Representations of Depth Maps: Spherical Harmonics and Wavelets

Author: Katrina Ashton

Supervisor: Jochen Trumpf

Course: ENGN3712 Engineering Research and Development Project

Submission: October 27, 2017

Acknowledgments

I'd like to thank my supervisor Jochen Trumpf for his help and guidance. Additionally, I'd like to thank Angus Gruen for proof reading my report and allowing me bounce ideas off him.

I would also like to acknowledge Sean O'Brien, whose work the techniques investigated in this report are intended to be used in conjunction with. Finally I would like to acknowledge Lessig and Fiume, whose work the spherical Haar wavelets portion of this report is based upon.

Abstract

This report investigates the suitability of spherical harmonics (SBFs) and spherical Haar wavelets (SHWs) as basis functions for a model to represent depth maps. Depth maps are useful in applications such as robotic mapping and 3D modelling.

The model is updated with new measurements using gradient descent. However, the measurements are taken from a rotated and/or translated camera, so in order to do this the model and the measurements need to be aligned. This report considers two ways of performing this alignment. The first method is transforming the model to be aligned with the camera. This is done by transforming the weights, but for both bases this can only be used when the camera is not translated. The second method is transforming the measurements to be aligned with the model. This method works for both rotation and translation of the camera.

The report investigates the convergence behaviour and computational time of each basis with each alignment method under ideal conditions, that is, with no measurement error and full field of view for the camera. For the SBFs, it also investigates the basin of attraction, robustness with respect to the scaling of the measurement gradients and measurement uncertainty, and the effect of field of view.

The only significant difference between the two alignment methods for spherical harmonic basis functions in these areas is in their computational time – the moved measurements method is much faster. For the SHWs the two methods take a very similar time, which is between the two speeds for the SBFs. The moved model method converges faster than the moved measurements method for SHWs. The main limitation of both methods for the SBFs is the difficulty and importance of choosing an appropriate step size for the gradient descent. Choosing an appropriate step size is also important for the SHWs, as well as the choice of data structure. For most situations using the SBFs with the moved measurements method is the best choice.

Contents

1 Glossary and Notation

1.1 Abbreviations

FOV: Field of View

FWT: Fast Wavelet Transform

MRA: Multiresolution Analysis

SBF: (Real) Spherical (Harmonic) Basis Function

SHW: Spherical Haar Wavelet

SOHO: Orthogonal and symmetric Haar wavelets [15]

1.2 Notation

Basis functions and weights, maximum likelihood estimation, gradient descent (see Sections 3.4, 3.5, 3.6)

$\mathbf{x} = (x_1, x_2, ..., x_N)^T$: data

$\mathbf{t} = (t_1, t_2, ..., t_N)^T$: associated targets

$\mathbf{b} = (b_0, b_1, ..., b_{M-1})^T$: basis functions

$B = \begin{bmatrix} b_0(\mathbf{x}) & b_1(\mathbf{x}) & \dots & b_{M-1}(\mathbf{x}) \end{bmatrix}$: matrix containing basis functions applied to data

$\mathbf{w} = (w_0, w_1, ..., w_{M-1})^T$: weights associated to each basis function such that $B\mathbf{w}$ approximates $\mathbf{t}$

∇F: gradient of F

η: step size

Spherical coordinates (see Section 3.7)

θ: polar angle

ϕ: azimuthal angle

r: radius

Euler Angles (see Section 3.8)

$[\alpha, \beta, \gamma]$: ZYZ-Euler angles

Spherical Harmonics (see Section 3.9)

Y_{lm}: complex spherical harmonic

N_{lm}: normalisation factor

$P_l^m(x)$: Legendre associated polynomials

S_{lm}: real spherical harmonic

Wavelets (see Sections 3.11, 3.12, 3.13)

Ψ: mother wavelet

$\Psi_{j,m}(t)$: wavelet basis (for $L_2(\mathbb{R})$)

$\phi_{j,m}(t)$: scaling function

$\psi_{j,m}(t)$: wavelet function

$S_{j,m}$: scaling coefficient

$W_{j,m}$: detail coefficient

$T_{j,k}$: spherical triangle

$\alpha_{j,k}$: area of $T_{j,k}$

$\tau_{j,k}$: characteristic (indicator) function of $T_{j,k}$

1.3 Terminology

Moved model method: brings model and measurements into alignment by transforming the model to be with respect to the body fixed frame of the camera.

Moved measurements method: brings model and measurements into alignment by transforming the measurements to be with respect to the space fixed frame.

2 Introduction

A depth map of a scene is a function provides the distance between a given (centre) point (O) in the environment (Σ) and the point (P) in the scene (Ω) in any given direction from the centre [20]. (See Figure 1). Being able to produce an accurate depth map has a variety of applications, including 3D modelling and guidance and control of unmanned vehicles.

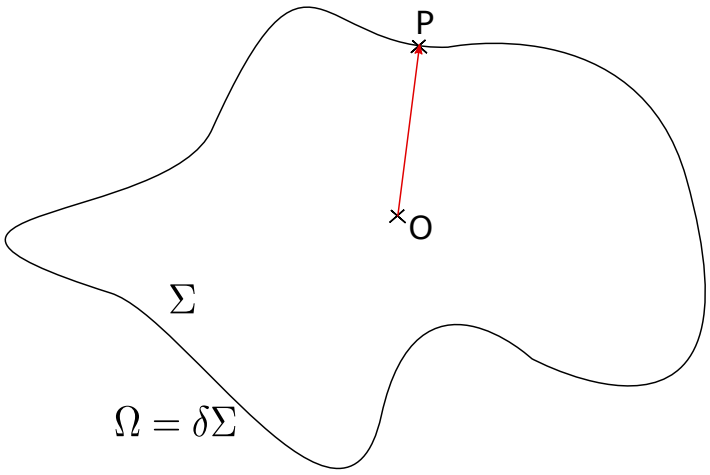


Figure 1: Scene map

Depth can be measured either actively or passively. Active methods interact with the environment directly, generally by emitting light or other radiation. For example, using a laser scanner. Passive methods do not emit anything, they measure aspects of the scene that are already present. For example, taking a photograph. Passive methods are generally preferred when they are available, as they do not interfere with the system being measured. However as passive methods are indirect, they can be more complicated to use than active methods. The techniques investigated in this report will focus on usage with passive methods, in particular using a light-field camera. Although they may also be applicable for depth measurements obtained via different means.

Light field cameras allow for depth information to be obtained in a single image capture. The depth information is not measured by the camera directly – it needs to

be computed from the light field data. The techniques described in this report are designed for use with the observer described in O'Brien et al. [20]. This observer uses a gradient observer approach. That is, given a current model of the scene and a light field, the observer gives gradients. Each gradient corresponds to how much the current scene model needs to be shifted in a certain direction in order to match the depth information. This gradient is guaranteed to point in the correct direction, but its magnitude may not be equal to the actual difference in depth between the model and the real scene. In addition, there is no guarantee that the scales of the gradients (i.e. how much the magnitude of the gradient differs from the difference in depth between the model and the scene) will be constant. Light fields do technically contain complete depth information, including accurate magnitudes. However, extracting this information relies on knowing the precise distance between points in the images captured by each lens/pin-hole camera. These distances are tiny and so cannot be measured without incurring some error. Because the distance is so small, any errors have a large impact on the calculated depth. Therefore extracting accurate depth information from light fields is not a trivial task and usually requires some form of iterative algorithm. For more information on light field cameras see Section 3.2.

Once the depth information has been obtained, it needs to be represented in a useful way. Which representation should be used and how to implement it is the focus of this report. A suitable representation should be able to accurately reflect the scene and be able to be updated efficiently with new depth measurements.

The aim of this report is to evaluate real spherical harmonic basis functions (SBFs) and spherical Haar wavelets (SHWs) as possible representations of a depth map for information obtained as described above.

3 Background and Literature Review

This section details the necessary background information and gives overview of relevant literature on scene depth, spherical harmonics and wavelets (see Sections 3.1, 3.9 and 3.11, respectively). More specifically, Haar wavelets and spherical Haar wavelets are discussed in some detail in Sections 3.12 and 3.13, respectively. These topics are core to this report, as the aim is to investigate the feasibility of using real spherical harmonic basis functions (SBFs) or spherical Haar wavelets (SHWs) to model scene depth.

This report also covers the procedure for updating these models from measurements from a camera in different poses. The first step in this process is to align the measurements and the model somehow.

Rotating spherical harmonics (see Section 3.10) and spherical Haar wavelets (see Section 3.14) allows for the SBF model and SHW model, respectively, to be brought into alignment with measurements from a rotated camera. Ideally the report would also discuss how to translate these models, however such a technique could not be located in the literature for either spherical harmonics or spherical Haar wavelets.

The other way to attain alignment is to transform the measurements so that they align with the model, this is discussed in Section 3.15. This method allows for both translating and rotating the camera.

There are also some other topics that need to be covered in order to understand the aforementioned sections. The techniques presented in this report are intended to be used with depth measurements from a light-field camera, so Section 3.2 covers the basics of how light-field cameras work. The spherical coordinate system is used for the depth measurements, and is outlined in Section 3.7. A number of sections cover the relevant theory for taking measurements with different camera poses: rigid body motion (Section 3.3) and Euler angles (Section 3.8).

Finally, the method for attaining the initial models and update them using the aligned measurements also needs to be discussed. The wavelet model is constructed using the Fast Wavelet Transform, discussed in the various sections on wavelets (3.11, 3.12, 3.13). The update method is also based on information from these sections, and is described in Section 4.9. The spherical harmonic model uses maximum likelihood estimation (see Section 3.5) for the initial model, and gradient descent (see Section 3.6) to update it. In order to use these methods an understanding of basis functions and weights (see Section 3.4) is necessary.

3.1 Scene depth

There are two main questions about scene depth: how to obtain depth information and how to model said information. The first is outside the scope of this report, whereas the second is central to it.

There have been many different modelling techniques applied to scene maps, changing to reflect both advances in measurement technology and what is being modelled. One early model uses only cubic voxels [23]. That is, the environment is split into cubes, and each cube is labelled as either Void, Full or Unknown. The model is built upon data from a range sensor, and only became practical due to the advancements in range sensing technology at the time, allowing for densely measured range data. This model is intended to work in any static environment.

This report, however, will focus on one specific type of scene depth model: depth maps. A Depth map is a function that take a direction and returns the depth in that direction. This approach is suitable because the measurements this report is intended to work with are tied to direction.

An example of a depth map representation is Newcombe and Davison [19], who use data from a single moving camera. The modelling starts with structure from motion

which generates a 3D feature point cloud. The surface is then approximated by fitting a function to the data points. The base mesh is formed by polygonising the function's zero level set. The base mesh allows for view predictions, which are then compared with true images, allowing for dense correspondence fields of sub-pixel accuracy to be obtained. This correspondence information is then used to update the base mesh into highly accurate local depth maps. Such a representation is only possible due to advances in both technology and modelling techniques.

Despite the development of new techniques, 3D voxels are still a very popular representation – although the depth reconstruction methods have also become more advanced. Polygonal meshes are also popular, and meshes are the standard representation used in computer graphics. Many algorithms use multiple depth representations, which are then merged into a 3D object model [28].

As light-field cameras are a new measurement technology, new depth representations should be explored in order to find one that is suitable for the new types of algorithms enabled by this technology.

3.2 Light-field Cameras

Light field cameras, also known as plenoptic cameras, allow for different focus or depth images to be obtained in a single capture [22]. The idea is similar to using two or more adjacent standard cameras, i.e. stereo systems. This allows for images to be captured from multiple view points, whose relative poses are known. The images can then be interpolated in order to retrieve depth information. Note that binocular stereo systems (2 cameras) exploit parallax along one axis and cannot offer depth estimates for contours parallel to this axis [1]. Light field cameras do not have this issue as they capture more than 2 images. They can also be made less bulky and require less calibration than stereo systems [1].

The idea of light field cameras has been around since 1908 [16], but only recently

has technology advanced to the point where they are feasible to construct. In 1992, Adelson and Wang [1] constructed a working prototype that used a single main lens along with a lenticular array placed at the sensor plane. Commercial light field cameras are now becoming available [17, 9].

3.3 Rigid Body Motion

"Rigid body dynamics is the study of the motion in space of one or several bodies in which deformation is neglected" [29]. That is, the distance between any two points on such a body remains constant in time. To properly describe rigid body motion, two frames are required: a space fixed (inertial) frame and a body fixed frame attached to the rigid body.

The movement of a rigid body can be represented as a rotation (which changes its orientation) and a translation (which changes its linear position). The orientation and position of the rigid body can be defined with respect to the space fixed frame using the rotation and translation that would give the space fixed frame the same orientation and translation as the body fixed frame. Together, orientation and position are referred to as pose.

In the case of this report, the light field camera is the rigid body. The rotation and translation referred to in this report are the transforms of the camera pose. This report seeks to update a model defined with respect to the space fixed frame with measurements that are made with respect to the body fixed frame of the camera. One way to do this is transforming the model to be with respect to the body fixed frame – we will call this the *moved model method* (see Sections 3.10 and 3.14). The other way is to transform the measurements to be with respect to the space fixed frame – we will call this the *moved measurements method* (see Section 3.15).

3.4 Basis functions and Weights

Basis functions give a basis of a function space, that is any element of that function space can be written as an infinite linear combination of basis functions. Note that for most practical applications (including this report) working with an infinite number of basis functions is not possible, so a truncated basis must be used. If we have prior information about what is being modelled then this can be used to select an optimal basis. For example, if it is known that the desired shape for a one-dimensional model is a polynomial, then an appropriate set of basis functions would be $1, x, x^2, ..., x^n$. Once a set of basis functions has been established, they can be used to model an element of the space they are a basis for using data sampled from that element. This is done by finding weights w_m for each basis function b_m such that the linear combination $\sum_m w_m b_m(x)$ best approximates the data (x, t). The model can be updated by changing the weights, but the basis functions remain fixed.

More formally, we start with a data set of N inputs $\mathbf{x} = (x_1, x_2, ..., x_N)^T$ and associated target values $\mathbf{t} = (t_1, t_2, ..., t_n)^T$. We also have M basis functions b_m, and define

$$b(x_n) = \begin{bmatrix} b_0(x_n) \\ b_1(x_n) \\ \vdots \\ b_{M-1}(x_n) \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_0(x_1) & b_1(x_1) & \dots & b_{M-1}(x_1) \\ b_0(x_2) & b_1(x_2) & \dots & b_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ b_0(x_N) & b_1(x_N) & \dots & b_{M-1}(x_N) \end{bmatrix}.$$

The aim is to find a vector of weights $\mathbf{w} = (w_1, ..., w_M)^T$ associated with each basis function such that $B\mathbf{w}$ best approximates $\mathbf{t}$. The associated *error function* is a measure of how close this approximation is. When using sum-squared error it is given by

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T b(x_n))^2 = \frac{1}{2}(\mathbf{t} - B\mathbf{w})^T(\mathbf{t} - B\mathbf{w})$$

3.5 Maximum Likelihood Estimation

Maximum likelihood estimation is a form of linear regression that seeks to find a set of weights $\mathbf{w}$ that maximise the likelihood of a target t given data x [26]. It assumes

that a target, t is then given by

$$t = y(x, \mathbf{w}) + \epsilon,$$

where $y(x, \mathbf{y})$ is deterministic and ϵ is a zero-mean Gaussian random variable with precision (inverse variance) β. Therefore the likelihood of a target t given data x is

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}(t|y(x, \mathbf{w}), \beta^{-1}).$$

This can be expanded for a set of N inputs, i.e. for $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ with associated target values $\mathbf{t}$ as defined in Section 3.4. However, this requires that the data is independent and identically distributed (iid). The process of generating points described in Section 4.1 will ensure that our points are iid, so we can use this method. The likelihood is given by

$$\begin{aligned} p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) &= \sum_{n=1}^{N} \mathcal{N}(t_n|y(x_n, \mathbf{w}), \beta^{-1}) \\ &= \sum_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T\mathbf{b}(x_n), \beta^{-1}). \end{aligned}$$

As a simplification, we take the logarithm of the likelihood. As the logarithm is a monotone function, a critical point of $\log(p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta))$ will correspond to a critical point of $p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$. Thus we can minimise $p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)$ by maximising $\log(p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta))$. The logarithm of the likelihood is

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E(\mathbf{w}).$$

Using the sum-squares error function $E_D(\mathbf{w})$ (detailed in Section 3.4), this is

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta \frac{1}{2}(\mathbf{t} - B\mathbf{w})^T(\mathbf{t} - B\mathbf{w}).$$

As β and N are constant, maximising $\log(p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta))$ is equivalent to minimising the error function E_D (see Section 3.4).

The directional derivative of the logarithm of the likelihood in direction ξ is

$$\mathcal{D} \ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)(\xi) = \beta \xi^T(B^T\mathbf{t} - B^T B\mathbf{w}).$$

We want the directional derivative to be 0 in all directions ξ, therefore

$$B^T\mathbf{t} - B^T B\mathbf{w} = 0,$$

so

$$\mathbf{w}_{ML} = (B^T B)^{-1} B^T \mathbf{t}.$$

Regularisation can be added to this solution in order to prevent over-fitting [26]. With regularisation constant λ, the error becomes

$$E = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w}).$$

The regulariser used in this report is the following quadratic regulariser:

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}.$$

Including this regularisation gives

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta(\frac{1}{2}(\mathbf{t} - B\mathbf{w})^T(\mathbf{t} - B\mathbf{w}) + \lambda\frac{1}{2}\mathbf{w}^T\mathbf{w}).$$

Then finding the directional derivative as above gives maximum likelihood solution

$$\mathbf{w}_{ML} = (\lambda I + B^T B)^{-1} B^T \mathbf{t}.$$

This Maximum Likelihood technique will be used to find the initial SBF model (see Section 4.5).

3.6 Gradient Descent

Gradient descent finds a local minimum of a given function near some starting point.

To update $\mathbf{w}$ using gradient descent, the following formula is used [26]:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_D(\mathbf{w}^{(\tau)}),$$

where η is the step size and E_D is the error function (see Section 3.4). The step size is a term that balances how much we trust $\mathbf{w}^{(\tau)}$, the old estimation of $\mathbf{w}$ compared to

the new data that is encoded in the gradient term $E_D(\mathbf{w}^{(\tau)})$. A small step size leads to slow convergence, however a large step size can lead to overshoot and therefore prevent convergence.

Differentiating E_D with respect to $\mathbf{w}$ yields the gradient expression

$$\nabla E_D(\mathbf{w}) = (-B)(\mathbf{t} - B\mathbf{w}) = B(B\mathbf{w} - \mathbf{t}).$$

Substituting this into the above formula gives

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta B(B\mathbf{w}^{(\tau)} - \mathbf{t}).$$

This gradient descent technique will be used to update the SBF model with new measurements $(\mathbf{x}, \mathbf{t})$ (see Section 4.6). Note that the new measurements $\mathbf{x}$ are used to calculate B (see Section 3.4).

3.7 Spherical Coordinates

The spherical coordinate system specifies the location of a point in $\mathbb{R}^3$ using two angles and a radius: θ, ϕ, r. The radius is the distance between the origin and the point, however there are a few different definitions of the angles in use in the literature. In this report, θ is the polar angle and ϕ is the azimuthal angle, as pictured in Figure 2. This report uses the angle ranges $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$. Note that this definition of polar and azimuthal angles must be made in relation to a given Cartesian coordinate system, as the angles are defined with respect to the x, y and z axes.
The spherical coordinate system and the Cartesian coordinate system can be converted between as follows:

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} & x &= r\sin\theta\cos\phi \\ \theta &= \cos^{-1}\frac{z}{r} & y &= r\sin\theta\sin\phi \\ \phi &= \tan^{-1}\frac{y}{x} & z &= r\cos\theta. \end{aligned}$$

Note that $\tan^{-1}$ must take into account the quadrant of (x, y). In MATLAB, the function `tan2` does this. Alternatively, `sph2cart` and `cart2sph` may be used. However,

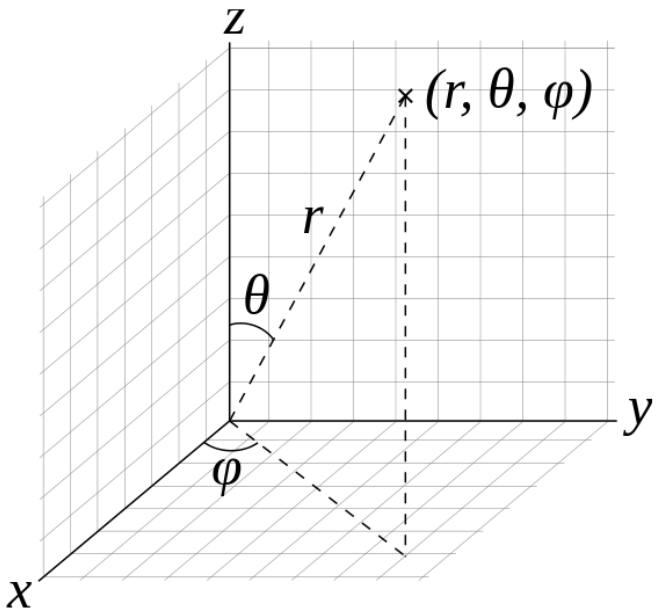


Figure 2: Spherical coordinate system – radial distance r, polar angle θ, azimuthal angle ϕ. Source: [2]

care must be taken as these functions use elevation instead of polar angle. Elevation is the angle from the xy-plane to the point as opposed to the angle from the z-axis.

