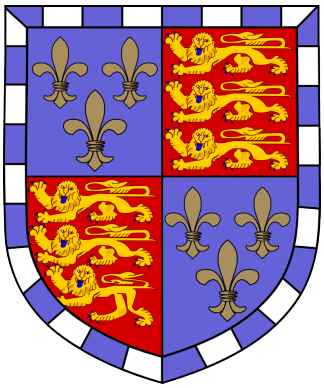
# Model Localisation & Terrain Mapping for Tabletop Wargames

F-JL-type(b)



**Oliver Jones**
Christ's College

Supervisor: Joan Lasenby

Department of Engineering
University of Cambridge

This report is submitted for the degree of
*Master of Engineering*

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

*Signed:*_____ *Date:* 27th May 2020

# Acknowledgments

I would like to thank my supervisor Joan for her constant support and encouragment, particularly her patience when I try to explain the rules of Warhammer. A thanks also goes to her PhD students Hugo and Alex for their input in meetings and presentations.

Finally, a special thanks to CAPE's Professor Chu and Huawei's Michael Hill-King for their financial support through the CAPE Acorn IIB Project prize, the money from which contributed towards the cameras used that were instrumental to this project.

# Abstract

*Tabletop wargames* covers many different games such as Warmachine, Riot Quest, Warhammer 40,000 (40k) and Age of Sigmar. While the rules for each game are often very different, they all consist of at least 2 players with a number of miniatures on a tabletop at least $4' \times 4'$ as well as various scenery items such as forests and buildings to create dynamic terrain. A game is played by units interacting with one another and the board based on the relative distance between objects and dice rolls to achieve given objectives. The size of a game itself can vary wildly from 4-5 miniatures under a certain ruleset, taking an hour to play a game, to a full weekend-long campaign using 1000+ miniatures and items of scenery to re-create a battle.

A key problem that can arise over the course of a game is the determination of *Line of Sight* (LOS) - a task that involves a player determining what a miniature can 'see' from its point of view, which sometimes cannot be done without disturbing the positions of other models on the board. This project seeks to solve this problem using contributions centred around making a digital reconstruction of a board from a series of images. Information from this digital board can then also be used to provide analysis of a game and suggestions to players.

Starting with the registration of point cloud data, variations on the popular algorithm Iterative Closest Point (ICP) are examined, making use of colour and normal information to improve the rate of convergence and quality of correspondence estimates. The quality of registrations are enhanced by providing a bespoke method based on a clustering-least-squares algorithm for locating common geometries on a wargames board for a strong initial alignment. Considerations are also made as to how the number of degrees of freedom can be reduced to speed up optimisation while still maintaining a robust, high-quality alignment.

Supported with DBSCAN clustering, a classification method is carried out using the novel constrained ICP to match clusters against a bank of precision photogrammetric scans of the relevant miniatures. This simultaneously estimates the position and orientation of miniatures on the board to complete reconstruction.

Several methods are then discussed and implemented for actually solving the LOS problem given a digital board. These methods include estimating the silhouettes of miniatures from a viewpoint and ray-casting to deal with arbitrarily complex shapes, the information from which forms the basis for a statistical treatise on interactions in the game. As a part of this statistical analysis, a number of concepts based on results from the Binomial distribution provide a framework from which actions can be considered and compared against one another in a discrete optimised control problem.

# Contents

# Chapter 1

# Introduction

*Tabletop wargames* cover many different games such as Warmachine, Riot Quest[1], Warhammer 40,000 (40k) and Age of Sigmar[2]. While the rules for each game are often very different, they all consist of at least 2 players with a number of miniatures on a tabletop at least $4' \times 4'$ (Figure 1.1) as well as various scenery items such as forests and buildings to create dynamic terrain. A game is played by units interacting with one another and the board based on the relative distance between objects and dice rolls to achieve given objectives. The size of a game itself can vary wildly from 4-5 miniatures under a certain ruleset, taking an hour to play a game, to a full weekend-long campaign using 1000+ miniatures and items of scenery to re-create a battle.

(a) Typical 40k game

(b) Large scale 40k diorama

Figure 1.1: Examples of Tabletop Wargaming boards

There are two key problems that can arise in the practicalities of playing a game leading to disputes between players:

- *Line of Sight* (LOS) of a miniature needs to be determined for certain interaction rules such as attacking or determining the benefits of cover. This requires players to be able to move their eye-line down to the model's-eye view. In some instances, this can be particularly difficult if the given miniature is in a densely packed part of the board, or a tightly enclosed area of scenery, where LOS cannot be determined without displacing other miniatures thus ruining the board state.

[1] `home.privateerpress.com`
[2] `games-workshop.com`

- *Distances between miniatures* are important for movements and close-quarters interactions, such measurements are taken between the circular bases of the miniatures. Similar to LOS, getting a measuring implement between miniatures when they are close together can be very difficult without disrupting the board state.

Often these disputes have to be resolved randomly with a dice roll or coin toss, despite being technically a deterministic problem. If it were possible to re-create the board in a digital space it would be possible to resolve these issues with much more precision. Moreover, a digital board can provide additional insights into a game, making use of the probabilistic nature of actions to generate performance metrics for miniatures and record their performance.

Tabletop wargaming as a past-time involves a variety of interlinked practices including collecting, crafting, painting and storytelling as well as the gameplay described above. As a result, players will often create characters behind their miniatures, and use the events of the battles played with them to craft a unique character within the lore of the universe in which the game is set. [8] & [9] demonstrate a detailed ethnographic study of digitally recording stories behind meaningful objects, including wargames miniatures, but require manual input to digitally record and track data. Being able to track a game in near real-time would provide an automated framework around which characters can be created.

The process of digitally re-creating a board will require 4 key tasks for which this report will be the focus of:

- *Data Acquisition* - gathering geometric information about the scene to generate 3D point clouds which can then be analysed to locate regions of interest.

- *Registration* - aligning multiple 3D views to reduce occlusion as well as part of a method for matching miniatures within a scene.

- *Segmentation* - breaking a scene into Regions of Interest (ROI) on which to conduct more precise analytics such as improving registration or classification.

- *Classification* - identifying if each ROI is a model or other useful feature that needs to be taken into account in the board state.

Chapter 2 outlines a specific version of the problems described here to help guide investigations in an insightful manner. Details are also given as to how datasets were created and the resources available for the task. Chapter 3 explains the different approaches for aligning views of a scene to combine them using *Iterative Closest Point* (ICP) and provides a method for getting a robust initial alignment for tabletop wargaming data. Chapter 4 looks at two non-parametric approaches for clustering the tabletop scene to help both with registration and classification. Chapter 5 discusses methods of creating reference/ matching data from which to the segmentation clusters can be matched. A breakdown of how ICP can be adapted to deal with constrained rotations and translations is given so that matching can be carried out reliably even with sparse data. Chapter 6 deals with how the digital board can be used to analyse the LOS of miniatures through both ray-casting and silhouette analysis. Finally, Chapter 7 provides some useful statistical results and a framework that can be built on top of the digital tabletop to analyse and compare the quality of actions that a player can take in-game.

## 1.1 Nomenclature

For clarity, a definition of common terms is provided in Table 1.1.

Table 1.1: Definition of typical Wargaming terms

| Term | Definition |
|---|---|
| attack | An action a unit can take against another unit consisting of a number of shots all with the same statistics. |
| board state | The information required to determine what actions are currently available to each player. Requires knowledge of the scene, the player whose turn it is, and the phase of the turn i.e. movement, shooting, combat etc. |
| damage | When a unit receives a certain amount of damage, individual models within the unit are removed from the game. |
| miniatures | A figurine, typically on a round base that represents a character/ soldier in the game. A game is carried out by miniatures interacting with one another to achieve an objective |
| models | Both miniatures and scenery - any object that can be interacted with or provides an effect on the board state |
| scene | Geometric information detailing the physical location of models on the board. |
| scenery | Fixed items that are placed on the board to obscure line of sight or provide objectives for the scenario in the game. These stay in place throughout the game. |
| shot | A single event that occurs during an attack which upon success causes a single damage against the target. |
| unit | A collection of miniatures that act together, typically with the same rules/ equipment/ statistics. A unit can also be a single miniature. |

# Chapter 2

# The Problem

## 2.1 Problem Statement

The task presented in the previous chapter is a large one; to cover all the aspects within the scope of this project, this report shall act as a feasibility study, looking at how each of the key tasks could be carried out. To focus investigations, the problems detailed previously can be distilled into the following problem statement:

> *Given a limited number of views of a scene, comprising of 4 miniatures and an item of scenery on a flat surface, digitally reconstruct the scene to allow the full board state to be determined.*

It is important that the statement captures the key aspects of the problem such that the solution is not trivial, but is still practical within the resources available. A successful solution to this statement should serve as a proof of concept for further development into a real-time application. The statement is justified as follows:

- *Given a limited number of views of a scene* - one of the key aspects of being able to scale this up to a real-time application is reducing the amount of data that needs to be processed. 100+ pictures of a scene can provide a very accurate reconstruction without any additional information about what the images contain, but will take many hours to produce accurately.

- *comprising of 4 miniatures and an item of scenery* - as discussed previously, while a number of wargames use 100+ miniatures, there are some skirmish-based games such as 40k Killteam [12] that use only 4-10 miniatures. There are also practicality issues with matching many miniatures and acquiring enough data for classification. Having a small number of miniatures and an item of scenery means that several ambiguous LOS problems can still be recreated without having to worry about large amounts of training data.

- *on a flat surface* - as a starting point for the problem, being able to identify simple geometries such as planes can be done efficiently without complex search algorithms and the focus can be put on processing miniatures' data. This is also not too onerous a restriction as boards tend to be flat so that miniatures can be placed on them without falling over.

## 2.2 Resources & Facilities

Point clouds give simple 3D data about a scene that can be further manipulated and segmented. There are numerous methods for creating point clouds from 2D images [16], but these can take a long time to process (See Chapter 5.1.2). Nonetheless, several devices can generate point clouds in real-time such as the Intel RealSense series [17] and Microsoft's Kinect devices [29]. These devices use either an Active IR stereo camera or time of flight to create a point cloud with RGB colour mapping on the data.

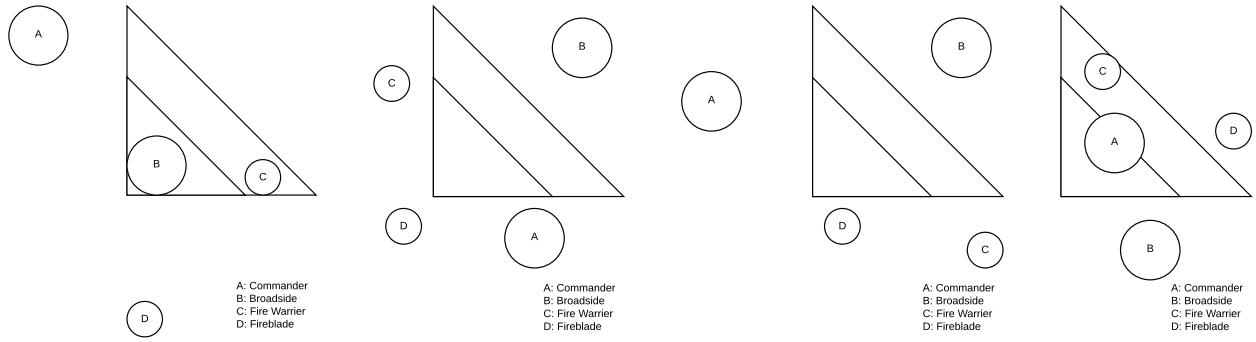
Due to the Kinect SDK being no longer supported by Microsoft except for the new (and more expensive) Azure Kinect device, as well as the wider support in several online applications for the RealSense series, the RealSense was used for the large majority of scene capture tasks. Specifically, the Intel RealSense D415 was chosen. This device makes use of 2 IR cameras to construct point clouds from points identified from a projected IR pattern and then matched with data from the RGB camera to colour the points.

As the project is largely a proof of concept for a wider wargaming framework, industry-standard C++ libraries such as PCL [5] were avoided in preference of Python-based libraries where implementation is more straightforward at the sacrifice of performance and flexibility. Open3d [26] was the library of choice, as it contains primitive point cloud manipulation functionality as well as high-level implementations of recent algorithms such as Coloured Iterative Closest Point [20] and built-in visualisation capabilities.

## 2.3 Dataset

The dataset upon which the experiments detailed in this report are run are sets of views taken with the RealSense of 4 different tabletop scenarios. Each scene contains the same 4 miniatures - 2 large and 2 small from the 40K miniatures range (Figure 2.2), and a laser-cut building[1] (Figure 2.3). A rough layout of each of the tabletop scenarios are given in Figure 2.1. Two of the layouts involve miniatures inside the building; a more ambiguous LOS and segmentation scenario, while the remaining two are much simpler with each model clearly separable from one another. Typically 5-8 point cloud images were then captured for a player's-eye view, spaced evenly around the scene to ensure sufficient geometric information about each model.

[1] Using a custom building rather than a consumer bought building such as this allows for the CAD model to easily be converted into a triangle mesh for referencing.




(a) Layout 1 (b) Layout 2 (c) Layout 3 (d) Layout 4

Figure 2.1: Each layout from which point cloud data is captured.

Figure 2.2: Miniatures used to create each scene. From left to right (a) Broadside (b) Commander (c) Fire Warrior (d) Fireblade. All miniatures are from Games Workshops's Warhammer 40,000 range.

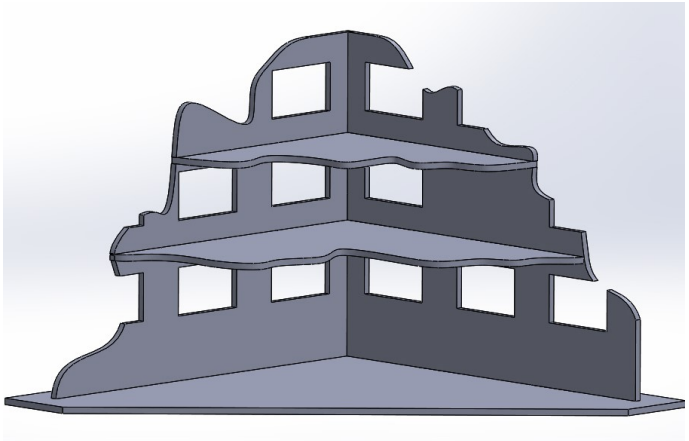
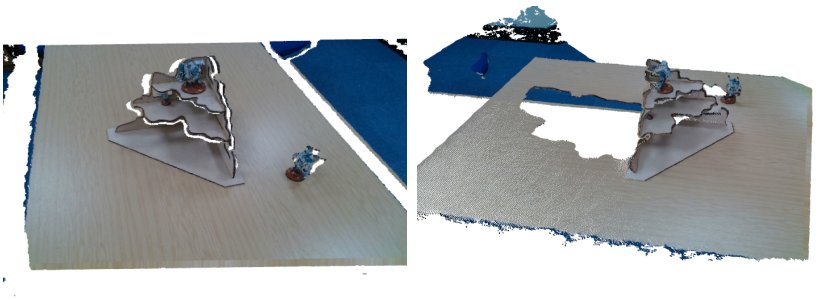


Figure 2.3: CAD model of the building created for use in scenes. The final building was laser cut from 4mm MDF. The height of the final building is 20cm.

# Chapter 3

# Registration

The point cloud views contain no direct information on the position of the cameras relative to one another. Since no view can capture all the information on the scene there are many occluded areas as can be seen in Figure 3.1. However, if these views can be aligned, making use of any overlap, then a clearer view of the scene from all angles can be found. Hence, the first part of the scene reconstruction process is to align all of the 3D point clouds to reduce occlusions in the scene and get a fuller picture of what is going on. If the correspondences between each cloud are known then calculating the alignment transformation is trivial. However, often the correspondences need to be estimated as a part of the algorithm and an iterative process is created.



(a) Front (b) Side

Figure 3.1: A single view from Layout 1. The Fireblade is not visible in this view, and so other views must be combined with it to reconstruct the whole scene.

## 3.1 Iterative Closest Point

From the early 90s, there has been a family of registration algorithms developed based around the method of Iterative Closest Point (ICP). There are two main flavours: Point to Point [4] and Point to Plane [6] - both operate in a very similar way but use a different objective function that is optimised.

Given a reference or target point cloud $\{\mathbf{t}_i\}_{i=1}^{N}$ with normals $\{\mathbf{n}_i\}_{i=1}^{N}$ and a roughly aligned or source point cloud $\{\mathbf{s}_i\}_{i=1}^{L}$, where $\mathbf{t}_i, \mathbf{n}_i, \mathbf{s}_i \in \mathbb{R}^4$ are homogenous coordinates of a point and $L \neq N$, a $4 \times 4$ transformation $\mathbf{M}$ can be found to align the two sets of points. With some policy to find a correspondence set of equal size for each cloud (discussed further below) the following objective functions can be optimised:

$$\mathbf{M}_{\text{point}} = \underset{\mathbf{M}}{\operatorname{argmin}} \sum_i (\mathbf{M}\mathbf{s}_i - \mathbf{t}_i)^T (\mathbf{M}\mathbf{s}_i - \mathbf{t}_i) \tag{3.1}$$

$$\mathbf{M}_{\text{plane}} = \underset{\mathbf{M}}{\operatorname{argmin}} \sum_i \left[ (\mathbf{M}\mathbf{s}_i - \mathbf{t}_i)^T \mathbf{n}_i \right]^2 \tag{3.2}$$

The correspondences can then be updated based on the **M** found, and then the transformation is recalculated based on the new correspondences. There are several closed-form solutions to both of these equations which will be discussed more in depth later (Chapter 5.3), but the method from which the correspondences are selected is a significant factor in how quickly the algorithms converge. In the simplest case, the nearest point can be selected, alternatively, if normals are known, the closest point projected along the normal can be selected. Because of the locality requirement for accurate correspondence estimates, a rough initial alignment is still required for consistent results.

