Noname manuscript No.
(will be inserted by the editor)

A Visual Odometry Algorithm using a Vector Model of the Static Scene and Camera

George L. Sechu

Received: date / Accepted: date

Abstract This paper presents an approach to visual odometry that relies on constructing rigid body models of the static scene and camera. Visual odometry plays an important role in Simultaneous Localisation and Mapping (SLAM) where it provides local estimates of the camera's trajectory from visual imagery alone. In this work, the static scene and the camera viewing it are each represented as a set of orthonormal vectors and a position vector. The position vector represents the position of the entity the model is refering to as measured from some reference. The orthornormal set represents the entity's orientation. Scene models are constructed for each RGB-D image frame in an input sequence. Since feature matching is an expensive processing operation, matching is not done on an entire image frame. Instead the frames are split into a programmable number of sections, and matching is perfomed on these sections. Frame splitting helps improve the estimation rate since the estimation may be completed before the entire image is scanned and matched. The motion estimation step of this approach differs from the majority of existing visual odometry algorithms which involve minimising the rigid body transformation between frames. In this approach, motion is estimated by comparing scene and camera models accross frames and refined using a moving average technique.

Keywords visual odometry · egomotion · rgbd · slam

George L. Sechu
University of Cape Town, Electrical Engineering, Cape Town South Africa 7700
E-mail: george.sechu@uct.ac.za

1 Introduction

Visual odometry is the process of estimating the position or motion of a vehicle using only visual imagery. It is the part of SLAM or specifically Visual SLAM that is involved with egomotion estimation. A SLAM system can be thought of as a larger system of which Visual Odometry is the part that computes local motion. The goal of the larger SLAM system in general is obtain a global, consistent estimate of the robot path (Scaramuzza and Fraundorfer, 2011). One of the earliest works on visual odometry was done by Moravec on the Stanford cart (Huang et al, 2011; Moravec, 1980). Moravec's work is used as the basic motion estimation pipeline even today. Most of the early research in Visual Odometry was done for planetary rovers to provide them with the capability to measure 6 degree of freedom motion in the presence of wheel slippage (Scaramuzza and Fraundorfer, 2011). The approach presented in this paper is a sparse visual odometry approach. As such it is similar to methods (Nister et al, 2004; Konolige et al, 2007; Huang et al, 2011; Engel et al, 2012; Weiss et al, 2012) and follows the basic pipeline: feature detection followed by feature matching then motion estimation.

The main contribution of this approach is an alternative to minimising the rigid body transformation or the feature reprojection error between two image frames. We instead model the scene and camera using an orthonormal vector set representing rotation and an another vector representing position. These models are constructed for each RGB-D frame, and then compared across frames to calculate camera motion.

The capturing device to be used is an RGB-D camera such as the Microsoft Kinect (Microsoft, 2014). To test the validity of the algorithm and evaluate it's perfomance, a dataset from the TUM RGB-D Benchmark

(Sturm et al, 2012; TUM, 2014) was used in test runs under different configurations.

2 Rigid Body Model

The computer program that implements this approach maintains model objects for the scene and the camera. Simply put, each of the model objects is a set of 4 vectors. The first vector in such a model represents the amount of translation as measured from some coordinate reference. The last 3 vectors are an orthonormal set, vectors of magnitude 1 which are orthogonal and represent the rigid body's orientation.

The 4 vectors in the rigid body model have been defined as the **Origin**, the **X Axis**, the **Y Axis** and the **Z Axis**. The Origin is a position vector representing the position of the entity the model is referring to. The X, Y and Z Axes are an orthonormal set of direction vectors representing rotation.

The scene rigid body model, denoted by $\boldsymbol{G}$, is shown in parts below:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G_O} \\ \boldsymbol{G_X} \\ \boldsymbol{G_Y} \\ \boldsymbol{G_Z} \end{bmatrix} \tag{1}$$

Where $\boldsymbol{G_O}$ is the scene origin and $\boldsymbol{G_X}$,$\boldsymbol{G_Y}$,$\boldsymbol{G_Z}$ are the scene x, y and z axes (orthonormal set).

Similarly, the camera rigid body model, denoted by $\boldsymbol{O}$, is shown in parts below:

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{O_O} \\ \boldsymbol{O_X} \\ \boldsymbol{O_Y} \\ \boldsymbol{O_Z} \end{bmatrix} \tag{2}$$

The approach works by capturing frames in an RGB-D sequence $M_0, M_1, M_2 ... M_{N-1}, M_N$ and then processing them 2 at a time. The previous and latest frames captured (M_{N-1} and M_N respectively) are the ones currently processed. The scene and camera model conventions are either appended the subscript $N-1$ or N to denote a model in the previous or latest frames respectively. We therefore have for the previous frame:

$$\boldsymbol{G_{N-1}} = \begin{bmatrix} \boldsymbol{G_{ON-1}} \\ \boldsymbol{G_{XN-1}} \\ \boldsymbol{G_{YN-1}} \\ \boldsymbol{G_{ZN-1}} \end{bmatrix} \tag{3}$$

$$\boldsymbol{O_{N-1}} = \begin{bmatrix} \boldsymbol{O_{ON-1}} \\ \boldsymbol{O_{XN-1}} \\ \boldsymbol{O_{YN-1}} \\ \boldsymbol{O_{ZN-1}} \end{bmatrix} \tag{4}$$

And for the latest frame:

$$\boldsymbol{G_N} = \begin{bmatrix} \boldsymbol{G_{ON}} \\ \boldsymbol{G_{XN}} \\ \boldsymbol{G_{YN}} \\ \boldsymbol{G_{ZN}} \end{bmatrix} \tag{5}$$

$$\boldsymbol{O_N} = \begin{bmatrix} \boldsymbol{O_{ON}} \\ \boldsymbol{O_{XN}} \\ \boldsymbol{O_{YN}} \\ \boldsymbol{O_{ZN}} \end{bmatrix} \tag{6}$$

The model can be visualised in 3D as a group of coordinate axes and a point as shown in Figure 1.

This setup is sufficient to fully represent the position and orientation of any rigid body in a 3D coordinate space. For zero translation, $\boldsymbol{G_O}$ is a null vector. For zero rotation, $\boldsymbol{G_X}$, $\boldsymbol{G_Y}$, and $\boldsymbol{G_Z}$ point in the same directions as the x, y and z axes of the coordinate reference.

Both the camera and the static scene are rigid bodies and are therefore represented in the same way. The difference is that values in the camera model are as measured from the world coordinate reference, which may be given to the system using an external input device or initialised to the camera's current position and orientation. The values in the scene model, on the other hand are as measured from the camera's coordinate reference.

3 Feature Detection and Matching

Suppose we have a sequence of RGB-D frames captured with an RGB-D camera. Each frame contains two 640 x 480 matrices of pixel data, one for colour and one for depth. These frames are processed two at a time, comparing the most recently captured frame with the previous one at each processing step. In Figure 2, the frames labeled $N-1$ and N are the two frames currently being processed. The previous frame and current frame respectively.