3.8 Euler Angles

The existing literature on rotating spherical harmonics uses Euler angles, so this report will also use them. Euler angles are three angles α, β, γ that define a rotation in 3D space; each angle is a rotation about an axis. However, there are 12 different conventions for Euler angles, depending on which axes are chosen for each angle. The Wigner D-Matrix method described in Section 3.10 uses the ZYZ convention, so this report will also use it. Figure 3 shows how α, β and γ are defined using the ZYZ convention.

In order to use the transforming measurements method, the Euler angles need to be converted back into a rotation matrix.

A rotation of θ counter-clockwise about a coordinate axis can be expressed as the

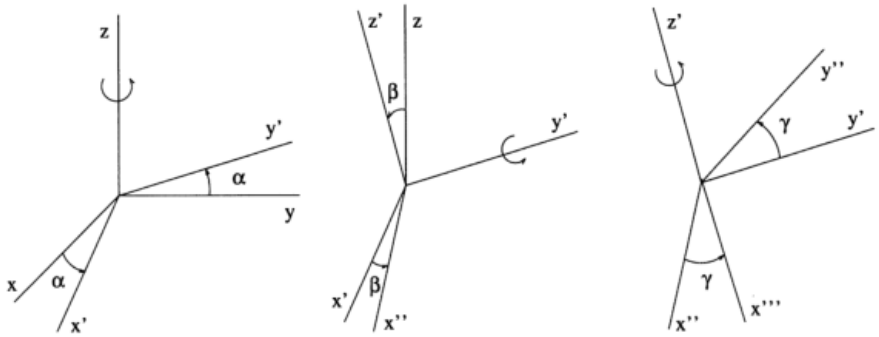


Figure 3: Euler angles – ZYZ convention. Source: [3]

following rotation matrices

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

These rotation matrices can be combined to form the overall rotation matrix R_{zyz}. To aid readability, let

$$c_a = \cos\alpha \qquad s_a = \sin\alpha$$

$$c_b = \cos\beta \qquad s_b = \sin\beta$$

$$c_g = \cos\gamma \qquad s_g = \sin\gamma.$$

Then

$$
\begin{aligned}
R_{zyz} &= R_z R_{y'} R_{z'} \\
&= \begin{bmatrix} c_a & -s_a & 0 \\ s_a & c_a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_b & 0 & s_b \\ 0 & 1 & 0 \\ -s_b & 0 & c_b \end{bmatrix} \begin{bmatrix} c_g & -s_g & 0 \\ s_g & c_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_a c_b & -s_a & c_a s_b \\ c_b s_a & c_a & s_a s_b \\ -s_b & 0 & c_b \end{bmatrix} \begin{bmatrix} c_g & -s_g & 0 \\ s_g & c_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_a c_b c_g - s_a s_g & -c_a c_b s_g - c_g s_a & c_a s_b \\ c_b c_g s_a + c_a s_g & c_a c_g - c_b s_a s_g & s_a s_b \\ -c_g s_b & s_b s_g & c_b \end{bmatrix}
\end{aligned}
$$

Note that this matrix pre-multiplies a column vector, i.e. the rotation of a point (x, y, z) is given by $R_{zyz}(x, y, z)^T$. z, y' and z' are as shown in Figure 3 (page 14).

3.9 Spherical Harmonics

The spherical harmonics are the angular portion of the solution to Laplace's equation in spherical coordinates [30]. They can be used for modelling 3D objects, as they provide an orthogonal basis for $L_2(\mathbb{S}^2)$.

Modelling using spherical harmonics is often used to compare objects [25, 18]. They are suitable for this because spherical harmonics only need a small set of homologous landmarks to register objects as similar to one another [25]. Wigner D-matrices are sometimes used to rotate the objects into a standard orientation so they can be compared [18].

Fourier spherical harmonic (SPHARM) functions are a related way to do modelling [25]. These functions, detailed in [4], expand an object surface into a series of spherical harmonic functions. Scale, translation and rotation invariant descriptors are obtained by rotating the parameter net and the object into standard positions. SPHARM func-

tions also allow for general simply-connected objects to be represented, whereas spherical harmonics used as basis functions for a depth map can only model star-shaped objects. SPHARM functions are good for modelling 3D objects when all of the measurements are available at the start, but are not suitable for updating a model with new data.

Spherical harmonics have been popular in physics and chemistry, they have also been used in geoscience and medical imaging and in computer graphics for applications such as environment map rendering and representing bidirectional reflectance distribution functions [15].

Spherical harmonic basis functions have global support, which prevents them from efficiently representing low-frequency signals [15]. This means that a large number of spherical harmonic basis functions are needed to model environments that are highly asymmetric. However they are relatively easy to understand and implement and have been relatively widely used for modelling. Thus they are worth investigating as a potential basis for modelling scene depth.

The complex spherical harmonic Y_{lm} can be written as [3]

$$Y_{lm}(\theta, \phi) = (-1)^m N_{lm} P_l^m(cos(\theta)) e^{im\phi},$$

where $l \in \{0, 1, 2, ..., n, ...\}$ and $m \in \{-l, -l+1, ..., 0, ...l-1, l\}$, N_{lm} is the normalisation factor

$$N_{lm} = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}}$$

and $P_l^m(x)$ are the Legendre associated polynomials, that is

$$P_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_l(x),$$

where $P_l(x)$ is the usual Legendre polynomial of degree l,

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l.$$

The real spherical harmonics can be found by combining complex conjugates, corresponding to opposite values of m. Then the real spherical harmonics S_{lm} can be defined as [3]

$$S_{lm}(\theta,\phi) = \begin{cases} N_{lm}P_l^m(\cos\theta)\sqrt{2}\cos(m\phi) & m > 0 \\ N_{lm}P_l^m(\cos\theta) & m = 0 \\ N_{l|m|}P_l^{|m|}(\cos\theta)\sqrt{2}\sin(|m|\theta) & m < 0 \end{cases}$$

3.10 Rotating Spherical Harmonic Basis Functions: Wigner D Functions

Wigner D functions, sometimes called Wigner D matrices, are well known in the literature as a way to rotate complex spherical harmonics. These functions are given by [3]

$$\begin{aligned} D^l_{mm'}(\alpha,\beta,\gamma) &= e^{-im\alpha}d^l_{mm'}(\beta)e^{-im'\gamma}, \\ d^l_{mm'}(\beta) &= (-1)^{m-m'}\sqrt{(l+m)!(l-m)!(l+m')!(l-m')!} \\ &\quad \times \sum_s (-1)^s \frac{\left(\cos\frac{\beta}{2}\right)^{2l-2s-m+m'}\left(\sin\frac{\beta}{2}\right)^{2s+m-m'}}{s!(l-m-s)!(l+m'-s)!(m-m'+s)!}, \end{aligned}$$

where s runs through all the integer values for which the factorials involved exist.

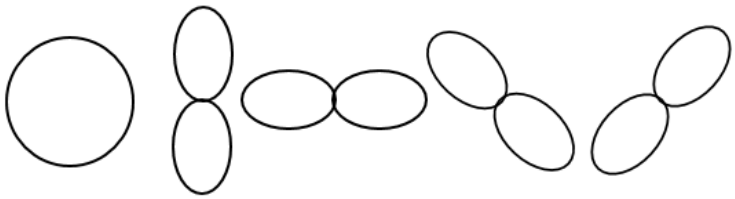
These D functions can be replaced with Δ functions that allow for the transformation of real spherical harmonics [3]:

$$\begin{aligned} \Delta^l_{mm'} &= \mathrm{sign}(m')\Phi_m(\alpha)\Phi_{m'}(\gamma)\frac{d^l_{|m'||m|} + (-1)^m d^l_{|m|(-|m|)}}{2} \\ &\quad - \mathrm{sign}(m)\Phi_{-m}(\alpha)\Phi_{-m'}(\gamma)\frac{d^l_{|m'||m|} - (-1)^m d^l_{|m|(-|m|)}}{2}, \end{aligned}$$

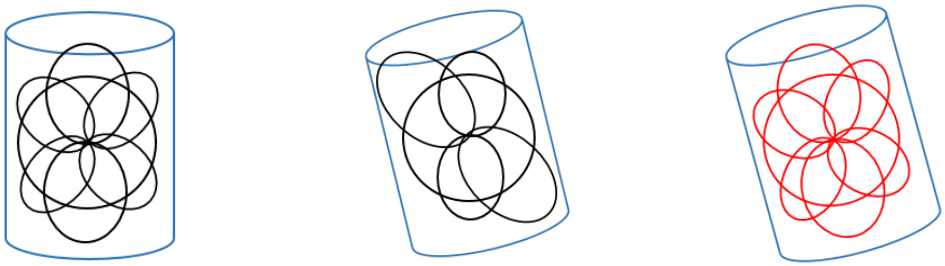
where $\mathrm{sign}(0) = 1$ and

$$\Phi_m(\phi) = \begin{cases} \sqrt{2}\cos(m\phi) & m > 0 \\ 1 & m = 0 \\ \sqrt{2}\sin(|m|\phi) & m < 0 \end{cases}$$

However, these equations are intended for use with rotated spherical harmonics i.e. they change the coefficients of an unrotated basis to the coefficients for a rotated basis. Whereas this report requires that the basis stays the same. (See Figure 4. We want a transform like (c), but we have one like (d)).



(a) Basis functions (untransformed)



(b) Initial model and scaled basis functions used to create it

(c) Transformed model with original basis functions and a new scaling

(d) Transformed model with new basis functions and original scaling

Figure 4: (Best viewed in colour). Difference between transforming coefficients of basis functions (b) and transforming the basis functions themselves (c). The original basis functions are shown in black, and the rotated ones in red. Note that all of the combinations of basis functions used to create the models (blue shapes) are very approximate.

Transforming the coefficients is also possible using D functions. If M_{lm} are the coefficients of the spherical harmonics in the old (unrotated) coordinate frame, then the new coefficients $M_{lm'}$ in the rotated coordinate frame are given by [7]

$$M_{lm'} = \sum_{m=-l}^{n} D^l_{mm'} M_{lm},$$

where

$$D^l_{mm'}(\alpha, \beta, \gamma) = e^{im\gamma} d^l_{mm'}(\beta) e^{im'\alpha}$$

and $d^l_{mm'}$ is the same as for when the D function is used to rotate spherical harmonics.

Propagating this change forwards to the real case results in the following equations

$$\Delta^l_{mm'} = \text{sign}(m')\Phi_{m'}(\alpha)\Phi_m(\gamma)\frac{d^l_{|m'||m|} + (-1)^m d^l_{|m|(-|m|)}}{2} - \text{sign}(m)\Phi_{-m'}(\alpha)\Phi_{-m}(\gamma)\frac{d^l_{|m'||m|} - (-1)^m d^l_{|m|(-|m|)}}{2},$$

where $\text{sign}(0) = 1$ and

$$\Phi_m(\phi) = \begin{cases} \sqrt{2}\cos(m\phi) & m > 0 \\ 1 & m = 0 \\ -\sqrt{2}\sin(|m|\phi) & m < 0 \end{cases}$$

Note that these rotation formulas are only exact when the order of the basis functions, l, approaches infinity. As finding and storing weights for an infinite number of basis functions is not practically possible, this means that any rotations performed on the weights of the SBFs incur an error.

3.11 Overview of Wavelets

The term wavelet encompasses a wide field of basis functions. However, all wavelets share common characteristics that can be useful in modelling. Wavelets can be defined on function spaces with a variety of domains, such as the real line, intervals, the square and the sphere. This report will be focusing on wavelets defined on the sphere. However a brief explanation of wavelets on the real line will be given, as that is where wavelets were originally defined and thus it is helpful in understanding the fundamental properties of wavelets.

The original definition of a wavelet is a function $\Psi \in L_2(\mathbb{R})$ such that the family of functions

$$\Psi_{j,m}(t) = 2^{j/2}\Psi(2^j t - m),$$

where j and m are arbitrary integers, is an orthonormal basis in the Hilbert space $L_2(\mathbb{R})$ [31]. Ψ is known as the mother wavelet.

Such wavelets are called first generation wavelets, whereas wavelets defined on a more general setting are referred to as second generation wavelets [27]. Unlike first generation wavelets, second generation wavelets are not necessarily translates and dilates of each other. However, they still preserve the following 4 key properties of first generation wavelets [27]:

1. Wavelets form a Riesz basis for a certain domain – for wavelets on the real line, this is $L_2(\mathbb{R})$, for spherical wavelets it is $L_2(\mathbb{S}^2)$. They also form an unconditional basis for a variety of normed function spaces $\mathcal{F}$. That is, for a wavelet basis denoted $\{\psi_{j,m}|j,m\}$, we can represent a general function $f \in \mathcal{F}$ as $f = \sum_{j,m} \gamma_{j,m}\psi_{j,m}$, with unconditional convergence in the norm of $\mathcal{F}$. Also, simple characterisations of the $\mathcal{F}$-norm of f in terms of the absolute value of its wavelet coefficients $\gamma_{j,m}$ exist.

2. One has explicit information on how to calculate the coefficients, i.e. on the coordinate functionals $\tilde{\psi}_{j,m}$ where $\gamma_{j,m} = \tilde{\psi}_{j,m}(f)$. The wavelets are either orthogonal, or the dual (biorthogonal) wavelets are known.

3. The wavelets and their duals are local in space and frequency.

4. Wavelets fit into the framework of multiresolution analysis (explained later in this section). This allows use of the fast wavelet transform, which takes linear time to go between samples of a function and its wavelet coefficients.

Note that these properties reference dual wavelets. For this report the wavelets and their duals coincide, so the report will not include much detail on dual wavelets in general. Interested readers are referred to Sweldens [27].

Wavelets do not only vary based on the domain of the function space they are a basis of. Even within spherical wavelets, there are a number of different types, such

as Haar, Lazy, Linear, Quadratic and Butterfly [24]. There can also be different ways to implement a single type of wavelet, which will lead to different properties. In the case of Haar wavelets, their scaling basis functions are constant on their support. So the key problem is choosing how to partition the sphere [15]. Both of the algorithms presented in [24] and [15], respectively, use triangular subdivisions, however the way they choose the positions of the vertices leads to different properties.

It is also possible to start with a relatively simple wavelet, such as the Haar wavelet, and then modify it to attain certain properties such as smoothness using a process called lifting [27]. This allows the wavelet's properties to be customised for an application.

Spherical wavelets are relatively new, and thus have not yet been used in as wide a variety of applications as spherical harmonics. Spherical wavelets have, however, been used relatively commonly for 3D model compression [13, 21, 11]. They have also been used as descriptors for 3D shapes [14] and watermarking for 3D meshes [12].

Wavelets are good for representing dissimilarities in a signal, due to their localisation in space and frequency [15]. This means that they are well suited for modelling most scenes both compactly and accurately.

Another aspect of wavelets that makes them good for modelling is their hierarchical structure, which combined with their afore-mentioned localisation leads to fast algorithms for basis projection and processing signals in the basis representation [15]. This hierarchical structure is very clear when considering the multiresolution analysis (MRA).

The vast majority of wavelets are constructed following a MRA [31]. This is another concept created for first generation wavelets that can be transferred to second generation wavelets, however modifications are required. The definition of the MRA for second

generation wavelets keeps most of the terminology and symbols of the first generation definition, however many of the meanings change [27]. The purposes of the parts is generally the same as their namesake, however the properties and implementation can be quite different. This report will only detail the MRA for second generation wavelets.

Let $L_2 = L_2(X, \Sigma, \mu)$ be a general function space, with $X \subset \mathbb{R}^n$ being the spatial domain, Σ a σ-algebra, and μ a nonatomic measure on Σ. Then a MRA $\mathbf{M}$ of L_2 is a sequence of closed subspaces $\mathbf{M} = \{V_j \subset L_2 | j \in \mathcal{J} \subset \mathbb{Z}\}$ such that [27]

1. $V_j \subset V_{j+1}$,
2. $\cup_{j\in\mathcal{J}} V_j$ is dense in L_2,
3. for each $j \in \mathcal{J}$, V_j has a Riesz basis given by scaling functions $\{\phi_{j,k} | k \in \mathcal{K}(j)\}$.

MRAs allow for the definition of approximation S and detail coefficients W [31]. The approximation coefficients at some level are the discrete approximation of the signal at that level. The detail coefficients are the coefficients of the wavelet basis functions.

The fast wavelet transform (FWT) is a way of finding these coefficients for each level of the MRA in linear time. The approximation coefficients at the lowest level are simply samples of the function. These approximation coefficients can then be combined to find the approximation and detail coefficients at the other levels, using the following deconstruction algorithm [31]

$$S_{j+1,m} = \frac{1}{\sqrt{2}} \sum_k c_k S_{j,2m+k} = \frac{1}{\sqrt{2}} \sum_k c_{k-2m} S_{j,k}$$
$$W_{j+1,m} = \frac{1}{\sqrt{2}} \sum_k b_k S_{j,2m+k} = \frac{1}{\sqrt{2}} \sum_k b_{k-2m} S_{j,k},$$

where c_k depends on the wavelet being used as the basis and $b_k = (-1)^k c_{N_k-1-k}$ (N_k is the number of scaling coefficients).

The inverse FWT allows the reconstruction of all of the approximation coefficients (including the original samples) from the approximation coefficients at the lowest level

and the detail coefficients, using the following reconstruction algorithm

$$S_{j-1,m} = \frac{1}{\sqrt{2}} \sum_k c_{n-2k} S_{j,k} + \frac{1}{\sqrt{2}} \sum_k b_{n-2k} W_{j,k}.$$

One limitation of the FWT is that the function samples generally have to follow some form of regular sampling scheme. Although it is possible to process irregular samples to allow them to work for wavelets that are designed to work with regular samples [5].

Also, the approximation of the underlying continuous signal $x(t)$ at a given level j can be found using a combination of the approximation and detail coefficients as follows [31]

$$x_j(t) = \sum_m S_{j,m} \phi_{j,m}(t) + \sum_m W_{j,m} \psi_{j,m}(t),$$

recall that $\phi_{j,m}(t)$ are the scaling functions and $\psi_{j,m}$ are the wavelet functions.

3.12 Haar Wavelets

Haar wavelets on the real line are the simplest example of an orthogonal wavelet [31]. The mother wavelet $\Psi(t)$ is one period of a block wave, and their scaling function $\phi(t)$ is constant on an interval. The equations are as follows [31]

$$\Psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2} \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & \text{elsewhere} \end{cases}$$

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & \text{elsewhere} \end{cases}$$

The corresponding FWT cofficients are $c_0 = c_1 = 1$.

The key property of Haar wavelets is that they divide the domain using a nested set of partitionings [27], the scaling functions are constant over their support and the wavelet basis functions associated with a partition are exclusively defined over the child partitions [15].

3.13 Haar Wavelets on the Sphere

There are multiple ways to find the required partitioning of the sphere, and thus multiple ways to define Haar wavelets on the sphere. This report will use the orthogonal and symmetric Haar wavelets (SOHO) defined in Lessig and Fiume [15].

The SOHO wavelet basis uses a novel partition scheme of spherical triangles $\mathcal{T} = \{T_{j,k} | j \in \mathcal{J}, k \in \mathcal{K}(j)\}$, as shown in Figure 5. The key part of this partitioning scheme is the placement of the vertices. They are chosen so that the areas of the three outer child triangles $T^k_{j+1,1}, T^k_{j+1,2}$ and $T^k_{j+1,3}$ are equal.

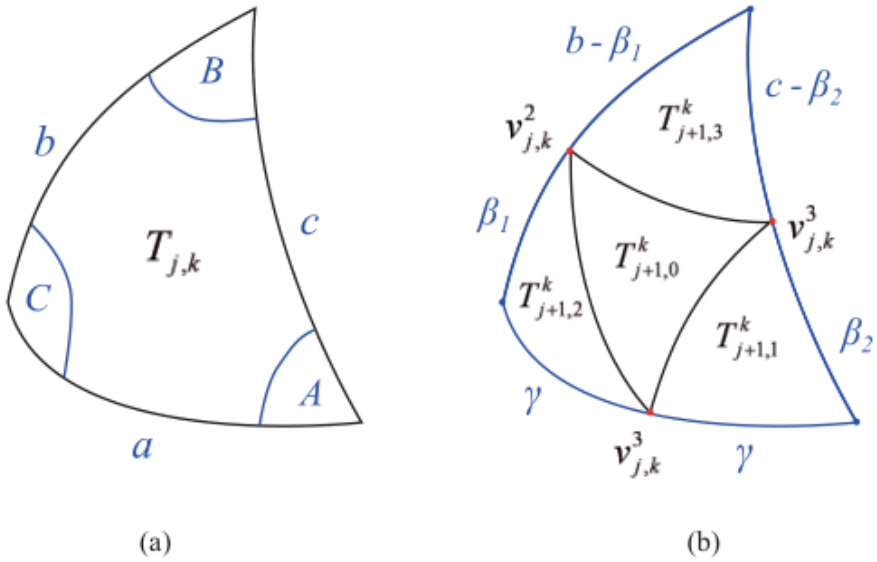


Figure 5: Subdivision of a spherical triangle. The labelling of the entities of a spherical triangle $T_{j,k}$ is shown in (a), the 4-fold subdivision yielding the child triangles in (b). Source: [15].

The area of a spherical triangle $T_{j,k}$ is denoted $\alpha_{j,k}$ and $\tau_{j,k}$ is the characteristic function of $T_{j,k}$.