As the name suggests Point to Point (Eq. 3.1) is a general method that merely aligns points - it tries to minimise the distance between points thought to be the same feature in the two different clouds. Conversely, Point to Plane (Eq. 3.2) makes use of the target point cloud's normals - this is less sensitive to the correspondence, as it will only align a point to make sure that it is within the same plane as the corresponding point, meaning the alignment of planes will converge more quickly as the correspondences do not have to be exact.

The general process can be summarised in Algorithm 1. There are many variations that can be used to enhance these initial ICP algorithms, often in independent building blocks that can be combined for a different scenario:

- *Matching* - determining similar points in each cloud.
- *Weighting/ Rejecting* - assessing the quality of the matches and weighting them appropriately (or reject them all together) for the optimisation step.
- *Error Metric* - two cost functions are presented above, in the following section variations on these are presented to make use of additional colour information.
- *Optimising* - the cost functions are non-linear optimisations and being able to solve these efficiently has a significant effect on runtime. Similarly, these functions are non-convex and so effort needs to be made to ensure a solution is the global minimum. The chances of reaching the global minimum are often improved if the initial alignment is good.

## 3.2 Colour-based ICP

The RealSense camera provides colour information for the points in the clouds, and there have been several enhancements to the standard ICP algorithms to try and make use of this; from simple extensions to 6-dimensional space [15] (3 spatial coordinates and 3 colour), to refining the correspondence finding process [20] [21].

As mentioned, the most straightforward option is extending the spatial coordinates with an extra 3 components for the colour space of the point. This method can be

**Algorithm 1:** Iterative Closest Point

**Data:** Target cloud $\mathcal{T} = \{\mathbf{t}_i\}_{i=1}^{N}$, Source cloud $\mathcal{S} = \{\mathbf{s}_i\}_{i=1}^{L}$
**Result:** $\mathbf{M}$
$C(\mathbf{M}, \mathcal{S}, \mathcal{T})$ := Objective function ;
**begin**
  $\mathbf{M} \longleftarrow \mathbf{I}$;
  **while** *not converged* **do**
    $\mathcal{S}^T \longleftarrow \{\mathbf{M}\mathbf{s}_i\}_{i=1}^{L}$ ;
    $\mathcal{S}^C, \mathcal{T}^C \longleftarrow$ correspondences of $\mathcal{S}^T, \mathcal{T}$ ;
    $\mathbf{M}_{\text{new}} \longleftarrow \underset{\mathbf{M}}{\operatorname{argmin}}\, C(\mathbf{M}, \mathcal{S}^C, \mathcal{T}^C)$;
    $\mathbf{M} \longleftarrow \mathbf{M}_{\text{new}}\mathbf{M}$;
  **end**
**end**

substituted into any of the ICP methods with the shape of $\mathbf{M}$ being changed as appropriate and constrained still to only manipulate the spatial coordinates. Godin, Rioux, and Baribeau [15] propose an algorithm using this. The main drawback of this method is reconciling the conflict between spatial and colour within a cost measure, and that colour spaces have a limit i.e. RGB has 3 values between 0 and 1, whereas a point in space can be at infinity.

An alternative choice is using colour only in the correspondence stage of the ICP algorithm. Korn, Holzkothen, and Pauli [20] use a 6-dimensional space for finding correspondences based on Euclidean distance between the 6-vectors, with a scaling term on the colour proportion. Their implementation makes use of a generalized plane to plane [34] ICP which involves a statistical model for correspondences. However, this still has the drawback of dimensional conflicts as the colour proportion will need to be hand-crafted for each dataset.

Łepicka, Kornuta, and Stefańczyk resolve the dimensional conflict by, rather than finding the nearest neighbour directly as in the original ICP algorithms, splitting the correspondence process into two parts. First, find the $k$-nearest neighbours in the corresponding cloud, and then pick the corresponding point from this set based on the point with the closest colour match. From here optimisation continues as normal. It will match similar coloured points that are already close together, giving more conservative registrations that will require a better initial approximation, but will be more robust to things such as repetitive patterns of colour.

The choice of algorithm used in the results presented is a more complicated algorithm by Park, Zhou, and Koltun [25]. Their implementation has two separate cost functions - one for colour and space respectively. Keeping correspondences in the geometric space, the colour information is optimised by taking a virtual camera normal to the point of interest and approximating a colour gradient, this colour gradient in the target cloud can then be used to more accurately match the colour of the corresponding point in the source cloud.

$\mathcal{X} = \{\mathbf{x}\}_{n=0}^{N}$ is a coloured point cloud with a discrete function $c(\mathbf{x})$ that retrieves the colour of that point. Park, Zhou, and Koltun introduce an orthogonal camera at each point observed along its normal $\mathbf{n}_\mathbf{x}$. $c(\mathbf{x})$ can be parametrised to a general point $\mathbf{u}$ in the

plane of $\mathbf{x}$ (i.e. ${\mathbf{n_x}}^T\mathbf{u} = 0$), $C_\mathbf{x}(\mathbf{u})$ using a first order approximation:

$$C_\mathbf{x}(\mathbf{u}) \approx c(\mathbf{x}) + {\mathbf{d_x}}^T\mathbf{u} \tag{3.3}$$

Where $\mathbf{d_x}$ is the gradient of $C_\mathbf{x}(\mathbf{u})$. This gradient is approximated by projecting points local to $\mathbf{x}$ onto the orthogonal image plane and fitting $C_\mathbf{x}(\mathbf{u})$ to this set of points. This then allows the colour and geometric objective functions to be defined, and then combined into a weighted sum for optimisation.

$$\mathbf{s'_i} = \mathbf{M}\mathbf{s}_i - [(\mathbf{M}\mathbf{s}_i - t_i)^T\mathbf{n}_i]\mathbf{n}_i \tag{3.4}$$

$$E_c(\mathbf{M}) = \sum_i \left[C_\mathbf{t}(\mathbf{s'_i}) - c(\mathbf{s_i})\right]^2 \tag{3.5}$$

$$E_g(\mathbf{M}) = \sum_i \left[(\mathbf{M}\mathbf{s}_i - \mathbf{t}_i)^T\,\mathbf{n}_i\right]^2 \tag{3.6}$$

$$\mathbf{M}_{\text{opt}} = \underset{\mathbf{M}}{\operatorname{argmin}}\left[\sigma E_g(\mathbf{M}) + (1-\sigma)E_c(\mathbf{M})\right] \tag{3.7}$$

Where $\mathbf{t}_i, \mathbf{s}_i, \mathbf{n}_i$ are defined the same as at the start of this chapter and $\sigma \in [0, 1]$.

## 3.3 Initial Alignment

As mentioned, the performance of ICP is greatly dependent on the initial alignment of the two clouds. Most commonly to get this rough initial alignment, global registration methods such as the *Fast Point Feature Histogram* (FPFH) [31] are used to find parts of each cloud which are geometrically similar and then performing a RANSAC fit to find the best alignment. Since the FPFH calculates a 33-dimensional vector that describes the local geometry of that point, it can be very computationally intensive and requires the clouds to be down-sampled. Due to the small size of miniatures with respect to the field of view of the RealSense, the level of down-sampling required means the algorithm would not be feasible for this dataset.

Instead, a more bespoke and application-specific approach would be required to get a rough general alignment. A typical wargames board will often have a number of simple geometric features on it - the surface must be largely flat for miniatures, buildings often have the shape of cuboid sections and the tabletop itself will typically be a rectangle. 3 degrees of freedom can very quickly be removed by aligning the planes of the table (2 rotations and a translation) which can be found using a RANSAC least-squares fit.

The general result of finding the rotation $\mathbf{R}$ that aligns the 3D unit vector $\mathbf{a}$ with unit vector $\mathbf{b}$ ($\mathbf{a} \neq -\mathbf{b}$) is also used to align buildings later on. First we define the axis of rotation as the vector perpendicular to $\mathbf{a}$ & $\mathbf{b}$

$$\mathbf{v} = \mathbf{a} \times \mathbf{b} \tag{3.8}$$

$$\sin\theta = ||\mathbf{v}|| \tag{3.9}$$

$$\cos\theta = \mathbf{a}^T\mathbf{b} \tag{3.10}$$

$$\mathbf{V} = \begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \tag{3.11}$$

Noting that $\mathbf{Vx} = \mathbf{v} \times \mathbf{x}$. We want to find the transformation such that $R(\mathbf{a}) = \mathbf{b}$. $\mathbf{a}$ can be split up into its components parallel and perpendicular to $\hat{\mathbf{v}} = \mathbf{v}/||\mathbf{v}||$.

$$\mathbf{a} = \mathbf{a}_\perp + \mathbf{a}_\parallel \tag{3.12}$$
$$\mathbf{a}_\parallel = (\mathbf{a}^T\hat{\mathbf{v}})\hat{\mathbf{v}} \quad (= 0) \tag{3.13}$$
$$\mathbf{a}_\perp = \mathbf{a} - \mathbf{a}_\parallel \tag{3.14}$$
$$= -\hat{\mathbf{v}} \times (\hat{\mathbf{v}} \times \mathbf{a}) = -\hat{\mathbf{V}}^2\mathbf{a} \tag{3.15}$$

Where the final line comes from the identity:

$$\mathbf{c} \times (\mathbf{b} \times \mathbf{a}) = (\mathbf{a}^T\mathbf{c})\mathbf{b} - (\mathbf{b}^T\mathbf{c})\mathbf{a} \tag{3.16}$$

Only the perpendicular part of $\mathbf{a}$ will be affected by the rotation, so if we define an orthogonal basis $\mathbf{a}_\perp$ and $\hat{\mathbf{V}}\mathbf{a}_\perp = \hat{\mathbf{V}}\mathbf{a}$ then:

$$\begin{aligned} R(\mathbf{a}) &= R(\mathbf{a}_\parallel) + R(\mathbf{a}_\perp) = \mathbf{a}_\parallel + R(\mathbf{a}_\perp) \\ &= \mathbf{a}_\parallel + \mathbf{a}_\perp \cos\theta + \hat{\mathbf{V}}\mathbf{a}\sin\theta \\ &= \mathbf{a} + \hat{\mathbf{V}}^2\mathbf{a}(1 - \cos\theta) + \hat{\mathbf{V}}\mathbf{a}\sin\theta \\ &= \left(\mathbf{I} + \frac{1 - \cos\theta}{\sin^2\theta}\mathbf{V}^2 + \mathbf{V}\right)\mathbf{a} \end{aligned} \tag{3.17}$$

Therefore the rotation matrix that maps $\mathbf{a}$ on $\mathbf{b}$ is given by:

$$\mathbf{R} = \mathbf{I} + \mathbf{V} + \frac{1}{1 + \cos\theta}\mathbf{V}^2 \tag{3.18}$$

Which is a special case of the Rodrigues formula [23] where the axis and angles are defined by the vectors $\mathbf{a}$ and $\mathbf{b}$. For ease, all the table planes are aligned with the $xz$ plane. With unit normal $\mathbf{n}$ for the plane at a distance $d$ from the origin, the rotation matrix $\mathbf{R}$ can be calculated from Eq. 3.18 by aligning $\mathbf{n}$ with $[0, 1, 0]^T$. The entire cloud can then be translated by $-d$ in the $y$ direction to align it with the origin.

To align the remaining 3 degrees of freedom an approach based on the principle of FPFH could be adopted - since each model could be considered a feature on the surface of the plane, the centres of mass of each of those clusters can be sampled and treated as a cloud of at most 5 points. With 2 of these low size point clouds (as in Figure 3.2) the correspondence policy can simply trial every possible combination of correspondences in a RANSAC fashion.

However, with only 5 models on the board, there can only be *at most* 5 points which can be tested, but a minimum of 3 is required to get an alignment. There may be miniatures inside the building (which requires a classification step before segmenting a second time) or those not seen in certain views as shown previously in Figure 3.1. So, under the conditions of the current problem, it is likely that there may not be 3 distinct models in a given view.

Alternatively, since the building will be the largest item in each scene, and hence can easily be identified by bounding boxes of segments pre-classification (see Chapter 4), geometric features within the building can be matched such as the planar walls and floors.

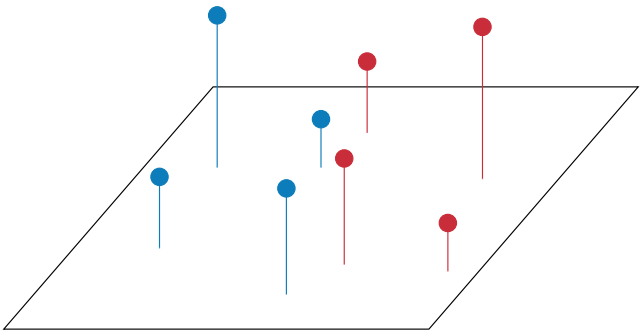


Figure 3.2: Two point clouds reduced to just the centres of mass of each object on a plane's surface. These reduced clouds are much easier to align even without an initial alignment due to the low number of points.

### 3.3.1 Normals Histogram

Under the assumption that the majority of the points on the building are within the walls, most of the normals of points in the cloud should be close to one of the normals of the walls. As a much more simplistic FPFH, if these normals are put into a histogram there should be a number of distinct maxima corresponding to the normals of the walls and floors. Figure 3.3 shows these histograms for several different views. For the full CAD model there are the 3 basis vectors, and the other views show at least 2 maxima which are sufficient to resolve the final rotational degree of freedom. However, as can also be seen the quality of these results vary. This discrepancy is down to which views contain more points on the walls such as those viewed from the outside, or those which contain more of the floor when viewed from the inside.

Because of the lack of clarity as to which vectors these maxima represent, alignments would then need to be trialled in combination to ensure the right vectors are put in correspondence. Moreover, the final linear translation would still need to be resolved perhaps as an alignment of the centres of mass of the clusters, but again, the proximity to the true centre of mass of the building will vary based on the view.

### 3.3.2 Corner Matching

As an alternative approach, the corner of the building can be located as well as the normals for each plane of the walls giving a guaranteed 3 equations to complete the global alignment. Some views may not contain full views of the walls of the building, but do contain points along them such as the edge of the floors - by using only the $xz$ data of the building points and taking the 2D convex hull we get a reduced set of points that form two lines along the walls. Finding the corner can be approached as a combination of a clustering and regression problem.

**Unconstrained case** Consider a collection of 2D points, each point as a row of matrix $\mathcal{X}_1 \in \mathbb{R}^{N_1 \times 2}$ or $\mathcal{X}_2 \in \mathbb{R}^{N_2 \times 2}$ belonging to line $l_1 : \mathbf{x}^t\mathbf{n} = 1$ and $l_2 : \mathbf{x}^t\mathbf{R}\mathbf{n}/\alpha = 1$ respectively, where $\mathbf{R}$ is a 90 degrees rotation matrix in the plane of the 2D points. This means that the distance of the line from the origin is given by one over the magnitude of $\mathbf{n}$ and $\alpha$ is the ratio of distances of the lines from the origin. We want to minimise the geometric

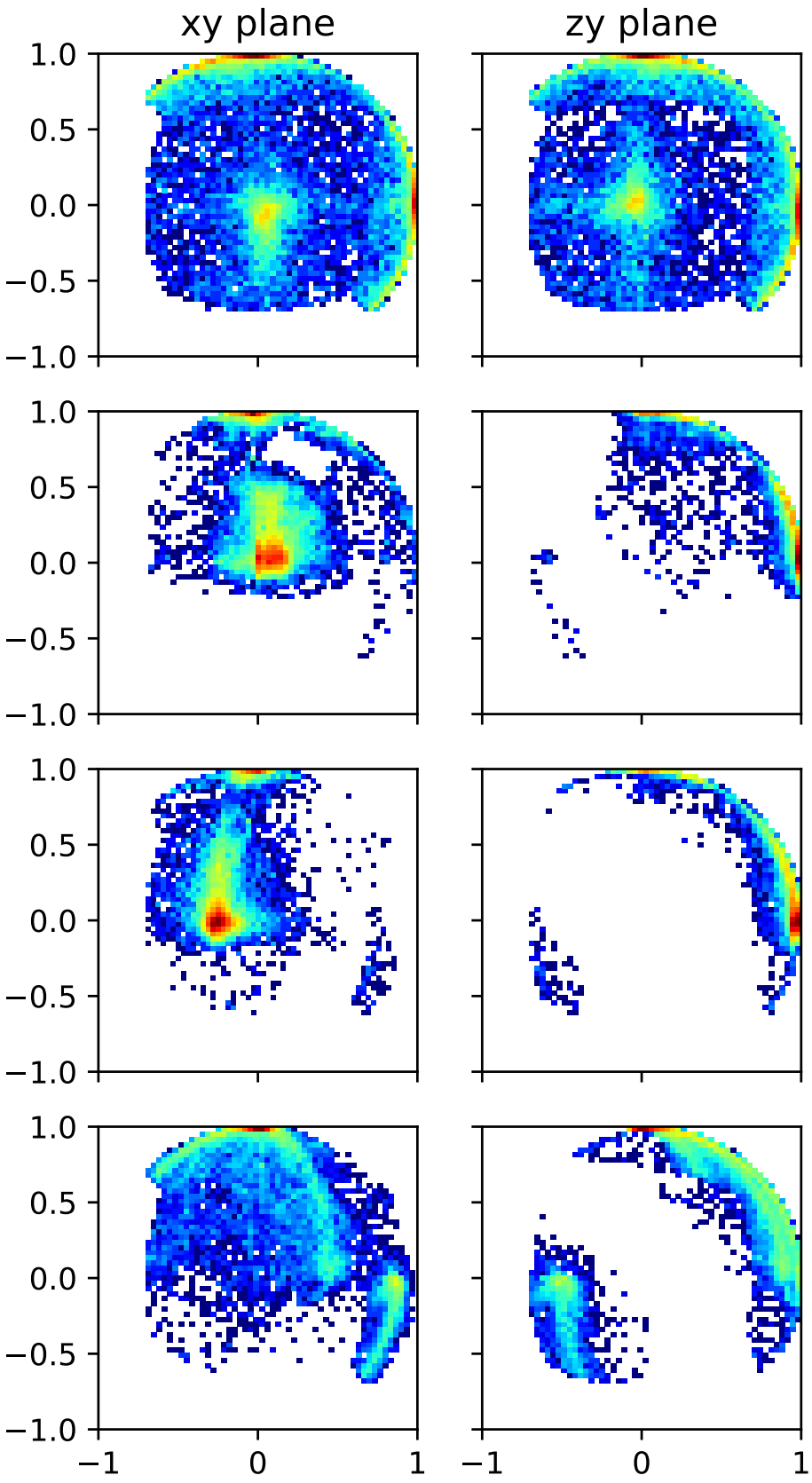



Figure 3.3: Histogram of unit normals from different building views projected onto a sphere. The top row is from taking the normals of the CAD model used to design the building, each other view is a different perspective captured with the RealSense. Colour scale is logarithmic, and directions are normalised by ensuring the largest component is always positive.