First, each frame is split into a given number of sections TS. TS must be an integer value such that it is possible to split the 640 x 480 pixel grid almost evenly about the rows and columns. Each of these sections becomes a candidate for use in creating the scene model. A random section is selected and searched for features using a SURF (Bay et al, 2006) feature detector. The depth maps in the frames will contain some pixels with no depth values. The feature detector is instructed to avoid these zones in the colour images by passing it a mask image matrix with zero values at the pixels that should not be searched.

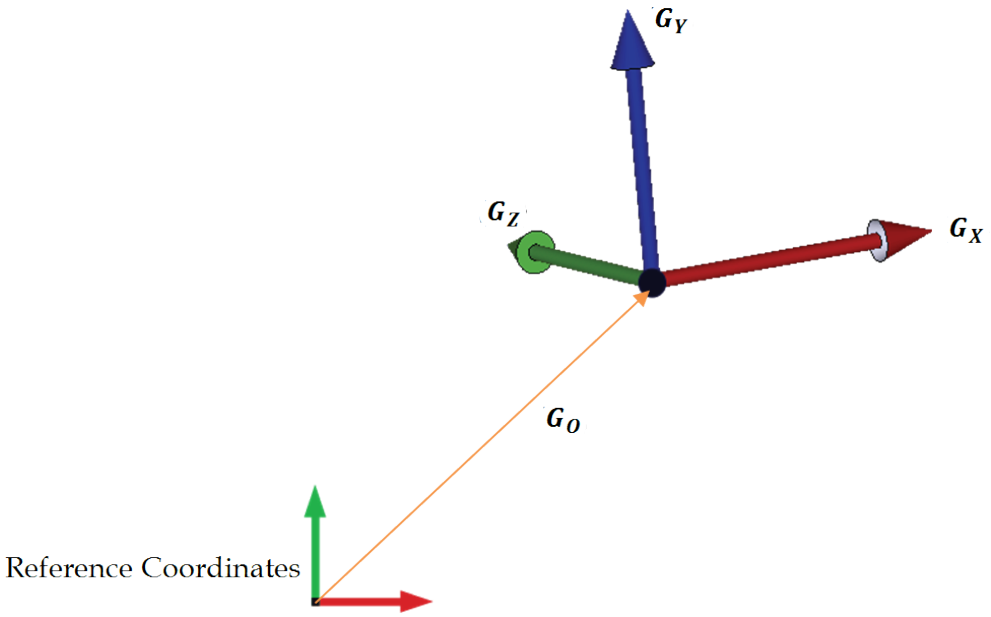

Fig. 1 An illustration of the rigid body model in 3D. G_0 is the origin and $[G_X, G_Y, G_Z]$ is the orthonormal set. G has been used here as an example but same model is used to represent the both the scene (G) and the camera (O).

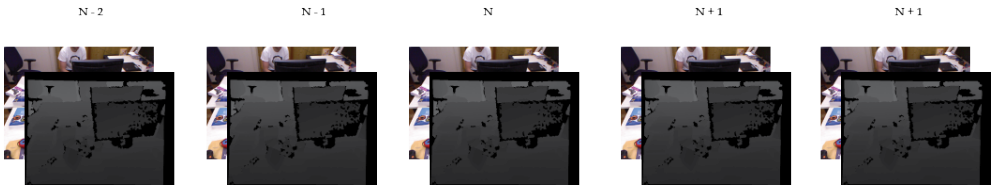

Fig. 2 An RGB-D sequence. Each frame contains a colour image and a depth map. Frames are processed 2 at a time. The currently processed frames are denoted M_{N-1} and M_N (previous and latest frame respectively). After each iteration, a new previous frame is read from the sequence and the latest one is dropped.

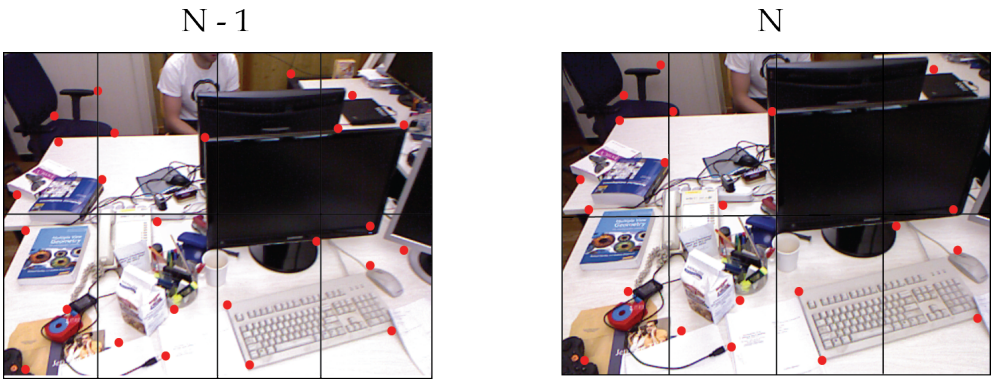

Fig. 3 The previous and latest frames split into TS = 8 sections. 3 of these sections that contain a minimum of FS matched features of sufficient quality, will be used to construct the scene model. If less than 3 sections meet the requirements then the system fails to compute egomotion.

Note that detected features have image pixel coordinates. This does not imply however, that their coordinates will be whole numbers. A feature can exist close to the intersection of many pixels hence their coordinate values are floating point numbers.

We define a variable FS, the number of features to be used per scene section to construct the scene model. The number of features detected in the randomly selected section must be at least FS. If a sufficient amount of features are detected, feature matching is performed on these image sections between the previous and current frames. The matching operation results in a list of matches, each containing the pixel coordinates of a feature in the previous and latest frames.

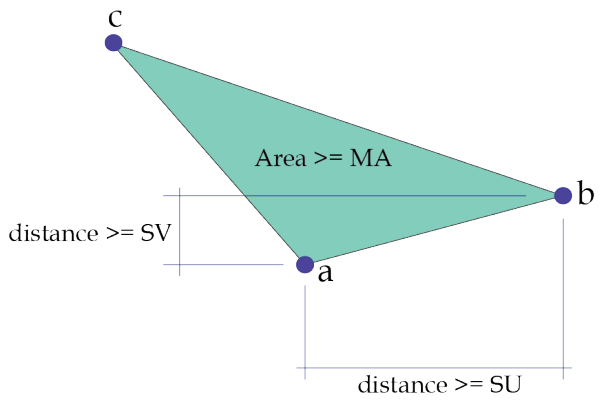

Fig. 4 Ensuring sufficient separation of scene sections. ***a***, ***b*** and ***c*** are in image coordinates and must be sufficiently distant from each other so as to not be considered a single point or line. One of the 2 distance inequalities must be met and the area inequality must also be met.