The scaling basis functions $\phi_{j,k}$ are constant over their support $T_{j,k}$, with their value

given by a normalisation constant $\eta_{j,k}$:

$$\phi_{j,k} = \eta_{j,k}\tau_{j,k}.$$

We choose $\eta_{j,k} = 1/\sqrt{\alpha_{j,k}}$, as this combined with the disjoint nature of the $T_{j,k}$ for fixed j gives that the $\phi_{j,k}$ on the same level are orthogonal [15].

The wavelet basis functions $\psi^l_{j,k}$ are as follows

$$\begin{aligned}
\psi^0_{j,k} &= \frac{\Lambda_1}{\Lambda_0}\tau_0 + \frac{1}{\Lambda_1}((-2a+1)\tau_1 + a\tau_2 + a\tau_3) \\
\psi^1_{j,k} &= \frac{\Lambda_1}{\Lambda_0}\tau_0 + \frac{1}{\Lambda_1}(a\tau_1 + (-2a+1)\tau_2 + a\tau_3) \\
\psi^2_{j,k} &= \frac{\Lambda_1}{\Lambda_0}\tau_0 + \frac{1}{\Lambda_1}(a\tau_1 + a\tau_2 + (-2a+1)\tau_3),
\end{aligned}$$

where

$$\begin{aligned}
a &= \frac{\alpha_0 \pm \sqrt{\alpha_0^2 + 3\alpha_0\alpha_1}}{2\alpha_0} \\
\Lambda_l &= \sqrt{\alpha^k_{j+1,l}}.
\end{aligned}$$

3.14 Rotating Haar Wavelets on the Sphere

Spherical Haar Wavelets (SHWs) are described completely by the location and associated values of the vertices of the subdividing spherical triangles (see Section 3.13). In this report, the value associated to a vertex is the modelled depth in the direction of that vertex. However, in a different application these values could be something else, such as colour data.

The value of the vertices does not change with a rotation, therefore SHWs can be rotated by simply rotating the positions of all of the vertices. This has to be done for all of the subdividing triangles, not just those on a particular level. Note that this is actually changing the basis, not the coefficients, as the scaling and wavelet functions are defined with respect to the positions of the triangles. This is not, however, an issue for the update method used (see Section 4.9).

The simplicity of rotating the SHWs may raise the question of whether it is also simple to translate them. Unfortunately, this is not the case. Translating the vertices would mean that they are no longer located on the unit sphere. Thus the values should change. However translation moves the vertices to points that are not aligned with the direction of the original vertices (see the orange lines in Figure 6). So the translated model cannot be formed just by changing the values. One possible solution is to define new vertices, in the direction of the translated points (see the purple lines in Figure 6). However, it is not guaranteed that these new vertices will form spherical triangles that can be used to construct a SHW basis. Thus translating SHWs is considered outside the scope of this report.

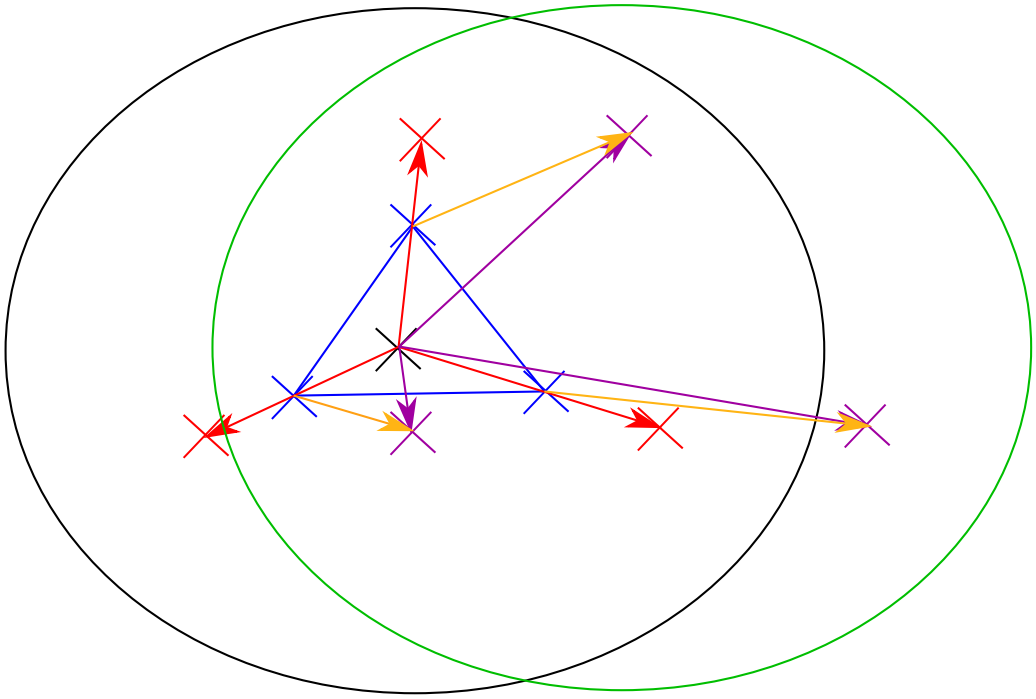


Figure 6: (Best viewed in colour). Difficulties in translating SHWs. The black point is the origin, with the unit sphere shown in black around it. The blue points are the vertices of a spherical triangle, with values equal to the magnitudes of the red arrows. This corresponds to a 3D position shown by the red points. The green circle is the translated unit sphere, and the corresponding translated red points are shown in purple. The orange lines show that the blue and purple points are no longer aligned radially. The purple lines show the new radial directions that pass through the purple points.

3.15 Transforming Measurements

Measurements with origin $O = [x, y, z]$ and rotation $[\alpha, \beta, \gamma]$ can be brought into alignment with a model with origin $[0, 0, 0]$ and rotation $[0, 0, 0]$ by first rotating the measurements by $[-\gamma, -\beta, -\alpha]$ and then translating by $[-x, -y, -z]$. Once the measurements are aligned with the model, they can be used to update it.

Unfortunately, the optimal way to update the model is not obvious. Spherical harmonics and spherical Haar wavelets can only be updated in a radial direction. That is, an arbitrary point $[\theta, \phi, r]$ can only be moved to $[\theta, \phi, r']$. This causes an issue, as the measurement data is a gradient in a radial direction for the rotated measurements. When the measurements are brought into alignment with the model, this gradient no longer points in a radial direction, and so cannot be used directly to update the model. This can be thought of as a point $[\theta, \phi, r]$ on the model being mapped to a point $[\theta', \phi', r']$ that does not lie along the same radial direction. There are two main ways of using this gradient to update the model. One is to move the point $[\theta, \phi, r]$ by $r' - r$ in the θ, ϕ direction. The second is to find the point of the model in the θ', ϕ' direction and update it to $[\theta', \phi', r']$ (see Figure 7, which shows the problem in 2D for simplicity).

A combination of these methods is also possible – moving a point in a direction between θ, ϕ and θ', ϕ'. The question of which of these methods is best is further complicated by the fact that updating the spherical harmonics model does not just move a single point; at the very least points in a small neighborhood around the target point will also be moved. Section 4.10 will discuss which method was chosen for this report.

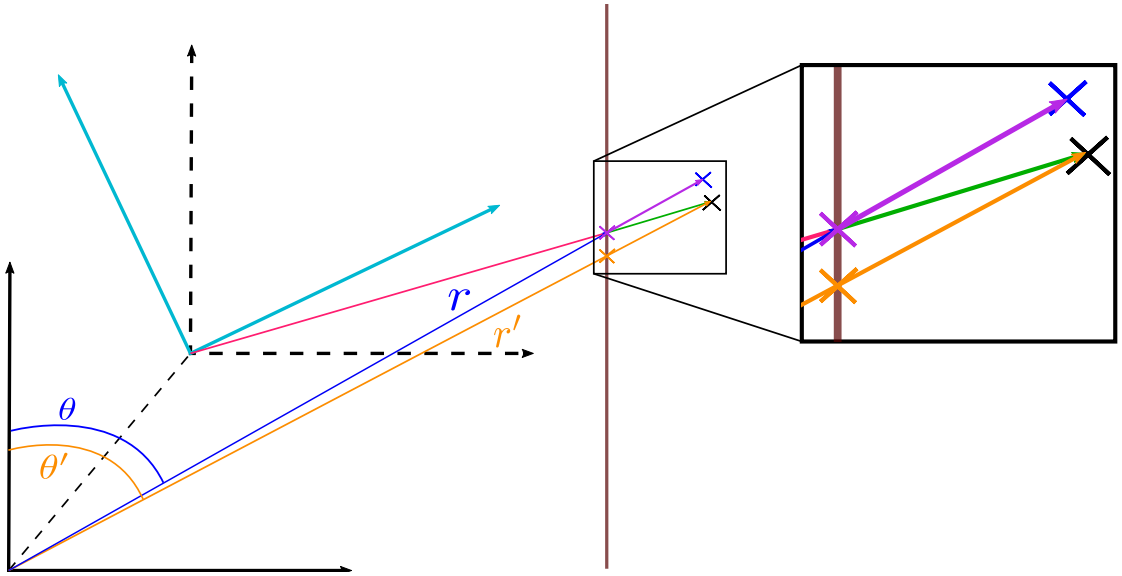


Figure 7: (Best viewed in colour). Two ways to use a transformed measurement to update a model in 2D. The space fixed frame is in black, with the body fixed frame in aqua. The model is the brown line to the right. The pink line shows the measurement, with the gradient shown in green. Ideally we would update the purple point ($[\theta, r]$) on the model to the black point ($[\theta', r]$). However we can only move the purple point to the blue point or the orange point to the black point. This might be clearer from looking at the zoomed inset.

4 Problem Set-up

This report aims to evaluate both methods of updating spherical harmonic basis functions (SBFs) and spherical Haar wavelets (SHWs), that is transforming the models or the measurements (see Sections 3.10, 3.14 and 3.15) relative to each other, and to compare SBFs and SHWs as basis functions. The code created for this report will generate a set of base, measurement and testing points. It will then find an initial model using the base points, which will be updated using the measurements. The error after each update will be found by comparing the model to the testing points.

In particular, for SBFs the initial model will be found using maximum likelihood estimation for least squares with regression (see Section 3.5). The model will be updated using gradient descent (see Section 3.6). For SHWs, the initial model will be found

using the FWT, which will also be used for updating the model (see Sections 3.11 and 3.13). For both SBFs and SHWs, the points will be chosen based on a simulated trajectory, given by a Fibonacci spiral, which changes the field of view (FOV). The measurements will then be selected in accordance with this new FOV.

4.1 Generating Points for Spherical Harmonic Model

The points need to be generated in spherical coordinates (see Section 3.7). That is, a θ, ϕ and corresponding r value. The first step in generating the points is to decide which shape to sample the points from. The shape used for this report is a cylinder, chosen due to its simplicity. The height H_c and radius R_c for the cylinder are both variable.

There are three types of points that must be generated: "base", "measurement" and "test" points. The "base" points are used to create the initial model (see Section 4.5). The "measurement" points are then used to update the model (see Section 4.6) - although note that multiple sets of measurement points must be generated (see Section 4.4). The "test" points are meant to represent the true shape – the model is compared with them in order to see how well it approximates the shape.

The "base" and "test" points are both generated for an upright cylinder. The "measurement" points are generated by first generating points for an upright cylinder and then rotating and/or translating those points according to given parameters. Therefore a way to sample points from an upright cylinder is required. Note that the sampling must be uniformly random, as spherical harmonics are extremely sensitive to symmetry in data.

The sampling is performed by splitting the cylinder into 3 parts: the top disk, the bottom disk and the mantle. The mantle can be thought of as a rectangle with one edge joined to the opposite edge. To sample uniformly from the disk, a random value for the radius R and an angle a need to be chosen. $R \in [0, R_c^2]$ and $a \in [0, 2\pi]$. Then

uniformly sampled Cartesian coordinates are

$$x = \sqrt{R}\cos(a), \qquad y = \sqrt{R}\sin(a).$$

This can then be converted into spherical coordinates using $z = H_c/2$ for the top disk and $z = H_c/2$ for the bottom disk (see Section 3.7).

Uniformly sampling from a rectangle is done by choosing a random $x_r \in [0, 2\pi R_c]$ and $y_r \in [-H_c/2, H_c/2]$. The spherical coordinates are found directly from x_r and y_r, instead of first converting them to the Cartesian coordinate system. x_r already samples a point on the circumference, so ϕ can be obtained by dividing by R_c. θ and r can be found using trigonometry (see Figure 8). This gives

$$\theta = \pi/2 - \tan^{-1}(y_r/R_c)$$
$$\phi = x_r/R_c$$
$$r = R_c/\sin(\theta).$$

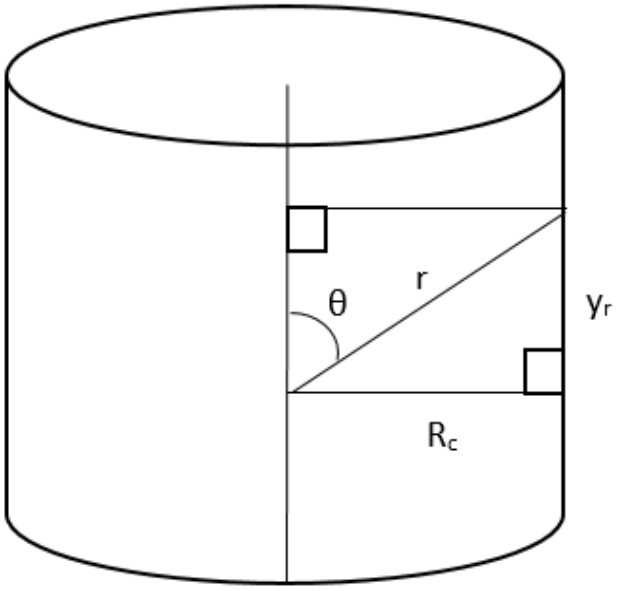


Figure 8: Sampling points from a cylinder – finding θ and r from y_r and R_c

Now a way of transforming the points to obtain the "measurement" points is needed. This is done by first transforming the points into Cartesian coordinates, then rotating

them using a rotation matrix (see Section 3.8 for the conversion from Euler angles to a rotation matrix). The points can then be translated by subtracting the desired coordinate shift and then transformed back into spherical coordinates.

If there is measurement uncertainty (u), it is included here. To do this, random numbers between $-u/2$ and $u/2$ are chosen, and then one of these numbers is added to each radius of the spherical coordinates we found earlier.

In order to simulate starting with an imperfect model, the "base" points are found with a smaller cylinder radius and height than the "measurement" and "test" points.

In order to get the measurement gradients, points for the model and environment are found with respect to the space fixed frame (no rotation or translation). These points are transformed to be in relation to the body fixed frame. Directions are then chosen for which to get measurement gradients (note that these must be uniform in order to get good updates). The closest point in the transformed model point cloud and the transformed environment point cloud are then found, and the gradient between them is returned.

This method is necessary for translations to work. Points on the model can only be found with respect to the space fixed frame. So finding the point on the model in a certain direction defined with respect to the body fixed frame is difficult (see literature on ray tracing such as Glassner's introductory text [8]). Thus the point cloud method is used. This method should be reasonably accurate when the point clouds are dense.

4.2 Generating Points for Wavelet Model

Generating points for the Spherical Haar Wavelet (SHW) model follows mostly the same process as for the Spherical Harmonic model (see Section 4.6).

However, one limitation of the SHW basis functions used in this report is that they

require a very specific sampling scheme. In order for them to work properly the sample points must form the vertices of the triangles described in Section 3.13. That is, they form groups of four triangles where the outer three triangles all have the same area.

The sampling scheme is implemented as follows [15]. First, the geodesic bisector $v^1_{j,k}$ is chosen. Then the positions of the other two vertices $v^2_{j,k}$ and $v^3_{j,k}$ are chosen to satisfy the following system of equations

$$\begin{aligned}
\cot\left(\frac{E}{2}\right) &= \cot(C) + \frac{\cot(\beta_1/2)\cot(\gamma/2)}{\sin(C)} \\
\cot\left(\frac{E}{2}\right) &= \cot(B) + \frac{\cot(\beta_2/2)\cot(\gamma/2)}{\sin(B)} \\
\cot\left(\frac{E}{2}\right) &= \cot(A) + \frac{\cot(b/2-\beta_1/2)\cot(c/2-\beta_2/2)}{\sin(A)},
\end{aligned}$$

where $\beta_1 = v^2_{j,k}$, $\beta_2 = v^3_{j,k}$, E denotes the spherical excess of the three outer child domains and the other variables are as shown in Figure 5 (page 24).

Thus in order to allow for arbitrary points to be sampled the environment is given as a function that takes θ and ϕ as inputs and returns the radius r in that direction. For a cylinder with radius R_c and height H_c, the function f_c is

$$f_c(\theta,\phi) = \left| \begin{cases} \frac{H_c}{2*\cos(\theta)} & \text{if } \theta \le \tan^{-1}\left(2\frac{R_c}{H_c}\right) \text{ or } \theta \ge \pi - \tan^{-1}\left(2\frac{R_c}{H_c}\right) \\ \frac{R_c}{\cos(\pi/2-\theta)} & \text{if } \theta > \tan^{-1}(2\frac{R_c}{H_c}) \text{ and } \theta \le \frac{\pi}{2} \\ \frac{R_c}{\cos(\theta-\pi/2)} & \text{if } \theta > \frac{\pi}{2} \text{ and } \theta < \pi - \tan^{-1}\left(2\frac{R_c}{H_c}\right) \end{cases} \right|.$$

This function was found based on the sampling scheme described in Section 4.1.

4.3 Setting the Field of View

The procedure described in Sections 4.1 and 4.2 generate points over the whole scene. However, physical cameras do not take a 360° photo of everything around them – they are limited by their FOV.

To implement this, the points were first generated over the whole scene as described

in Section 4.1 and 4.2. Points were then filtered so that only points whose θ value was within the vertical FOV and whose ϕ value was within the horizontal FOV were kept. The center of the cameras was defined at $\theta = \pi/2$ (i.e. on the horizontal plane) and $\phi = \pi$. $\phi = \pi$ was chosen instead of $\phi = 0$ to simplify the filtering.

4.4 Choosing measurement locations

The new measurements are chosen such that the trajectory follows a Fibonacci spiral. The latitude and longitude of the i^{th} point in such a trajectory are given by [10]

$$\text{lat}_i = \sin^{-1}\left(\frac{2i}{2N+1}\right)$$
$$\text{lon}_i = 2\pi i \Phi^{-1},$$

where $\Phi = 1 + \Phi^{-1} = (1+\sqrt{5})/2$ and i ranges from $-N$ to N. N is chosen so that the desired number of points is produced.

This corresponds to Euler angles $[(\text{lon} + \pi) \mod 2\pi, |\text{lat} - \pi/2|, 0]$. Also, note that no points are produced at the poles.

These Euler angles determine the rotation between each set of measurements, however a translation is also required. For simplicity, translation occurs in a loop, with the i^{th} position of the center of the shape given by

$$x_i = r\cos(a_i), \qquad y_i = r\sin(a_i), \qquad z_i = 0,$$

where $a_i = 2\pi i/8$.

4.5 Creating the Initial Spherical Harmonic Model

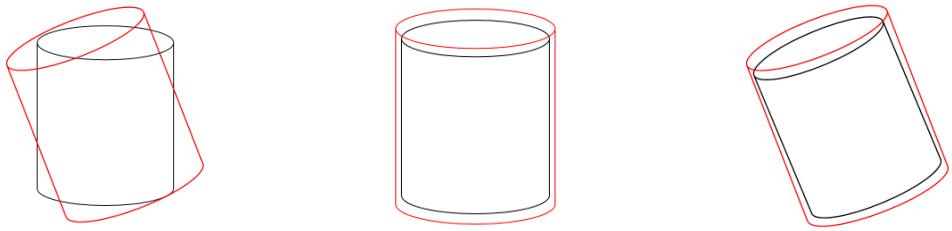
The model used for the spherical harmonic basis functions (SBFs) is a set of basis functions with associated weights, as described in Section 3.4. In the case of this report, each data-point x_n is a row vector containing a θ value in the first column and a ϕ value in the second. The associated target t_n is then the corresponding radius (r

value). The $M = (k+1)^2$ basis functions are spherical harmonics of up to order k.

The initial weights (and thus the initial model) are found using maximum likelihood estimation, as described in Section 3.5. The data used in this step is the "base" data, as described in Section 4.1. As this is for the initial model, the FOV covers the entire environment (i.e. the vertical FOV is chosen as π and the horizontal FOV as 2π).

4.6 Updating the Spherical Harmonic Model

The SBF model is then updated based on new data, the "measurement" data described in Sections 4.1 and 4.4. There are two different ways the model can be updated. The first is transforming the weights in order to make the current model align with the new measurements (see Section 3.10) and then using the "measurement" data directly to update it. The second is transforming the measurement data to align with the current model (see Section 3.15) and then using this aligned measurement data to update the current model. Figure 9 gives an example of each of these methods.



(a) Initial model with unaligned measurements

(b) Transform the measurements so they are aligned with the model

(c) Transform the model so it is aligned with the measurements

Figure 9: Two ways to update a model with new unaligned measurements. The model is shown in black, and the measurements are shown in red.

Once the model and "measurement" data are aligned, gradient descent is then used to update the weights of the model. Note that the formula for gradient descent provided in Section 3.6 uses the actual measurements, but we only have the gradients (that is, the amount that a point on the model needs to move in a radial direction to match the

measurement). However, this gradient is the same as $B\mathbf{w} - \mathbf{t}$, so if we let the gradient be $\Delta\mathbf{t}$, then we can use the following formula for gradient descent

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta B \Delta\mathbf{t}$$

When using gradient descent it is very important to choose the correct step size. If the step size is too large, it can result in overshoot, if it is too small then the method will take too long to converge. A step size of 10^{-3} was chosen for this report, using trial and error.