squared-error cost function:

$$C = ||\mathcal{X}_1\mathbf{n} - \mathbf{1}||_2^2 + ||\mathcal{X}_2\mathbf{R}\mathbf{n}/\alpha - \mathbf{1}||_2^2 \tag{3.19}$$

$$\mathbf{n}^*, \alpha^* = \underset{\mathbf{n},\alpha}{\operatorname{argmin}}(C) \tag{3.20}$$

Where $\mathbf{1}$ is a vector of ones with appropriate length. By differentiating with respect to each of the unknown parameters the solution is:

$$\mathbf{n}^* = (\mathcal{X}_1^T\mathcal{X}_1)^{-1}\mathcal{X}_1^T\mathbf{1} \tag{3.21}$$

$$\alpha^* = ||\mathcal{X}_2\mathbf{R}\mathbf{n}||_1/N_2 \tag{3.22}$$

In other words, $\mathbf{n}$ is found by a simple least squares of the points belonging to one line, and then $\alpha$ is formed by averaging the projection of the points along the other. This formulation inherently calculates the distance of the lines from the origin in the magnitude of $\mathbf{n}$ and hence saves the use of Lagrange multipliers to constrain $\mathbf{n}$ to be a unit vector or using more complex quadratic programmes. Nonetheless, it does mean that there may be numerical errors if the line comes close to passing through the origin.

Since the fit resembles a least squares estimator, an optimal solution can easily be adapted using a RANSAC based estimator [14]. Inspired[1] by the Scikit-Learn implementation [27] outliers are classified by thresholding points to be within the *Median Absolute Deviation* (MAD) of the data as this allows the algorithm to adapt its threshold as the statistics of the data changes as line classifications converge.

$$\text{MAD} = \text{median}(|x - \text{median}(\mathcal{X})|) \tag{3.23}$$

Since this is robust to outliers, and a least squares model minimises gaussian noise (in which the mean and median are the same), this works as a good estimate of the variance of the inlier residuals. The final algorithm used for classifying a point to each line is shown in Algorithm 2, the results of which can be seen in Figure 3.5. Convergence is fast, taking typically 4-7 iterations on the first pass and usually 1-3 iterations on the inverted pass. A second pass rotating $\mathbf{n}$ by 90 degrees about the intersection of the two lines is done as it is clear from the derivations above that the solution is not symmetric since the second line is already constrained in its direction.

**Constrained case** As an alternative approach, if $\mathbf{n}$ is constrained to be a unit vector, such that the two lines are $l_1 : \mathbf{x}^t\mathbf{n} = d_1$ and $l_2 : \mathbf{x}^t\mathbf{R}\mathbf{n} = d_2$ (this is equivalent to saying $\alpha = d_2/d_1$), the constrained cost function can be written as:

$$C = ||\mathcal{X}_1\mathbf{n} - \mathbf{d_1}||_2^2 + ||\mathcal{X}_2\mathbf{R}\mathbf{n} - \mathbf{d_2}||_2^2 + \lambda(\mathbf{n}^T\mathbf{n} - 1) \tag{3.24}$$

$$= \sum_{\mathbf{x}\in\mathcal{X}_1}(\mathbf{n}^T\mathbf{x} - d_1)^2 + \sum_{\mathbf{x}\in\mathcal{X}_2}(\mathbf{n}^T\mathbf{R}^T\mathbf{x} - d_2)^2 + \lambda(\mathbf{n}^T\mathbf{n} - 1) \tag{3.25}$$

$$\mathbf{n}^*, d_1^*, d_2^* = \underset{\mathbf{n},d_1,d_2}{\operatorname{argmin}}(C) \tag{3.26}$$

Where $\lambda$ is a Lagrange multiplier to enforce the constraint. Inspired by Eq. 3.22 calculating a ratio of distances as the average of one of the datasets, we can redefine the points

[1]The Scikit-Learn RANSAC estimators could not be used directly as it makes use of Ordinary Least Squares rather than Geometric Least Squares, and hence its calculation of the MAD is based on the error in the $y$ direction only. In the case of a line being near vertical this leads to very few outliers.

**Algorithm 2:** Corner Fitting

**Data:** 2D Points $\mathcal{P} = \{\mathbf{p}_i\}_{i=1}^{M}$
**Result:** $\mathbf{n}$, $\alpha$
$\mathbf{R}$ := 90 degree rotation matrix ;
$\mathbf{R}_{45}$ := 45 degree rotation matrix ;
$\#\mathcal{X}$ := Number of points in $\mathcal{X}$ ;
$f_1(.)$ := 1D RANSAC function returning inliers and outliers $\mathcal{I}, \mathcal{O}$ ;
$f_2(.)$ := 2D RANSAC function returning inliers and outliers $\mathcal{I}, \mathcal{O}$ ;
**begin**
  Initialise $n$ and $\alpha$;
  $\mathcal{X}_1 \longleftarrow \mathcal{P}$;
  $\mathcal{I}_1, \mathcal{O} \longleftarrow f_2(\mathcal{X}_1)$;
  $\mathbf{n} \longleftarrow (\mathcal{I}_1^T \mathcal{I}_1)^{-1} \mathcal{I}_1^T \mathbf{1}$;
  $\mathcal{X}_2 \longleftarrow \mathcal{O}$;
  $\mathcal{I}_2, \mathcal{O} \longleftarrow f_1(\mathcal{X}_2 \mathbf{R}\mathbf{n})$;
  $\alpha \longleftarrow ||\mathcal{I}_2||_1 / \#\mathcal{I}_2$;
  **while** *Tolerance or iteration condition not met* **do**
    $\mathbf{c} \longleftarrow$ solution to $\mathbf{x}^t \mathbf{n} = \mathbf{x}^t \mathbf{R}\mathbf{n}$;
    $\varepsilon \longleftarrow \mathbf{c}^T \mathbf{R}_{45} \mathbf{n}$;
    $\mathcal{A}_1 \longleftarrow \{\mathbf{p} \in \mathcal{P} \mid \mathbf{p}^T \mathbf{R}_{45} \mathbf{n} > \varepsilon\}$;
    $\mathcal{A}_2 \longleftarrow \{\mathbf{p} \in \mathcal{P} \mid \mathbf{p}^T \mathbf{R}_{45} \mathbf{n} < \varepsilon\}$;
    $\varepsilon \longleftarrow \mathbf{c}^T \mathbf{R}_{45}^T \mathbf{n}$;
    $\mathcal{B}_1 \longleftarrow \{\mathbf{p} \in \mathcal{P} \mid \mathbf{p}^T \mathbf{R}_{45}^T \mathbf{n} > \varepsilon\}$;
    $\mathcal{B}_2 \longleftarrow \{\mathbf{p} \in \mathcal{P} \mid \mathbf{p}^T \mathbf{R}_{45}^T \mathbf{n} < \varepsilon\}$;
    `/* Choose the bisection that divides the input points evenly */`
    **if** $|\#\mathcal{A}_1 - \#\mathcal{A}_2| < |\#\mathcal{B}_1 - \#\mathcal{B}_2|$ **then**
      $\mathcal{X}_1 \longleftarrow \mathcal{A}_1$;
      $\mathcal{X}_2 \longleftarrow \mathcal{A}_2$;
    **else**
      $\mathcal{X}_1 \longleftarrow \mathcal{B}_1$;
      $\mathcal{X}_2 \longleftarrow \mathcal{B}_2$;
    **end**
    $\mathcal{I}_1, \mathcal{O} \longleftarrow f_2(\mathcal{X}_1)$;
    $\mathbf{n} \longleftarrow (\mathcal{I}_1^T \mathcal{I}_1)^{-1} \mathcal{I}_1^T \mathbf{1}$;
    $\mathcal{I}_2, \mathcal{O} \longleftarrow f_1(\mathcal{X}_2 \mathbf{R}\mathbf{n})$;
    $\alpha \longleftarrow ||\mathcal{I}_2||_1 / \#\mathcal{I}_2$;
  **end**
  $C = ||\mathcal{I}_1 \mathbf{n} - \mathbf{1}||_2^2 / \#\mathcal{I}_1 + ||\mathcal{I}_2 \mathbf{R}\mathbf{n}/\alpha - \mathbf{1}||_2^2 / \#\mathcal{I}_2$;
  $\alpha', \mathbf{n}', C' \longleftarrow$ output of the while loop initialising with
   $\alpha' = ||\mathbf{n}||_2 / \alpha, \mathbf{n}'' = \mathbf{R}\mathbf{n}/\alpha$ ;
  **if** $C' < C$ **then**
    $\mathbf{n} \longleftarrow \mathbf{n}'$ ;
    $\alpha \longleftarrow \alpha'$ ;
  **end**
**end**

with the means $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}$ subtracted from the points in the appropriate matrix to create the 0 mean points $\hat{\mathbf{x}}$ as a row in matrix $\hat{\mathcal{X}}_i$:

$$\boldsymbol{\mu_i} = \frac{1}{N_i} \sum_{\mathbf{x} \in \mathcal{X}_i} \mathbf{x} \tag{3.27}$$

$$\hat{\mathcal{X}}_i = \mathcal{X}_i - \mathbf{1}\boldsymbol{\mu}_i^T \tag{3.28}$$

$$d_1' = d_1 - \mathbf{n}^T \boldsymbol{\mu_1}, \;\; d_2' = d_2 - \mathbf{n}^T \mathbf{R}^T \boldsymbol{\mu_2} \tag{3.29}$$

Substituting this back into $C$:

$$\begin{aligned} C &= \sum_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}_1} (\mathbf{n}^T \hat{\mathbf{x}} - d_1')^2 + \sum_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}_2} (\mathbf{n}^T \mathbf{R}^T \hat{\mathbf{x}} - d_2')^2 + \lambda(\mathbf{n}^T \mathbf{n} - 1) \\ &= \mathbf{n}^T \left( \hat{\mathcal{X}}_1^T \hat{\mathcal{X}}_1 + \mathbf{R}^T \hat{\mathcal{X}}_2^T \hat{\mathcal{X}}_2 \mathbf{R} + \lambda \mathbf{I} \right) \mathbf{n} - 2\mathbf{n}^T \left( d_1' \sum_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}_1} \hat{\mathbf{x}} + d_2' \sum_{\hat{\mathbf{x}} \in \hat{\mathcal{X}}_2} \hat{\mathbf{x}} \right) + N_1 d_1'^2 + N_2 d_2'^2 \\ &= \mathbf{n}^T \left( \hat{\mathcal{X}}_1^T \hat{\mathcal{X}}_1 + \mathbf{R}^T \hat{\mathcal{X}}_2^T \hat{\mathcal{X}}_2 \mathbf{R} + \lambda \mathbf{I} \right) \mathbf{n} + N_1 d_1'^2 + N_2 d_2'^2 \end{aligned} \tag{3.30}$$

Where the final line comes from the fact that the data in each set now has 0 mean. With the sum of 3 expressions, each in only one of the unknown parameters it is clear that:

$$d_1 = \frac{1}{N_1} \mathbf{n}^T \sum_{\mathbf{x} \in \mathcal{X}_1} \mathbf{x} \tag{3.31}$$

$$d_2 = \frac{1}{N_2} \mathbf{n}^T \mathbf{R}^T \sum_{\mathbf{x} \in \mathcal{X}_2} \mathbf{x} \tag{3.32}$$

$$\mathbf{n} = \text{E-vector w/ min. e-value of } \hat{\mathcal{X}}_1^T \hat{\mathcal{X}}_1 + \mathbf{R}^T \hat{\mathcal{X}}_2^T \hat{\mathcal{X}}_2 \mathbf{R} \tag{3.33}$$

This formulation has the benefit of being more numerically stable as $d_1$ and $d_2$ can be 0, it also makes use of both sets of points for determining the direction of $\mathbf{n}$ by finding the direction of minimum variance in the data, providing a symmetric result. Nonetheless, now being in 3 parameters rather than 2 it will require an additional RANSAC step to solve which may result in longer runtime per iteration. Moreover, there is a chicken and egg scenario when adapting this for a RANSAC implementation - the means of each set need to be subtracted before $\mathbf{n}$ can be calculated, but to calculate the mean one needs to know which points are inliers based on its dot product with $\mathbf{n}$. There are two possible ways to alleviate this:

- Subtract the median of the data in each set since the least squares model assumes a gaussian noise distribution, the median can be used which is more robust to outliers and is equal to the mean in gaussian noise.
- Within the RANSAC algorithm ensure that at least 2 points from each set are selected, allowing the mean for each set to be estimated and then subtracted before the $\mathbf{n}$ for those samples are calculated.

It should also be noted that while the median (rather than the mean) of $\mathbf{n}^T \mathbf{x}$ could be used to estimate $d_i$, reducing the number of RANSAC runs but still being robust to outliers, the precision of results were much better when a random sample of inliers

were used to calculate the distances. Due to the low amount of noise in the data (bar outliers on the convex hull), and hence the fast convergence of all the algorithms, it is currently difficult to compare the rates of convergence and speed of the unconstrained and constrained algorithms. All variants described above provide visually similar results in a similar number of iterations when a minimum of 50 data points are used.

The results in Figure 3.5 show that the algorithm is able to find the corner consistently in many different orientations and occlusion environments, and hence using a similar method to the one used to align the table planes, an initial alignment of the full scene can be realised.

## 3.4 Registration Results

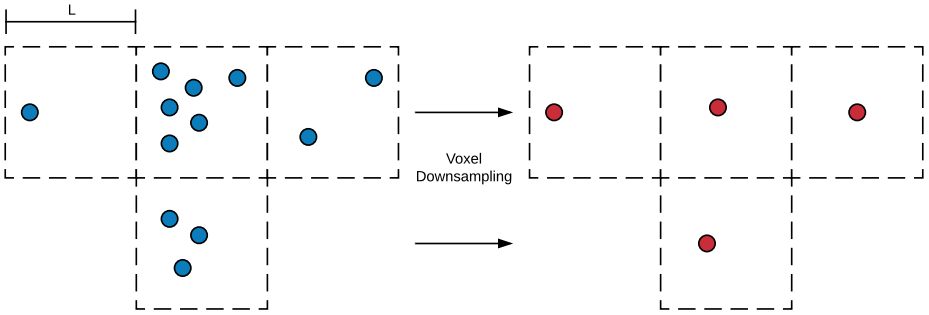



Figure 3.4: 2D example of the Voxel downsampling process. The space is split up into an even grid of length $L$ and the centroid of points in each cell is taken.

For each view in each of the scenes, the scene is segmented (Chapter 4) to identify the building and an initial alignment is found using the Corner Fitting algorithms detailed previously. One of the views is then taken as the anchor. For each other view a 'point cloud pyramid' is constructed, that is, a set of downsampled clouds are created from a given view based on a varying length scale $l$. The voxel downsampling is carried about by splitting the space into cubes of vertex length $l$ and then replacing all points in each cube with the centroid of those points. This allows the number of points to be dramatically reduced, and make the density of points across a surface more homogenous as demonstrated in Figure 3.4.

Since the complexity of ICP is $\mathcal{O}(n^2)$, the views are registered to the anchor by taking the initial alignment, and then updating it using the most coarse downsampling in its pyramid, and repeating the process with the next-most downsampled. This allows for most of the iterations to be carried out on the sparser clouds which can be done much quicker and only fine-tuning on the more computationally expensive dense clouds.

As can be seen in Figure 3.6 the initial global alignment provides a good match for the scene, which the Colour-based ICP is then able to tighten up to get a single point cloud with reduced occlusion in Figure 3.7. The quality of results does vary based on which view within the dataset is used as the anchor, since there is minimal information in the first pair of clouds, without sufficient overlap the ICP may take the clouds out of alignment. Nonetheless, since typically 4-6 views are needed to get a dense enough scene the process can be carried out with several different clouds starting as the anchor and selecting the best overall fit based on cost.