Every match also carries a score representing its quality. The score is a floating point number called the match distance. The lower the match distance the higher the quality of the match therefore we set a minimum value MQ for it.

We need to find a total of FS matches with match distances less than MQ in a section for it to qualify as one of the scene sections ***a***, ***b*** and ***c*** used to construct the scene model. If we only find less than FS matches of match distance less than MQ, we discard the section and move on to the next random section. If we find at least FS matches of match distance less than MQ, the section is labeled scene section ***a***.

The next steps are to search for ***b*** and then ***c*** and are done in the same way as ***a*** but with some additional constraints. To ensure sufficient spacing between the 3 scene sections, we set minimum values of pixel separation SU and SV in the horizontal and vertical directions respectively for the centers of scene sections ***a*** and ***b***, and a minimum area MA of the resulting triangle ***abc***. This concept is illustrated in Figure 4

The constraints applied can be summerised as follows:

Constraint 1: An ***ab*** pair will be accepted if their horizontal separation is not less than SU or their vertical separation is not less than SV. This means that if any of the two distance inequalities shown in Figure 4 are met, an ***ab*** pair will be accepted.

Constraint 2: An ***abc*** triad will be accepted if ***ab*** satisfies constraint 1 and the area of the triangle ***abc*** is not less than MA.

To reduce the complexity of tuning the overall system with configuration parameters, the value of MA is calculated at runtime using equation 7.

$$MA = \frac{1}{2} \cdot SU \cdot SV \tag{7}$$

This is essentially the area of a right triangle with side lengths SU and SV. The values of SU and SV are supplied to the program as configuration parameters.

Constraints 1 and 2 are applied to candidate sections. The points checked against the constraints are in image coordinates. Any points in the scene that are separated in image coordinates will be proportionally separated in 3D world coordinates. This is because each pixel in the image represents a single light ray that made it onto the camera's sensing plane. Each light ray entering the camera comes from a scene point with 3D coordinates distinct from any other scene point whose light ray also entered the camera. We should therefore expect to get sufficient spacing between the world coordinates of ***a***, ***b*** and ***c*** as well.

4 Creating the Scene Model

To construct the scene model, 3 sufficiently separated sections with at least FS matched features each must have been found in the previous and latest frame. We calculate the center pixel coordinates of each of the 3 sections by averaging the pixel coordinates of the matches within them.

A depth value is then attached to each of the sections by averaging the depth values of pixel positions in the neighbourhood of the matched features. If any one of the three sections has missing depth by any chance, it is discarded and the system searches the frame for other sections.

Once all three sections have a depth value, their 3D coordinates can be calculated from the following equations.

$$Z = depth \tag{8}$$

$$X = \frac{u - c_x}{f_x} \cdot Z \tag{9}$$

$$Y = \frac{u - c_y}{f_y} \cdot Z \tag{10}$$

Where $[X, Y, Z]$ are the 3D coordinates, $[u, v]$ are the image coordinates of the scene section, $[c_x, c_y]$ is the location of the camera's optical center in image coordinates and $[f_x, f_y]$ are the focal lengths of the camera.

The resulting 3D points form a virtual triangle encoded with information about the scene's position and orientation as measured from the camera. After 3D coordinates of scene sections **a**, **b** and **c** have been calculated. The next step is to compute the scene model.

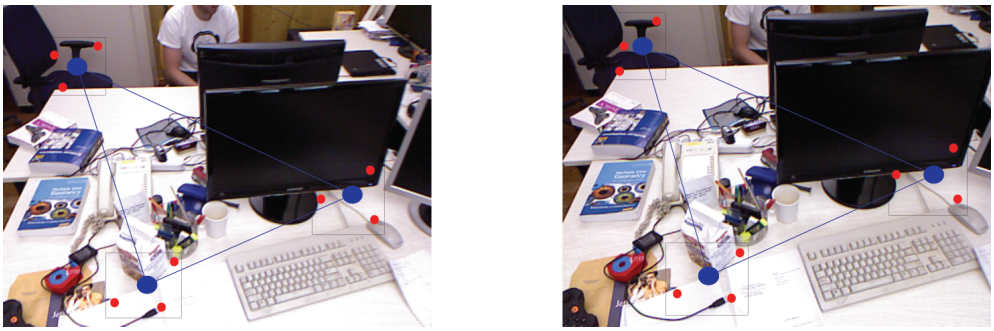

Fig. 5 Pixel positions of 3 scene sections with their centers. The center pixel positions are calculated by averaging the pixel positions of the matched features within each section.

Fig. 6 Depth is calculated from neighbour image patches. Note that not the entire section is used to calculate depth, only the neighbourhood of the matched features within the section.

The scene model $\boldsymbol{G}$ is constructed from $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ as follows:

The origin $\boldsymbol{G_O}$ is the point $\boldsymbol{a}$.

The x axis $\boldsymbol{G_X}$ is the unit vector in the direction $\boldsymbol{ab}$.

The z axis $\boldsymbol{G_Z}$ is the unit vector in the direction of the cross product of $\boldsymbol{ab}$ and $\boldsymbol{ac}$.

The y axis $\boldsymbol{G_Y}$ is the cross product of the x and z axes.

This concept is illustrated in Figure 7.

As the camera moves, it views the scene model at different positions and orientations so it appears to the camera as though the scene model is moving. Computing this apparent motion will ultimately lead to computing the camera's motion.

5 Creating the Camera Model

Assuming we already have the odometry estimates for the previous frame M_{N-1}. The translation of the camera in the previous frame is represented by the 3D vector $\boldsymbol{s_{N-1}}$ and the rotation is a unit quaternion $\boldsymbol{q}$. The camera model $\boldsymbol{O_{N-1}}$ in the previous frame is constructed as shown in the following equations:

$$\boldsymbol{O_{ON-1}} = \boldsymbol{s_{N-1}} \tag{11}$$

$$\boldsymbol{O_{XN-1}} = \boldsymbol{q_{N-1}} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \boldsymbol{q'_{N-1}} \tag{12}$$

$$\boldsymbol{O_{YN-1}} = \boldsymbol{q_{N-1}} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \boldsymbol{q'_{N-1}} \tag{13}$$

$$\boldsymbol{O_{ZN-1}} = \boldsymbol{q_{N-1}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \boldsymbol{q'_{N-1}} \tag{14}$$

6 Rotation Calculation

At this point we have constructed the scene models $\boldsymbol{G_{N-1}}$ and $\boldsymbol{G_N}$ for the previous and latest frames respectively. Translational motion of the camera only affects the origin $\boldsymbol{G_{ON}}$ whereas rotation only affects the orthonormal vectors $\boldsymbol{G_{XN}}$,$\boldsymbol{G_{YN}}$ and $\boldsymbol{G_{ZN}}$. In this section, methods for calculating the rotation axis and rotation angle are presented.