When the weights are transformed, they need to be transformed back after the update step in order to allow for comparisons with the other method and the "test" data. However, as some error is introduced with every transformation, the transformed weights are used for the next update step. This requires that the next transform is in terms of the previous transform, instead of in terms of the initial state of the model. How to do this is detailed in Section 4.7.

4.7 Getting Euler Angles in Terms of a Rotated Frame

The aim of this section is to detail how to transform Euler angles $[\alpha, \beta, \gamma]$ defined with respect to an original frame into Euler angles $[\alpha', \beta', \gamma']$ defined with respect to a rotated frame (described in Euler angles $[\alpha_0, \beta_0, \gamma_0]$) of that frame. (See Section 3.8 for an overview of Euler angles).

This can be done using rotation matrices and then transforming them back into Euler angles. Recall that the rotation matrix corresponding to Euler angles $[\alpha, \beta, \gamma]$ is given by

$$R_{zyz} = R_z R_{y'} R_{z'} = \begin{bmatrix} c_a c_b c_g - s_a s_g & -c_a c_b s_g - c_g s_a & c_a s_b \\ c_b c_g s_a + c_a s_g & c_a c_g - c_b s_a s_g & s_a s_b \\ -c_g s_b & s_b s_g & c_b \end{bmatrix}$$

For the initial rotation, this matrix corresponds to angles $[\alpha_0, \beta_0, \gamma_0]$. So for the initial rotation matrix R_{zyz0} we will write

$$R_{zyz0} = \begin{bmatrix} c_{a0}c_{b0}c_{g0} - s_{a0}s_{g0} & -c_{a0}c_{b0}s_{g0} - c_{g0}s_{a0} & c_{a0}s_{b0} \\ c_{b0}c_{g0}s_{a0} + c_{a0}s_{g0} & c_{a0}c_{g0} - c_{b0}s_{a0}s_{g0} & s_{a0}s_{b0} \\ -c_{g0}s_{b0} & s_{b0}s_{g0} & c_{b0} \end{bmatrix}$$

The inverse of a rotation matrix is its transpose, so the inverse rotation is

$$R_{zyz0}^{-1} = R_{zyz0}^{T} = \begin{bmatrix} c_{a0}c_{b0}c_{g0} - s_{a0}s_{g0} & c_{b0}c_{g0}s_{a0} + c_{a0}s_{g0} & -c_{g0}s_{b0} \\ -c_{a0}c_{b0}s_{g0} - c_{g0}s_{a0} & c_{a0}c_{g0} - c_{b0}s_{a0}s_{g0} & s_{b0}s_{g0} \\ c_{a0}s_{b0} & s_{a0}s_{b0} & c_{b0}. \end{bmatrix}$$

Therefore the new rotation matrix R_{zyzn} with respect to $[\alpha_0, \beta_0, \gamma_0]$ is given by

$$\begin{aligned} R_{zyzn} &= R_{zyz}R_{zyz0}^{-1} \\ &= \begin{bmatrix} c_a c_b c_g - s_a s_g & -c_a c_b s_g - c_g s_a & c_a s_b \\ c_b c_g s_a + c_a s_g & c_a c_g - c_b s_a s_g & s_a s_b \\ -c_g s_b & s_b s_g & c_b \end{bmatrix} \begin{bmatrix} c_{a0}c_{b0}c_{g0} - s_{a0}s_{g0} & c_{b0}c_{g0}s_{a0} + c_{a0}s_{g0} & -c_{g0}s_{b0} \\ -c_{a0}c_{b0}s_{g0} - c_{g0}s_{a0} & c_{a0}c_{g0} - c_{b0}s_{a0}s_{g0} & s_{b0}s_{g0} \\ c_{a0}s_{b0} & s_{a0}s_{b0} & c_{b0} \end{bmatrix} \end{aligned}$$

We can work out this matrix product using MATLAB, so for simplicity let

$$\begin{aligned} R_{zyzn} &= \begin{bmatrix} c_a c_b c_g - s_a s_g & -c_a c_b s_g - c_g s_a & c_a s_b \\ c_b c_g s_a + c_a s_g & c_a c_g - c_b s_a s_g & s_a s_b \\ -c_g s_b & s_b s_g & c_b \end{bmatrix} \begin{bmatrix} c_{a0}c_{b0}c_{g0} - s_{a0}s_{g0} & c_{b0}c_{g0}s_{a0} + c_{a0}s_{g0} & -c_{g0}s_{b0} \\ -c_{a0}c_{b0}s_{g0} - c_{g0}s_{a0} & c_{a0}c_{g0} - c_{b0}s_{a0}s_{g0} & s_{b0}s_{g0} \\ c_{a0}s_{b0} & s_{a0}s_{b0} & c_{b0} \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{aligned}$$

We then set this equal to

$$\begin{bmatrix} c_{a'}c_{b'}c_{g'} - s_{a'}s_{g'} & -c_{a'}c_{b'}s_{g'} - c_{g'}s_{a'} & c_{a'}s_{b'} \\ c_{b'}c_{g'}s_{a'} + c_{a'}s_{g'} & c_{a'}c_{g'} - c_{b'}s_{a'}s_{g'} & s_{a'}s_{b'} \\ -c_{g'}s_{b'} & s_{b'}s_{g'} & c_{b'} \end{bmatrix}$$

This gives [6, p. 45]

$$\begin{aligned} \beta' &= \operatorname{atan2}(\sqrt{r_{31}^2 + r_{32}^2}, r_{33}) \\ \alpha' &= \operatorname{atan2}(r_{23}/\sin\beta', r_{13}/\sin\beta') \\ \gamma' &= \operatorname{atan2}(r_{32}/\sin\beta', -r_{31}/\sin\beta'). \end{aligned}$$

Note that β' must be calculated first.

4.8 Creating Initial Wavelet Model

The initial wavelet model is found using the FWT (the FWT is discussed in Section 3.11). When considering only one partition $T_{j,k}$ (see Section 3.13), the analysis and synthesis steps of the FWT can be expressed as compact matrix-vector products, with analysis matrix $A_{j,k}$ and synthesis matrix $S_{j,k}$. As the SHWs used in this report are orthonormal, $A_{j,k} = S_{j,k}^T$ [15]. An analysis step is given by

$$\begin{bmatrix} \lambda_{j,0} \\ \lambda_{j,1} \\ \lambda_{j,2} \\ \lambda_{j,3} \end{bmatrix} = \begin{bmatrix} h_{j,k,0} & h_{j,k,1} & h_{j,k,2} & h_{j,k,3} \\ h_{j,k,0} & h_{j,k,1} & h_{j,k,2} & h_{j,k,3} \\ h_{j,k,0} & h_{j,k,1} & h_{j,k,2} & h_{j,k,3} \\ h_{j,k,0} & h_{j,k,1} & h_{j,k,2} & h_{j,k,3} \end{bmatrix} \begin{bmatrix} \lambda_{j+1,0} \\ \lambda_{j+1,1} \\ \lambda_{j+1,2} \\ \lambda_{j+1,3} \end{bmatrix}$$

and

$$\begin{bmatrix} \gamma_{j,0} \\ \gamma_{j,1} \\ \gamma_{j,2} \end{bmatrix} = \begin{bmatrix} g_{j,0,0} & g_{j,0,1} & g_{j,0,2} & g_{j,0,3} \\ g_{j,1,0} & g_{j,1,1} & g_{j,1,2} & g_{j,1,3} \\ g_{j,2,0} & g_{j,2,1} & g_{j,2,2} & g_{j,2,3} \end{bmatrix} \begin{bmatrix} \lambda_{j+1,0} \\ \lambda_{j+1,1} \\ \lambda_{j+1,2} \\ \lambda_{j+1,3} \end{bmatrix}$$

and a synthesis step is given by

$$\begin{bmatrix} \lambda_{j+1,0} \\ \lambda_{j+1,1} \\ \lambda_{j+1,2} \\ \lambda_{j+1,3} \end{bmatrix} = \begin{bmatrix} h_{j,k,0} & g_{j,0,0} & g_{j,1,0} & g_{j,2,0} \\ h_{j,k,1} & g_{j,0,1} & g_{j,1,1} & g_{j,2,1} \\ h_{j,k,2} & g_{j,0,2} & g_{j,1,2} & g_{j,2,2} \\ h_{j,k,3} & g_{j,0,3} & g_{j,1,3} & g_{j,2,3} \end{bmatrix} \begin{bmatrix} \lambda_{j,k} \\ \gamma_{j,0} \\ \gamma_{j,1} \\ \gamma_{j,2} \end{bmatrix}$$

where $\lambda_{j,k}$ are the scaling function coefficients and $\gamma_{j,m}$ are the basis function coeffi-

cients. Here, h_l and g_l^m are the filter coefficients. The values are given by [15]

$$h_{j,k,l} = \frac{\sqrt{\alpha_{j+1,l}^k}}{\sqrt{\alpha_{j,k}}}$$

$$g_{j,0,0} = -\frac{\sqrt{\alpha_1}}{\sqrt{\alpha_0}} \qquad g_{j,1,0} = g_{j,0,0} \qquad g_{j,2,0} = g_{j,0,0}$$

$$g_{j,0,1} = -2a + 1 \qquad g_{j,1,1} = a \qquad g_{j,2,1} = a$$

$$g_{j,0,2} = a \qquad g_{j,1,2} = g_{j,0,1} \qquad g_{j,2,2} = a$$

$$g_{j,0,3} = a \qquad g_{j,1,3} = a \qquad g_{j,2,3} = g_{j,0,1}$$

The synthesis step gives the initial model; only the coarsest level of the scaling function coefficients needs to be remembered, along with all of the basis function coefficients. After these coefficients have been found they can then be filtered, retaining only the most significant coefficients and setting the others to 0. This saves time and space, but obviously prevents perfect reconstruction during the synthesis phase.

4.9 Updating Wavelet Model

Due to the four different types of coefficients and the localised nature of the Spherical Haar Wavelet (SHW) functions, the gradient descent algorithm described in Section 3.6 is not used. However the method that is used implements a similar idea.

Each SHW function is zero outside of a specific spherical triangle (see Section 3.13). Thus updating with a new measurement point can be done by adjusting the wavelet coefficients of each spherical triangle that the point lies in. First, the point is mapped to the unit sphere, with its distance out from the the model in the radial direction stored as an associated gradient. Next, the spherical triangle on the highest level that this point lies in is found and its wavelet coefficients adjusted accordingly (how exactly this is done will be discussed later). Then the process is repeated with the children of this spherical triangle. This process continues until the second lowest level is reached. Note that the lowest level cannot be used with the current implementation, as the SHW

functions for a spherical triangle are defined with respect to the children of that triangle and these are not found for the lowest level. The scaling coefficients are not altered.

The adjustment of the wavelet coefficients for $T_{j,k}$ is as follows:

$$\psi^0_{j,k} = \psi^0_{j,k} + \eta_n \eta_l \nabla r(\tau_0/3 + \tau_1)$$
$$\psi^1_{j,k} = \psi^1_{j,k} + \eta_n \eta_l \nabla r(\tau_0/3 + \tau_2)$$
$$\psi^2_{j,k} = \psi^2_{j,k} + \eta_n \eta_l \nabla r(\tau_0/3 + \tau_3),$$

where η_n is a scaling constant that is used across all levels, and η_l is a scaling constant that varies depending on the level of the triangle. η_n depends on the number of measurements, it is chosen as $\frac{\text{lr}}{n_{meas}}$. Where lr is the step size, as with gradient descent (see Section 3.6). It is found using trial and error. n_{meas} is the number of measurements, this term is necessary as the measurement gradients are found with respect to the original model, but the update only does one point at a time. So this term counteracts the way that updating one point changes the gradient for the other points. This still introduces some error, as each point moves the model by a different amount in different directions (ways to fix this will be discussed later).

η_l takes into account the fact that the wavelet functions are localised, and have differently sized supports in each level. As this method updates one point at a time, it makes sense to move the smaller triangles more than the bigger ones. Thus η_l is chosen as $\frac{j^2}{j^3_{max}}$, where j is the current level and j_{max} is the lowest level. Note that the denominator has a power one higher than the numerator. This accounts for the fact that the wavelet coefficients are being updated on each level.

There are ways to combat the error introduced by updating one point at a time. The most obvious is to use the un-updated model and the gradient to find the points which the model would be moved to. Then when adding a new point the gradient can be recalculated for that point based on the current model. This was judged to have too large a time cost to be worth implementing at this stage, however it would be worth investigating later to see how much it affects the convergence characteristics

of the model. Another possibility is finding a way to use this method with batch updates, however this would be difficult due to the fact that triangle which a single point lies in needs to be located across different levels. So working out which triangles to scan on the lower levels is a bit more involved than simply taking the children of the current triangle. Also, the batch method would probably work by taking into account how many points lie in each triangle. Thus if points are located one by one, wavelet coefficients cannot be updated immediately. Instead, there would need to be some way of storing which points (and the associated gradients) lie in which triangle and then updating using this information (although it should still be possible to update one level at a time).

4.10 Choosing Moved Measurements Method Implementation

Recall that there are two main ways of updating the model with a new transformed point $[\theta', \phi', r']$ (see section 3.15). The first is to take the point $[\theta, \phi, r]$ that was in alignment with the untransformed measurement point and move it to $[\theta, \phi, r']$. The second is to find the point of the model in the θ', ϕ' direction and update it to $[\theta', \phi', r']$. A combination of these methods is also possible.

Intuitively, if the environment is locally similar (i.e. given a point $[\theta, \phi, r]$ there will be similar radius values to r for directions close to $[\theta, \phi]$) then moving the point $[\theta, \phi, r]$ to $[\theta, \phi, r']$ will be best. This is because the radius value should be reasonably close for the θ, ϕ and θ', ϕ' directions. Likewise, if the model is locally similar then moving the point of the model in the θ', ϕ' direction to $[\theta', \phi', r']$ will be better.

Note that both of these methods will work well when the θ, ϕ direction is close to the θ', ϕ' direction. However, as the amount the point is moved increases, the effect of a difference in directions is magnified.

This report moves the point in the θ', ϕ' direction to $[\theta', \phi', r']$. This was a somewhat arbitrary choice, decided mainly by ease of implementation. A further investigation of the benefits of each of these methods could be a good extension to this report.

5 Simulation Study

In this section, the moved model method and moved measurements methods, using both a spherical harmonic basis (SBF) and a spherical Haar wavelet (SHW) basis, are compared using MATLAB. The scene itself is simulated. This allows for test values to be generated, reduces possible sources of error and allows for the effect of different parameter values to be tested easily. The results in this section use the parameter values in Table 1 for both types of basis functions, with SBF-specific parameters given in Table 2 and SHW-specific parameters given in Table 3, except where otherwise stated. The full code of the simulation study is provided in Appendices 8.1 and 8.2.

Name	Value	Description
N	20	N for Fibonacci spiral (see Section 4.4)
nmeas	2000	Number of measurement points for each camera movement
nbase	10000	Number of points used to find initial model
ntest	10000	Number of test points
R	1.5	Radius of actual cylinder (used for test and measurement)
h	3	Height of actual cylinder (used for test and measurement)
R0	1	Radius of initial cylinder (used for initial model)
h0	2	Height of initial cylinder (used for initial model)
fovt	π	Vertical FOV
fovp	2π	Horizontal FOV
scale	1	Scale for gradient (see Section 3.2)
uncer	0	Uncertainty in measurements (standard deviation)

Table 1: Parameter values used in simulation study, except where otherwise stated

Name	Value	Description
n	40	Highest order of SBFs
lam	0.1	regularisation constant (see Section 3.5)
lr	0.001	step size (see Sections 3.6 and 4.6)

Table 2: Parameter values used in simulation study for spherical harmonic (SBF) basis, except where otherwise stated

Name	Value	Description
level	4	highest level for SHWs
nc	1024	number of non-zero coefficients for SHWs
lr	100	step size (see Section 4.9)

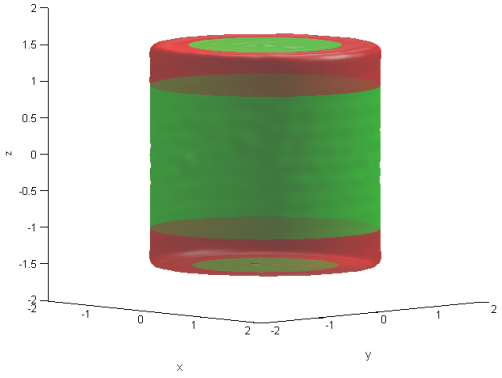
Table 3: Parameter values used in simulation study for spherical Haar wavelet (SHW) basis, except where otherwise stated

5.1 Resultant Observer Error

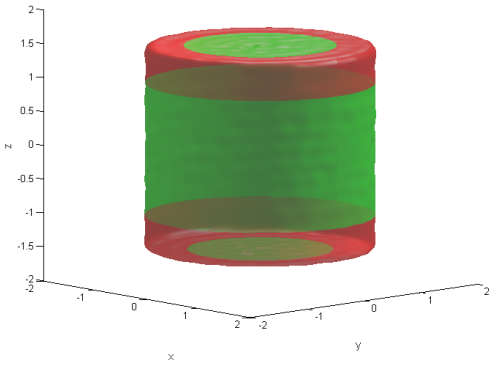
This section investigates the convergence rate with the parameters listed in Tables 1, 2 and 3. Note that these parameters are not realistic. This section focuses purely on comparing the two methods and sets of basis functions under ideal circumstances. The effect of error in the measurements, introducing a realistic FOV for the camera and other parameter changes will be investigated for spherical harmonics (SBFs) in Section 6, however they could not be represented for spherical Haar wavelets (SHWs) due to time constraints.

There was difficulty when choosing the step size for the SHWs, which lead to insufficient time to run a simulation until convergence. With $lr = 500$, the error appeared to decrease normally (albeit slowly), until it reached about 0.29, after which it began increasing. This indicates that the step size is too high (see Section 14). A step size of 100 is used in this section, but it is only a guess based on some trial and error for values below 500. There was not enough time to run the simulation until convergence.

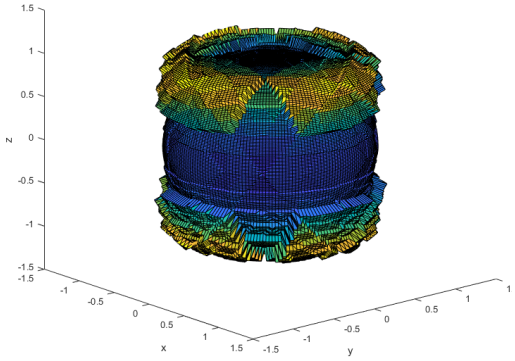
Thus it is yet to be confirmed whether the update method actually works as intended.



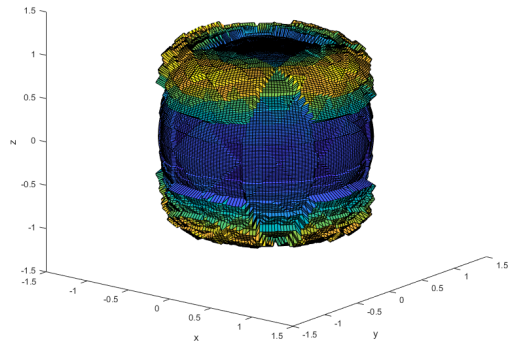
(a) Model found using moved model method with SBF



(b) Model found using moved measurements method SBF



(c) Model found using moved model method with SHW basis



(d) Model found using moved measurements method SHW basis

Figure 10: Models obtained by each method, under conditions listed in Tables 1, 2 and 3.

Figure 10 shows the models that are obtained after all 41 of the camera movements. The models are supposed to be of upright cylinders with radius 1.5 and height 3, centered on the origin. Visually inspecting them, the two different methods of updating seem to have little effect on the overall shape, however there are clear differences between the different bases. The SBF does well although there is a slight "ripple" effect on top and bottom discs and along the sides. The wavelet model has rounded sides, and blocky parts around the edges of the top and bottom disks.

These issues with the SHWs are at least partially due to the limitations of the basis functions, not the update methodology. Even when the environment is modelled directly (i.e. the "base" points are sampled from the correctly sized cylinder and no further updates are made) these effects are still visible (see Figure 19 in Appendix 8.4). Increasing the level of the SHWs to 7 and having 4096 non-zero coefficients improves the accuracy of the model greatly (again, see Figure 19). These values were not chosen for the main study as they take longer to run and require more memory to store the coefficients. However in an application where accuracy is much more important than time or memory constraints it is recommended to use a high value for these parameters.