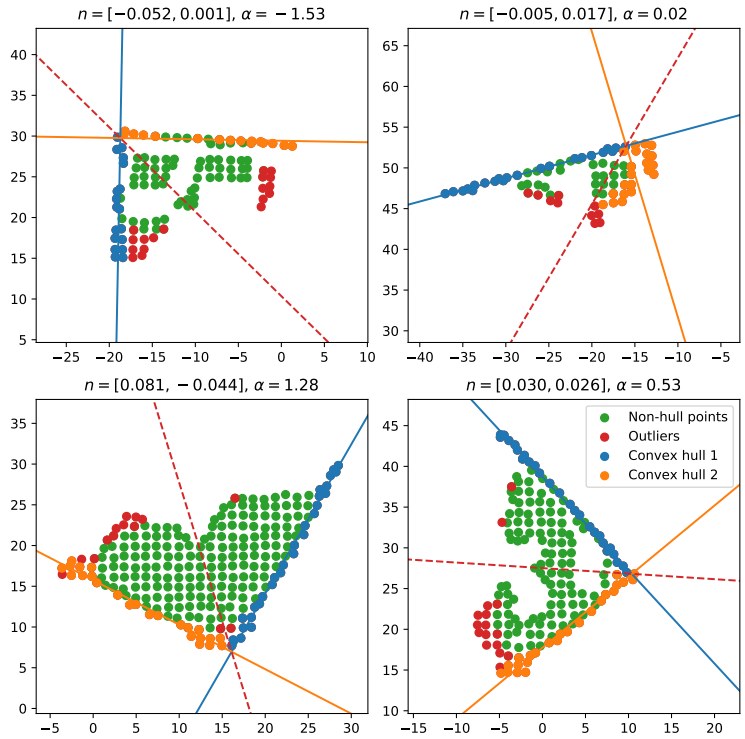
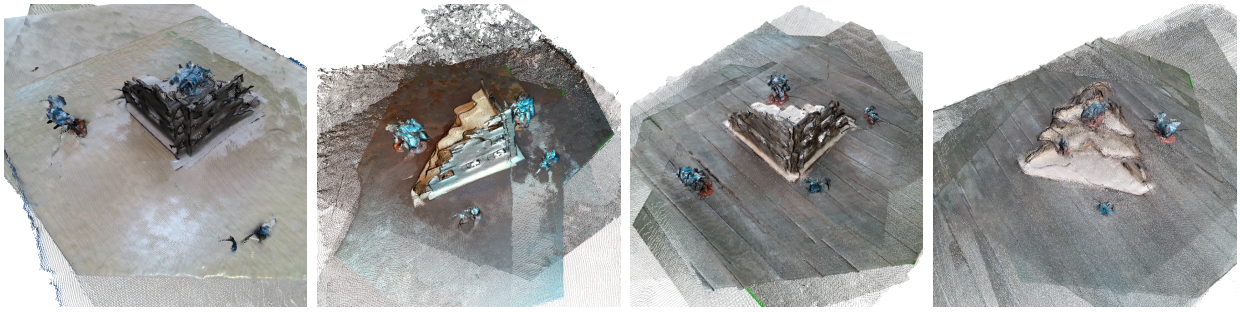
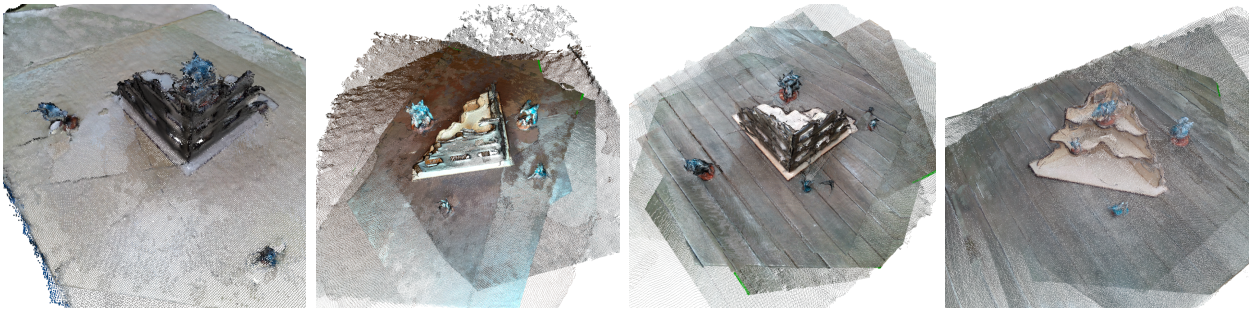



Figure 3.5: Building clusters of 4 different views with lines fitted to each wall and the classification boundary. Buildings are scaled such that distances are in centimetres. Convex hull sets are generated by taking repeated hulls of the data without the current hull until there are at least 50 points in the set.



(a) Layout 1 (b) Layout 2 (c) Layout 3 (d) Layout 4

Figure 3.6: Initial alignment of mutliple views using corner fitting



(a) Layout 1 (b) Layout 2 (c) Layout 3 (d) Layout 4

Figure 3.7: Final alignment of multiple views after ICP

# Chapter 4

# Segmentation

Segmentation is carried out at two stages of the reconstruction pipeline: to identify the building pre-registration (based on bounding box volume and height) and to generate *Regions of Interest* (ROIs) once a scene has been registered for classification. For this application there are some important factors that any implementation needs to meet:

- A non-parametric model that does not require the number of clusters to be specified *a priori* as opposed to $k$-means where the number of clusters is fixed at run-time. Since the number of models on a board can change throughout a game, it is impractical to have to specify the clusters between each turn of a game, particularly if a game is scaled up.
- Robust to outliers. Both RealSense depth images and photogrammetry (see Chapter 5) can produce several floating artefacts within a scene that are not associated with a real surface, so the algorithm shouldn't be biased by these points.
- Can find an arbitrarily shaped cluster as models can have many different geometric structures and sizes.

Two algorithms were explored for segmentation: Region growing and *Density-based spatial clustering of applications with noise* (DBSCAN) which are discussed further below.

## 4.1 Region Growing

Region growing is a very simple algorithm which works by picking a random starting seed and examining the scalar product of its normal with the normals of the $k$-nearest neighbours. If the result is above a threshold it is classified as the same region and the process repeats on this new point until there are no new additions to a given region. When there are no new points added, the loop is restarted with a new seed from points that are yet to be classified until every point has been categorised into a ROI.

Since the only parameter for the algorithm is the scalar product tolerance, Region Growing satisfies all the conditions of a practical implementation defined above. Algorithm 3 was adapted from [5] for use in Python. As can be seen in Figure 4.2 the algorithm can effectively separate different parts of the scene. However, smaller models which have a lot of small geometric variation are split into several sections and so will require an additional clustering step to combine these many small regions into a single ROI. This splitting of the building observed in the results may be a useful feature when trying to

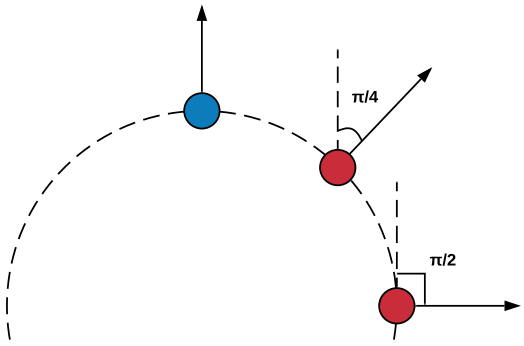
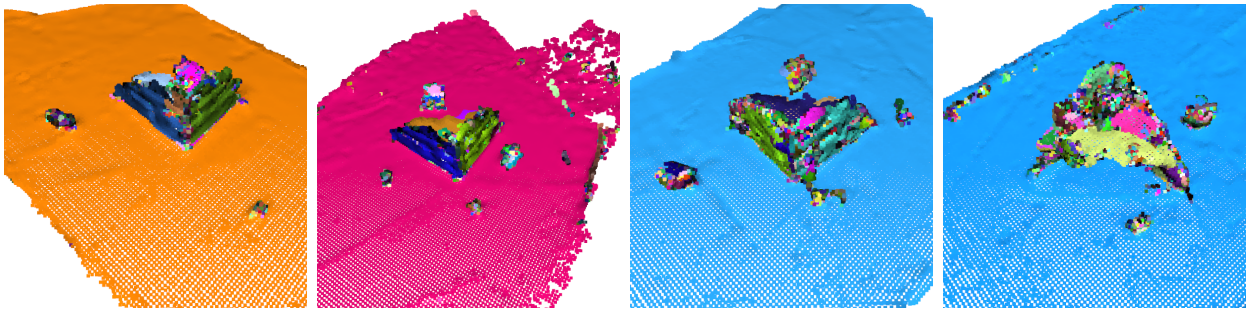



Figure 4.1: A seed point (blue) and 2 candidate points (red) in the region on the surface of a circle. Despite the curvature being constant around a circle, without taking distance into account the absolute angle between the seed points will vary.



(a) Layout 1 (b) Layout 2 (c) Layout 3 (d) Layout 4

Figure 4.2: Region Growing segmentation applied to each layout.

extract miniatures that are inside the building, as each of the floors and walls has been separated into large chunks which can easily be extracted leaving only the geometrically complex miniatures inside.

All the normals are unit vectors, and the scalar product gives $\cos\theta$ between them; the algorithm is effectively looking at the curvature of surfaces in a non-linear way, and splitting areas accordingly. The benefit of only looking at the angles between the vectors is that it is preserved under scaling, and hence it is robust to deal with clouds from several sources with different scaling policies. However, there is some drawback to not considering the spacing between points; consider Figure 4.1, by looking at the scalar product only, the rightmost point will give a value of 0 and not be included in the region. If $\cos^{-1}(\mathbf{n_1}^T\mathbf{n_2})/d$ is used instead, the true curvature value would be comparable for each of the candidate points. Use of this linear threshold was trialled initially, but results created additional ROIs that were outlines of larger sections because of variations in point normals based on their approximation.

## 4.2 DBSCAN

As the name suggests, the DBSCAN algorithm [13] excels at clustering regions that are densely packed relative to the rest of the cloud. It determines 'core' points based on how many neighbours a point has in a certain region, and then connecting core points into a cluster if they are reachable based on certain conditions as in Figure 4.3. Similar to Region Growing it has a geometric parameter, but this is related to the relative separation

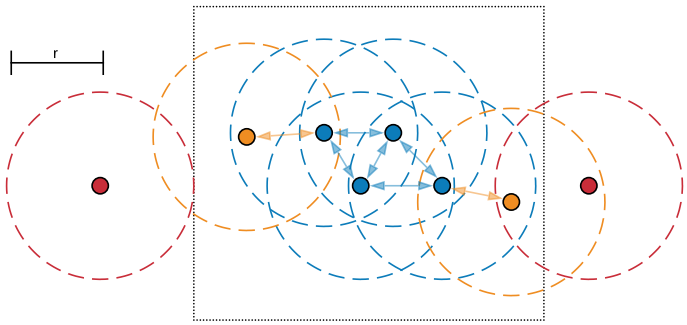
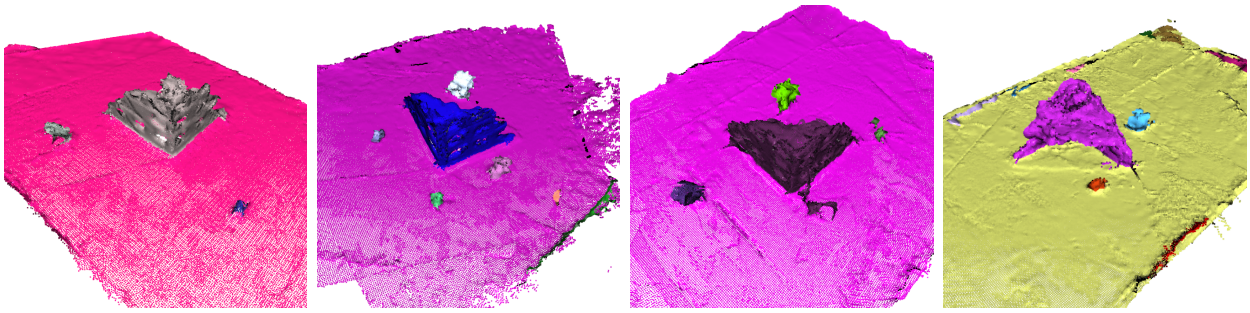



Figure 4.3: Demonstration of DBSCAN algorithm with minPts= 3 and eps= $r$. Core points (blue) are points with at least 3 points within $r$ of it, including itself. Reachable points (yellow) are within $r$ of a core point, outliers (red) are not within $r$ of any core point. All points in the box are part of the same cluster.



(a) Layout 1 (b) Layout 2 (c) Layout 3 (d) Layout 4

Figure 4.4: DBSCAN segmentation applied to each layout

of points which is not dimensionless and hence not invariant under scaling, unlike Region Growing.

Since the algorithm will connect clusters that have a 'path' between them, the table plane (as well as everything below it since they must be outliers) needs to be removed before clustering commences otherwise only a single cluster is returned due to the table acting as a path of core points between all the models. Fortunately, as detailed in Chapter 3.3, the table is very straightforward to find, and so does not have a significant effect on the performance of the algorithm.

Figure 4.4 shows that the algorithm can extract a clear ROI for each model in the scene. There are some erroneous ROIs but these come from artefacts captured by the RealSense. Since they often have a constrained height or are linear, during classification these sections can largely be filtered out. One issue that this algorithm does present is in the case of models in buildings, where the building and connected miniatures are presented as a single ROI. Efforts to split up the building can be made using Region Growing on that specific ROI as described above, or after classification by determining which points actually belong to a model.

**Algorithm 3:** Region Growing

**Data:** Point cloud with normals $\mathbf{P} = \{(\mathbf{t}, \mathbf{n})_i\}_{i=1}^{M}$, curvature tolerance $\theta$
**Result:** $\mathbf{R} = \{\mathbf{r}_i\}_{i=1}^{K}$ where each $\mathbf{r}_i$ is a region proposal
**begin**
  $\mathbf{R} \longleftarrow \emptyset$;
  **while** $\mathbf{P} \neq \emptyset$ **do**
    $\mathbf{S} \longleftarrow \{\mathbf{p}_0\}$;
    $\mathbf{P} \longleftarrow \mathbf{P} \setminus \mathbf{S}$;
    $\mathbf{r} \longleftarrow \emptyset$;
    **while** $\mathbf{S} \neq \emptyset$ **do**
      $s \longleftarrow \mathbf{S}_0$;
      $\mathbf{S} \longleftarrow \mathbf{S} \setminus s$;
      /* $\mathbf{t}_s$ is the point associated with seed $s$ */
      $\mathbf{U} \longleftarrow$ normals of K nearest neighbours of $\mathbf{t}_s$ in $\mathbf{P}$ ;
      $\mathbf{V} \longleftarrow \emptyset$;
      **foreach** $\mathbf{u} \in \mathbf{U}$ **do**
        /* $\mathbf{n}_s$ is the normal associated with seed $s$ */
        $z \longleftarrow \mathbf{u}^T \mathbf{n}_s$;
        **if** $z > \theta$ **then**
          $\mathbf{V} \longleftarrow \mathbf{V} \cup \{\mathbf{u}\}$;
        **end**
      **end**
      $\mathbf{S} \longleftarrow \mathbf{V} \setminus \mathbf{r}$;
      $\mathbf{P} \longleftarrow \mathbf{P} \setminus \mathbf{V}$;
      $\mathbf{r} \longleftarrow \mathbf{r} \cup \mathbf{V}$;
    **end**
  **end**
  $\mathbf{R} \longleftarrow \mathbf{R} \cup \mathbf{r}$;
**end**

# Chapter 5

# Classification

With several views registered to maximise the information of the scene and ROI proposed from segmenting, the process of identifying these regions and hence reconstructing areas of the scene currently captured can be carried out. Depending on the lighting conditions, a scene can have many artefacts that are not present in real life. This is particularly prominent on reflective surfaces where the segmentation algorithm will identify the structure as a ROI due to its point density. Initial filtering of ROI can be carried out with simple geometric tests:

- The height of a ROI can be compared against possible matches, if it is in none of the possible height ranges it is likely an artefact.
- Sometimes the segmentation may cluster the edge of the table as it curves slightly, or artefacts appear in a linear fashion - taking the SVD of the points in the region, a true 3D object will have 3 well defined singular values, lines and planes will have at least one 0 singular value.

After this ROI filtering, any remaining clusters can be matched to a pre-made bank of data. To collect reference data on each of the miniatures two methods were identified and discussed below. With these reference models, the ROIs can be matched using a version of ICP constrained to a translation and single axis of rotation. This also generates the required transformation to match the orientation of the ROI.

## 5.1 Reference Models

### 5.1.1 Point Clouds

Initially, reference models were created by taking a series of point cloud images of just the miniature on the table. After cropping and isolating the image from the background by finding the table plane, each model view was combined using Colour ICP. As in Chapter 3.3, an initial alignment can be found by aligning ground planes. However, unlike for whole scene registrations the small size of the clouds relative to the rest of a scene (Figure 5.1) meant there were no particularly strong alignments that the algorithm would consistently converge to. As a result, this produced clouds where one view's table plane becomes completely misaligned with the table plane of the other.

To stop this from happening the ICP can be constrained to a rotation about the y-axis by $\theta$ and a small translation $\boldsymbol{\alpha}$ given by transformation matrix $\mathbf{M}$.

$$\mathbf{M} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & \uparrow \\ 0 & 1 & 0 & \boldsymbol{\alpha} \\ \sin\theta & 0 & \cos\theta & \downarrow \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.1}$$

This constraint behaviour can be mimicked by re-creating the ground plane in each view, which allows library implementations of ICP such as those in Open3d to be used in a constrained setting. The presence of the plane in each view will then introduce a penalisation in the cost function if there is any rotation not along the $y$-axis. This is because the plane in each view will have a very high probability of being given a correspondence with a point in the plane in the other view and so will not add to the cost function provided the planes are not rotated out of alignment. This produces reliable results for the first few views, however, poor alignments occur once views with minimal overlap of the current cloud are registered. CloudCompare [7] is an open-source piece of software that allows for direct point cloud manipulation, and more importantly ICP that allows the proportion of overlap to be specified to inform the optimisation. Figure 5.2 shows the full reconstruction of 2 miniatures in CloudCompare; a lot of the large structure of the model has been recovered - limbs can be identified and certain weapons are visible. Nonetheless, smaller details such as the head are hard to distinguish. This problem is further exacerbated with smaller infantry sized miniatures, where the RealSense is simply unable to pick up sufficient detail for models to be registered.