6.1 Rotation Axis

According to Euler's rotation theorem, any number of rigid body rotations about a fixed point are equivalent

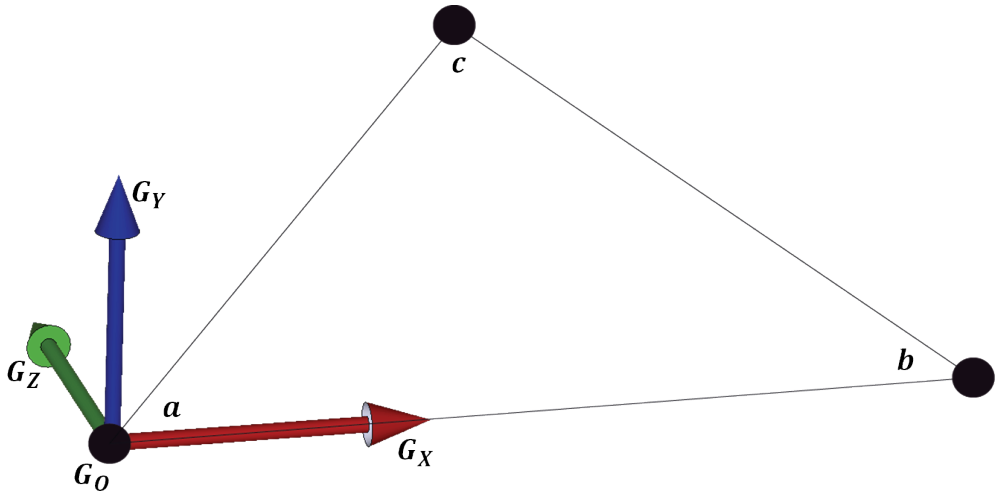

Fig. 7 Scene model constructed from the centers of 3 scene sections **a**, **b** and **c**. The origin $\boldsymbol{G_O}$ is the point $\boldsymbol{a}$. The x axis $\boldsymbol{G_X}$ is the unit vector in the direction $\boldsymbol{ab}$. The z axis $\boldsymbol{G_Z}$ is the unit vector in the direction of the cross product of $\boldsymbol{ab}$ and $\boldsymbol{ac}$. The y axis $\boldsymbol{G_Y}$ is the cross product of the x and z axes.

Fig. 8 Scene models in the previous and latest frames. As the camera moves, it views the scene from a different pose therefore the scene appears to move relative to the camera. The scene model follows this relative motion.

to a single rotation about an axis through the point. Rotation of the scene relative to the camera's viewpoint is equivalent to the rotation of the scene model. Since we are dealing with a rigid body rotation, we expect that there is an axis of rotation and an angle.

The 3 unit vectors $\boldsymbol{G_{XN-1}}$, $\boldsymbol{G_{YN-1}}$ and $\boldsymbol{G_{ZN-1}}$ in the model touch the surface of an abstract sphere with radius 1 unit. During rotation, the heads of these vectors will move on the surface of the unit sphere along circular arcs and end up on the sphere surface locations touched by $\boldsymbol{G_{XN}}$,$\boldsymbol{G_{YN}}$ and $\boldsymbol{G_{ZN}}$. The three orthonormal vectors in the previous frame model are therefore connected to those in the latest frame model by circular arcs. These 3 arcs become 3 full circles if the rotation angle is 360 degrees.

In addition to the 3 circular arcs, each vector pair is also connected by a difference vector calculated by applying vector subtraction to the paired unit vectors. We call these vector differences $\boldsymbol{d_X}$,$\boldsymbol{d_Y}$ and $\boldsymbol{d_Z}$ where:

$$\boldsymbol{d_X} = \boldsymbol{G_{XN}} - \boldsymbol{G_{XN-1}} \tag{15}$$

$$\boldsymbol{d_Y} = \boldsymbol{G_{YN}} - \boldsymbol{G_{YN-1}} \tag{16}$$

$$\boldsymbol{d_Z} = \boldsymbol{G_{YN}} - \boldsymbol{G_{YN-1}} \tag{17}$$

$\boldsymbol{d_X}$, $\boldsymbol{d_Y}$ and $\boldsymbol{d_Z}$ are illustrated on a scene model in Figure 9.

Due to the nature of rigid body rotations, the 3 arcs formed have to be parallel because every point in the rigid body moves in a perfect circle around a single axis resulting in circles that look like the parallels of the earth. The 3 vector differences are not parallel but they lie on 3 parallel planes. The 3 parallel planes intersect the abstract sphere at 3 circles, each containing one of the 3 circular arcs.

The axis of rotation will be a line running through the centers of the 3 circles. The direction of this line is a vector that fully describes the rotation axis. Since the circles and planes are all parallel, a normal to these planes is also the axis of rotation. The 3 difference vectors also lie on these planes, therefore the rotation axis

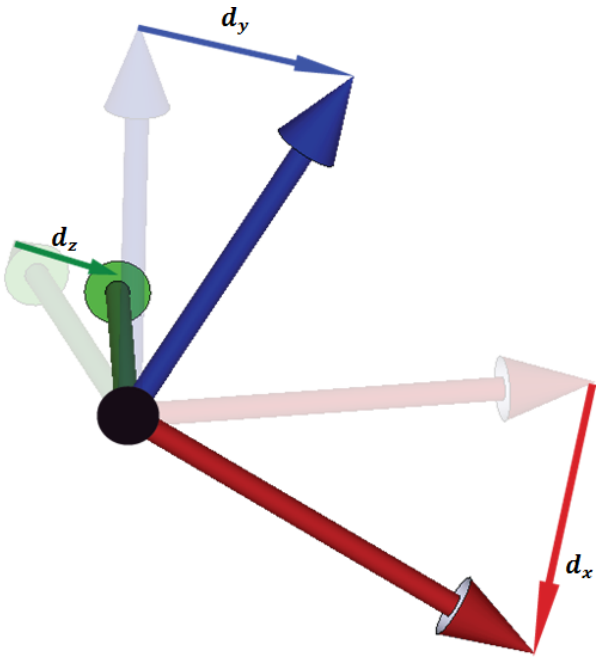

Fig. 9 Rotation of the scene model showing the 3 difference vectors between scene models of the previous and latest frames. Note that translational motion does not in any way affect $\boldsymbol{d_X}$, $\boldsymbol{d_Y}$ and $\boldsymbol{d_Z}$.

is the cross product of any two of these difference vectors.