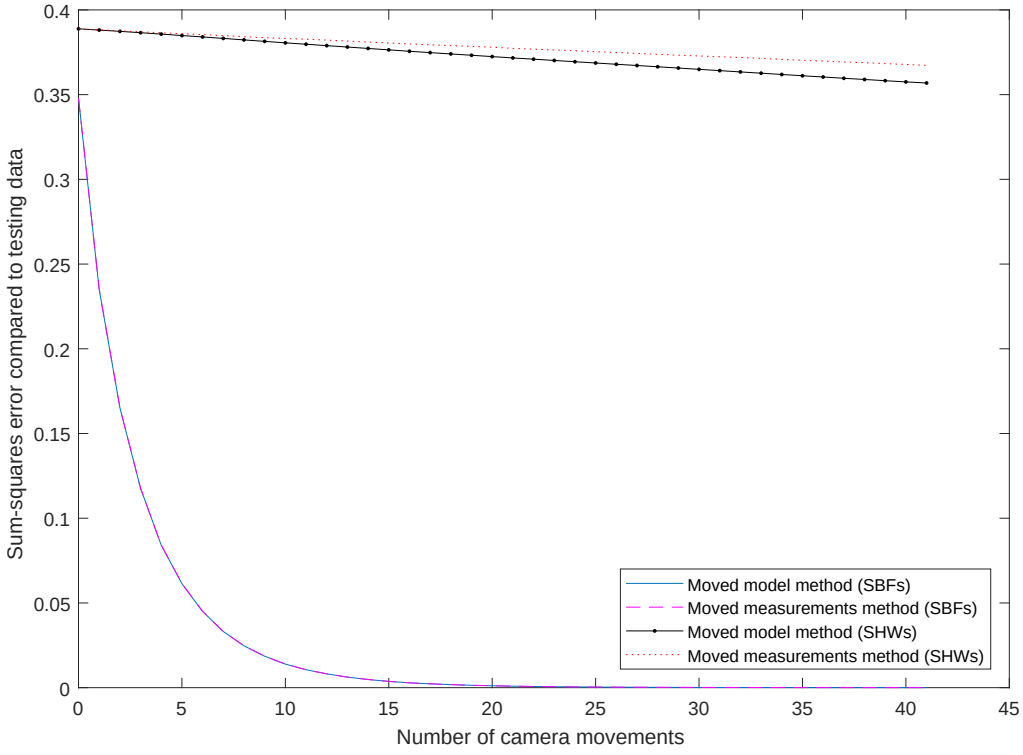


Figure 11: Sum-squared error for both methods and bases, under conditions listed in Tables 1, 2 and 3

Figure 11 shows the sum-squared error between the models and the test points. There is a large difference between the two bases. The SHWs have a much slower convergence rate than the SBFs, in fact their convergence rate appears linear which is obviously not ideal. However, whether the convergence rate is linear or not cannot be determined without seeing the full shape of the error graph before convergence.

For the SBFs, the convergence rate of the two update methods is so similar that they cannot be distinguished by eye. Table 4 quantifies the difference between the sum-squared error for each method. This table shows that the difference is small enough to be negligible, so the convergence rate and noise floor are essentially the same for both methods.

Total difference	Average difference	Standard deviation
2.8505×10^{-10}	6.7869×10^{-12}	1.2245×10^{-11}

Table 4: Difference in sum-squared error between moved model method and moved measurements method for SBFs, under conditions listed in Tables 1 and 2

For the SHWs, it can be seen that the moved model method converges faster than the moved measurements method for the SHWs, which is not the case for the SBFs. The moved model method for the SHWs is exact, which is not the case for the SBFs. However as this simulation only included rotation (no simulation), the moved measurements method should also not have any error (see Section 3.15). Thus this difference may be an indication that the update method for SHWs is not working as expected, and has some subtle interaction with the orientation of the model being updated.

Figure 12 shows the log of the sum-squared error plotted against the number of camera movements for the SBFs, The slope is approximately linear for the first part, but then begins flattening out after about 12 camera movements. This means that the convergence rate is slower than exponential. Gradient descent generally has exponential convergence. However, in this case we are optimising the least-squares distance to the measurement points at each camera movement, whereas Figure 12 has the log of the sum-squares error between the model and the test points. The error also plateaus after about 17 camera movements, corresponding to when the error stops decreasing.

The first part of the log plot is approximately linear, so we can approximate the first part of the convergence as exponential. This part has a gradient of $\frac{-8.7165-(-3.9858)}{30-10}$ –

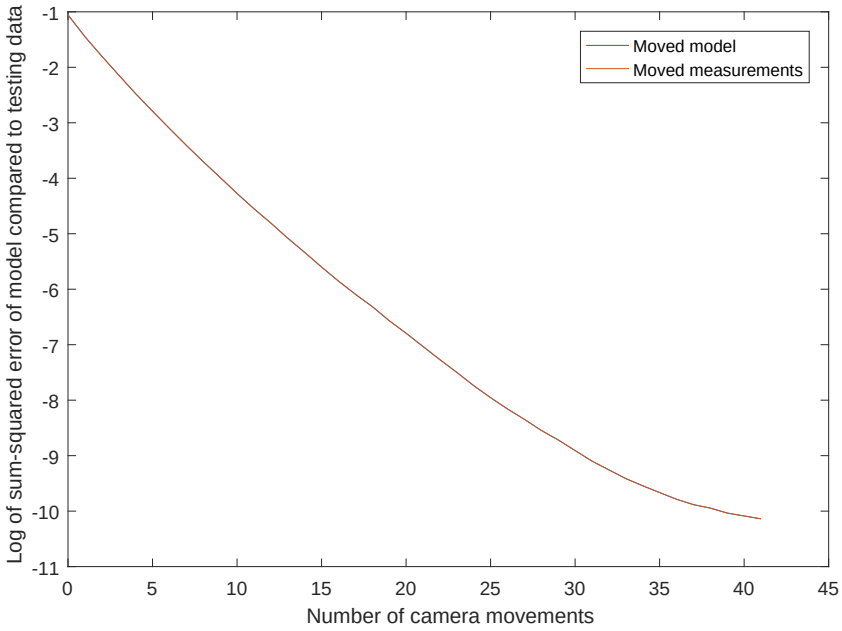


Figure 12: Log of sum-squares error for SBFs with each method, under conditions listed in Tables 1 and 2

0.2365 and a (projected) y-intercept of -1.0555. So we can approximate the convergence of the sum-squared error as following $\exp(-0.2365x - 1.0555)$, where x is the number of camera movements. (Note that this can be written as Ce^{ax}, where $C = e^{-1.0555}$ and $a = -0.2365$).

5.2 Computational Time

The data structure used to implement the spherical Haar wavelets (SHWs) is not optimised to take advantage of the sparse nature of the coefficients. Thus the results in this section may not be indicative of how a well-optimised implementation would perform. This is mostly relevant to comparing the times taken for the SHWs and and spherical harmonic basis functions (SBFs), as improving the data structure would likely improve the computational time of both update methods by a similar amount. Also, conclusions can still be drawn about the current implementations of each method and basis.

Table 5 shows the computational times for each method with each basis. (These times correspond to the results shown in Figures 11 and 10).

Basis	Method	Average computational time (s)	Standard deviation
SBF	Moved model	82.9707	2.5056
SBF	Moved measurements	1.1474	0.0618
SHW	Moved model	30.4041	0.6168
SHW	Moved measurements	29.9540	0.6480

Table 5: Computational time, under conditions listed in Tables 1, 2 and 3

As can be seen, the moved model method for spherical harmonics (SBFs) takes 72.312 times longer than the moved measurements method. This is with 2000 measurements and basis functions of up to order 40 (so 1681 basis functions). If the number of measurements increases or the number of basis functions decreases, then this ratio will become lower. (This is shown in Figure 20). The difference in methods is much smaller for the spherical Haar wavelets (SHWs), for which the moved model method is only 1.015 times slower (and this difference may be due to error, as the values are within a standard deviation). However both of the methods for the SHWs are significantly slower (about 26 times) than the moved measurements method for the SBFs, although they are faster than the moved model method for the SBFs (about 2.8 times).

Figure 20 in Appendix 8.4 shows the relationship between the ratio of the times taken for each method using the SBFs and the number of measurements (with $n = 40$) is approximately linear. One possible reason that the relationship is not more linear in this graph is that the data was collected in different batches. The 1000, 3000 and 4000 measurements data was collected together, as was the 1500, 2500, 3500 and 5000 measurement data, whereas the 2000 measurement data was collected by itself. These results are sensitive to the state of the computer when the code is run, so that could have resulted in errors. Although the ratio between the times for each method should be less sensitive to this than the absolute times for each of the methods (as differences in the state of the computer should effect both methods similarly). The line of best fit for the model is $-0.0091m + 94.175$, where m is the number of measurements (found

using the MATLAB function `polyfit`). This means the ratio will be 1 (i.e. the moved model method will be just as fast as the moved measurements method) when $m \approx 10^4$.

Figure 21 in Appendix 8.4 shows the relationship between the ratio of the times taken for each method and the maximum order of the basis functions (with 2000 measurements at each camera movement). Note that if the maximum order of basis functions is n, then the number of basis functions is $(n+1)^2$. The relationship appears to be quadratic, with line of best fit $0.0369n^2 + 0.3169n + 1.1451$ (also found using the MATLAB function `polyfit`). This means that even with only one basis function ($n = 0$), the moved measurements method will be faster for 2000 measurements. However, this relationship can be combined with the relationship between the ratio and the number of measurements (found above) to see which method will be faster for different combinations of n and m.

How important computational time is depends on the application. If the application seeks to run in or close to real-time, then a short computational time is extremely important. For example, an autonomous robot constructing a model of its environment and using said model to navigate as it travels. For other applications, computational time may be less important. For example, constructing a 3D model of an environment for use in a movie.

For applications that highly value computational time and do take an extremely large number of measurements on each camera movement (i.e. around 10^4), the most suitable method would be the moved model method with SHWs. In other situations, the moved measurements method with SBFs will likely be the fastest.

It may still be possible to use the moved model method with SBFs if accuracy is not a priority, by decreasing the number of basis functions. However the moved measurements method will almost certainly be a better choice for SBFs when speed is required.

6 Additional Results for Spherical Harmonic Basis Functions

This section presents results for additional scenarios that were investigated for spherical harmonic basis functions (SBFs) but not spherical Haar wavelets (SHWs). These include translation with the moved measurements method, basin of attraction, various robustness characteristics and field of view. This Section uses the parameter values from Section 5 in Tables 1 and 2, except where stated otherwise.

6.1 Translation

The moved measurements method allows for the camera to be translated as well as rotated. Allowing for translation is very important, as for applications such as robotic mapping camera translation is essential.

Figure 13 shows the sum-squares error when the camera is translated in a circle of radius 0.01, with all other aspects remaining the same as in Section 5.1. The convergence behaviour shown in Figure 13 is very similar to that shown in Figure 11. The total difference between the sum-squares error for the moved measurements method with and without translation is 0.0065. Considering the fact that adding translation modifies the trajectory, this difference is negligible.

6.2 Basin of attraction

This section investigates how the initial estimate of the model effects the convergence to the test model. A wide basin of attraction is important for applications where a good estimate of the scene is not available at the start.

Due to the large staring errors, N was increased for some of tests so that the overall shape of the convergence could be observed. The overall shapes of the error graphs are similar to Section 5.1, but with different starting points and gradients. Table 6 shows

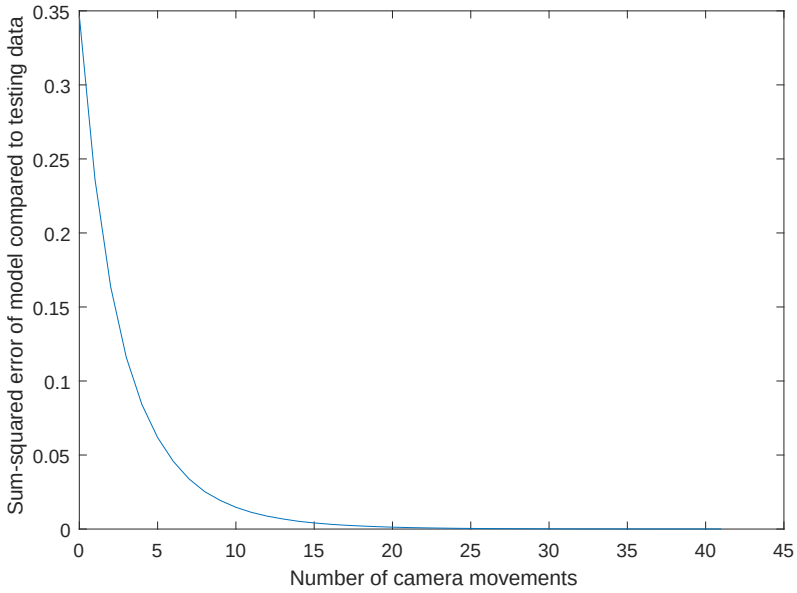


Figure 13: Sum-squared error for moved measurements method, with camera translation in a circle and under conditions listed in Tables 1 and 3

the approximate convergence rates of the first part (the approximately linear downwards slope in the log graph of the sum-squared error) when starting with different initial models. Note that the convergence rate applies for both methods – as in Section 5.1, any difference in convergence between the two methods was found to be negligible.

These results show that even if the initial model is far different to the actual scene, the model will still converge. However, it will take longer. Note that the coefficient of the x-term in the exponent is fairly similar across most of the initial models, but the constant term varies by quite a bit. This indicates that the reduction in error after a new set of measurements is fairly similar for most choices of initial model, even though the initial error differs.

The fact that the convergence rates are the same for both methods means that they are equally robust to changes in the initial model.

Shape	Radius	Height	N	Convergence rate
Cylinder	150	300	50	$\exp(-0.2221x + 10.3264)$
	75	150	50	$\exp(-0.2212x + 8.925)$
	15	30	50	$\exp(-0.1857x + 5.5328)$
	0.3	0.06	30	$\exp(-0.2290x + 1.1024)$
	0.015	0.03	50	$\exp(-0.1236x + 1.1194)$
	0.0015	0.003	30	$\exp(-0.2347x + 1.1380)$
Sphere	1	N/A	30	$\exp(-0.2112x - 0.4981)$
	3	N/A	30	$\exp(-0.2280x + 0.4604)$

Table 6: Convergence rates for various initial models. Note that the desired model in all cases is a cylinder with radius 1.5 and height 3. The convergence rate is worked out using the log of the sum-squares error as in Section 5.1.

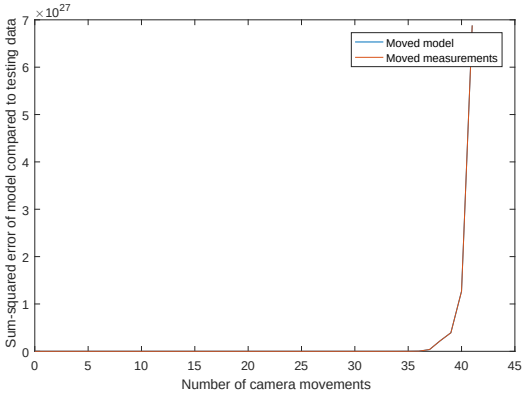
6.3 Robustness: Measurement Scaling

This section investigates how scaling the measurement gradients effects the convergence to the test model. Here, the measurement gradient is the amount a point on the model needs to be moved in a certain radial direction in order for it to match the measurement. This is important as the light field cameras that this work aims to be used with can only provide relative gradient information. This testing is done with a constant measurement scaling, which may not be accurate to real gradient information from light field cameras. However, if the gradient scalings do not differ greatly these results should still give an idea of the robustness with respect to different average gradient scalings.

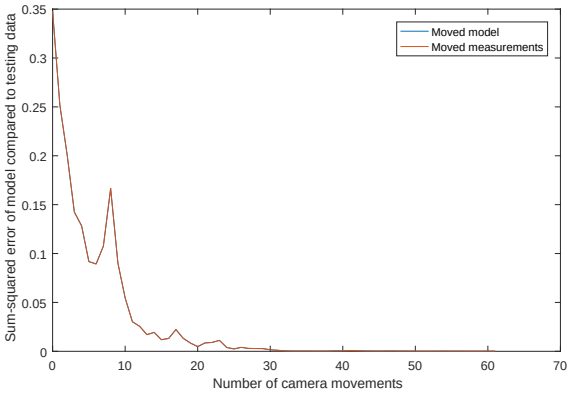
Note that changing the scaling has the same effect as changing the step size of the gradient descent. Recall that we are using the following formula for gradient descent (see Sections 3.6 and 4.6)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta B \Delta \mathbf{t}$$

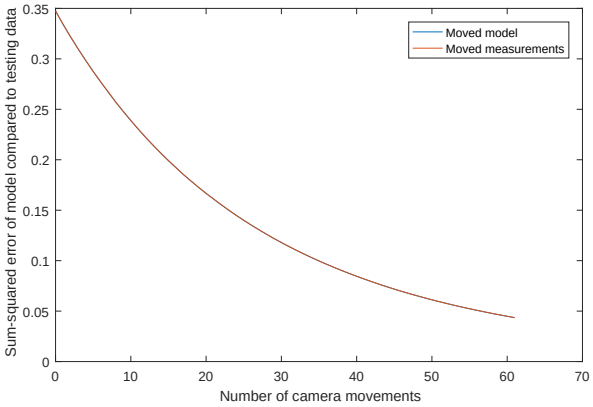
Where $\Delta\mathbf{t}$ is the measurement gradient and η is the step size, for this section $\eta = \text{lr} = 0.01$ as in Table 2 in Section 5. It is clear that multiplying $\Delta\mathbf{t}$ by a constant produces the same effect as multiplying η by that same constant. This means that the effect of changing the scaling can be offset by changing the step size.



(a) Measurement gradient scaling of 10



(b) Measurement gradient scaling of 5



(c) Measurement gradient scaling of 0.1

Figure 14: Sum-squares error for both methods for different measurement gradient scalings, but all other parameters as listed in Tables 1 and 3

With a gradient scale of 5, overshoot occurs but the method ends up converging. With a gradient scale of 10, convergence no longer occurs. On the other hand, with small gradient scales, convergence is very slow. (See Figure 14). Note that the sum-squared error of the two methods is still close enough that they cannot be distinguished by eye.

6.4 Robustness: Measurement Uncertainty

This section investigates how introducing uncertainties to the measurements effects the convergence to the test model. Robustness to measurement errors is important because in a physical system there will be errors in the measurements. This section used $N = 30$ in order to ensure that full convergence behaviour can be observed.

As in the previous sections, the sum-squared error of the two methods is close enough to be negligible. (With a measurement uncertainty of 1, the total difference is 8.6756×10^{-9} for 61 camera movements). The overall shape of the convergence is the same as without measurement uncertainty, but the method converges to a noise floor instead of 0 (see Figure 22 in Appendix 8.4). Figure 23 in Appendix 8.4 shows the relation between this noise floor and the measurement uncertainty. This relationship appears to be quadratic, with line of best fit $0.0067u^2 - 0.0001u + 0.0001$, where u is the standard deviation of the measurements (found using the MATLAB function `polyfit`). However, more simulation data would be needed to confirm whether this is actually the case.

The measurement uncertainty appears to have the same effect for both methods. It does not appear to significantly effect the convergence rate, but does result in a noise floor. These methods appear to scale quadratically with the standard deviation (and thus linearly with the variance), which is often the best that can be expected.

6.5 Field of View

This section investigates how limiting the FOV effects the convergence to the test model. Robustness to a limited FOV is important as physical cameras do not have a full FOV of everything around them.

When a limited FOV is introduced the methods take much longer to converge. The values investigated for (vertical FOV, horizontal FOV) were $(\pi/4, \pi/4), (\pi/4, \pi/3), (\pi/3, \pi/3), (\pi/3, \pi/2), (\pi/2, \pi/2)$. Both methods were tested for $N = 70$ for these

FOVs, and the difference between the sum-squared error of each method was again found to be negligible. The methods did not converge with this many camera movements. $N = 500$ was then used, but this was only done for the moved measurements method due to time constraints. The results are shown in Figure 15. As expected, a larger FOV results in faster convergence. Interestingly, the vertical FOV appears to matter much more for the convergence than the horizontal FOV. This may just be due to that fact that the scene being modelled is an upright cylinder, so the distance to the scene in a direction varies vertically but not horizontally. Another interesting feature is that the shape of the curve is different to when the FOV is not limited. It starts reasonably steeply, then flattens out for a bit, becomes steeper again and finally slows down and flattens out as it converges. The flat middle section likely indicates that there is an overlap in information between camera movements. Camera movements that look at entirely new points will provide more information, and therefore allow for faster convergence, than when the camera is looking at an area that contains points that have already been used to update the model. However, implementing an optimal scheme for moving the camera by taking this into account is not a trivial task.

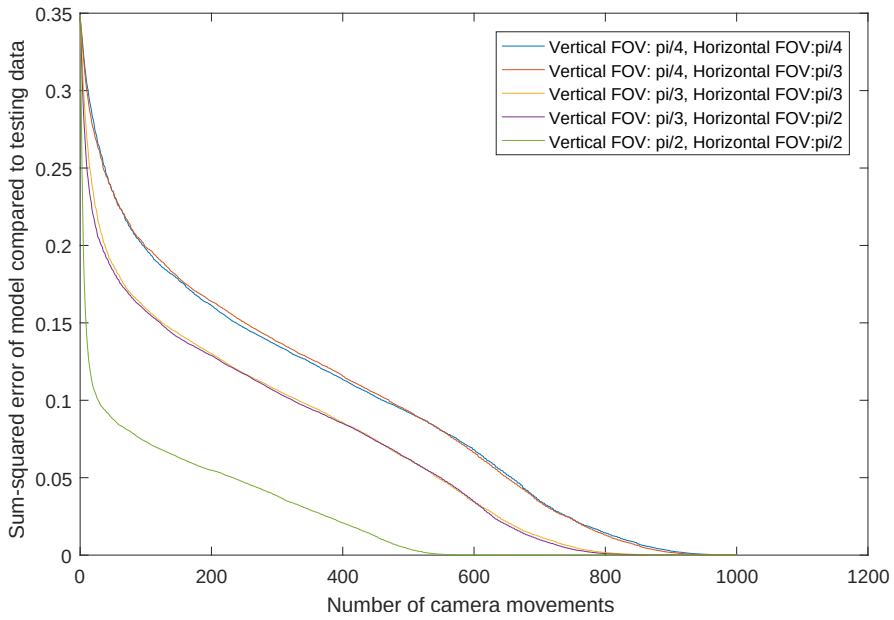


Figure 15: Sum-squared error for moved measurements method. $N = 5000$ and FOV varied. All other paramters are as listed in Tables 1 and 2.