Perhaps the biggest problem with this method, however, is that the reference models are still only point clouds themselves and so need to then be processed to become water-tight meshes for determining Line of Sight properly (Chapter 6). Most commonly meshes can be constructed from point clouds using Poisson surfaces [19] but the noise in the cloud means geometric parameters need to be made very coarse, and further detail is lost. An alternative attempt to make use of ball pivoting algorithms [3] [10] (rolling a ball of fixed radius across the points, and when 3 points touch the ball they are considered a triangle in the mesh) resulted in a more realistic looking mesh, but the directions of triangle normals were often inverted, and the whole cloud would not be fully meshed.

### 5.1.2 Photogrammetry

An alternative approach to producing the reference models is through photogrammetry. This is the process of creating a 3D model through a series of images taken on conventional cameras. Darzentas [8] [9] has used photogrammetry with wargaming miniatures previously for digital record keeping of a miniature's life cycle (performance in games, paint schemes, fictional backstory etc.) and the process provides a much greater improvement on Chapter 5.1.1.

MeshLab provides a framework that implements the AliceVision pipeline [18] [24], streamlining a significant part of the photogrammetry process. Figure 5.3 shows the near photorealistic results, far superior to previous methods creating a full mesh containing over 2 million vertices. The process is a time consuming one, with a reconstruction taking 6+ hours each, so is clearly not feasible for the wider project when mapping boards. Nonetheless, low noise models like this will make the task of matching region proposals in a scene more reliable as there is information on every possible angle the RealSense might

capture from a board. Moreover, as the process works from standard RGB images, it is physically scalable to any sized model, removing the problems that the RealSense was having with capturing sufficient detail of smaller miniatures.

## 5.2 Scaling

With point clouds being generated from several different sources (CAD models, photogrammetry and RealSense), the scaling of clouds relative to one another needs to adopt a uniform policy. Being able to scale before classification and registration are also useful to achieve consistency in any parameters i.e. geometric constants used in DBSCAN segmentation, or distance thresholds for plane finding. The simplest method for estimating the correct scale is based on heights. As mentioned several times before the building can be identified easily after segmentation and so its height acts as a good initial estimate for scaling.

Nonetheless once classification has been carried out further measurements of each model's apparent height can be taken and, depending on the quality of the cluster that has been classified, a circle fitted to the base of the miniature. A Kasa fit [35] provides a simple least squares method for estimating the parameters of the circle equation centred at $(a, b)$ with radius $r$:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{5.2}$$

With a set of points on the circle $\{(x_i, y_i)\}_{i=1}^{N}$ the least squares problem can be formulated thus:

$$\begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & 1 \end{pmatrix} \boldsymbol{\theta} = \begin{pmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ \vdots \\ x_N^2 + y_N^2 \end{pmatrix} \tag{5.3}$$

Where $\boldsymbol{\theta}$ can recover the original parameters of Eq. 5.2 from:

$$a = -\frac{\theta_1}{2} \tag{5.4}$$

$$b = -\frac{\theta_2}{2} \tag{5.5}$$

$$r = \frac{\theta_1^2 + \theta_2^2 - 4\theta_3}{4} \tag{5.6}$$

The fits on clusters from photogrammetry-based scenes are incredibly tight as can be seen in Figures 5.4 & 5.5 where the large proportion of samples are within 0.5mm of the estimated radii - far better than can be often managed by two players. Since the bases of all miniatures are standardised into a small number of sizes, taking readings like this across a scene after classification can generate many noisy scaling samples from which the true scaling can be more accurately estimated. Scaling is not estimated automatically in the ICP transformations (Chapter 5.3) since it should be applied on a whole scene uniformly rather than on a cluster by cluster basis to keep proportions and distance ratios between models the same for scene reconstruction.

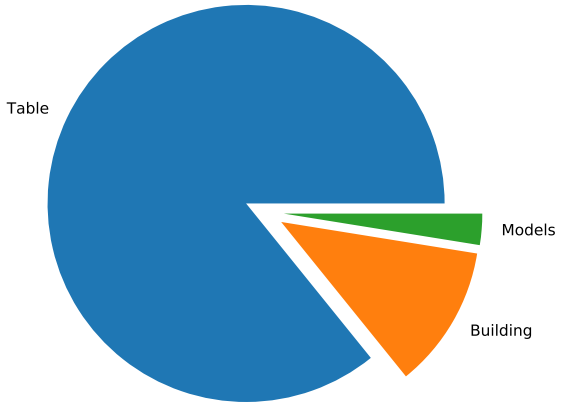
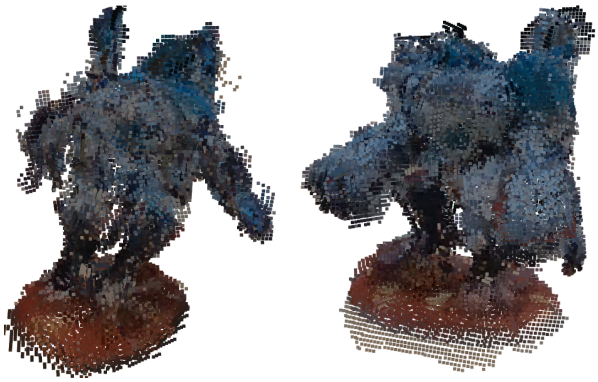



Figure 5.1: Pie chart showing the split between different types of object in a scene. 'Models' refers to all points associated with an of the 4 miniatures in the layout dataset.



(a) Commander (b) Broadside

Figure 5.2: Reference models created in CloudCompare

(a) Commander (b) Broadside (c) Fire Warrior (d) Fireblade

Figure 5.3: Reference models created with photogrammetry

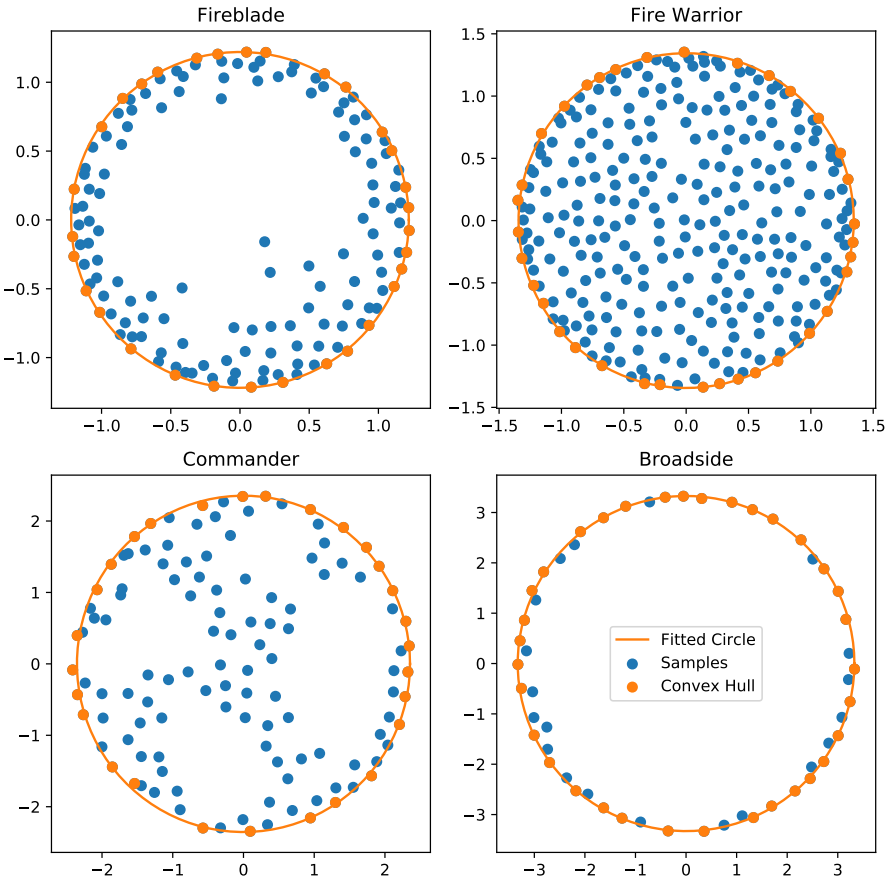



Figure 5.4: Circles fitted to the bases of clusters classified from a photogrammetry-based scene. Initial scaling is taken by matching the height of the cluster to its real world model height.

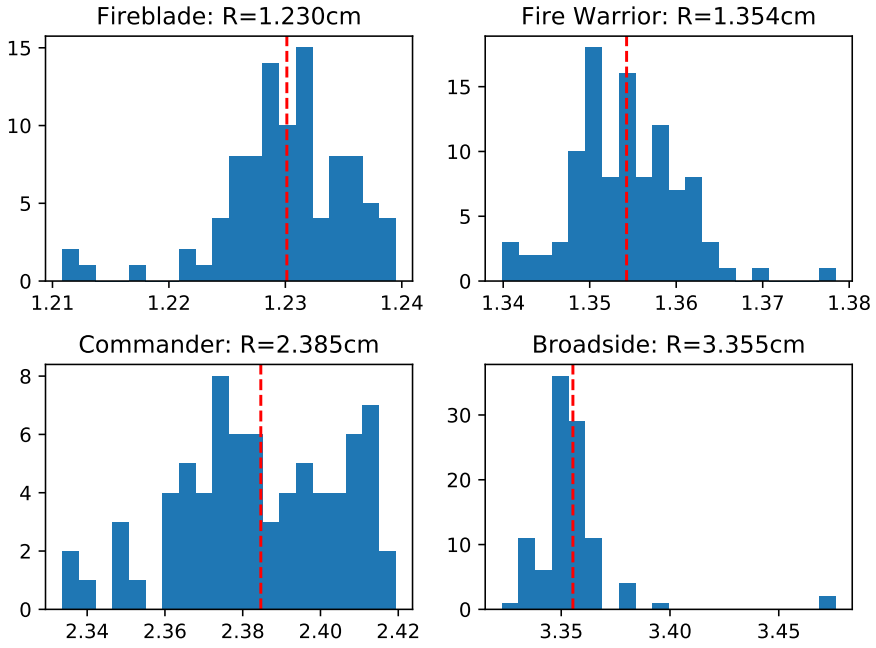



Figure 5.5: Histogram of radii samples from Circle fitting. Red line indicates the estimated $R$.

## 5.3 ICP Revisited

Finally, as mentioned at the start of this section, as well as being able to classify regions of interest, the transformation that maps the match to the ROI is required to successfully reconstruct the scene. By performing ICP registration, mapping the ROI to each of the reference models in turn and selecting the reference model with the lowest overall cost value (Eq. 3.1 & 3.2) we can simultaneously calculate the transformation, but there is still the issue of initial alignment. However, as already shown in Figure 5.1, all models in a scene typically contain less than 10% of the points of the entire scene and hence ICP on just the models should be orders of magnitude faster than ICP on a whole scene. Since the ground planes can be aligned as before, a search can be conducted to find the optimal registration across a one-dimensional search space by trialling ICP at evenly spaced initial alignments rotating about the $y$ axis.

To use the reference models in ICP they are sampled to form a point cloud across its surface where each point is an approximately equal distance from all its neighbours [38]. The output of photogrammetry provides a texture material rather than a colour for each vertex/ triangle, and so the sampled reference model has no colour information. This means colour-based methods can't be used. As a result, rather than adding back in the ground plane to constrain the optimisation (as in Chapter 5.1.1 where the colour can provide useful correspondence information) a specific constrained ICP algorithm was written for both Point to Point and Point to Plane.

**Point to Point** Taking Eq. 3.1 and breaking up $\mathbf{M}$ into a rotation $\mathbf{R}$ and translation $\boldsymbol{\alpha}$ as well as taking the mean of the error gives:

$$C = \frac{1}{N}\sum_i ||\mathbf{R}\mathbf{s}_i - \mathbf{t}_i + \boldsymbol{\alpha}||_2^2 \tag{5.7}$$

For clarity we re-define the points with their means[1] subtracted:

$$\boldsymbol{\alpha}' = \boldsymbol{\alpha} + \frac{1}{N}\mathbf{R}\sum_i \mathbf{s}_i - \frac{1}{N}\sum_i \mathbf{t}_i \tag{5.8}$$

$$\mathbf{s}_i' = \mathbf{s}_i - \frac{1}{N}\mathbf{R}\sum_j \mathbf{s}_j,\ \mathbf{t}_i' = \mathbf{t}_i - \frac{1}{N}\mathbf{R}\sum_j \mathbf{t}_j \tag{5.9}$$

$$\begin{aligned} C &= \frac{1}{N}\sum_i ||\mathbf{R}\mathbf{s}_i' - \mathbf{t}_i' + \boldsymbol{\alpha}'||_2^2 \\ &= \frac{1}{N}\sum_i ||\mathbf{R}\mathbf{s}_i' - \mathbf{t}_i'||_2^2 + \frac{2}{N}\boldsymbol{\alpha}'^T\sum_i (\mathbf{R}\mathbf{s}_i' - \mathbf{t}_i') + ||\boldsymbol{\alpha}'||_2^2 \\ &= \frac{1}{N}\sum_i ||\mathbf{R}\mathbf{s}_i' - \mathbf{t}_i'||_2^2 + ||\boldsymbol{\alpha}'||_2^2 \end{aligned} \tag{5.10}$$

Where the last line comes from the fact that the points are referred to the centroid and so have 0 mean. This gives an addition of two expressions, one in each of the unknown parameters. By inspection, $\boldsymbol{\alpha}'$ is minimised when:

$$\boldsymbol{\alpha} = \frac{1}{N}\sum_i \mathbf{t}_i - \frac{1}{N}\mathbf{R}\sum_i \mathbf{s}_i \tag{5.11}$$

[1] That is, the mean of the points in the correspondence set

The summation in Eq. 5.10 is a form of the Orthogonal Procrustes problem [33] and hence $\mathbf{R}$ is minimised by taking the singular value decomposition of the multiplication of the source and target points:

$$\mathbf{S}^T\mathbf{T} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{5.12}$$

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \tag{5.13}$$

Where each row of $\mathbf{S}$ and $\mathbf{T}$ are corresponding source and target points with their means subtracted respectively. Therefore, to constrain the point to rotate only about the $y$ axis, take the matrix multiplication of the $x$ and $z$ components of each point only, and putting the $2 \times 2$ rotation matrix $\mathbf{R}$, along with the translation together to make $\mathbf{M}$ as

$$\mathbf{M} = \begin{pmatrix} R_{11} & 0 & R_{21} & \uparrow \\ 0 & 1 & 0 & \boldsymbol{\alpha} \\ R_{12} & 0 & R_{22} & \downarrow \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.14}$$

**Point to Plane** In the case of point to plane, normally each iteration is slower than point to point as a non-linear optimisation has to occur. However, Low [22] provides a method for approximating the solution using linear least squares appropriate for angular displacements less than 30 degrees. For the constrained version consider Eq. 5.1 for small angles:

$$\mathbf{M} \approx \begin{pmatrix} 1 & 0 & -\theta & \uparrow \\ 0 & 1 & 0 & \boldsymbol{\alpha} \\ \theta & 0 & 1 & \downarrow \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.15}$$

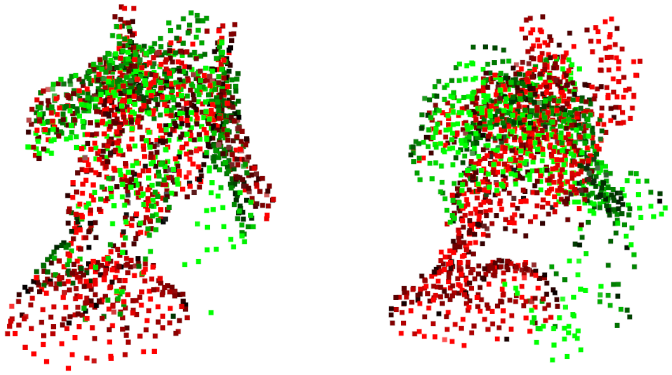
When expanded out this transforms the cost function (Eq. 3.2) to:

$$C = \sum_i \left[\boldsymbol{\alpha}^T\mathbf{n}_i + \theta(n_{xi}s_{zi} - n_z s_{xi}) + \mathbf{n_i}^T(\mathbf{s}_i - \mathbf{t}_i)\right]^2 \tag{5.16}$$

And hence the parameters can be found by solving the least squares equation:

$$\begin{pmatrix} n_{x1}s_{z1} - n_{z1}s_{x1} & \leftarrow \mathbf{n}_1 \rightarrow \\ n_{x2}s_{z2} - n_{z2}s_{x2} & \leftarrow \mathbf{n}_2 \rightarrow \\ \vdots & \vdots \\ n_{xN}s_{zN} - n_{zN}s_{xN} & \leftarrow \mathbf{n}_N \rightarrow \end{pmatrix} \begin{pmatrix} \theta \\ \uparrow \\ \boldsymbol{\alpha} \\ \downarrow \end{pmatrix} = \begin{pmatrix} \mathbf{n_1}^T(\mathbf{t}_1 - \mathbf{s}_1) \\ \mathbf{n_2}^T(\mathbf{t}_2 - \mathbf{s}_2) \\ \vdots \\ \mathbf{n_N}^T(\mathbf{t}_N - \mathbf{s}_N \end{pmatrix} \tag{5.17}$$

Currently for both algorithms correspondence is determined by finding the nearest point, calculated using a KD tree [2], and then taking the top 80% of points with the smallest Euclidean distance between their correspondence pairs. There are no checks as to the uniqueness of a correspondence or that ordering constraints are maintained. In the case of point to plane this is less of an issue provided the correspondences are in the same plane. As a result, the optimal transformation for point to point is not always the true optimal, and hence the correct classification is not consistent as can be seen in Figure 5.6. This is particularly noticeable for the larger models: the Broadside has a large chest cavity which a number of points can sit inside at an 'optimal' position regardless of the actual model the source point cloud came from, which is not an issue for point to plane.



