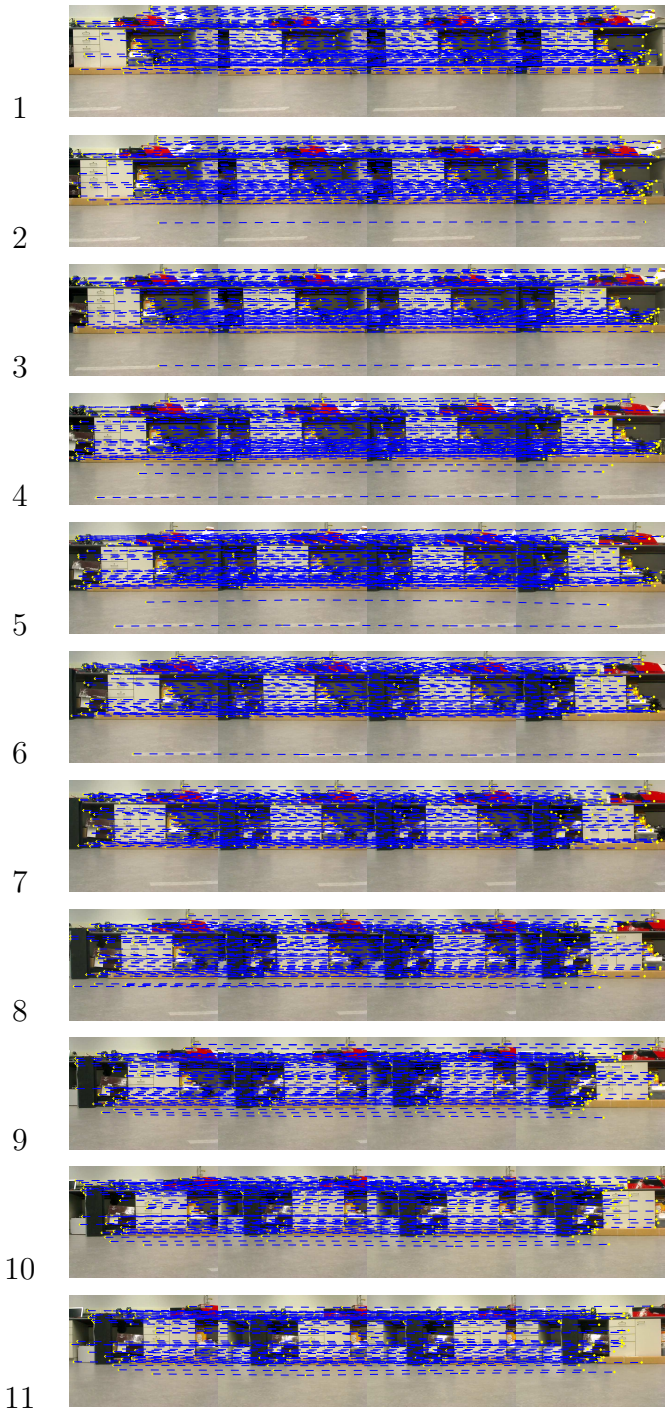


Figure 5.8: SIFT matching results for the second test case. The number to the left of each image represents the number of iterations of the PSR algorithm.

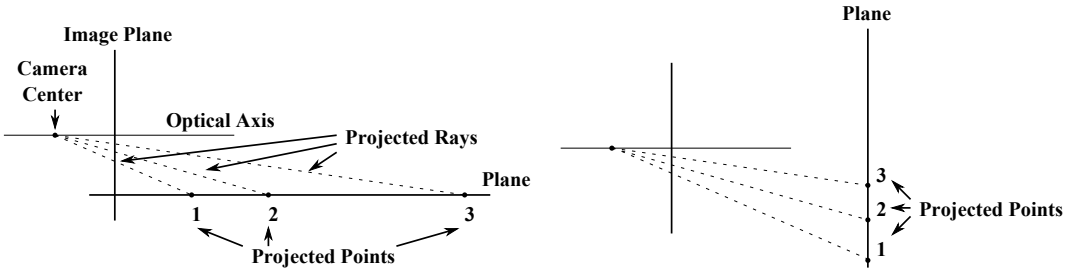


Figure 5.9: Illustration of the affects that the orientation of the plane of best fit have on the projection result. (a) A plane which is parallel and (b) a plane which is perpendicular to the optical axis of the camera.