Since we have three difference vectors, we have 3 cross products. Ultimately we have to choose which cross product to take as the rotation axis. The answer depends on the rotation itself. Consider a situation where the axis of rotation is parallel or close to parallel with one of the three orthonormal vectors $\boldsymbol{G_X}$,$\boldsymbol{G_Y}$ and $\boldsymbol{G_Z}$. The vector it is parallel to will experience little or no motion during rotation and the resulting difference vector will be close to the zero vector in magnitude. Using this small difference vector amplifies the error scale since it will result in a very small vector with noise values similar to its magnitude. On the other hand, the other 2 orthonormal vectors will be far from parallel with the axis of rotation, since the rotation axis is almost parallel to a vector that is orthogonal to both of them. The remaining two vectors will experience large movement for the rotation and the cross product of their difference vectors can safely be used as the axis of rotation. The problem is solved by simply taking the largest of the 3 cross products shown below as the rotation axis.

$$RotationAxis1 = \boldsymbol{d_X} \times \boldsymbol{d_Y} \tag{18}$$

$$RotationAxis2 = \boldsymbol{d_X} \times \boldsymbol{d_Z} \tag{19}$$

$$RotationAxis3 = \boldsymbol{d_Y} \times \boldsymbol{d_Z} \tag{20}$$

The rotation axis calculated so far has an ambiguous direction since cross products are anticommutative. To produce a consistent direction and angle, we find the components of $\boldsymbol{G_{XN-1}}$, $\boldsymbol{G_{YN-1}}$, $\boldsymbol{G_{ZN-1}}$ and $\boldsymbol{G_{XN}}$, $\boldsymbol{G_{YN}}$, $\boldsymbol{G_{ZN}}$ that are perpendicular to the rotation axis. These vertical components are the radiuses of the circles mentioned above and are labeled $\boldsymbol{r_{XN-1}}$,$\boldsymbol{r_{YN-1}}$,$\boldsymbol{r_{ZN-1}}$ and $\boldsymbol{r_{XN}}$,$\boldsymbol{r_{YN}}$,$\boldsymbol{r_{ZN}}$. Figure 10 highlights the component of $\boldsymbol{G_{YN}}$ perpendicular to the rotation axis.

The cross products $\boldsymbol{r_{XN-1}} \times \boldsymbol{r_{XN}}$, $\boldsymbol{r_{YN-1}} \times \boldsymbol{r_{YN}}$ and $\boldsymbol{r_{ZN-1}} \times \boldsymbol{r_{ZN}}$ all yield the rotation axis in a consistent direction since we take the cross product from $N-1$ to N.

$$RotationAxis1 = \boldsymbol{r_{XN-1}} \times \boldsymbol{r_{XN}} \tag{21}$$

$$RotationAxis2 = \boldsymbol{r_{YN-1}} \times \boldsymbol{r_{YN}} \tag{22}$$

$$RotationAxis3 = \boldsymbol{r_{ZN-1}} \times \boldsymbol{r_{ZN}} \tag{23}$$

The largest of the three cross products is taken as the axis of rotation and normalised to get the unit axis of rotation $\boldsymbol{u}$.

6.2 Rotation Angle

After obtaining the rotation axis in a consistent direction, the next step is to calculate the rotation angle. To work efficiently with quaternions, we calculate the values of $\cos\frac{1}{2}\theta$ and $\sin\frac{1}{2}\theta$ where θ is the rotation angle.

In Figure 11 $\boldsymbol{r_N}$ and $\boldsymbol{r_{N-1}}$ are the largest of the 3 radius pairs illustrated previously. There may be smaller pairs or even a pair with zero vectors. The largest pair is chosen to calculate the sine and cosine of half the rotation angle.

From Figure 11 the value of $\sin\theta$ can be calculated from the cross product of $\boldsymbol{r_N}$ and $\boldsymbol{r_{N-1}}$.

$$\sin\theta = \frac{\boldsymbol{r_{N-1}} \times \boldsymbol{r_N}}{|\boldsymbol{r_{N-1}}|\,|\boldsymbol{r_N}|} \tag{24}$$

The *adjacent* is the projection of $\boldsymbol{r_N}$ onto the vector sum of $\boldsymbol{r_{N-1}}$ and $\boldsymbol{r_N}$.

$$adjacent = \boldsymbol{r_N} \cdot \frac{\boldsymbol{r_{N-1}} + \boldsymbol{r_N}}{|\boldsymbol{r_{N-1}} + \boldsymbol{r_N}|} \tag{25}$$

The *hypotenuse* is the length of $\boldsymbol{r_{N-1}}$ or $\boldsymbol{r_N}$.

$$hypotenuse = |\boldsymbol{r_N}| \tag{26}$$

The cosine of the half angle can then be calculated using simple trigonometry:

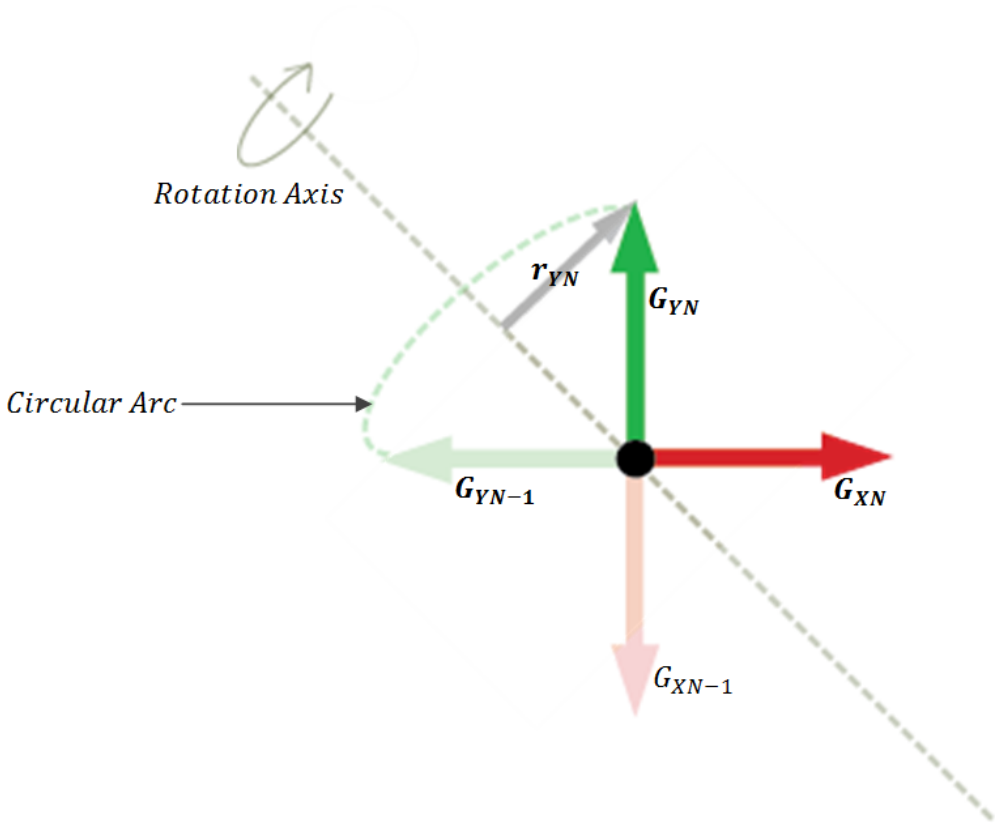

Fig. 10 A scene model rotation highlighting the component of G_{YN} perpendicular to the rotation axis. G_{XN} and G_{ZN} also have components perpendicular to the rotation axis. The three components are r_{XN},r_{YN} and r_{ZN}. The largest of the 3 cross products involving these components yields the rotation axis in a consistent direction.