(a) Point to Plane (b) Point to Point

Figure 5.6: ICP alignment of Commander reference model with Commander cluster segmented from Layout 2

**Comparison with non-constrained case** It could be asked, why not take the results of the original 6-DOF algorithm and manipulate the results to apply the constraints? For simplicity take the point to point case for 3D, where the result is calculated by the SVD of:

$$\mathbf{S}^T\mathbf{T} = \begin{pmatrix} \sum s_x t_x & \sum s_x t_y & \sum s_x t_z \\ \sum s_y t_x & \sum s_y t_y & \sum s_y t_z \\ \sum s_z t_x & \sum s_z t_y & \sum s_z t_z \end{pmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{5.18}$$

This can be reduced to the case where only $x$ and $z$ components are considered by pre and post multiplying by $\mathbf{P}$:

$$\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{5.19}$$

$$\hat{\mathbf{S}}^T\hat{\mathbf{T}} = \mathbf{P}^T\mathbf{S}^T\mathbf{T}\mathbf{P} = \mathbf{P}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{P} \tag{5.20}$$

Since this transforms $\mathbf{U}$ and $\mathbf{V}$ to be non-square, the SVD of the constrained case is not a simple sub-matrix of the 6-DOF case. This is to be expected, however, as rotations are non-commutative and not linearly independent. Alternatively, could the result of the full 6 DOF can be approximated as a constrained transformation? This could be done either by decomposing the transformation into its Euler angles or by minimising some cost function such as the Frobenius norm between the two matrices. Having some way of estimating the closest 3-DOF case from a 6-DOF case would allow for library implementations such as those in Chapter 3.1 to be directly used. Nonetheless, it is clear that the constrained case has a complexity saving[2] and should be used where practical. In fact, it is easy to adapt the results here to be constrained to rotate about any arbitrary axis by aligning the axis of rotation with one of the principle axes (in this case $y$) using Eq. 3.18 and extending the rotation matrix to be the $4 \times 4$ transformation $\mathbf{R}$, performing constrained ICP on the rotated points to find $\mathbf{M}$, and then rotating back, to get the overall transformation $\mathbf{M}'$ as:

$$\mathbf{M}' = \mathbf{R}\mathbf{M}\mathbf{R}^T \tag{5.21}$$

[2]Least Squares has a complexity $\mathcal{O}(C^2N)$ where $N$ is the number of data points, $C$ the number of parameters

## 5.4 Classification Results

Figure 5.7 shows the cost function at each iteration over the classification process. Since the least squares point to plane is only suitable for rotations less than 30 degrees [22], the photogrammetry reference model is initialised at 36-degree intervals around the circle. Determining convergence as when the change in the cost function is less than 0.1%, it is clear that point to plane reaches a solution much faster, taking nearly half the number of iterations. It appears in the point to point case that the cost function has several peaks and troughs. However, Figure 5.8 shows that even where the cost function is moving up and down $\theta$ is still converging towards the same value. It is also clear from Figure 5.8 that for both algorithms the angle changes by no more than a few degrees i.e. the small-angle assumption holds. It is likely that the increase in cost in point to point is due to significant changes in the correspondence allocations.

An additional observation is that the initialisation that leads to the optimal solution starts with the lowest cost function of all the initialisation points. While there is no guarantee that this will always be true, in the case of large point clouds that may take a long time to match or converge, a single iteration can be done at each initialisation and then iterations can then be allowed to continue at the initialisation that starts lowest - reducing the number of iterations by an order of magnitude.

Table 5.1 shows the results of classification with point to plane and point to point ICP against the 25 different un-registered views taken across all 4 layouts. While there is at most 25 possible items of test data for each model (even less given some models are occluded in certain views), and much more test data would be required to truly validate the model, this data shows at least 80% are matched correctly in the point to plane case. Conversely, it is clear that the matching point clouds are much too sparse for meaningful classification using point to point ICP, where the results skew to one of the large or small models respectively. Moreover, in the point to point case even when classification is correct, the orientation of the model is still wrong, implying it was more luck than the algorithm finding correct correspondences.

Table 5.1: Total confusion matrix for each view across all layouts. Only cases where the model was correctly segmented to be classified in the scene are counted.

| | | Estimated Label | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Point to Plane | | | | Point to Point | | | |
| | | A) | B) | C) | D) | A) | B) | C) | D) |
| True Label | A) Commander | 12 | 2 | | 1 | 12 | 2 | 1 | |
| | B) Broadside | 3 | 14 | | 1 | 11 | 6 | | 1 |
| | C) Fire Warrior | | | 8 | 3 | | | 8 | 3 |
| | D) Fireblade | | | 2 | 18 | | | 12 | 8 |

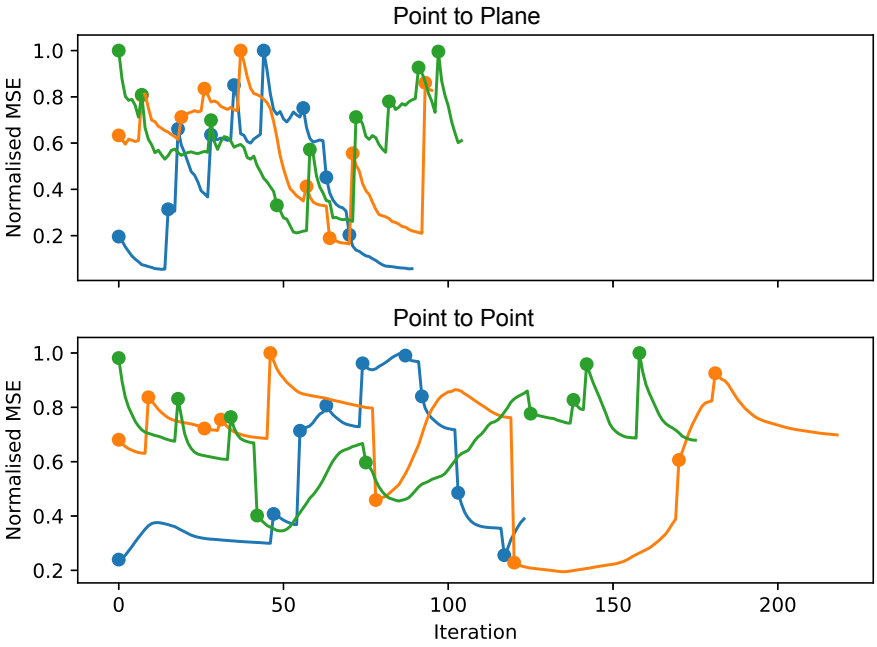



Figure 5.7: The cost functions at each iteration for matching 3 miniatures to their ground truth clouds for point to point and point to plane. Each dot represents the algorithm re-initialising at a new angle evenly spaced at 36 degree increments. Convergence and subsequent re-initialisation occurs when the change in cost function between iterations is less than 0.1%. Corresponding colours refer to the same model being matched in each case.

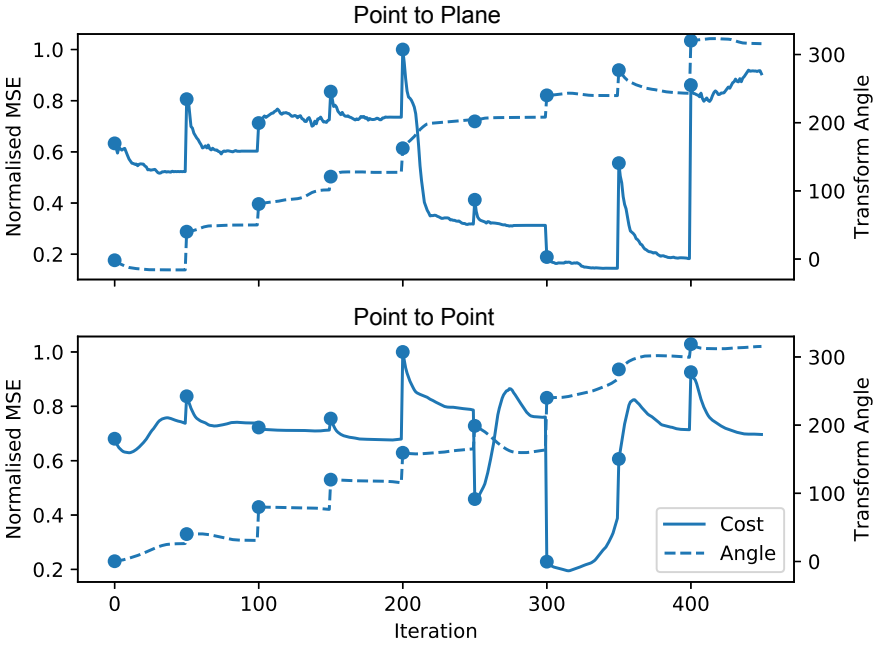



Figure 5.8: The cost function and angle of rotation at each iteration. Each dot represents the algorithm re-initialising at a new angle evenly spaced at 36 degree increments. Each initialisation is left to run for 50 iterations.

# Chapter 6

# Line of Sight

With a fully reconstructed scene, locating several reference models within the digital space, the process of resolving ambiguity in *Line of Sight* (LOS) and position can now begin. Figures 6.1 & 6.2[1] shows a comparison of several views digitally reconstructed vs. pictures of the physical scenes. As can be seen in the cases where classification is correct, results are very close to one another, but now there is the added advantage that a user can explore any camera location on the board. Determining the distance between two miniatures is now trivial - using the bases found in Chapter 5.2 the minimum distance is simply the distance between the centres of each base, minus their radii. The more difficult challenge is Line of Sight. A number of methods from computer graphics rendering can help determine properties of meshes in a scene.

The first step is to classify all the faces that are pointing away from the camera view in a process call back-face culling [11]. In a triangle mesh, the triangle's surface normal can be determined easily using the connecting vertices. The direction of the normal is determined by the order of the vertices specified, creating a screw rule. A triangle can then be labelled as not visible if it points away from the viewer, by checking the sign of the scalar product of the normal with the vector from the camera centre to the triangle's centroid.

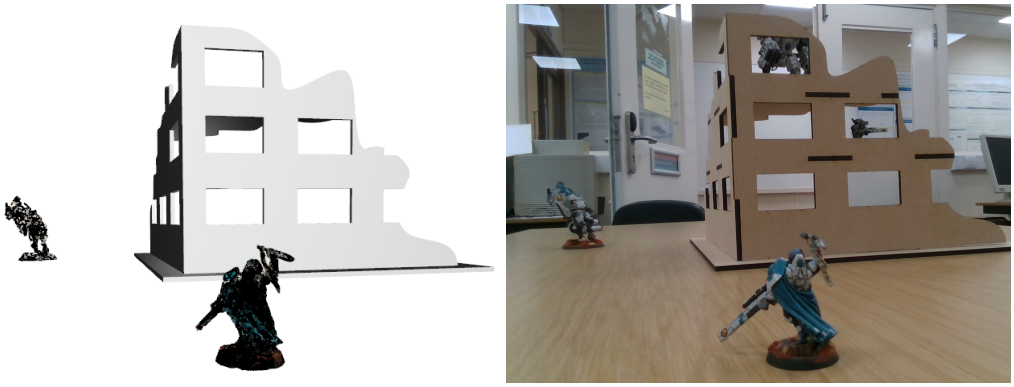
With knowledge of which points are facing away, there are two possible ways to proceed that form the basis of the LOS investigation. First is making use of the silhouettes of models, the second is making use of ray-casting to try and draw lines from the origin to the desired object.

## 6.1 Silhouettes

Calculating the silhouettes of 3D models is a common task in 3D graphics for user-interface enhancement such as when objects are highlighted or for artistic styles such as cel-shading graphics [1] [32]. These 'silhouettes' produce several intersecting loops along vertices where the mesh turns away from the camera view.

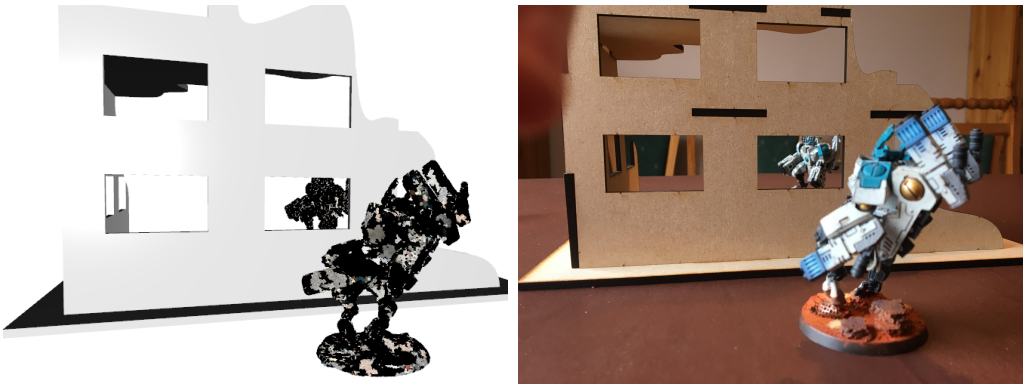
The output silhouette should, therefore, be a single list of vertices which can trace the outline of the triangle mesh in a given view. Consider 2 triangles with vertices $\{\mathbf{z}, \mathbf{x}_1, \mathbf{x}_2, \mathbf{y}\} \in \mathbb{R}^3$ where the edge $l$ between $\mathbf{x}_1$ & $\mathbf{x}_2$ is shared between each triangle, and $\mathbf{z}$ is the point further from the viewpoint than $\mathbf{y}$. $l$ is on the silhouette if and only if $\mathbf{z}$ is on a back-facing triangle and $\mathbf{y}$ is on a front facing triangle as in Figure 6.4. An

[1] Currently Open3d does not support multi-image textures that AliceVision produces and hence miniatures look 'patchy' in these visualisations.



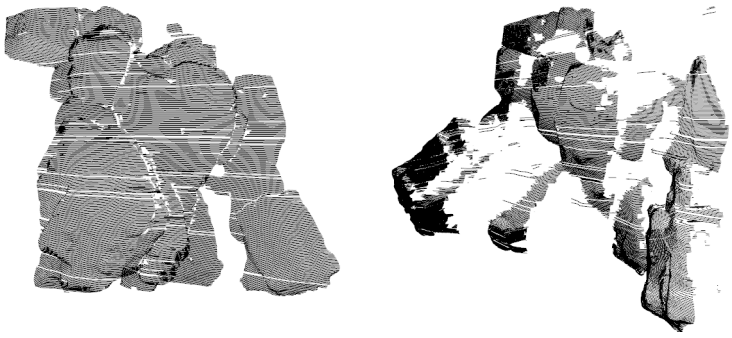
(a) Reconstructed View (b) Real View

Figure 6.1: A model's view from Layout 1



(a) Reconstructed View (b) Real View

Figure 6.2: A model's view from Layout 2



(a) Front (b) Side

Figure 6.3: Ray-Casting on the Broadside reference model

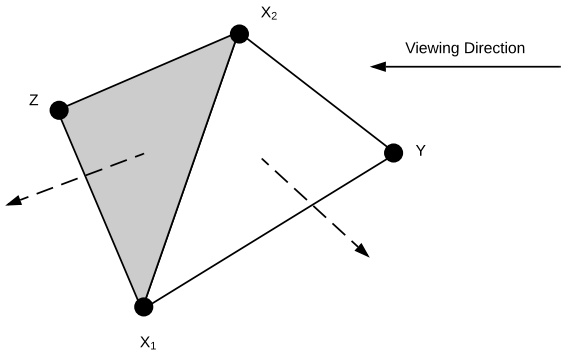



Figure 6.4: Determining silhouette vertices based on back and front facing adjacent triangles. $X_1, X_2$ are on the silhouette since triangle $X_1X_2Z$ is back-facing and $X_1X_2Y$ is front-facing.

algorithm can then follow the line around until it forms a closed loop. With a set of vertices on the silhouette, perspective projection can then map these vertices onto the image plane.

To determine LOS each silhouette is added to the image plane in order of closest to the viewpoint first as in Figure 6.5. If silhouettes intersect, a model is obscured if all the vertices on the further silhouette are inside the silhouette of the closer, if there is any region not intersecting then the model can be seen. This method is very simple and allows the degree of visibility to be determined as the area of the model that is not intersecting. Being able to quantify this helps with certain games interactions where a model receives a benefit of cover if a certain amount of the model is obscured by intervening terrain. However, for large models such as buildings where, in the case of miniatures being inside them, different model's 3D bounding boxes can intersect, determining the order of adding a silhouette to an image plane becomes ambiguous. Resolving this conflict might take additional steps where the silhouette needs to be split up.