5.3 Algorithm Processing Durations

The processing times of the main procedures used in the algorithm are shown in Figures 5.10 and 5.11, for the clean and cluttered scenarios, respectively. As shown in the two figures, the main bottlenecks are the SIFT feature extraction and mean shift segmentation. Both of these procedures can be sped up through parallelization and GPU implementation. The GPU implementation of SIFT extraction and mean shift segmentation would allow for processing speeds of about 10 fps to 20 fps or 0.05s to 0.1s sample rates. It is also noted that the Kuwahara filter, feature matching, planar fitting and surface projection can all be implemented in parallel to further increase the speed of the algorithm. The segmentation step can be a separate process altogether (as it is independent of the 3D mapping), which can result in further improvements in execution speeds. Therefore it is possible to speed up the algorithm significantly for it to be practical to be implemented in real-time mobile robotics applications.

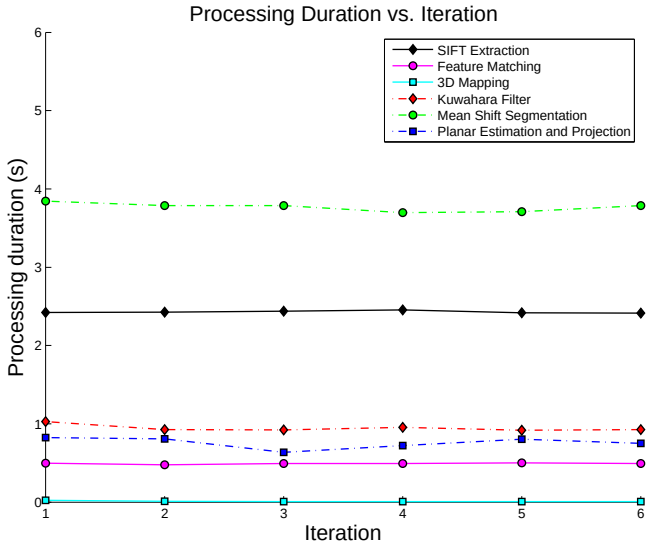


Figure 5.10: Processing durations of the main methods in the PSR algorithm for the first test case.

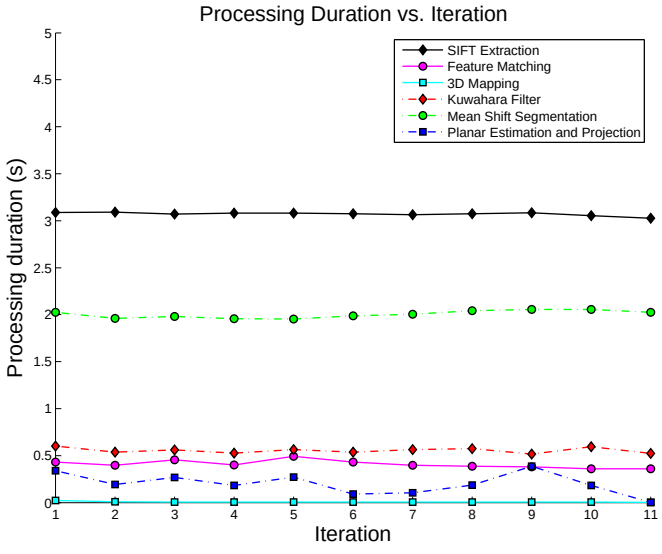


Figure 5.11: Processing durations of the main methods in the PSR algorithm for the second test case.

Chapter 6

Conclusion

Environment perception is both an essential and complicated task for autonomous vehicles. It is necessary for the vehicle to be able to perceive the obstacles in the environment in order for it to navigate around them and avoid possible collisions. This allows for the possibility of many practical applications that involves navigation through environments where there are potentially dangerous objects. Indoor environment perception for surveillance applications was chosen as the primary focus for this thesis with the vehicles considered being wheeled motor vehicles and quadrotor helicopters. Vehicle limitations include payload, size, power and cost restrictions, which are all important factors for determining the feasibility of the possible techniques to be used for obstacle detection. The dynamics of the vehicles were also taken into account. It is assumed that the environment has no moving objects with sufficient lighting conditions during the vehicle's surveillance routine. It was also assumed that the vehicle has accurate and reliable pose estimation.