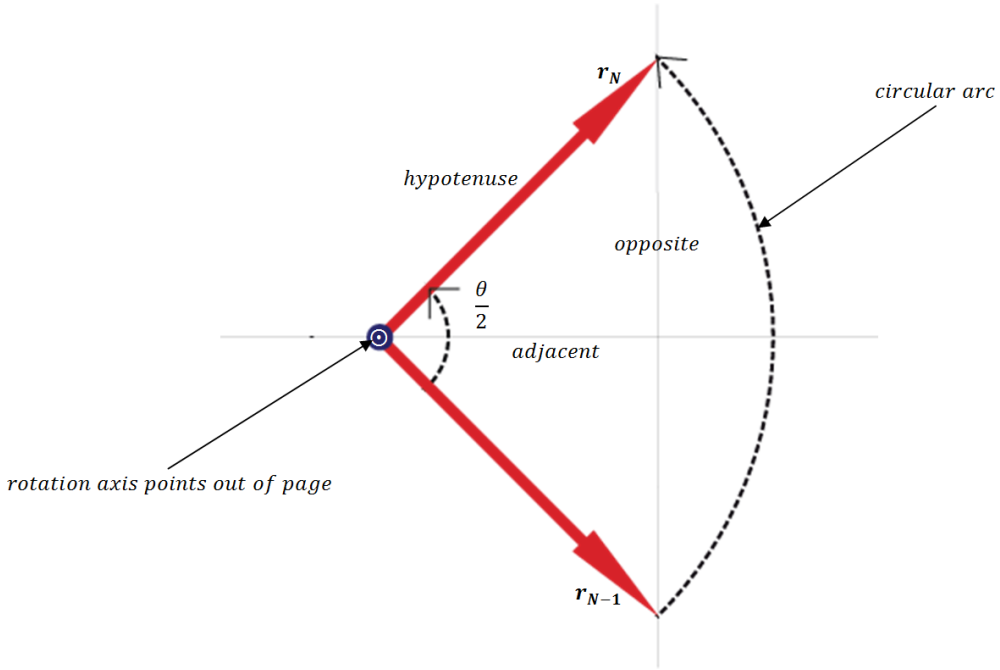

Fig. 11 Calculating the sine and cosine of the half angle of rotation. In this illustration, the rotation axis points out of the page therefore the circular arcs and the planes are parallel to the page. The sine and consine of $\frac{1}{2}\theta$ are calculated using trigonometry.

$$\cos\frac{1}{2}\theta = \frac{adjacent}{hypotenuse} \tag{27}$$

$$\cos\frac{1}{2}\theta = \frac{\boldsymbol{r_N}\cdot(\boldsymbol{r_{N-1}}+\boldsymbol{r_N})}{\boldsymbol{r_{N-1}}+\boldsymbol{r_N}} \tag{28}$$

And from the double angle formulas we have:

$$\sin\theta = 2\cos\frac{1}{2}\theta\sin\frac{1}{2}\theta \tag{29}$$

Therefore:

$$\sin\frac{1}{2}\theta = \frac{\sin\theta}{2\cos\frac{1}{2}\theta} \tag{30}$$

Now that we have the sine and cosine of the half rotation angle, and the axis of rotation $\boldsymbol{u}$ calculated previously, we apply these values to the unit quaternion $\boldsymbol{q}$ used to represent rotation as follows:

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{31}$$

$$\boldsymbol{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{32}$$

Where:

$$q_w = \cos\frac{1}{2}\theta \tag{33}$$

$$q_x = u_x\sin\frac{1}{2}\theta \tag{34}$$

$$q_y = u_y\sin\frac{1}{2}\theta \tag{35}$$

$$q_z = u_z\sin\frac{1}{2}\theta \tag{36}$$

7 Translation Calculation

At this point we have obtained a rotation estimate. The translation will be estimated using simple vector operations. The illustration in Figure 12 is in 2D for ease of visualisation but the underlying concept also works 3D.

In Figure 12, $\boldsymbol{O_{N-1}}$ and $\boldsymbol{O_N}$ are the camera models at the previous and latest frames respectively. The camera has translated and rotated in the world frame and now observes the scene model from a different location and pose. The scene model is fixed in the world reference but appears to move relative to the camera.

From the previous iteration, we know the global position and orientation at $\boldsymbol{O_{N-1}}$. $\boldsymbol{O_N}$ is the camera model in the latest frame. We have already calculated its global rotation and we now seek its global translation. The vector $\boldsymbol{\Delta O}$ is the displacement vector between the models $\boldsymbol{O_{N-1}}$ and $\boldsymbol{O_N}$. The global translation can be found by computing the vector sum of the coordinates at $\boldsymbol{O_{N-1}}$ and $\boldsymbol{\Delta O}$. From Figure 12 the value of $\boldsymbol{\Delta O}$ is also the vector difference between $\boldsymbol{\tau_N}$ and $\boldsymbol{\tau_{N-1}}$ and can be calculated using the following equation:

$$\boldsymbol{\Delta O} = \boldsymbol{\tau_{N-1}} - \boldsymbol{\tau_N} \tag{37}$$

where:

$$\boldsymbol{\tau_{N-1}} = \begin{bmatrix} \boldsymbol{\tau_{N-1[X]}} \\ \boldsymbol{\tau_{N-1[Y]}} \\ \boldsymbol{\tau_{N-1[Z]}} \end{bmatrix} \tag{38}$$

$$\boldsymbol{\tau_N} = \begin{bmatrix} \boldsymbol{\tau_{N[X]}} \\ \boldsymbol{\tau_{N[Y]}} \\ \boldsymbol{\tau_{N[Z]}} \end{bmatrix} \tag{39}$$

The z components of $\boldsymbol{\tau_N}$ and $\boldsymbol{\tau_{N-1}}$ can be thought of as pointing into the page.