Also, note that the way that the FOV is implemented, the number of measurements sets the number of points generated over the whole scene. The points within the FOV are then kept. This means that a wider FOV will also take in more measurements. The number of measurements was set to 5000, although as stated earlier not all of these measurements were used to update the model. This distorts the results, as it means the number of measurement points is also changing with the FOV, which is not generally the case for real cameras. Thus to get more meaningful results comparing FOVs the code should be rewritten to allow for the number of measurement points within the field of view to be set and these experiments repeated. Nevertheless, these results do show that the overall shape is fairly consistent between different FOVs.

7 Conclusion

The only significiant performance difference found between the moved model method and the moved measurements method for SBFs is in their computational time. The moved measurements method was found to be approximately 72 times faster when using the parameters in Tables 1 and 2. This ratio changes depending on the number of measurements and basis functions (see Section 5.2), but for the vast majority of applications the moved measurements method will be significantly faster. In addition, the moved measurements method can be used with measurements taken from a camera that has been translated, whereas currently the moved model method cannot. Therefore it appears that the moved measurements method is superior.

This computational time difference between the two methods is barely present for the SHWs. The moved measurements method is only about 1.015 times faster, and this difference may just be due to error as it is well within 1 standard deviation of the average times. Both of these methods are much slower (about 26 times) than the moved measurements method for SBFs, but they are slightly faster than the moved model method for SBFs (about 2.8 times). The relative slowness of the SHWs is likely due to poor choice of data structure when implementing the coefficient storage.

Both methods and bases have some issues. The main one is the importance of choosing an appropriate step size and how this interacts with the measurement gradient scaling. As explained in Section 6.3, changing the step size produces the same result as changing the gradient scaling. So if the gradient scaling is not known then choosing an appropriate step size will be difficult. If the step size is too large, it can cause overshoot, which can delay convergence or even stop the method from converging. If the step size is too small, it will take too long to converge. The step size should therefore be chosen conservatively to ensure that the method will actually converge. Also, note that the gradient scaling may not be constant. However, if the model is updated appropriately for the majority of measurements, then the error introduced by measurements with large gradient scalings will eventually be overcome.

Unfortunately the SHWs were not fully investigated due to time constraints. Only performance in ideal situations and computational time under those conditions were found. There was also only one shape that was modelled (a cylinder), thus more shapes should be tested for both the SHWs and the SBFs in order to find which ones each basis is suited to modelling.

A slightly longer-term goal is investigating why the wavelet update method is not working as desired. This could be due to not having found the right step size. However if this is not the (only) problem then clearly a new method of updating should be found and tested.

An interesting area of this project to pursue further would be investigating different spherical wavelet bases. In particular, looking at lifting the spherical Haar wavelet basis to get desirable properties such as smoothness (for an explanation of lifting see Sweldens and Schroder's work [24, 27]).

References

[1] E. H. Adelson and J. Y. Wang. Single lens stereo with a plenoptic camera. *IEEE transactions on pattern analysis and machine intelligence*, 14(2):99–106, 1992.

[2] Andeggs. 3d spherical.svg, 2009. URL https://en.wikipedia.org/wiki/File:3D_Spherical.svg.

[3] M. A. Blanco, M. Flórez, and M. Bermejo. Evaluation of the rotation matrices in the basis of real spherical harmonics. *Journal of Molecular Structure: THEOCHEM*, 419(1):19–27, 1997.

[4] C. Brechbühler, G. Gerig, and O. Kübler. Parametrization of closed surfaces for 3-d shape description. *Computer vision and image understanding*, 61(2):154–170, 1995.

[5] W. Chen, S. Itoh, and J. Shiki. Irregular sampling theorems for wavelet subspaces. *IEEE Transactions on Information Theory*, 44(3):1131–1142, 1998.

[6] J. J. Craig. *Introduction to Robotics Mechatronics and Control*. Pearson Education, Inc, third edition, 1955.

[7] Z. Gimbutas and L. Greengard. A fast and stable method for rotating spherical harmonic expansions. *Journal of Computational Physics*, 228(16):5621–5627, 2009.

[8] A. S. Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[9] R. GmbH. Raytrix — 3d light field camera technology, 2016. URL https://www.raytrix.de/.

[10] Á. González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical Geosciences*, 42(1):49–64, 2010.

[11] H. Hoppe and E. Praun. Shape compression using spherical geometry images. In *Advances in Multiresolution for Geometric Modelling*, pages 27–46. Springer, 2005.

[12] J. Jian-qiu, D. Min-ya, B. Hu-jun, and P. Qun-sheng. Watermarking on 3d mesh based on spherical wavelet transform. *Journal of Zhejiang University-Science A*, 5(3):251–258, 2004.

[13] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 271–278. ACM Press/Addison-Wesley Publishing Co., 2000.

[14] H. Laga, H. Takahashi, and M. Nakajima. Spherical wavelet descriptors for content-based 3d model retrieval. In *Shape Modeling and Applications, 2006. SMI 2006. IEEE International Conference on*, pages 15–15. IEEE, 2006.

[15] C. Lessig and E. Fiume. SOHO: orthogonal and symmetric Haar wavelets on the sphere. *ACM Transactions on Graphics (TOG)*, 27(1):4, 2008.

[16] G. Lippmann. Epreuves reversibles donnant la sensation du relief. *J. Phys. Theor. Appl.*, 7(1):821–825, 1908.

[17] I. Lytro. Lytro - home. URL https://www.lytro.com/.

[18] R. J. Morris, R. J. Najmanovich, A. Kahraman, and J. M. Thornton. Real spherical harmonic expansion coefficients as 3d shape descriptors for protein binding pocket and ligand comparisons. *Bioinformatics*, 21(10):2347–2355, 2005.

[19] R. A. Newcombe and A. J. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1498–1505. IEEE, 2010.

[20] S. O'Brien, J. Trumpf, R. Mahony, and V. Ila. An infinite-dimensional observer for depth estimation using plenoptic cameras. Unpublished draft, 2017.

[21] E. Praun and H. Hoppe. Spherical parametrization and remeshing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 340–349. ACM, 2003.

[22] R. Raghavendra, B. Yang, K. B. Raja, and C. Busch. A new perspective—face recognition with light-field camera. In *Biometrics (ICB), 2013 International Conference on*, pages 1–8. IEEE, 2013.

[23] Y. Roth-Tabak and R. Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, 1989.

[24] P. Schröder and W. Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 161–172. ACM, 1995.

[25] L. Shen, H. Farid, and M. A. McPeek. Modeling three-dimensional morphological structures using spherical harmonics. *Evolution*, 63(4):1003–1016, 2009.

[26] C. Soon Ong and C. Walder. Introduction to statistical machine learning, 2017. URL https://machlearn.gitlab.io/isml2017/lectures/05_Linear_Regression_1.pdf. Lecture notes.

[27] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM journal on mathematical analysis*, 29(2):511–546, 1998.

[28] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[29] G. Vilmart. Rigid body dynamics. 2015.

[30] E. W. Weisstein. Spherical harmonic. URL http://mathworld.wolfram.com/SphericalHarmonic.html.

[31] P. Wojtaszczyk. *A Mathematical Introduction to Wavelets*. Cambridge University Press, 1997.

8 Appendices

8.1 Code: Spherical Harmonics

This simulation study includes various functions, split into ones for spherical harmonic basis functions (SBFs) and spherical Haar wavelets (SHWs). This section will present the code for the SBFs. Figure 16 shows the dependencies of the various functions.

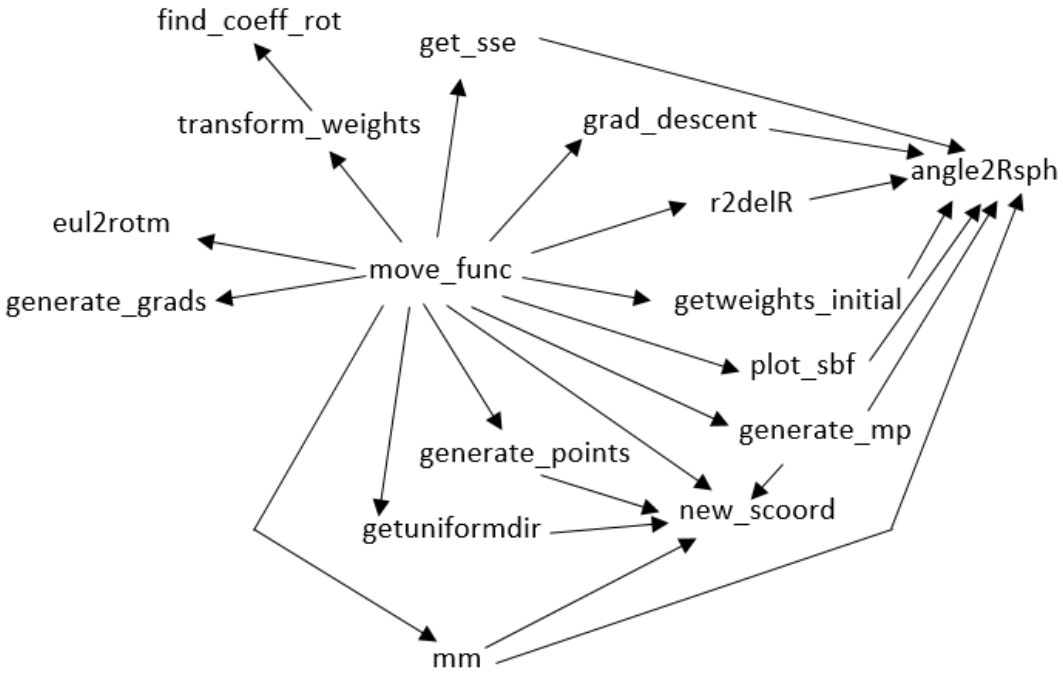


Figure 16: Code dependencies diagram. Arrows go from function A towards function B iff A calls B

A brief explaination of each function is listed below. The code itself is included on the following pages

1. `move_func`: this is the main function, can set the values of the parameters listed in Table 1 and 2. Finds and saves sum-squared error and models for both methods.

2. `generate_points`: generate points, can set shape, camera orientation and position, FOV and measurement uncertainty.

3. `new_scoord`: takes points in spherical coordinates, a rotation and a translation as input. Returns those points under the transformations, also in spherical coordinates.

4. `transform_weights`: takes weights and a rotation as input. Returns new weights that correspond to a rotated model.

5. `find_coeff_rot`: use Wigner D-functions to find a single rotated weight

6. `eul2rotm`: takes ZYZ Euler angles as input and returns the corresponding rotation matrix

7. `plot_sbf`: takes weights as input and plots the corresponding model.

8. `getweights_initial`: takes points and maximum order of basis functions as input and returns weights for a model fitting the data using Maximum Likelihood

9. `r2delR`: takes a measurement, the model and a scale. Returns the amount the model needs to move in the direction of the measurement in order for the model to match the measurement, multiplied by the scale.

10. `grad_descent`: takes a model and measurements as input, updates the model to better fit the measurements using gradient descent.

11. `get_sse`: takes test points and a model as input, returns the sum-squared difference between the radius of the model and the test points in the directions of the test points

12. `angle2Rsph`: takes a direction (θ, ϕ) and basis weights as input and returns the radius of the model in that direction.

13. `generate_mp`: generate points on a model.

14. `generate_grads`: takes a point cloud for a model, a point cloud for an environment and directions, generates gradients between the model and environment in those directions.

15. `getuniformdir`: takes a shape, returns uniform directions for that shape.

16. `mm`: takes gradients, a model and the camera rotation and translation, returns aligned measurement points using the moved measurements method.

8.2 Code: Spherical Haar Wavelets

This simulation study includes various functions, split into ones for spherical harmonic basis functions (SBFs) and spherical Haar wavelets (SHWs). This section will present the code for the SHWs. Figure 17 shows the dependencies of the various functions.

Note that this code is based on Lessig's implementation, available at http://www.dgp.toronto.edu/~lessig/soho/. Some of the functions had to be modified in order to allow them to work with depth information instead of colour information. That is, making them work with a function $f(\theta, \phi) = r$, instead of sampling colour data from a coloured sphere that has been mapped to a rectangular image. Code for unmodified functions is not provided in this report, but many such functions are necessary in order for my code to be able to run.

A brief explaination of each function is listed below. The code itself is included on the following pages

1. `move_func`: this is the main function, can set the values of the parameters listed in Table 1 and 3. Finds and saves sum-squared error and models for both methods.

2. `stri`: this is a class which represents a spherical triangle. It has a field which points to its children.

3. `generate_points`: (same as for SBFs) generate points, can set shape, camera orientation and position, FOV and measurement uncertainty.

4. `new_scoord`: (same as for SBFs) takes points in spherical coordinates, a rotation and a translation as input. Returns those points under the transformations, also in spherical coordinates.

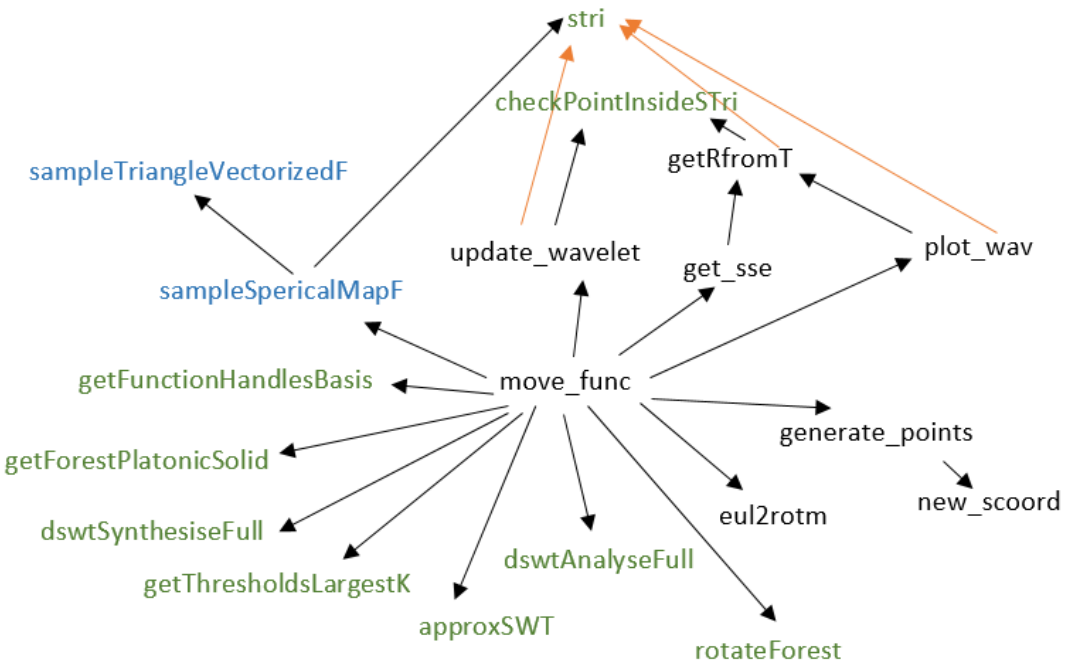


Figure 17: (Best view in colour). Code dependencies diagram. Black arrows go from function A towards function B iff A calls B. Orange arrows go from function A to class B iff A is a method of B. Function names in black were written entirely by me. Function names in blue are modified from Lessig's SOHO code. Function names in green are unmodified from Lessig's SOHO code, the dependencies for these functions are not shown in order to avoid over-complicating the diagram.

5. `eul2rotm`: (same as for SBFs) Takes ZYZ Euler angles as input and returns the corresponding rotation matrix

6. `plot_wav`: takes a stri and maximum level as input and plots the corresponding model.

7. `get_sse`: takes test points and a stri as input, returns the sum-squared difference between the radius of the model and the test points in the directions of the test points

8. `getRfromT`: takes a direction (θ, ϕ), a stri and a maximum level as input and returns the radius of the model in that direction.

9. `update_wavelet`: takes a stri, a maximum level, a point on the unit sphere and

its corresponding radius, and a step size as input and returns an updated stri.

10. `checkPointInsideSTri`: takes a stri and a point and returns whether that point lies within that stri.

11. `sampleSphericalMapF`: takes a stri, a function, a maximum level and the number of samples as input, returns a stri with samples from the function mapped onto the stri.

12. `sampleSphericalMapF`: takes the vertices of a stri, barycentric weights on that stri, a function and a number of samples, returns a stri with samples from that function.

13. `getFunctionHandlesBasis`: get function handles for given basis.

14. `getForestPlatonicSolid`: takes a platonic solid and level as input, and returns a forest of partitions derived from the platonic solid up to the level.

15. `dswtSynthesiseFull`: takes a forest, performs the synthesis step of the fast wavelet transform.

16. `getTresholdsLargestK`: takes a synthesised forest and a number of coefficients, returns the threshold for which there are exactly that number of coefficients over the threshold in the forest.

17. `approxSWT`: takes a synthesised forest and a threshold, sets all coefficients in that forest which are lower than the threshold to 0.

18. `dswtAnalyseFull`: takes a synthesised forest, performs the analysis step of the fast wavelet transform

19. `rotateForest`: takes a forest and a rotation, returns that forest with the rotation applied