Indoor environments consist of many kinds of objects with different structural and material characteristics. The placement of the objects can also produce cluttered scenes and occlusions. These make the task of developing a generalized method for identifying all objects a challenge. There are many different kinds of sensors that can be used to determine obstacles in the scene, with the camera selected because it minimizes payload and power requirements and captures abundant information from the scene. This thesis introduced a monocular vision-based obstacle detection algorithm that is designed to work for unknown indoor environments and slow moving vehicles. What had not been explored by previous works was the coupling of a generalized object surface detection method with a 3D point map to produce more informative and additional surface details of the scene. This notion was evaluated here, through the development of the PSR algorithm, as the main contribution to the field.

The PSR algorithm was divided into four main steps: feature extraction, feature tracking, 3D mapping, and surface fitting. The key assumptions made for simplifying the

algorithm were that the indoor scene consists mainly of flat and chromatically unique surfaces. The feature extraction step extracts SIFT features from each image in the sequence and then tracked in the feature tracking step. The 3D mapping step takes these features to form a 3D point cloud of the scene. Segmentation using a combination of the Kuwahara filter and mean shift segmentation was used to find segments that are assumed to represent flat object surfaces, and then coupled with the 3D point cloud to project these surfaces into the environment map. The resultant map consists of both surfaces and points that are assumed to represent obstacles in the scene.

An autonomous vehicle platform was developed to assess the algorithm. The testbed is a four wheeled ground vehicle, based off a RC chassis and retrofitted to contain control hardware. Because of irreparable collision, an alternate vehicle platform known as the Chameleon from Clearpath Robotics was used to perform the experiments offline, while the second version of the testbed is being developed for the real-time implementation of the algorithm. The Chameleon obtained video data using a Logitech Quickcam Pro 9000 camera and odometry data using the OptiTrack system integrated in the lab space. Two scenarios were assessed for this thesis. The first scenario is a clean scenario where the key assumptions made (planar surfaces where each surface consists colours that do not vary much) are valid, which involves the vehicle driving and capturing video data of a box placed on the floor. The second scenario tests robustness of the algorithm in a cluttered environment. Surfaces generated from the first scenario were fairly accurate, proving that the algorithm is able to reconstruct surfaces when the key assumptions are valid. For the second scenario, however the resultant surface map produced less satisfactory results mainly because the scene does not hold true to the key assumptions.

The PSR algorithm works reasonably well under the assumptions it was developed for, but is less effective in other environments. The 3D mapping produced reasonable results but the segmentation process was susceptible to view changes, scale changes, light affects, noise and clutter. This led to some differences in the segments pertaining to the same object surface on separate iterations, which adversely affected the quality of the surfaces produced. The surface projection was also more sensitive to fitted planes that were parallel to the camera's optical axis, which led to many sharp surfaces generated due to errors in the segmentation process. Currently the algorithm's processing bottlenecks are the SIFT extraction and the mean shift segmentation. Overall the experimental results prove that the coupling of object surface detection with 3D point estimates can produce a more informative and detailed surface map.

It is recommended that the algorithm be tested in a quadrotor testbed in future research. The speed of the algorithm can be improved significantly through parallelization and GPU implementation of the SIFT extraction, mean shift segmentation, Kuwahara filter, feature matching, planar fitting and surface projection. Another recommendation is to reduce the complexity of the algorithm by reducing the number of parameters used for

its input, as the algorithm is heavily reliant on the tuning of the parameters. This can be done by performing additional experiments to determine reliable values of specific parameters. Future investigation can be done to include additional methods or change the current methods of the PSR algorithm. Surface fusion techniques can be included in a future version of the algorithm to reduce the number of redundant surfaces. The linking of each surface can be done by tracking a subset of the features associated to a surface between frames. In addition, a generalized technique for detecting object surfaces can be developed to replace the current one, where it encompasses the detection of non-planar surfaces and surfaces that consists of various different scales and colours. More complex surface fitting techniques, such as cubic-spline and B-spline interpolations, can be implemented to account for smooth surfaces.

In conclusion, vision-based obstacle detection remains a challenge for autonomous vehicles. The contributions for this thesis include the development of an autonomous ground vehicle testbed and the introduction, development and evaluation of the PSR algorithm, which incorporates object surface recognition into dense 3D reconstruction to improve the quality of the generated surface map. It is hoped that in the near future the goal of detecting and avoiding obstacles is achieved, which will bring humanity one step closer to the realization of a fully autonomous vehicle.

References

[1] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin. The Stanford testbed of autonomous rotorcraft for multi-agent control (STARMAC). In *Proc. IEEE 23rd Conference on Digital Avionics Systems*, Salt Lake City, UT, USA, November 2004.