The magnitude of the vectors $\boldsymbol{\tau_{N[X]}}$, $\boldsymbol{\tau_{N[Y]}}$ and $\boldsymbol{\tau_{N[Z]}}$ are values $\left|\boldsymbol{G_{O[X]}}\right|$, $\left|\boldsymbol{G_{O[Y]}}\right|$ and $\left|\boldsymbol{G_{O[Z]}}\right|$ which are the components of the scene model's origin $\boldsymbol{G_O}$ as observed from the camera at frame M_N. The directions of $\boldsymbol{\tau_{N[X]}}$, $\boldsymbol{\tau_{N[Y]}}$ and $\boldsymbol{\tau_{N[Z]}}$ are the 3 orthonormal vectors $\boldsymbol{O_{XN}}$, $\boldsymbol{O_{YN}}$ and $\boldsymbol{O_{ZN}}$ of the scene model at M_N. The vectors $\boldsymbol{\tau_{N-1[X]}}$, $\boldsymbol{\tau_{N-1[Y]}}$ and $\boldsymbol{\tau_{N-1[Z]}}$ are therefore calculated for frame M_{N-1} by scaling the observed scene model orthonormal vectors $\left|\boldsymbol{G_{ON-1[X]}}\right|$, $\left|\boldsymbol{G_{ON-1[Y]}}\right|$ and $\left|\boldsymbol{G_{ON-1[Z]}}\right|$.

$$\boldsymbol{\tau_{N-1[X]}} = \boldsymbol{O_{XN-1}}\cdot\left|\boldsymbol{G_{ON-1[X]}}\right| \tag{40}$$

$$\boldsymbol{\tau_{N-1[Y]}} = \boldsymbol{O_{YN-1}}\cdot\left|\boldsymbol{G_{ON-1[Y]}}\right| \tag{41}$$

$$\boldsymbol{\tau_{N-1[Z]}} = \boldsymbol{O_{ZN-1}}\cdot\left|\boldsymbol{G_{ON-1[Z]}}\right| \tag{42}$$

The vectors $\boldsymbol{O_{XN-1}}$, $\boldsymbol{O_{YN-1}}$ and $\boldsymbol{O_{ZN-1}}$ are already known from the previous odometry estimate. To calculate $\boldsymbol{\tau_{N[X]}}$, $\boldsymbol{\tau_{N[Y]}}$ and $\boldsymbol{\tau_{N[Z]}}$ for M_N, the vectors $\boldsymbol{O_{XN}}$, $\boldsymbol{O_{YN}}$ and $\boldsymbol{O_{ZN}}$ are first obtained by rotating $\boldsymbol{O_{XN-1}}$, $\boldsymbol{O_{YN-1}}$ and $\boldsymbol{O_{ZN-1}}$ by the rotation value already computed.

$$\boldsymbol{O_{XN}} = \boldsymbol{q_{N-1}}\times\boldsymbol{O_{XN-1}}\times\boldsymbol{q'_{N-1}} \tag{43}$$

$$\boldsymbol{O_{YN}} = \boldsymbol{q_{N-1}}\times\boldsymbol{O_{ZN-1}}\times\boldsymbol{q'_{N-1}} \tag{44}$$

$$\boldsymbol{O_{ZN}} = \boldsymbol{q_{N-1}}\times\boldsymbol{O_{ZN-1}}\times\boldsymbol{q'_{N-1}} \tag{45}$$

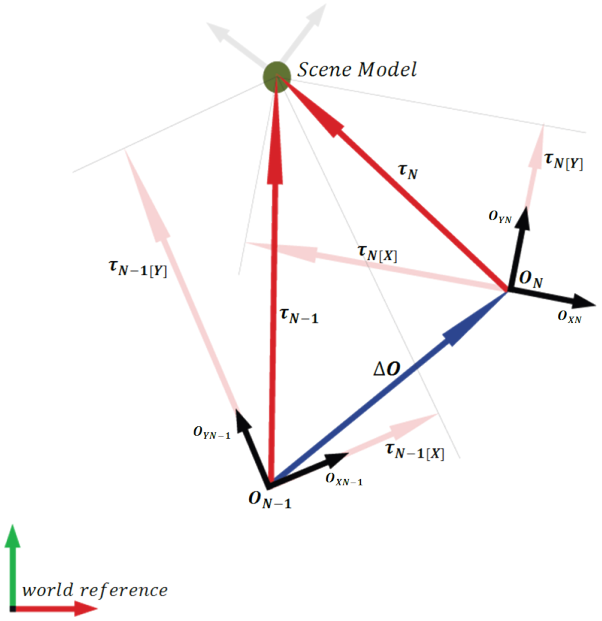

Fig. 12 Camera position change between 2 frames. The camera measures different distances to the scene model in the directions of its principle axes. The principle axes are also the 3 vectors in the orthonormal set. The vector $\boldsymbol{\Delta O}$ is the amount of camera translation between frames $N-1$ and N. This illustration is in 2D but the concept can be extended to 3D. Here, the z axes of the camera models are pointing into the page.

Similarly, we scale the orthonormal vectors for the M_N scene model.

$$\boldsymbol{\tau_{X[N]} = O_{XN} \cdot \left| G_{ON[X]} \right|} \tag{46}$$

$$\boldsymbol{\tau_{Y[N]} = O_{XN} \cdot \left| G_{ON[Y]} \right|} \tag{47}$$

$$\boldsymbol{\tau_{Z[N]} = O_{XN} \cdot \left| G_{ON[Z]} \right|} \tag{48}$$

Where $\boldsymbol{q}$ is the unit quaternion representing the estimated camera rotation between M_{N-1} and M_N and $\boldsymbol{q'}$ is its quaternion conjugate.

After the values of $\boldsymbol{\tau}$ have been calculated for M_{N-1} and M_N, we have the vectors $\boldsymbol{\tau_N}$ and $\boldsymbol{\tau_{N-1}}$, whose difference is the displacement vector $\boldsymbol{\Delta 0}$. The position estimate in M_N is the vector sum of $\boldsymbol{\Delta 0}$ and the origin of $\boldsymbol{O_{N-1}}$.

$$\boldsymbol{s_N = O_{ON-1} + \Delta 0} \tag{49}$$

For the RGB-D frame sequence $M_O, M_1, M_2 ... M_N$ we obtain odometry estimates $\varepsilon_O, \varepsilon_1, \varepsilon_2 ... \varepsilon_N$ each holding a position vector $\boldsymbol{s}$ and unit quaternion $\boldsymbol{q}$. We call these the raw odometry estimates and perform further processing on them to produce the final estimated trajectory. It is intended that this post processing step reduces randomly distributed noise over time. The noise post filter is described in the next section.

$$\varepsilon_0, \varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \boxed{\varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8, \varepsilon_9,} \varepsilon_{10}, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{13}, \varepsilon_{14}$$

$$\downarrow$$

Average of this block updates ε_7

Fig. 13 Estimate refinement using $\Delta = 5$. Here, the average of estimates ε_5 - ε_9 is used to update estimate ε_7. This moving average technique is used to update the trajectory after a sufficient number of raw estimates have been obtained.