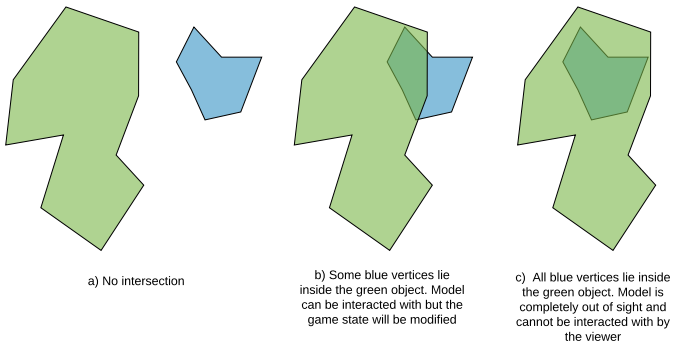



Figure 6.5: Two arbitrary polygon silhouettes of a closer model (green) and a distant model (blue). By looking at whether vertices are internal to the closer object's polygon, it can be determined whether a model is in full view, partially obscurred or cannot be seen by the viewer.

Nonetheless, taking the convex hull of mesh vertices projected onto the image plane can provide an initial indication as to whether a model is not obscured at all, and act as an initial filtering step as to whether a more detailed analysis and the more computationally intensive ray-casting is required

## 6.2 Ray-Casting

Ray-casting is a rendering algorithm originally proposed in 1982 by Roth [30] to trace what a camera can view in a digital space by mimicking rays of light. By projecting lines or rays from the camera centre, a point is drawn in a 3D space at the position the ray intersects with the first surface as in Figure 6.6. *Wolfenstein 3D* [36] was the first commercial use of the algorithm, and took a 2D plan view of a room to project rays and render the objects it hits. It makes the assumption that all the objects in the room have a uniform cross-section so that the process only needs to be carried out in 2D once. However, this can be adapted for 3D by taking a slice of a mesh along the plane to produce a set of lines and then determining which of those lines are occluded from the viewpoint or not.

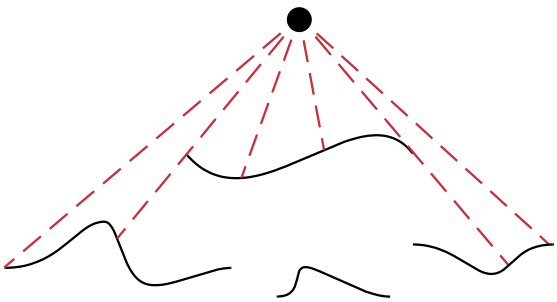


Figure 6.6: Rays (red) projected from an optical centre to a number of planar line segments. What can be seen can then be determined by where these rays intersect with the line.

A plane that passes through the optical centre, $\mathbf{c}$ with normal $\mathbf{n}$ will intersect with any triangle that has 1 or 2 vertices on one side of the plane. If the mesh has already had its back faces removed, the triangles can be grouped base on whether they connect. To find specifically the line where the plane intersects the triangle, define vertex $\mathbf{x}$ as the only vertex on one side of the plane and $\mathbf{y}_1$, $\mathbf{y}_2$ as the other vertices that are together on the other side of the plane. Defining the line between $\mathbf{x}$ and $\mathbf{y}_i$:

$$\mathbf{r} = \mathbf{x} + \alpha\mathbf{d} \tag{6.1}$$

$$\mathbf{d} = \mathbf{y}_i - \mathbf{x} \tag{6.2}$$

Then the intersection of $\mathbf{r}^T\mathbf{n} = \mathbf{c}^T\mathbf{n}$ is at:

$$\alpha_i = \frac{(\mathbf{c} - \mathbf{x})^T\mathbf{n}}{(\mathbf{y}_i - \mathbf{x})^T\mathbf{n}} \tag{6.3}$$

Yielding 2 points that form a line section across the triangle as in Figure 6.7. Repeated across every triangle, and keeping the line sections from adjacent triangles together, produces a collection of non-intersecting lines for which occlusion can then be calculated.

Traditionally, ray-casting will determine occlusion using a set of discrete rays, usually one for each pixel in an image, traced from the viewpoint through the pixel. This can be done by repeating the process above for an orthogonal plane, finding the intersection between the two line sets (which will simply be the triangles that are split by both planes) and keeping the closest intersection point. However, this discrete approach

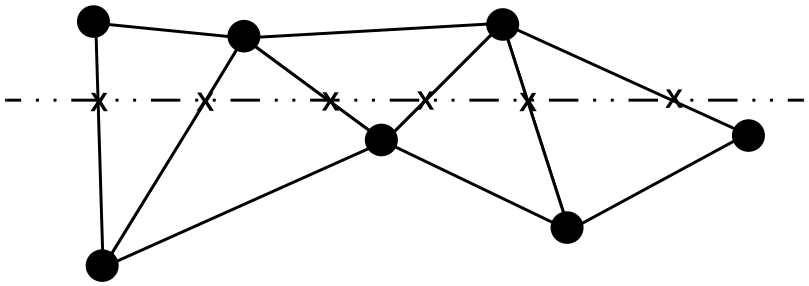


Figure 6.7: A plane (dashed line) through a triangle mesh can trace a line across the mesh's surface

could miss important holes depending on the resolution of the algorithm. Instead, the method presented here is akin to 'plane-casting' making use of the planar lines calculated previously.

Suppose there is a set of non-intersecting 2D line sections which do not self-occlude[2] $\mathcal{L}$. The vertices shall be defined in polar coordinates with the polar axis being the vector from the viewpoint to the centroid $\mathcal{L}$. To calculate which parts of the lines are occluded, a mask is maintained such that if a point has its angle $\theta$ within the mask then it is occluded. Each line section is considered in order of its distance from the origin, any vertices that are within the mask are removed (splitting the line in two if necessary), and then the maximum and minimum $\theta$ on that line set in then added to the mask. Provisional results in Figure 6.3 show how lines are only created on parts of the model that are visible, sections of the leg do not have lines on them where arms have obscured them from the viewer. While overall more computationally intensive than silhouette intersection (which use a much smaller subset of vertices in the mesh), ray-casting inherently determines what is visible, requiring no additional modifications to deal with holes in arbitrarily complex surfaces.

[2]Any line section created from a mesh that has been backface-culled will satisfy this.

# Chapter 7

# Game Analysis

Finally to conclude this report, a brief analysis of dice statistics used in tabletop wargames is provided. This can act as the basis for integration with the digital reconstruction to issue advice and information to players on the current options available. The specific ruleset used is the popular *Warhammer 40,000* (40k) [37], nonetheless, the principles are easily applicable to other rulesets that are available.

40k makes use of a hierarchal model for its interactions - a sequence of dice rolls are carried out, where each role affects the next to determine the final outcome of an action. Typically a dice is rolled to hit and has a probability of success $\theta_H$, if the hit is successful it is rolled again to wound, with probability of success $\theta_W$, and finally, if the wound is successful it is rolled again and causes one damage with a probability $\theta_D$. The result of one damage against a unit is that a model from that unit is removed from the game[1]. This process can be modelled as several Bernoulli random variables: a roll to hit $H$, to wound $W$, and an armour save/ damage $D$ such that:

$$H \sim \text{Ber}(\theta_H) \tag{7.1}$$
$$W|H = 1 \sim \text{Ber}(\theta_W) \tag{7.2}$$
$$D|W = 1 \sim \text{Ber}(\theta_D) \tag{7.3}$$

In the case of a model/unit having $n$ shots, this becomes a sequence of conditional Binomial distributions

$$H \sim \text{B}(n, \theta_H) \tag{7.4}$$
$$W|H \sim \text{B}(H, \theta_W) \tag{7.5}$$
$$D|W \sim \text{B}(W, \theta_D) \tag{7.6}$$

The specific values of probability set $\boldsymbol{\theta} = \{\theta_H, \theta_W, \theta_D\}$ are determined based on combinations of a model's stats that are provided as a part of the game's rules - they provide a value on which a 6-sided die needs to be greater than to be considered a success, for example a roll to hit of a 3 or more would be the equivalent of $\theta_H = 2/3$.

The first step is to find the distribution $p_D(d|n, \boldsymbol{\theta})$ so there is a direct mapping of an initial action to the outcome as a probability. When writing probability distributions the convention $p_X(x|a, b)$ is taken to mean the probability the random variable $X$ takes the value $x$ given variables $a, b$.

[1]There are cases where larger models require several damage to be removed but such cases complicate the maths without providing additional insight so are not considered here.

Consider first $p_W(w|n,\theta_H,\theta_W)$, that is the probability distribution of the number of successful wounds, calculated by taking the joint distribution and summing over every possible value of $H$:

$$p_W(w|n,\theta_H,\theta_W) = \sum_{h=0}^{n} p_W(w|H=h,\theta_W)p_H(h|n,\theta_H) \tag{7.7}$$

$$= \sum_{h=0}^{n} \left[{}^hC_w\theta_W^w(1-\theta_W)^{h-w}\right]\left[{}^nC_h\theta_H^h(1-\theta_H)^{n-h}\right] \tag{7.8}$$

$${}^nC_h = \frac{n!}{h!(n-h)!} \tag{7.9}$$

Of course $H \geq W$ and so the index of the summation can be shifted. Let $k = h - w$:

$$p_W(w|n,\theta_H,\theta_W) = \sum_{k=0}^{n-w} \left[{}^{k+w}C_w\theta_W^w(1-\theta_W)^k\right]\left[{}^nC_{k+w}\theta_H^{k+w}(1-\theta_H)^{n-k-w})\right] \tag{7.10}$$

Note also that:

$${}^{k+w}C_w\ {}^nC_{k+w} = \frac{(k+w)!}{w!k!}\frac{n!}{(k+w)!(n-k-w)!}$$

$$= \frac{n!}{w!k!(n-k-w)!} = \frac{n!}{w!(n-w)!}\frac{(n-w!)}{k!(n-k-w)!}$$

$$=^n C_w\ {}^{n-w}C_k \tag{7.11}$$

$$p_W(w|n,\theta_H,\theta_W) = {}^nC_w(\theta_H\theta_W)^w \sum_{k=0}^{n-w} {}^{n-w}C_k \left[\theta_H(1-\theta_W)\right]^k (1-\theta_H)^{n-k-w}$$

$$= {}^nC_w(\theta_H\theta_W)^w \left[\theta_H(1-\theta_W)+1-\theta_H\right]^{n-w} \tag{7.12}$$

$$= {}^nC_w(\theta_H\theta_W)^w(1-\theta_H\theta_W)^{n-w} \tag{7.13}$$

Where Eq. 7.12 comes from the application of the binomial theorem. Therefore the consecutive binomial distributions, conditional on the previous, are just a binomial with the probabilities of success multiplied. Hence for the case where there are $n$ shots in an attack all with the same weapon:

$$W \sim \mathrm{B}(n,\theta_H\theta_W) \tag{7.14}$$

$$D \sim \mathrm{B}(n,\theta_H\theta_W\theta_D) \tag{7.15}$$

This result makes sense intuitively since for a single initial 'to hit' dice roll to subsequently result in a damage requires 3 consecutive successes and so the 3 sequential trials can be merged into a single trial encompassing the 3 related events with probability $\theta_H\theta_W\theta_D$. To keep equations terse, herein $\theta$ without a subscript will refer to the product of all elements in $\boldsymbol{\theta}$.

In the case of a model having $M$ different *types* of attacks (this may represent additional weapons that generate a different $\boldsymbol{\theta}$ with a distinct number of shots), the total damage model $x$ inflicted on model $y$, $D_{xy}$ is given as the sum of individual damages:

$$D_{xy} = \sum_{i=1}^{M} D_i \tag{7.16}$$

$$p_D(d|x,y) = p_D(d|n_1,\boldsymbol{\theta}_1) \circledast p_D(d|n_2,\boldsymbol{\theta}_2) \circledast ...p_D(d|n_M,\boldsymbol{\theta}_M) \tag{7.17}$$

Where $a \circledast b$ denotes the convolution of $a$ and $b$. This is a case of the Poisson Binomial distribution. While a full closed form definition of the distribution would be intractable except in special cases, it is very rare in a game that a single unit makes more than 50 attacks in total (across all weapons), and hence it would not be difficult to generate the individual binomial distributions and combine them using a *Fast Fourier Transform/ Moment Generating Function*. In most cases $n_i$ is either equal to the number of models in the unit $n$ or 1 where a single model in the unit has some sort of special weapon.

Finally, when there is more damage than models in the targeted unit (referred to as overkill) the remaining damage is lost, this means that the total models lost, $L$ from a unit of $m$ models when attacked with a combined total of $N$ shots is:

$$L = \min\{D, m\} \tag{7.18}$$

$$p_L(l|x,y) = \begin{cases} p_D(l|x,y), & 0 \leq l < m \\ \sum_{i=m}^{N} p_D(i|x,y), & l = m \\ 0, & \text{otherwise} \end{cases} \tag{7.19}$$

i.e. the overkill-binomial is a standard binomial up until the limit, where the final value is the sum of probabilities of all remaining values. These results can provide the basis for calculating a number of metrics in a certain situation and then provide suggestions to maximise the opportunity of the board state.

## 7.1 Threat and Opportunity

As mentioned previously, the outcome of an attack results in models being removed from the game dependent on how much damage occurs from the dice rolls. Suppose we have two armies $\mathcal{X}, \mathcal{Y}$ which are a set of units in states $\{x_i\}_{n=1}^{N}, \{y_i\}_{n=1}^{M}$ respectively - where a unit is a collection of models with the same statistics that must all perform the same action[2]. We can define a threat $T_{xy}$ between two units in states $x, y$ as the expected damage that model $x$ can do to $y$ if they were to attack now, and the opportunity as the change in that threat from taking action $m$ that transitions $x$ to state $x'$.

$$T_{xy} = \mathbb{E}_{p_L(l|x,y)}[L_{xy}] \tag{7.20}$$

$$\Delta T_{xy} = T_{xy} - T_{x'y} \tag{7.21}$$

In the case of the Poisson Binomial distribution, this is simply the sum of expectations of the individual binomial distributions. The convention is followed that when unit states are listed (either as parameters to a distribution $p_L(l|x,y)$ or subscripts like in $L_{xy}$) it refers to the first unit taking an action on the second unit. Each action that a unit can take will effect $\boldsymbol{\theta}$ in some way. For example, moving $x$ to within range of $y$ will increase its threat from 0, giving a large opportunity. However, it is likely that this action will also affect the opportunity of $y$, since $x$ may then come into the range of $y$'s weapons as well. Alternatively, if that move brings $x$ in range, but also puts it in some position of cover, $x$'s opportunity will be the same as before, but $y$ will not be changed as much. This leads to an optimisable goal - to maximise the difference in each army's opportunity in one player's favour.

[2]This can also apply to rulesets where a unit can split its attacks between different targets by considering the unit as two smaller units when attacking.

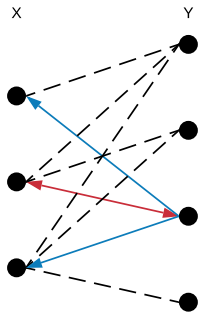



Figure 7.1: The interactions between armies $\mathcal{X}$ and $\mathcal{Y}$ can be visualised as a factor graph. When a unit $x_i$ makes an attack (red) against $y_i$ messages can be passed along the edges to estimate trickle down effects on the threats between each unit (blue).

The movement actions described above are deterministic, affecting the damage distributions by changing $\boldsymbol{\theta}$ in a predictable way - the distribution still stays in the same form (i.e. Binomial, Poisson-Binomial) as it was before. Attack actions on the other hand have a random result - it is easy to calculate the threat of $x$ attacking $y$, $T_{xy}$ using the results at the start of this chapter but focus is now turned to the distribution of $y$'s returning fire to calculate $T_{yx'}$, that is, the threat of $y$ attacking $x$ *after* $x$ has attacked. This is more difficult since the parameters of $y$'s damage distribution are now themselves random variables. If $y$ has 1 shot per model and $n_{y,0}$ models before $x$ attacks, $y$'s return fire will have a number of shots, $n_y$ based on the remaining models after $x$ attacks.

$$n_y = n_{y,0} - L_{xy} \tag{7.22}$$

As established in Eq. 7.13 the damage distribution for $y$ given $n_y$ is a binomial and so the returning fire distribution then becomes:

$$p_L(l|y, x') = \sum_k p_L(k|x, y) p_L(l|n_y - k, \boldsymbol{\theta}) \tag{7.23}$$

$$\mathbf{p}_{yx'} = \mathbf{p}_{xy}\mathbf{P}_{yx'} \tag{7.24}$$

$\mathbf{P}_{yx'}$ is a matrix with rows equal the discrete values of $y$'s damage distribution conditional on the outcome of $x$'s initial attack and $\mathbf{p}_{xy}$ is the distribution of $x$ attacking $y$ as a vector. Eq. 7.24 shows how the solution can be simplified into a matrix form for quick computation.

The notion that these distributions can be vectorised leads to a further enhancement to look at the interactions between action and threat. In a rather abstract fashion, one can consider each model as the node on a graph, with connections between models that can interact with each other as in Figure 7.1. A controller of army $\mathcal{X}$ considers several actions (messages passed from left to right in Figure 7.1) and the resulting change in threat from army $\mathcal{Y}$ (message passed from right to left). Once the true outcome of an action taken has been determined (the result of the dice rolls), whether the other considered actions should also be taken can then be evaluated as the network becomes more deterministic. While this does little to actively structure and hence reduce the space of actions to choose from, this networked consideration of threats and opportunities provides a framework to assess proposed actions against one another.

To summarise, the process of evaluating the outcome of $x$ taking an action is as follows:

1. Calculate the threat $x$ poses to each enemy model and vice versa *before* the action is taken.

2. Calculate the new threat after $x$ has taken the action. If the action involves an attack use the appropriate marginal distribution as in Eq. 7.24. In the case of the threat, the enemy models pose to $x$ this is effectively asking 'What is the threat from the enemy if they could immediately return fire after the action?'