[2] C. Plagemann, F. Endres, J. Hess, C. Stachniss, and W. Burgard. Monocular range sensing: A non-parametric learning approach. In *Proc. IEEE International Conference on Robotics and Automation*, pages 929–934, Pasadena, CA, USA, May 2008.

[3] B. Call, R. Beard, C. Taylor, and B. Barber. Obstacle avoidance for unmanned air vehicles using image feature tracking. In *Proc. AIAA Conference on Guidance, Navigation, and Control*, Keystone, CO, USA, August 2006.

[4] S. Hrabar. 3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 807–814, Nice, France, September 2008.

[5] D. J. Lee, R. W. Beard, P. C. Merrell, and P. Zhan. See and avoidance behaviors for autonomous navigation. In *Proc. SPIE Volume 5609: Mobile Robots XVII*, pages 23–34, Philadelphia, PA, USA, October 2004.

[6] S. Yoon, S. Roh, and Y. Shim. Vision-based obstacle detection and avoidance: Application to robust indoor navigation of mobile robots. *Advanced Robotics*, 22(4):477–492, 2008. VSP, an imprint of Brill.

[7] J. Byrne, M. Cosgrove, and R. Mehra. Stereo based obstacle detection for an unmanned air vehicle. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2830–2835, Orlando, FL, USA, May 2006.

[8] C. Stiller, J. Hipp, C. Rössig, and A. Ewald. Multisensor obstacle detection and tracking. *Image and Vision Computing*, 18(5):389–396.

[9] H. Baltzakis, A. Argyros, and P. Trahanias. Fusion of laser and visual data for robot motion planning and collision avoidance. *Machine Vision and Applications*, 15(2):92–100, 2003. Springer-Verlag.

[10] G. Fasano, D. Accardo, A. Moccia, and L. Paparone. Airborne multisensor tracking for autonomous collision avoidance. In *Proc. IEEE 9th International Conference on Information Fusion*, pages 1–7, Florence, Italy, July 2006.

[11] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 1981. Elsevier.

[12] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. IJCAI International Joint Conference on Artificial Intelligence*, volume 2, pages 674–679, Vancouver, B.C., Canada, August 1981.

[13] A. Verri and T. Poggio. Against qualitative optical flow. In *Proc. IEEE 1st International Conference on Computer Vision*, pages 171–180, London, UK, June 1987.

[14] J. C. Zufferey and D. Floreano. Toward 30-gram autonomous indoor aircraft: Vision-based obstacle avoidance and altitude control. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2594–2599, Barcelona, Spain, April 2005.

[15] R. C. Nelson and J. Aloimonos. Obstacle avoidance using flow field divergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1102–1106, October 1989. IEEE Computer Society.

[16] J. J. Koenderink and A. J. van Doorn. Affine structure from motion. *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, 8(2):377–385, February 1991. OSA.

[17] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. IEEE Computer Society.

[18] R. O. Duda and P. E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. ACM Press.

[19] S. M. Smith and J. M. Brady. SUSAN A new approach to low Level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997. Springer-Verlag.

[20] H. Wang and M. Brady. Real-time corner detection algorithm for motion estimation. *Image and Vision Computing*, 13(9):695–703, 1995. Elsevier.

[21] M. Trajkovi and M. Hedley. Fast corner detection. *Image and Vision Computing*, 16(2):75–87, 1998. Elsevier.

[22] C. Harris and M. Stephens. A bombined corner and edge detection. In *Proc. 4th Alvey Vision Conference*, volume 15, pages 147–151, Manchester, Greater Manchester, UK, August 1988.

[23] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *Computer Vision ECCV 2006*, 3951:404–417, 2006. Springer-Verlag.

[24] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. Springer-Verlag.

[25] J. Shi and C. Tomasi. Good features to track. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, Seattle, WA, USA, June 1994.

[26] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle Adjustment A Modern Synthesis. *Vision Algorithms: Theory and Practice*, 1883:153–177, 2000. Springer-Verlag.

[27] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2003.

[28] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[29] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1498–1505, San Francisco, CA, USA, June 2010.

[30] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.-M. Frahm, R. Yang, D. Nister, and M. Pollefeys. Real-time visibility-based fusion of depth maps. In *Proc. IEEE 11th International Conference on Computer Vision*, pages 1–8, Rio de Janeiro, Brazil, October 2007.

[31] M. Pollefeys, D. Nistr, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewnius, R. Yang, G. Welch, and H. Towles. Detailed real-time urban 3D reconstruction from video. *International Journal of Computer Vision*, 78(2):143–167, 2008. Springer-Verlag.