8.3 Source code

Spherical harmonic basis functions:

```
function move_func(typeb, typem, R0, h0, R, h, N, n, fovt, fovp, scale, ...
    p1n, p2n, sse1n, sse2n, varn, uncer, nmeas, modflag)
%Set parameters
%Number of points
ntest = 10000;
nbase = 10000;
npc = 10^5;
%n = 40;
lam = 0.1;
lr = 10^(-3);
resplot = 150;

%Generate data points
base = generate_points(typeb, [0,0,0], eye(3), nbase, pi, 2*pi, 0, R0, h0);
test = generate_points(typem, [0,0,0], eye(3), ntest, pi, 2*pi, 0, R, h);
pceb = generate_points(typem, [0,0,0], eye(3), npc, pi, 2*pi, 0, R, h);

%Find initial model using base data
wb = getweights_initial(base(:,1:2), base(:,3), n, lam);

w1o = wb;
w1 = wb;
w2 = wb;

err_ms = zeros(N+1,1);
err_mm = zeros(N+1,1);

sse = get_sse(wb, n, test);
sse = sse/size(test,1);

err_ms(1) = sse;
err_mm(1) = sse;

times = zeros(N, 3);
tic
```

```
roto = eye(3);
Oo = [0, 0, 0];
for i = -N:N
    %Rotate according to Fibonacci Spiral
    invrat = (1+sqrt(5))/2 -1;
    lat = asin(2*i/(2*N+1));
    lon = 2*pi*i*invrat;
    lon = mod(lon+pi, 2*pi);
    lat = abs(lat-pi/2);
    %opposite rotation
    invrote = [lon, lat, 0];

    rote = [-invrote(3), -invrote(2), -invrote(1)];
    invrot = eul2rotm(invrote);
    rot = invrot';

    %Translate in a loop
%     r = 0.01;
%     ang = 2*pi*i/8;
%     O = [r*cos(ang), r*sin(ang), 0];
    O = [0,0,0];

    pce = new_scoord(pceb(:,1), pceb(:,2), pceb(:,3), O, rot);
    pcm1 = generate_mp(w1, n, npc, uncer, O, rot);
    pcm2 = generate_mp(w2, n, npc, uncer, O, rot);
    [thetam, phim] = getuniformdir(nmeas, fovt, fovp, 'cylinder', O, rot, R, h);
    meas1 = generate_grads(pce, pcm1, thetam, phim, scale);
    meas2 = generate_grads(pce, pcm2, thetam, phim, scale);
    times(i+N+1,1) = toc;

    if modflag==1
    rotn = rot*roto';
    bn = atan2(sqrt(rotn(3,1)^2+rotn(3,2)^2), rotn(3,3));

    if bn == 0
```

```
        an = 0;
        gn = atan2(-rotn(1,2), rotn(1,1));
    elseif bn==pi
        an = 0;
        gn = atan2(rotn(1,2), -rotn(1,1));
    else
        an = atan2(rotn(2,3)/sin(bn), rotn(1,3)/sin(bn));
        gn = atan2(rotn(3,2)/sin(bn), -rotn(3,1)/sin(bn));
    end

    an = an + pi;
    gn = gn+pi;

    roten = [an,bn,gn];

    %Get the transformed weights
    w1 = transform_weights(w1o, n, roten);


    %Input measurements to transformed model
    r1 = meas1(:,3) + angle2Rsphfull(w1, n, meas1(:,1), meas1(:,2));
    w1 = grad_descent(meas1(:,1:2), r1, n, w1, lr);
    w1o = w1;

    w1 = transform_weights(w1, n, invrote + [pi, 0, pi]);


    sse = get_sse(w1, n, test);
    sse = sse/size(test,1);
    err_ms(i+1+N+1) = sse;
    end

    times(i+N+1,2) = toc;


    %Input measurements to original model
```

```
    [X, r2] = mm(meas2, w2, n, invrot, -O);

%     ms = [X, r2];
%     plot_sbf(w2, n, resplot);
%     plot_points(ms);

    w2 = grad_descent(X, r2, n, w2, lr);


    sse = get_sse(w2, n, test);
    sse = sse/size(test,1);
    err_mm(i+1+N+1) = sse;

    times(i+N+1,3) = toc;

    roto = rot;
    Oo = O;
end

if modflag==1
plot_sbf(w1, n, resplot);
title('Updated model (transform model method)')
xlabel('x')
ylabel('y')
zlabel('z')
saveas(gcf, p1n);
end

plot_sbf(w2, n, resplot);
title('Updated model (transform measurements method)')
xlabel('x')
ylabel('y')
zlabel('z')
saveas(gcf, p2n);

if modflag==1
```

```
figure;
plot(err_ms)
title('Sum-squared Error (transform model method)')
xlabel('Number of camera movements')
ylabel('Sum-squared error of model compared to testing data')
saveas(gcf, sse1n);
end

figure;
plot(err_mm)
title('Sum-squared Error (transform measurements method)')
xlabel('Number of camera movements')
ylabel('Sum-squared error of model compared to testing data')
saveas(gcf, sse2n);


save(varn, 'times', 'w1', 'w2', 'err_ms', 'err_mm', 'base', 'test', 'N');
end


function points = generate_points(type, O, rot, n, fovt, fovp, uncer, varargin)
if strcmp(type,'cylinder')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
        h = 2;
    elseif nvar < 2
        R = varargin{1};
        h = 2;
    else
        R = varargin{1};
        h = varargin{2};
    end

    Ac = 2*pi*R^2;    %Area of circles (top and bottom)
    Ar = 2*pi*R*h;    %Area of rectangle (side)
```

```
nr = int32(n/((Ac/Ar)+1)); %Number of points in rectangle
nc = int32((n - nr)/2);    %Number of points in each circle
rrt = rand(nc,1);             %radius (random)
rrt = R^2*rrt;                %Scale up
art = rand(nc,1);
art = 2*pi*art;
xt = sqrt(rrt).*cos(art);
yt = sqrt(rrt).*sin(art);
zt = h/2;

rt = sqrt(xt.^2+yt.^2+zt.^2);
thetat = acos(zt./rt);
phit = atan2(xt,yt);

xrr = rand(nr,1);
xrr = 2*pi*R*xrr;
yrr = rand(nr,1);
yrr = h*yrr - h/2;
thetar = pi/2 - atan(yrr/R);
phir = xrr/R;

st = sin(thetar);
rr = R./st;

rrb = rand(nc,1);
rrb = R^2*rrb;
arb = rand(nc,1);
arb = 2*pi*arb;
xb = sqrt(rrb).*cos(arb);
yb = sqrt(rrb).*sin(arb);
zb = -h/2;

rb = sqrt(xb.^2+yb.^2+zb.^2);
thetab = acos(zb./rb);
phib = atan2(xb,yb);
```

```
    %Make pit and phib go from 0 to 2pi instead of -pi to pi
    phit = pi + phit;
    phib = pi+phib;

    theta = [thetat; thetar; thetab];
    phi = [phit; phir; phib];
    r = [rt; rr; rb];

elseif strcmp(type,'sphere')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
    else
        R = varargin{1};
    end

    delta = pi/sqrt(n);
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;

    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);

    r = R*ones(N, 1);


elseif strcmp(type,'spherep')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
    else
        R = varargin{1};
    end
```

```
    delta = pi/sqrt(n);
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;

    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);

    r = R*ones(N,1) + rand(N, 1)/100;

else
    error('Invalid type. Valid types are: cylinder, sphere')
end

    %Combine into points array and match rotation, translation and fov

    points = [theta, phi, r];

    points = new_scoord(points(:,1), points(:,2), points(:,3), O, rot);

    theta = points(:,1);
    phi = points(:,2);
    r = points(:,3);

    %theta = pi/2 is center of camera
    fovti = find(abs(theta-pi/2)<= fovt/2);
    %phi = pi is center of the camera to make it easier to find (instead of
    %0)
    fovpi = find(abs(phi-pi)<= fovp/2);

    fovi = intersect(fovti, fovpi);
    theta = theta(fovi);
    phi = phi(fovi);
    r = r(fovi);
```

```
    r = r + (uncer*rand(numel(r),1)-uncer/2);

    points = [theta, phi, r];


end


function out = new_scoord(theta, phi, r, t, rot)
[x,y,z] = sph2cart(phi-pi, pi/2-theta, r);

v = [x,y,z]';
% rotM = eul2rotm(rot);
v = rot*v;

% v = rot(:,1:3)*v;
% v = rot(:,4:6)*v;
% v = rot(:,7:9)*v;

v = v';
x = v(:,1);
y = v(:,2);
z = v(:,3);

x = x-t(1);
y = y-t(2);
z = z-t(3);

[phi, theta, r] = cart2sph(x,y,z);

theta = abs(theta-pi/2);
phi = pi + phi;

out = [theta, phi, r];
end
```

```
function w = transform_weights(wo, n, rot)
w = wo;
for l = 0:n
    wo = w;
    for m = -l:l
        i = l^2+l+m+1;
        w(i) = find_coeff_rot(wo, l, m, rot);
    end

end
end



function c = find_coeff_rot(co, L, M, rot)
%zyz convention
alpha = rot(1);
beta = rot(2);
gamma = rot(3);

c = 0;
for m = -L:L
   c = c+ D(L, m, M, alpha, beta, gamma)*co(L^2+L+m+1);
end

end

function Dlmk = D(l,m,k, alpha, beta, gamma)
dlmk1 = d(l, abs(k), abs(m), beta) + (-1)^m*d(l, abs(m), -abs(k), beta);
dlmk2 = d(l, abs(k), abs(m), beta) - (-1)^m*d(l, abs(m), -abs(k), beta);
Dlmk = sig(k)*az_op(k, alpha)*az_op(m, gamma)*dlmk1/2 ...
    -sig(m)*az_op(-k, alpha)*az_op(-m, gamma)*dlmk2/2;
end

function dlmk = d(l, m, k, beta)
sum = 0;
for s = 0:(l+abs(m))
```

```
    if (l-m-s)<0 || (l+k-s)<0 || (m-k+s)<0
        continue
    end
    denom = factorial(s)*factorial(l-m-s)*factorial(l+k-s)*factorial(m-k+s);
    sum = sum + (-1)^(s)*cos(beta/2)^(2*(l-s)-m+k)*sin(beta/2)^(2*s+m-k)/denom;
end

dlmk = (-1)^(m-k)*sqrt(factorial(l+k)*factorial(l-k)*factorial(l+m)*factorial(l-m))*su
end

function Phi = az_op(m, ang)
if m>0
    Phi = sqrt(2)*cos(m*ang);
elseif m==0
    Phi = 1;
else
    Phi = -sqrt(2)*sin(abs(m)*ang);
end
end

function sign = sig(x)
if x < 0
    sign = -1;
else
    sign = 1;
end
end



function M = eul2rotm(eul)
a = eul(1);
b = eul(2);
g = eul(3);


s1 = sin(a);
```

```
s2 = sin(b);
s3 = sin(g);
c1 = cos(a);
c2 = cos(b);
c3 = cos(g);

%ZYZ
M = [c1*c2*c3-s1*s3, -c3*s1-c1*c2*s3, c1*s2;
    c1*s3+c2*c3*s1, c1*c3-c2*s1*s3, s1*s2;
    -c3*s2, s2*s3, c2];

end



function plot_sbf(w, n, res)
    delta = pi/res;
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;
    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    M = size(theta, 1);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);
    R = zeros(N,1);

    for l = 0:n
        r = angle2Rsph(l, theta, phi);
        R = R + (w(l^2+1:l^2+2*l+1)'*r)';
    end

    figure
    hold on

    r = abs(R);
    x = r.*sin(theta).*cos(phi);
    y = r.*sin(theta).*sin(phi);
```

```
    z = r.*cos(theta);

    x = reshape(x, [M,M]);
    y = reshape(y, [M,M]);
    z = reshape(z, [M,M]);
    r = reshape(r, [M,M]);

    h = surf(x,y,z,r);

    camlight left
    camlight right
    lighting phong

    alpha(h, 0.5)


    % map positive regions to red, negative regions to green
    colormap(redgreencmap([2]))
    set(h, 'LineStyle','none')

    grid off

end


function [w,R] = getweights_initial(X, r, n, lam)
%getweights_initial takes
%   X= [theta, phi] measurement directions
%   r= coressponding radii
%   n = maximum order of basis functions
%   lam = regularisation constant
theta =X(:,1);
phi = X(:,2);
R = zeros(size(X,1), (n+1)^2);

for l=0:n
```

```
    R(:,l^2+1:l^2+2*l+1) = angle2Rsph(l, theta, phi)';
end
w = (lam*eye((n+1)^2) + R'*R)\(R'*r);
end


function delR = r2delR(X, r, w, n, scale)
theta =X(:,1);
phi = X(:,2);
Rmod = zeros(size(X,1), 1);

for l = 0:n
    rl = angle2Rsph(l, theta, phi);
    Rmod = Rmod + (w(l^2+1:l^2+2*l+1)'*rl)';
end

delR = scale*(Rmod-r);
end


function [w, R] = grad_descent(X, delR, n, w, lr)
%X contains the theta and phi values, one set of values in each row
%t contains the corresponding r values
theta = X(:,1);
phi = X(:,2);
R = zeros(size(X,1), (n+1)^2);
%ae = (n+1)^2/sum(abs(w));

for l=0:n
    R(:,l^2+1:l^2+2*l+1) = angle2Rsph(l, theta, phi)';
end

w = w - lr*((R'*(delR)));

end
```

```
function sse = get_sse(w, n, test)
theta = test(:,1);
phi = test(:,2);
r = test(:,3);
R = zeros(size(r,1), 1);
for l = 0:n
    rl = angle2Rsph(l, theta, phi);
    R = R + (w(l^2+1:l^2+2*l+1)'*rl)';
end

sse = (r-R).^2;
sse = sum(sse);

end



function R = angle2Rsph(l, theta, phi)
%angle2Rsph takes
%   l= degree of spherical harmonic
%   theta = matrix of theta (polar angle) values
%   phi = matrix of corresponding phi (azimuthal angle) values
%and returns
%   R= matrix of corresponding radial distance values for order m =-l:l
%   The orders vary along the rows, and theta and phi values vary along the
%   columns

len = size(theta, 1);
P = legendre(l,cos(theta));
N = zeros(l, len);
S = zeros(l, len);
C = zeros(l, len);
N0 = sqrt((2*l+1)/(4*pi));
for m = 1:l
    N(m,:) = repmat(sqrt((2*l+1)/(4*pi)*factorial(l-m)/factorial(l+m)), [1,len]);
    S(m,:) = sin(m*phi);
    C(m,:) = cos(m*phi);
```

```
end

if l>0
R0 = sqrt(2)*N.*P(2:end,:).*S;
R1 = N0.*P(1,:);
R2 = sqrt(2)*N.*P(2:end,:).*C;
R = [flipud(R0); R1; R2];
else
R = N0.*P;
end

end



function points = generate_mp(w, nm, n, uncer, t, rot)

    delta = pi/sqrt(n);
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;

    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);

    %radius is point on the model
    r = angle2Rsphfull(w,nm, theta, phi);

    %uncertainty
    r = r + (uncer*rand(numel(r),1)-uncer/2);

    points = new_scoord(theta,phi,r,t,rot);

end
```

```
function grads = generate_grads(pce, pcm, theta, phi, scale)
    %radius_e and radius_m are given by closest point in each cloud
    %r is the difference (delR)
    r = zeros(numel(phi), 1);

    for i = 1:numel(theta)
        [~,ie] = min((pce(:,1)-theta(i)).^2+(pce(:,2)-phi(i)).^2);
        re = pce(ie, 3);
        [~,im] = min((pcm(:,1)-theta(i)).^2+(pcm(:,2)-phi(i)).^2);
        rm = pcm(im, 3);
        r(i) = re-rm;
    end


    %uncertainty
    r = scale*r;

    grads = [theta, phi, r];

end


function [theta, phi] = getuniformdir(n, fovt, fovp, type, O, rot, varargin)

    if strcmp(type,'cylinder')
        nvar = numel(varargin);
        if nvar < 1
            R = 1;
            h = 2;
        elseif nvar < 2
            R = varargin{1};
            h = 2;
        else
            R = varargin{1};
            h = varargin{2};
        end
```

```
n= n*(fovt*fovp)/(2*pi^2);

Ac = 2*pi*R^2;   %Area of circles (top and bottom)
Ar = 2*pi*R*h;   %Area of rectangle (side)

nr = int32(n/((Ac/Ar)+1)); %Number of points in rectangle
nc = int32((n - nr)/2);    %Number of points in each circle
rrt = rand(nc,1);             %radius (random)
rrt = R^2*rrt;                %Scale up
art = rand(nc,1);
art = 2*pi*art;
xt = sqrt(rrt).*cos(art);
yt = sqrt(rrt).*sin(art);
zt = h/2;

rt = sqrt(xt.^2+yt.^2+zt.^2);
thetat = acos(zt./rt);
phit = atan2(xt,yt);

xrr = rand(nr,1);
xrr = 2*pi*R*xrr;
yrr = rand(nr,1);
yrr = h*yrr - h/2;
thetar = pi/2 - atan(yrr/R);
phir = xrr/R;

st = sin(thetar);
rr = R./st;

rrb = rand(nc,1);
rrb = R^2*rrb;
arb = rand(nc,1);
arb = 2*pi*arb;
xb = sqrt(rrb).*cos(arb);
yb = sqrt(rrb).*sin(arb);
```

```
        zb = -h/2;

        rb = sqrt(xb.^2+yb.^2+zb.^2);
        thetab = acos(zb./rb);
        phib = atan2(xb,yb);

        %Make pit and phib go from 0 to 2pi instead of -pi to pi
        phit = pi + phit;
        phib = pi+phib;

        theta = [thetat; thetar; thetab];
        phi = [phit; phir; phib];
        r = [rt; rr; rb];

        %transform into view point
        points = [theta, phi, r];

        points = new_scoord(points(:,1), points(:,2), points(:,3), O, rot);

        theta = points(:,1);
        phi = points(:,2);
        r = points(:,3);

        %theta = pi/2 is center of camera
        fovti = find(abs(theta-pi/2)<= fovt/2);
        %phi = pi is center of the camera to make it easier to find (instead of
        %0)
        fovpi = find(abs(phi-pi)<= fovp/2);

        fovi = intersect(fovti, fovpi);
        theta = theta(fovi);
        phi = phi(fovi);

    elseif strcmp(type,'sphere')
        theta = rand(n,1);
        phi = rand(n,1);
```

```
        theta = theta*(fovt);
        phi = phi*(fovp);
    end

end


function [X, r] = mm(meas, w, n, rot, t)
sgn = sign(meas(:,3));
p = new_scoord(meas(:,1), meas(:,2), abs(meas(:,3)), t, rot);
p(:,3) = sgn.*p(:,3);

r = p(:,3) + angle2Rsphfull(w, n, p(:,1), p(:,2));
X = p(:,1:2);

end
```