3. Find the opportunity this action creates for each model.

4. Summarise the opportunity as the swing $\Delta T_{xy} - \Delta T_{yx}$.

## 7.2 Mock Scenario

Suppose we have a scenario of 5 miniatures in a unit we control $X$, an enemy unit with 3 miniatures $Y_1$ and an enemy unit with 10 miniatures $Y_2$. For the sake of simplicity, all miniatures in each unit have the exact same statistics, so the only variation in each unit's $\boldsymbol{\theta}$ is purely down to a unit's size and position. There is also an item of scenery that can provide cover if a unit moves into it as in Figure 7.2. The following movement and attack combinations will be considered:

0. Stay put. This provides no bonuses and is used as the control case to calculate opportunity.

1. Move towards unit $Y_1$ putting $X$ in rapid fire range - this will double the number of attacks that $X$ can make. $Y_1$ will be in rapid fire range also on its turn. It is assumed that $X$ will then attack $Y_1$. While this may not be the optimal attack, it is clear such a move is aimed to maximise damage against $Y_1$.

2. Move towards unit $Y_2$, this will put $X$ in rapid fire range as above, but also put $X$ in cover. $Y_1$ will be in rapid fire range also on its turn. There is a trade-off between an additional number of shots from $Y_2$ in return, but a reduction in the likelihood of damage. It is assumed $X$ will attack $Y_2$ for the same reasons outlined above.

3. Move behind cover. $X$ will only be able to see $Y_1$, but $Y_2$ will not be able to attack next turn

4. As 2. but $X$ does not get the benefit of cover. This is clearly a strictly worse choice of action but is used for comparative purposes.

The effects of each action on each unit's $(n, \theta)$ are given in Table 7.1, noting that the $n$ given for the return value is the maximum, and the true $n$ is conditional on the outcome of the action. Table 7.2 shows the breakdown of threat and opportunities using the steps outlined previously. It is clear that Action 2 provides the largest swing and would agree with the intuition of a player in the game. More surprisingly is that Action 3 is comparable despite offensive output being the smallest under this action. Since these are point estimates, it could be enhanced by considering variances of the underlying attack distributions.

While this is clearly a very simple example with a small action space, it provides tentative evidence that this framework can be used to suggest reasonable actions in the eyes of an experienced player. In this specific example, each action results in an instant

change in the attack distributions of the models, and hence an instant effect on the threat. However, it currently cannot quantify actions which may affect subsequent turns, for example, if $x$ were out of range and its movement distance would not bring it in range this turn but would put it in a position to move into range next turn.

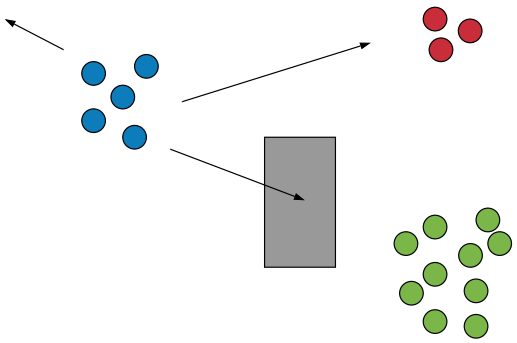


Figure 7.2: $X$ (blue) is on a tabletop with $Y_1$ (red) and $Y_2$ (green). They can choose to move towards cover (grey) but risk more return fire

Table 7.1: $n, \theta$ for $X$ shooting to $Y_i$ and for $X$ receiving fire from $Y_i$. Note the value of $n$ for receiving fire is a maximum possible value, since it may be reduced after attacks from $X$

| | | To $Y_1$ | To $Y_2$ | From $Y_1$ | From $Y_2$ |
|---|---|---|---|---|---|
| Action | 0) | 5, 1/9 | 5, 1/9 | 3, 1/9 | 10, 1/9 |
| | 1) | 10, 1/9 | 5, 1/9 | 6, 1/9 | 10, 1/9 |
| | 2) | 5, 1/9 | 10, 1/9 | 3, 1/18 | 20, 1/18 |
| | 3) | 5, 1/9 | 0, - | 3, 1/9 | 0, - |
| | 4) | 5, 1/9 | 10, 1/9 | 3, 1/9 | 20, 1/9 |

Table 7.2: Threat and Opportunity for each action

| | | $T_{XY_1}$ | $T_{XY_2}$ | $T_{Y_1X}$ | $T_{Y_1X}$ | $\Delta T_{XY_1}$ | $\Delta T_{XY_2}$ | $\Delta T_{Y_1X}$ | $\Delta T_{Y_1X}$ | Swing |
|---|---|---|---|---|---|---|---|---|---|---|
| Action | 0) | 0.55 | 0.55 | 0.33 | 1.11 | - | - | - | - | - |
| | 1) | 1.09 | 0.55 | 0.42 | 1.11 | 0.54 | 0 | 0.09 | 0 | 0.45 |
| | 2) | 0.55 | 1.11 | 0.17 | 0.99 | 0 | 0.56 | -0.16 | -0.12 | 0.83 |
| | 3) | 0.55 | 0 | 0.27 | 0 | 0 | -0.55 | -0.16 | -1.11 | 0.72 |
| | 4) | 0.55 | 1.11 | 0.33 | 1.96 | 0 | 0.56 | 0 | 0.85 | -0.29 |

# Chapter 8

# Conclusion

Presented in this report is a framework for the digital reconstruction of boards for tabletop wargames. Through the use of 3D point cloud data, it is possible to find simple 3D geometries as the basis for an initial alignment to combine separate views of a board state. This can then be refined through the use of a number of variations of Iterative Closest Point. As a part of this alignment process, a novel algorithm for isolating corners using a RANSAC least squares is provided. Combined with an initial plane alignment, the need for a full 6 degree of freedom registration is removed and a more computationally efficient optimisation on 3-DOF is derived instead for both Point to Point and Point to Plane methods.

Segmentation was originally trialled using region growing, which provided promising results for splitting up scenery models and would be the basis for searching for miniatures within buildings. Nonetheless, the more reliable DBSCAN algorithm provides a more robust segmentation for finding appropriate *Regions of Interest* (ROI). For these precision clusters, a matching method utilising the constrained ICP was devised. Matching is determined by registering the clusters with photogrammetrically produced reference models, and this reliable method provides a visually accurate reconstruction of the board state. However, this method does currently have issues with smaller miniatures captured in noisier environments which provide less consistent matches with either incorrect alignments or incorrect reference matches altogether. Moreover, although it is suitable for a small number of miniatures it may not scale well to larger scenes.

With reference to the initial problem statement, to complete the task of fully determining the board state, the reconstruction can then be used to determine Line of Sight. The most promising method is ray casting due to it being able to deal with arbitrarily complex shapes, although there was some investigation into how silhouettes can be also used to determine what a miniature can see.

Finally, a brief statistical analysis is shown as the next step on how information from the digital reconstruction can be used to influence a game. A method for determining the likely outcomes of actions as well as the advantages of certain actions gives can be used to formalise a game as an optimal control problem, with some simple general results. This statistical work can also provide the basis for further analysis into power balancing systems used in tabletop wargames that make sure both players have an army with similar power levels and stand a 50:50 chance of winning at the start of a game.

## 8.1 Further Work

The next logical step to enhance the framework presented is to scale it up to larger systems. Currently, the matching algorithm has a complexity $\mathcal{O}(nm)$ where $n$ is the number of ROIs and $m$ is the number of reference models. Despite being simply parallelisable, clearly, this does not scale well and so a number of steps should be taken to improve the matching.

**Probabilistic Models** In a typical game while there might be 100+ miniatures, there may only be 10-15 unique miniatures, that is, miniatures with different rules to one another. However, two miniatures that may be the same in terms of rules can have different poses and hence require 2 separate reference models in the current framework. A more probabilistic classifier could be created that combines each of these poses into a single high-dimensional probability field. This could be point-wise i.e. the probability of each point in a ROI being independently generated from the reference model, or a more complex joint model across all the points.

**2D cameras** For the framework to be accessible to the average tabletop wargames player, adapting the processes in this report to make use of 2D camera images is crucial. It has been demonstrated that the AliceVision photogrammetry pipeline can create high-quality useable point clouds from 2D images, but the processing time is prohibitive and requires a large number of photographs. It is well known how to find simple geometries in 2D images such as the surface of the table [16], and there are already some applications in wargaming such as *Rightful Ruler* [28] that make use of *Augmented Reality* (AR) to solve the measurement problem, but require user input to identify miniatures in the space.

The classification task in 2D should also be a much simpler one. Photogrammetry results in Chapter 5.1.2 show how photo-realistic 3D models can be created. From this, a bank of synthetic training images can be generated by capturing the 3D meshes at different angles. This training data can fuel a *Convolutional Neural-Network* (CNN) for matching and combining its location in the image with the geometry of the table. A CNN approach would also be useful for the *many model one rule* case as the training data can be enhanced with images of different poses, and the CNN can be left to process appropriate features.

**Low-Level implementation** All the results in this report have been presented from offline systems - datasets are created all at once and then algorithms are run on each of them as a demonstration of the key concepts. Currently, this means that runtime is not a significant issue and hence all code was dealt with in Python. Several optimisations can be made to the algorithms largely through parallelisation at a lower level, and possibly even GPU acceleration. Transferring implementations into C++ can make use of lower-level libraries such as PCL [5] and push performance to a speed where online capability can be created.

# Appendix A

# Digital Resources

A copy of all code and the logbook documenting this project can be found at `github.com/falcoso/Vision-Hammer`. As a part of this repository is a Python Notebook running the experiments used to generate the results in this project. Earlier experiments may no longer run due to later updates in the codebase, but are kept as a record of original work.

Data used in the experiments can be found at `shorturl.at/rszNU` along with 3 digital notebooks used to record mathematical derivations, and geometry notes. An attempt has been made to split the notebooks into geometry, algorithms and statistics but there is some overlap between them and repeated derivations to ensure they are correct. Copies of the notebooks and logbook have also been submitted alongside this report on moodle.

# Appendix B

# Risk Assessment Retrospective

The only risk assessed surrounding this project was extensive computer work that would be required. While largely this was the only risk, early in the project it was noted that the laser cutter was required to make the building used in the datasets. For this one-off task, it was ensured that proper training of the equipment was undertaken, taught by Dyson Centre staff and appropriate eye protection was maintained at all times while using the laser cutter.

Measures for working safely with computers such as taking regularly breaks and maintaining good posture were followed with no injuries sustained over the project.

# Appendix C

# Impact of COVID-19

As most of the work on this project involved software development, there has been little impact from the COVID-19 outbreak on the work in this project. The only caveat to this was in collecting datasets. It was initially hoped to create all the datasets under the same lighting conditions in the same lab, but as can be seen in the results this was not possible due to the closure of departmental facilities.

It is not believed that this has a significant impact on results gathered, and naturally, robustness to different lighting conditions would be important for a more widely applicable application anyway.

# Bibliography


[1] P. Alliez et al. "Efficient view-dependent refinement of 3D meshes using -subdivision". In: *The Visual Computer* 19 (July 2003), pp. 205–221. DOI: 10.1007/s00371-002-0165-z.

[2] J. L. Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517. DOI: 10.1145/361002.361007. URL: https://doi.org/10.1145%2F361002.361007.

[3] F. Bernardini et al. "The ball-pivoting algorithm for surface reconstruction". In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), pp. 349–359.

[4] P. J. Besl and N. D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256. ISSN: 1939-3539. DOI: 10.1109/34.121791.

[5] R. Bogdan Rusu and S. Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, May 2011.

[6] Y. Chen and G. Medioni. "Object Modeling by Registration of Multiple Range Images". In: *Image Vision Comput.* 10 (Jan. 1992), pp. 145–155. DOI: 10.1109/ROBOT.1991.132043.

[7] *CloudCompare*. Version 2.11. GPL software, 2019. URL: http://www.cloudcompare.org/.

[8] D. P. Darzentas. "The Lives of Objects: Designing for Meaningful Things". PhD thesis. University of Nottingham, Sept. 2017. URL: http://eprints.nottingham.ac.uk/50282/.

[9] D. P. Darzentas et al. "The Data Driven Lives of Wargaming Miniatures". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. CHI '15. Seoul, Republic of Korea: Association for Computing Machinery, 2015, pp. 2427–2436. ISBN: 9781450331456. DOI: 10.1145/2702123.2702377. URL: https://doi.org/10.1145/2702123.2702377.

[10] J. Digne. "An Analysis and Implementation of a Parallel Ball Pivoting Algorithm". In: *Image Processing On Line* 4 (2014), pp. 149–168. DOI: 10.5201/ipol.2014.81.

[11] D. H. Eberly. *3D game engine design : a practical approach to real-time computer graphics / David H. Eberly*. eng. Second edition. 2015. ISBN: 9781482267303.

[12] *Elites : special operatives in skirmish combat : Warhammer 40,000 Kill Team*. eng. Games Workshop, 2019. ISBN: 9781788264853.

[13] M. Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[14] M. A. Fischler and R. C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[15] G Godin, M. Rioux, and R. Baribeau. "Three-dimensional registration using range and intensity information". In: *Videometrics III*. Ed. by Sabry F. El-Hakim. Vol. 2350. International Society for Optics and Photonics. SPIE, 1994, pp. 279–290. DOI: 10.1117/12.189139. URL: https://doi.org/10.1117/12.189139.

[16] R. Hartley. *Multiple view geometry in computer vision / Richard Hartley, Andrew Zisserman.* eng. 2nd ed. Cambridge: Cambridge University Press, 2003. ISBN: 9780521540513.

[17] *Intel RealSense D400 Series Product Family*. 337029-005. Rev. 5. Intel. Jan. 2019.

[18] M. Jancosek and T. Pajdla. "Multi-view reconstruction preserving weakly-supported surfaces". In: *CVPR 2011*. IEEE, June 2011. DOI: 10.1109/cvpr.2011.5995693. URL: https://doi.org/10.1109/cvpr.2011.5995693.

[19] M. Kazhdan and H. Hoppe. "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graph.* 32.3 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2487228.2487237. URL: https://doi.org/10.1145/2487228.2487237.

[20] M. Korn, M. Holzkothen, and J. Pauli. "Color Supported Generalized-ICP". In: *Proceedings of the 9th International Conference on Computer Vision Theory and Applications - Volume 3: VISAPP, (VISIGRAPP 2014)*. INSTICC. SciTePress, 2014, pp. 592–599. ISBN: 978-989-758-009-3. DOI: 10.5220/0004692805920599.

[21] M. Łepicka, T. Kornuta, and M. Stefańczyk. "Utilization of Colour in ICP-based Point Cloud Registration". In: *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*. Ed. by Robert Burduk et al. Cham: Springer International Publishing, 2016, pp. 821–830. ISBN: 978-3-319-26227-7.

[22] K. L. Low. *Linear least-squares optimization for point-toplane ICP surface registration*. Tech. rep. University of North Carolina at Chapel Hill, 2004.

[23] J. E. Mebius. *Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations*. 2007. arXiv: math/0701759 [math.GM].

[24] P. Moulon, P. Monasse, and R. Marlet. "Adaptive Structure from Motion with a Contrario Model Estimation". In: *Proceedings of the Asian Computer Vision Conference (ACCV 2012)*. Springer Berlin Heidelberg, 2012, pp. 257–270. DOI: 10.1007/978-3-642-37447-0_20.

[25] J. Park, Q. Zhou, and V. Koltun. "Colored Point Cloud Registration Revisited". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 143–152. DOI: 10.1109/ICCV.2017.25.

[26] J. Park, Q. Zhou, and V. Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[27] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] N. Pihil. *Rightful Ruler*. Minitaur Gaming, 2019.

[29] M. Rahman. *Beginning Microsoft Kinect for Windows SDK 2.0 : Motion and Depth Sensing for Natural User Interfaces / Mansib Rahman.* eng. 2017. ISBN: 9781484223161.

[30] S. Roth. "Ray casting for modeling solids". In: *Computer Graphics and Image Processing* 18.2 (1982), pp. 109–144. ISSN: 0146-664X. DOI: https://doi.org/10.1016/0146-664X(82)90169-1. URL: http://www.sciencedirect.com/science/article/pii/0146664X82901691.

[31] R. B. Rusu, N. Blodow, and M. Beetz. "Fast Point Feature Histograms (FPFH) for 3D registration". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217.

[32] P. Sander et al. "Silhouette Clipping". In: *Computer Graphics (SIGGRAPH 2000)* (July 2004). DOI: 10.1145/344779.344935.

[33] P.H Schönemann. "A Generalized Solution of the Orthoganal Procrustes Problem". In: *Psychometrika* 31.1 (Mar. 1966), pp. 1–10. DOI: 10.1007/BF02289451.

[34] A. Segal, D. Hähnel, and S. Thrun. "Generalized-ICP". In: June 2009. DOI: 10.15607/RSS.2009.V.021.

[35] A. Al-Sharadqah and N. Chernov. "Error analysis for circle fitting algorithms". In: *Electron. J. Statist.* 3 (2009), pp. 886–911. DOI: 10.1214/09-EJS419. URL: https://doi.org/10.1214/09-EJS419.

[36] iD Software. *Wolfenstein 3D*. Apogee Software, 1992.

[37] *Warhammer 40,000*. eng. Games Workshop, 2016. ISBN: 9781785818493.

[38] C. Yuksel. "Sample Elimination for Generating Poisson Disk Sample Sets". In: *Comput. Graph. Forum* 34.2 (May 2015), pp. 25–32. ISSN: 0167-7055. DOI: 10.1111/cgf.12538. URL: https://doi.org/10.1111/cgf.12538.