[32] S. M. Seitz, B. Curless, J. Diebel, D.Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 519–528, New York, NY, USA, June 2006.

[33] P. Premaratne and F. Safaei. Stereo correspondence using moment invariants. 15:447–454, 2008. Springer-Verlag.

[34] E. Hanna, P. Straznicky, and R. Goubran. Obstacle detection for low flying unmanned aerial vehicles using stereoscopic imaging. In *Proc. IEEE Conference on Instrumentation and Measurement Technology*, pages 113–118, Victoria, BC, Canada, May 2008.

[35] T. E. Boult and G. Wolberg. Correcting chromatic aberrations using image warping. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 684–687, Champaign, IL, USA, June 1992.

[36] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000. IEEE Computer Society.

[37] G. A. Mastin. Adaptive filters for digital image noise smoothing: An evaluation. *Computer Vision, Graphics, and Image Processing*, 31(1):103–121, July 1985. Elsevier.

[38] B. Bascle, A. Blake, and A. Zisserman. Motion deblurring and super-resolution from an image sequence. *Computer Vision ECCV '96*, 1065:571–582, 1996. Springer-Verlag.

[39] L. Muratet, S. Doncieux, Y. Briere, and J. A. Meyer. A contribution to vision-based autonomous helicopter flight in urban environments. *Robotics and Autonomous Systems*, 50(4):195–209, 2005. Elsevier.

[40] K. Souhila and A. Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):13–16, 2007. Vienna University of Technology.

[41] J. Merchant and F. Pope. Micro UAV collision avoidance. In *Proc. SPIE Volume 6561: Unmanned Systems Technology IX*, page 65610K, Orlando, FL, USA, April 2007.

[42] S. J. Galvin, R. P. O'Shea, A. M. Squire, and D. G. Govan. Sharpness overconstancy in peripheral vision. *Vision Research*, 37(15):2035–2039, 1997. Elsevier.

[43] T. Gandhi, M. T. Yang, R. Kasturi, O. Camps, L. Coraor, and J. McCandless. Detection of obstacles in the flight path of an aircraft. *IEEE Transactions on Aerospace and Electronic Systems*, 39(1):176–191, January 2003. IEEE Computer Society.

[44] P. C. Merrell, D. J. Lee, and R. W. Beard. Obstacle avoidance for unmanned air vehicles using optical flow probability distributions. In *Proc. SPIE Volume 5609: Mobile Robots XVII*, pages 13–22, Philadelphia, PA, USA, October 2004.

[45] S. Hrabar, G. S. Sukhatme, P. Corke, K. Usher, and J. Roberts. Combined optic-flow and stereo-based navigation of urban canyons for a UAV. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Alberta, Canada.

[46] J. Byrne and C. J. Taylor. Expansion segmentation for visual collision detection and estimation. In *Proc. IEEE International Conference on Robotics and Automation*, pages 875–882, Kobe, Japan, May 2009.

[47] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE International Conference on Robotics and Automation*, pages 116–121, St. Louis, MO, USA, March 1985.

[48] J. Candamo, R. Kasturi, D. Goldgof, and S. Sarkar. Vision-based on-board collision avoidance system for aircraft navigation. In *Proc. SPIE Volume 6230: Unmanned Systems Technology VIII*, page 62300X, Orlando, FL, USA, April 2006.

[49] A. Rankin, A. Huertas, and L. Matthies. Evaluation of stereo vision obstacle detection algorithms for off-road autonomous navigation. In *Proc. AUVSI 32nd Unmanned Systems Symposium*, Baltimore, MD, USA, June 2005.

[50] J. Michels, A. Saxena, and A. Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proc. ACM 22nd international Conference on Machine Learning*, pages 593–600, Bonn, Germany, August 2005.

[51] M. Tomono. 3D object mapping by integrating stereo SLAM and object segmentation using edge points. *Advances in Visual Computing*, 5875:690–699, 2009. Springer-Verlag.

[52] T. G. McGee, R. Sengupta, and K. Hedrick. Obstacle detection for small autonomous aircraft using sky segmentation. In *Proc. IEEE International Conference on Robotics and Automation*, pages 4679–4684, Barcelona, Spain, April 2005.

[53] C. Braillon, C. Pradalier, J. L. Crowley, and C. Laugier. Real-time moving obstacle detection using optical flow models. In *Proc. IEEE Intelligent Vehicle Symposium*, pages 466–471, Tokyo, Japan, June 2006.