Spherical Haar wavelets:

```
function move_func(fb, N, level, nc, fovt, fovp, scale, ...
    p1n, p2n, sse1n, sse2n, varn, uncer, modflag, resplot, nmeas, type, varargin)

nvar = numel(varargin);
if nvar < 1
    Rc = 1;
    hc = 2;
elseif nvar < 2
    Rc = varargin{1};
    hc = 2;
else
    Rc = varargin{1};
    hc = varargin{2};
end
```

```
%Set parameters
platonic_solid = 'octahedron';
basis = 'osh';
fhs = getFunctionHandlesBasis( basis);


%Generate data points
forest = getForestPlatonicSolid( platonic_solid, level, fhs.enforce_equal_area);
base = sampleSphericalMapF( forest, fb, 1, level, 0);

if nvar < 1
    test = generate_points(type, [0,0,0], eye(3), 10000, pi, 2*pi, 0);
elseif nvar < 2
    test = generate_points(type, [0,0,0], eye(3), 10000, pi, 2*pi, 0, Rc);
else
    test = generate_points(type, [0,0,0], eye(3), 10000, pi, 2*pi, 0, Rc, hc);
end

%Find initial model using base data
ab = dswtAnalyseFull(base, level, fhs.filters_analysis, fhs.normalize);
thresholds = getThresholdLargestK( ab, level, nc, fhs.approx );
ab = approxSWT( ab, level, thresholds, fhs.approx );
ab = dswtSynthesiseFull(ab, level, fhs.filters_synthesis, fhs.denormalize, 0, 1);

a1o = ab;
a1 = ab;
a2 = ab;

err_ms = zeros(N+1,1);
err_mm = zeros(N+1,1);

sse = get_sse(ab, test, level);

err_ms(1) = sse;
err_mm(1) = sse;
```

```
times = zeros(N, 3);
tic


roto = eye(3);
Oo = [0, 0, 0];
for i = -N:N
    %step size for update_wavelet
    lr = 100/(nmeas);

    %Rotate according to Fibonacci Spiral
    invrat = (1+sqrt(5))/2 -1;
    lat = asin(2*i/(2*N+1));
    lon = 2*pi*i*invrat;
    lon = mod(lon+pi, 2*pi);
    lat = abs(lat-pi/2);
    %opposite rotation
    invrote = [lon, lat, 0];

    rote = [-invrote(3), -invrote(2), -invrote(1)];
    invrot = eul2rotm(invrote);
    rot = invrot';

    %Translate in a loop
%     r = 0.01;
%     ang = 2*pi*i/8;
%     O = [r*cos(ang), r*sin(ang), 0];
    O = [0,0,0];

    %get measurements
    a1r = rotateForest(a1o, rot);
    a2r = rotateForest(a2, rot);

    if nvar < 1
        meas = generate_points(type, O, rot, nmeas, fovt, fovp, uncer);
    elseif nvar < 2
```

```
    meas = generate_points(type, O, rot, nmeas, fovt, fovp, uncer, Rc);
else
    meas = generate_points(type, O, rot, nmeas, fovt, fovp, uncer, Rc, hc);
end
thetam = meas(:,1)';
phim = meas(:,2)';
R_m = meas(:,3)';

%Find the gradients

R_b1 = getRfromT(thetam,phim,a1r, level);
delR1 = scale*(R_m-R_b1);

R_b2 = getRfromT(thetam,phim,a2r,level);
delR2 = scale*(R_m-R_b2);

times(i+N+1,1) = toc;

%% moved model method
if modflag==1
rotn = rot*roto';
bn = atan2(sqrt(rotn(3,1)^2+rotn(3,2)^2), rotn(3,3));

if bn == 0
    an = 0;
    gn = atan2(-rotn(1,2), rotn(1,1));
elseif bn==pi
    an = 0;
    gn = atan2(rotn(1,2), -rotn(1,1));
else
    an = atan2(rotn(2,3)/sin(bn), rotn(1,3)/sin(bn));
    gn = atan2(rotn(3,2)/sin(bn), -rotn(3,1)/sin(bn));
end

an = an + pi;
gn = gn + pi;
```

```
roten = [an,bn,gn];
rotn = eul2rotm(roten);

%Get the transformed forest
a1 = rotateForest(a1o, rotn);

x1 = sin(thetam).*cos(phim);
y1 = sin(thetam).*sin(phim);
z1 = cos(thetam);

for j = 1:numel(thetam)
    a1 = update_wavelet(a1, stri(a1), level, [x1(j);y1(j);z1(j)], ...
        delR1(j), [], {}, lr);
end


%zeroing all except nc coefficients
thresholds = getThresholdLargestK(a1, level, nc, fhs.approx );
a1 = approxSWT(a1, level, thresholds, fhs.approx );

% reconstruct the approximated signal
a1o = dswtSynthesiseFull( a1, level, ...
                                    fhs.filters_synthesis, fhs.denormalize, 0, 1);

invrote1 = invrote + [pi, 0, pi];
invrot1 = eul2rotm(invrote1);
a1 = rotateForest(a1o, invrot1);

sse = get_sse(a1, test, level);
err_ms(i+1+N+1) = sse;
end

times(i+N+1,2) = toc;
```

```
    %% moved measurements method

    %Note that do not multiply by the delR2 values, as want a point on the
    %unit sphere
    x2 = sin(thetam).*cos(phim);
    y2 = sin(thetam).*sin(phim);
    z2 = cos(thetam);

    meas_align = invrot*[x2;y2;z2];
    x2 = meas_align(1,:); y2 = meas_align(2,:); z2 = meas_align(3,:);

    for j = 1:numel(thetam)
        a2 = update_wavelet(a2, stri(a2), level, [x2(j);y2(j);z2(j)], ...
            delR2(j), [], {}, lr);
    end

    %zeroing all except nc coefficients
    thresholds = getThresholdLargestK( a2, level, nc, fhs.approx );
    a2 = approxSWT( a2, level, thresholds, fhs.approx );

    % reconstruct the approximated signal
    a2 = dswtSynthesiseFull( a2, level, ...
                                    fhs.filters_synthesis, fhs.denormalize, 0, 1);



    sse = get_sse(a2, test, level);
    err_mm(i+1+N+1) = sse;

    times(i+N+1,3) = toc;

    roto = rot;
    Oo = O;
end

if modflag==1
    figure;
```

```
plot_wav( a1, level, resplot)
title('Updated model (transform model method)')
xlabel('x')
ylabel('y')
zlabel('z')
saveas(gcf, p1n);
end

figure;
plot_wav( a2, level, resplot)
title('Updated model (transform measurements method)')
xlabel('x')
ylabel('y')
zlabel('z')
saveas(gcf, p2n);

if modflag==1
figure;
plot(err_ms)
title('Sum-squared Error (transform model method)')
xlabel('Number of camera movements')
ylabel('Sum-squared error of model compared to testing data')
saveas(gcf, sse1n);
end

figure;
plot(err_mm)
title('Sum-squared Error (transform measurements method)')
xlabel('Number of camera movements')
ylabel('Sum-squared error of model compared to testing data')
saveas(gcf, sse2n);


save(varn, 'times', 'a1', 'a2', 'err_ms', 'err_mm', 'base', 'test', 'N');
end
```

```
function points = generate_points(type, O, rot, n, fovt, fovp, uncer, varargin)
if strcmp(type,'cylinder')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
        h = 2;
    elseif nvar < 2
        R = varargin{1};
        h = 2;
    else
        R = varargin{1};
        h = varargin{2};
    end

    Ac = 2*pi*R^2;   %Area of circles (top and bottom)
    Ar = 2*pi*R*h;   %Area of rectangle (side)

    nr = int32(n/((Ac/Ar)+1)); %Number of points in rectangle
    nc = int32((n - nr)/2);    %Number of points in each circle
    rrt = rand(nc,1);             %radius (random)
    rrt = R^2*rrt;                %Scale up
    art = rand(nc,1);
    art = 2*pi*art;
    xt = sqrt(rrt).*cos(art);
    yt = sqrt(rrt).*sin(art);
    zt = h/2;

    rt = sqrt(xt.^2+yt.^2+zt.^2);
    thetat = acos(zt./rt);
    phit = atan2(xt,yt);

    xrr = rand(nr,1);
    xrr = 2*pi*R*xrr;
    yrr = rand(nr,1);
    yrr = h*yrr - h/2;
    thetar = pi/2 - atan(yrr/R);
```

```
    phir = xrr/R;

    st = sin(thetar);
    rr = R./st;

    rrb = rand(nc,1);
    rrb = R^2*rrb;
    arb = rand(nc,1);
    arb = 2*pi*arb;
    xb = sqrt(rrb).*cos(arb);
    yb = sqrt(rrb).*sin(arb);
    zb = -h/2;

    rb = sqrt(xb.^2+yb.^2+zb.^2);
    thetab = acos(zb./rb);
    phib = atan2(xb,yb);

    %Make pit and phib go from 0 to 2pi instead of -pi to pi
    phit = pi + phit;
    phib = pi+phib;

    theta = [thetat; thetar; thetab];
    phi = [phit; phir; phib];
    r = [rt; rr; rb];

elseif strcmp(type,'sphere')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
    else
        R = varargin{1};
    end

    delta = pi/sqrt(n);
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;
```

```
    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);

    r = R*ones(N, 1);


elseif strcmp(type,'spherep')
    nvar = numel(varargin);
    if nvar < 1
        R = 1;
    else
        R = varargin{1};
    end

    delta = pi/sqrt(n);
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;

    [phi,theta] = meshgrid(phi,theta);
    N = numel(theta);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);

    r = R*ones(N,1) + rand(N, 1)/100;

else
    error('Invalid type. Valid types are: cylinder, sphere')
end

    %Combine into points array and match rotation, translation and fov

    points = [theta, phi, r];
```

```
    points = new_scoord(points(:,1), points(:,2), points(:,3), 0, rot);

    theta = points(:,1);
    phi = points(:,2);
    r = points(:,3);

    %theta = pi/2 is center of camera
    fovti = find(abs(theta-pi/2)<= fovt/2);
    %phi = pi is center of the camera to make it easier to find (instead of
    %0)
    fovpi = find(abs(phi-pi)<= fovp/2);

    fovi = intersect(fovti, fovpi);
    theta = theta(fovi);
    phi = phi(fovi);
    r = r(fovi);
    r = r + (uncer*rand(numel(r),1)-uncer/2);

    points = [theta, phi, r];


end


function out = new_scoord(theta, phi, r, t, rot)
[x,y,z] = sph2cart(phi-pi, pi/2-theta, r);

v = [x,y,z]';
% rotM = eul2rotm(rot);
v = rot*v;

% v = rot(:,1:3)*v;
% v = rot(:,4:6)*v;
% v = rot(:,7:9)*v;

v = v';
```

```
x = v(:,1);
y = v(:,2);
z = v(:,3);

x = x-t(1);
y = y-t(2);
z = z-t(3);

[phi, theta, r] = cart2sph(x,y,z);

theta = abs(theta-pi/2);
phi = pi + phi;

out = [theta, phi, r];
end



function M = eul2rotm(eul)
a = eul(1);
b = eul(2);
g = eul(3);



s1 = sin(a);
s2 = sin(b);
s3 = sin(g);
c1 = cos(a);
c2 = cos(b);
c3 = cos(g);

%ZYZ
M = [c1*c2*c3-s1*s3, -c3*s1-c1*c2*s3, c1*s2;
    c1*s3+c2*c3*s1, c1*c3-c2*s1*s3, s1*s2;
    -c3*s2, s2*s3, c2];

end
```

```
function plot_wav( stris, level, res)

    delta = pi/res;
    theta = 0:delta:pi;
    phi = 0:2*delta:2*pi;
    [phi,theta] = meshgrid(phi,theta);

    N = numel(theta);
    M = size(theta, 1);
    theta = reshape(theta, [N, 1]);
    phi = reshape(phi, [N, 1]);



    rm = getRfromT(theta,phi,stris,level);



    x = rm.*sin(theta).*cos(phi);
    y = rm.*sin(theta).*sin(phi);
    z = rm.*cos(theta);


    x = reshape(x, [M,M]);
    y = reshape(y, [M,M]);
    z = reshape(z, [M,M]);
    r = reshape(rm, [M,M]);

    surf(x,y,z,r);
    xlabel('x');
    ylabel('y');
    zlabel('z');
    grid off;
```

```
end


function sse = get_sse(a, test, level)

theta = test(:,1);
phi = test(:,2);
R_t = test(:,3);

R_a = getRfromT(theta,phi,a,level);

sse = (R_t-R_a).^2;
sse = mean(sse);

end


function R = getRfromT(theta,phi,stris,level)

x = sin(theta).*cos(phi-pi);
y = sin(theta).*sin(phi-pi);
z = cos(theta);

N = numel(theta);

%Interpolation.

%%From coeffs
R = zeros(size(theta));
for i = 1:N
    [scaleRi, waveRi] = rscale( stris, level, [x(i);y(i);z(i)], 0, 0);

    R(i) = (scaleRi+waveRi)/4;

end
```

```
end

%%
function [scaler, waver] = rscale( stris, level, p, scaler, waver)
%
% data = getPartitionDataPoint( partition, level, p)
%
% Get the value of the partition on level \a level in which \a p lies.

  for( i = 1 : numel( stris))

    if( 1 == checkPointInsideSTri( stris(i), p))

      if( getLevel( stris) < level)

        % recursively traverse tree
        childs = getChilds( stris(i));
        c = mean(stris(i).s_coeff);
        scaler = scaler + c/sqrt(getArea(stris(i)));

        cs = stris(i).w_coeffs;
        A1 = (sqrt(getArea(childs(2)))+sqrt(getArea(childs(3))) + ...
            sqrt(getArea(childs(4))))/3;
        %work out which subtriangle point is in
        %in t0
        if (1 == checkPointInsideSTri( childs(1), p))
            A0 = sqrt(getArea(childs(1)));
            waver = waver + cs(1,1)*A1/A0;
            waver = waver + cs(1,2)*A1/A0;
            waver = waver + cs(1,3)*A1/A0;
        end

        %in t1
        if (1 == checkPointInsideSTri( childs(2), p))
            a0 = getArea(childs(1));
            a1 = getArea(childs(2));
```

```
            a = (a0 - sqrt(a0^2+3*a0*a1))/(3*a0);
            waver = waver + cs(1,1)*(-2*a+1)/A1;
            waver = waver + cs(1,2)*a/A1;
            waver = waver + cs(1,3)*a/A1;
        end

        %in t2
        if (1 == checkPointInsideSTri( childs(3), p))
            a0 = getArea(childs(1));
            a1 = getArea(childs(2));
            a = (a0 - sqrt(a0^2+3*a0*a1))/(3*a0);
            waver = waver + cs(1,1)*a/A1;
            waver = waver + cs(1,2)*(-2*a+1)/A1;
            waver = waver + cs(1,3)*a/A1;
        end

        %in t3
        if (1 == checkPointInsideSTri( childs(4), p))
            a0 = getArea(childs(1));
            a1 = getArea(childs(2));
            a = (a0 - sqrt(a0^2+3*a0*a1))/(3*a0);
            waver = waver + cs(1,1)*a/A1;
            waver = waver + cs(1,2)*a/A1;
            waver = waver + cs(1,3)*(-2*a+1)/A1;
        end

        [scaler, waver] = rscale( childs, level, p, scaler, waver);



    end

    return;

  end  % end point is inside current stri
end  % end for all elements in partition
```

```
end


function basen = update_wavelet(base, stris, level, p, delR, is, child_arr, scale)
basen = stri(base);

for i = 1 : numel(stris)

    if( 1 == checkPointInsideSTri(stris(i), p))

      levelc = getLevel( stris);
      if(levelc  < level)
          scalel = (levelc)^2/level^3;

        % recursively traverse tree
        childs = getChilds(stris(i));

        %work out which subtriangle point is in
        %in t0 - center triangle: need to change all 3 wavelets
        if (1 == checkPointInsideSTri( childs(1), p))
            for j=1:3
                stris(i).w_coeffs(:,j) = stris(i).w_coeffs(:,j) + ...
                    scale*delR*scalel/3;
            end
        end

        %in t1
        if (1 == checkPointInsideSTri( childs(2), p))
            stris(i).w_coeffs(:,1) = stris(i).w_coeffs(:,1) + scale*delR*scalel;
        end

        %in t2
        if (1 == checkPointInsideSTri( childs(3), p))
            stris(i).w_coeffs(:,2) = stris(i).w_coeffs(:,2) + scale*delR*scalel;
        end
```

```
        %in t3
        if (1 == checkPointInsideSTri( childs(4), p))
            stris(i).w_coeffs(:,3) = stris(i).w_coeffs(:,3) + scale*delR*scalel;
        end

        is = [is, i];
        child_arr{end +1} = stris;

        basen = update_wavelet(basen, childs, level, p, scale*delR, is, ...
            child_arr, scale);
      else

          basen = set_children_level(basen, level, is, child_arr);
          return;
      end


    end
end


end

function basen = set_children_level(base, level, is, child_arr)
basen = stri(base);
if level > 1
    c = base;
    for l = 1:level-2
        li = is(l);
        c = getChilds(c(li));
    end
    li = is(level-1);
    bc = child_arr{level-1};
    c(li).w_coeffs = bc.w_coeffs;
    c(li).childs = child_arr{level};
```

```
    child_arr{level-1} = c;
    basen = set_children_level(basen, level-1, is, child_arr);
end

li = is(1);
bc = child_arr{1};
basen(li).w_coeffs = bc(li).w_coeffs;
basen(li).childs = child_arr{2};

end
```

8.4 Additional figures

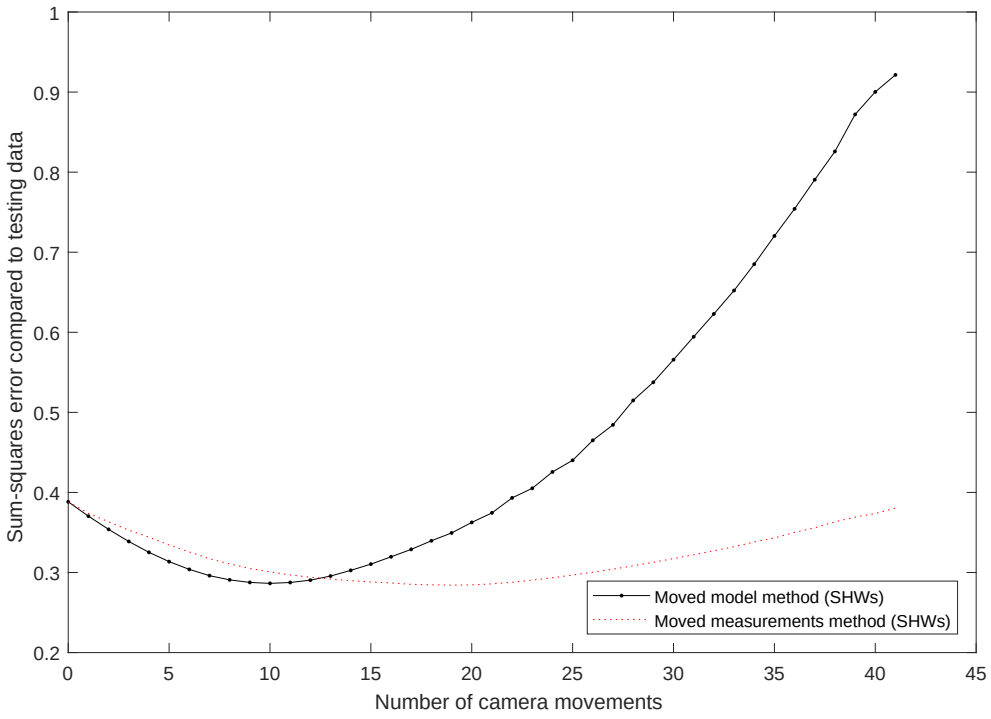
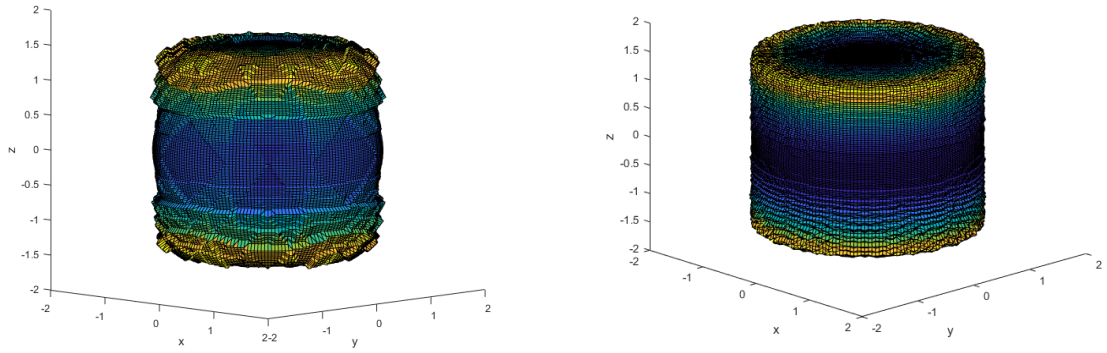


Figure 18: Sum-squares error for SHWs with step size 500, all other parameters as in Tables 1 and 3.



(a) Model found for SHWs with level = 4, nc = 1024

(b) Model found for SHWs with level = 7, nc = 4096

Figure 19: Models obtained using points sampled from the environment with no update step

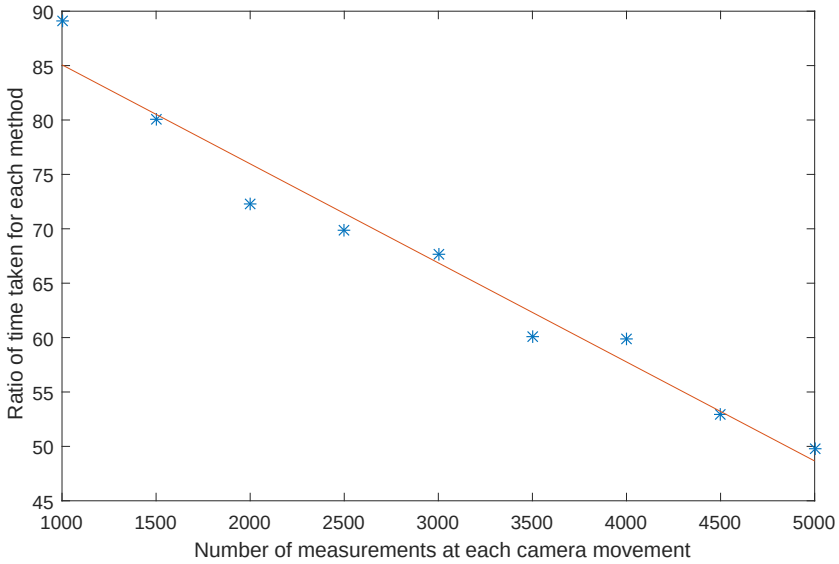


Figure 20: Ratio of time taken for the moved model method to time taken for the moved measurements method for varying numbers of measurements at each camera movement. All other parameters are as listed in Tables 1 and 2. The ratios are averaged over 41 camera movements. The line of best fit $-0.0091m + 94.175$ is also shown.

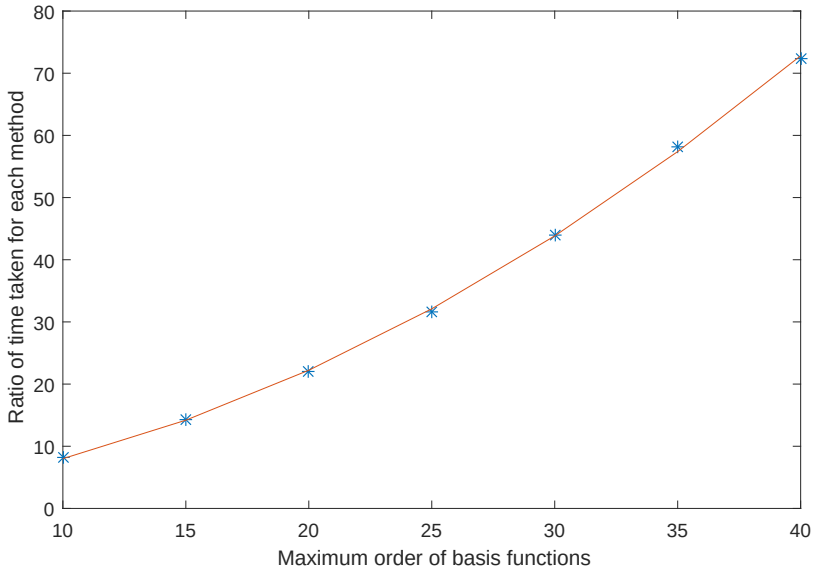


Figure 21: Ratio of time taken for the moved model method to time taken for the moved measurements method for varying maximum order of basis functions. 2000 measurements were taken at each camera movement. All other parameters are as listed in Tables 1 and 2. The ratios are averaged over 41 camera movements. The line of best fit $0.0369n^2 + 0.3169n + 1.1451$ is also shown.

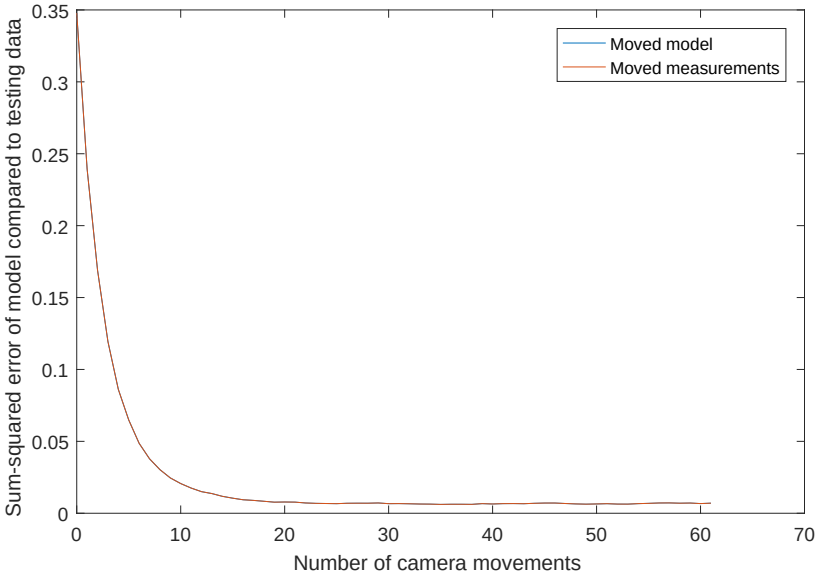


Figure 22: Sum-squares error for a measurement uncertainty of 1, with $N = 30$. Note that this corresponds to 61 batches of measurements. All other parameters are as listed in Table 1 and 3.

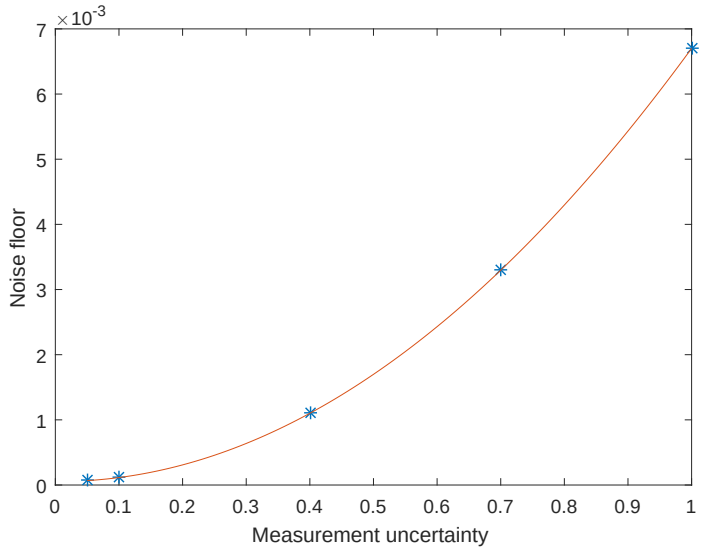


Figure 23: Noise floor found by averaging sum-squares error between 29 and 61 camera movements. $N = 30$ and measurement uncertainty varied. All other parameters are as listed in Table 1 and 3. The line of best fit $0.0067u^2 - 0.0001u + 0.0001$ is also shown, where u is the standard deviation of the measurements.