[54] J. K. Anderson, K. M. Iftekharuddin, E. Threlkeld, and B. Montgomery. Single camera-based object detection and tracking for mobile robots. In *Proc. SPIE Volume 7072: Optics and Photonics for Information Processing II*, page 70720T, San Diego, CA, USA, August 2008.

[55] K. Konolige, M. Agrawal, R. Bolles, C. Cowan, M. Fischler, and B. Gerkey. Outdoor mapping and navigation using stereo vision. *Experimental Robotics*, 39:179–190, 2008. Springer-Verlag.

[56] R. Steffen and W. Förstner. On visual real time mapping for unmanned aerial vehicles. In *Proc. 21st Congress of the International Society for Photogrammetry and Remote Sensing*, pages 57–62 Part B3a, Beijing, China, 2008.

[57] J. Dornauer, G. Kotsis, C. Bernthaler, and M. Naderhirn. A comparison of different computer vision methods for real time 3D reconstruction for the use in mobile robots. In *Proc. ACM SIGMM 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 136–141, Linz, Austria, November 2008.

[58] R. Rossi, X. Savatier, J. Y. Ertaud, and B. Mazari. Real-time 3D reconstruction for mobile robot using catadioptric cameras. In *Proc. IEEE International Workshop on Robotic and Sensors Environments*, pages 104–109, Lecco, Italy, November 2009.

[59] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1067–1073, San Juan, Puerto Rico, June 1997.

[60] S. Se and P. Jasiobedzki. Stereo-vision based 3D modeling and localization for unmanned vehicles. *International Journal of Intelligent Control and Systems*, 13(1):47–58, March 2008. Westing Publishing Co.

[61] D. G. Lowe. Object recognition from local scale-invariant features. In *Proc. IEEE 7th International Conference on Computer Vision*, volume 2, pages 1150–1157, Corfu, Greece, September 1999.

[62] J. Babaud, A. P. Witkin, M. Baudin, and R. O. Duda. Uniqueness of the Gaussian Kernel for Scale-Space Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1):26–33, January 1986. IEEE Computer Society.

[63] T. Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1993.

[64] J. E. Kyprianidis, H. Kang, and J. Döllner. Image and video abstraction by anisotropic Kuwahara filtering. *Computer Graphics Forum*, 28(7):1955–1963, October 2009. Wiley-Blackwell.

[65] M. Kuwahara, K. Hachimura, S. Eiho, and M. Kinoshita. Digital processing of biomedical images. pages 187–203, 1976. Plenum Press.

[66] D. Comanicu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002. IEEE Computer Society.

[67] P. Bakker, L. J. van Vliet, and P. W. Verbeek. Edge preserving orientation adaptive filtering. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 1535, Fort Collins, Colorado, USA, June 1999.

[68] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, January 1975. IEEE Computer Society.

[69] C. M. Christoudias, B. Georgescu, and P. Meer. Synergism in low level vision. In *Proc. IEEE 16th International Conference on Pattern Recognition*, volume 4, pages 150–155, Quebec, Canada, August 2002.

[70] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience, New York, 2nd edition, 1991.

[71] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 18(1):54–64, 1969. JSTOR.

[72] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, 1986.

[73] R. Pradhan, R. Agarwal, S. Kumar, M. P. Pradhan, and M. K. Ghose. Contour line tracing algorithm for digital topographic maps. *International Journal of Image Processing (IJIP)*, 4(2):156–163, 2010. CSC Journals.

[74] E. S. Spelke. Principles of object perception. *Cognitive Science*, 14(1):29–56, 1990. Elsevier.

[75] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[76] S. Bagon. Matlab interface for EDISON. http://www.wisdom.weizmann.ac.il/~bagon/matlab.html, 2011.

[77] Gerardo Salas. Optimization of 3-Axis Vertical Milling of Sculptured Surfaces. Master's thesis, University of Waterloo, Canada, 2010.

[78] S. Limsoonthrakul, M. N. Dailey, M. Srisupundit, S. Tongphu, and M. Parnichkun. A modular system architecture for autonomous robots based on blackboard and publish-subscribe mechanisms. In *Proc. IEEE International Conference on Robotics and Biomimetics*, pages 633–638, Bangkok, Thailand, February 2009.

[79] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Proc. IEEE International Conference on Robotics and Automation: Workshop on Open Source Software*, Kobe, Japan, May 2009.

[80] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the DARPA Grand Challenge. *The 2005 DARPA Grand Challenge*, 36:1–43, 2007. Springer-Verlag.