8 Noise Post Filter

The steps laid out in the previous section result in a sequence of raw odometry estimates $\varepsilon_O, \varepsilon_1, \varepsilon_2 ... \varepsilon_N$. Each estimate contains the unit quaternion $\boldsymbol{q}$ representing estimated rotation, the vector $\boldsymbol{s}$ for estimated translation, a timestamp and the scene sections $\boldsymbol{[a, b, c]_{N-1}}$ and $\boldsymbol{[a, b, c]_N}$ that were used to create the scene model.

The raw odometry estimates will contain randomly distributed noise over time. For better accuracy, we compute an average estimate over a fixed number of estimates Δ. The computed average is used as the estimate at the center of the indices used for the average. Here a buffer of capacity Δ will be used to hold the raw estimates. The buffer moves over the estimates i.e in each iteration, every estimate in the buffer is moved to a lower index. The last index becomes empty and is assigned the latest estimate. Note that Δ has to be an odd number to have a center index. Figure 13 illustrates the averaging method.

An odometry estimate ε_N is refined as follows:

$$\varepsilon_N = \frac{1}{\Delta} \sum_{i=\Delta_-}^{\Delta_+} \varepsilon_i \tag{50}$$

$$\Delta_+ = N + \frac{\Delta - 1}{2} \tag{51}$$

$$\Delta_- = N - \frac{\Delta - 1}{2} \tag{52}$$

The average values have to be computed for $\boldsymbol{q}$ (the unit quaternion representing rotation) and for $\boldsymbol{s}$ (the displacement vector).

This refinement can be thought of as a low pass filter, where all rapid fluctuations in the trajectory are suppressed to some extent. Increasing the value of Δ makes the resulting trajectory more rigid to rapid changes. Two disadvantages become apparent when using extremely high values of Δ . The first is that rapid changes that are legitimate are also suppressed. This becomes even more apparent during tight bends in the ground truth trajectory. The refined trajectory will spend more time during the bend, and probably have a larger turning radius.

The second disadvantage is that it will come with a waiting period when the estimates buffer is not yet full. For example with a Δ value of 23, $\varepsilon_1 - \varepsilon_{10}$ which is a total of 10 estimates will have to remain as noisy estimates or be discarded. In the case where they are discarded, the system will experience a startup time interval where raw estimates are filled into the buffer before real-time refined odometry can begin. Also, the accumulated noise between ε_0 and ε_{10} will probably amount to a lot of drift since the refinement process started late.

9 Results

In this section, results are presented for the algorithm test runs done on the fr1/xyz sequence of the TUM-RGB-D Benchmark. The sequence contains mostly translatory motion along the principle axes of the Kinect while the orientation was kept mostly fixed (TUM, 2014). It is well suited for debugging purposes since it is very simple (TUM, 2014).

TUM describes 2 error metrics: the absolute trajectory error (ATE) and the relative pose error (RPE). Relative pose error measures the local accuracy of the trajectory over a fixed time interval and corresponds to the drift of the trajectory whereas the absolute trajectory error measures the global consistency of the estimated trajectory. (Sturm et al, 2012)

For these experiments we use a sliding average width $\Delta = 7$, which proved to be a good general value in some preliminary tests. 4 main configuration sets are used in the experiments. Under each configuration, the number of features per section FS is varied and we observed the estimation rate in in frames per second (fps), the absolute trajectory error (ATE) in meters and the relative pose error (RPE) for the translation in meters and for the rotation in degrees.

9.1 Perfomance

The results from the test runs show a major tradeoff between estimation rate and accuracy. The algorithm runs much faster with a smaller number of visual features per section, but produces a poorer trajectory. The extreme scenarios are when we use 1 feature per section. The results show that such a setup defines the peak values for estimation rate.

When more features per section are used, the algorithm runs slower but produces a better trajectory. There are also limits to the number of features per section that can be used beyond which odometry estimation fails due to lack of a sufficient number of features.

The total number of sections used also affects the estimation rate. A larger number of sections generally produces faster estimates since smaller image patches are matched. When using a high number of sections, the number of features per section becomes more limited to smaller numbers, since more features may jump across sections when the camera moves.

9.2 Error Accumulation

The experimental results prove the validity of the algorithm since the estimated trajectory closely follows the ground truth trajectory and only drifts slightly over time. Drift is present in any visual odometry algorithm and it is necessary in a practical implementation to update the motion estimate over long periods of time. This can be done using an inertial measurement unit (IMU) equipped on a mobile robot.

10 Conslusions

This paper presented an approach to visual odometry using a compact model containing a position vector and an orthonormal set of 3 direction vectors. We labeled these the x, y and z axes of the model. Such a model was derived by selecting 3 sections in the scene in the previous and latest frames, treating them as points and assigning them a depth value from neighbouring pixels in the depth map. The model encodes information about

Table 1 Results under configuration 1. A total of 4 sections was used, each section 320 x 240 pixels with $\Delta = 7$.

FS	Rate (fps)	ATE (m)	s RPE (m)	θ RPE(deg)
15	0.90	0.097	0.108	3.012
13	0.92	0.076	0.083	3.349
11	0.95	0.098	0.105	3.205
9	1.00	0.036	0.063	2.112
7	1.03	0.064	0.074	2.550
5	0.97	0.044	0.088	4.653
3	1.20	0.216	0.123	6.832
1	1.16	0.144	0.152	3.761

Table 2 Results under configuration 2. A total of 8 sections was used, each section 128 x 240 pixels with $\Delta = 7$.

FS	Rate (fps)	ATE (m)	s RPE (m)	θ RPE(deg)
15	0.80	0.161	0.271	3.894
13	0.81	0.255	0.395	5.455
11	0.97	0.142	0.224	3.717
9	1.30	0.183	0.262	6.060
7	1.50	0.150	0.204	5.154
5	1.70	0.100	0.136	5.242
3	1.83	0.197	0.136	7.004
1	2.45	0.272	0.151	6.770

Table 3 Results under configuration 3. A total of 15 sections was used, each section 128 x 60 pixels with $\Delta = 7$.

FS	Rate (fps)	ATE (m)	s RPE (m)	θ RPE(deg)
15	0.87	0.172	0.321	4.037
13	0.99	0.105	0.187	4.548
11	1.01	0.120	0.225	5.647
9	1.06	0.399	0.376	5.694
7	1.26	0.279	0.328	5.721
5	1.59	0.226	0.234	6.398
3	1.96	0.187	0.217	6.269
1	2.60	0.238	0.149	5.728

Table 4 Results under configuration 4. A total of 25 sections was used, each section 128 x 96 pixels with $\Delta = 7$.

FS	Rate (fps)	ATE (m)	s RPE (m)	θ RPE(deg)
7	1.25	0.390	0.502	5.560
5	1.51	0.281	0.367	8.724
3	2.02	0.459	0.219	9.266
1	3.38	0.206	0.157	8.913

the scene's orientation and position. The reason for selecting a group of points rather than individual points is sensor noise. The Kinect and similar sensors provide useful depth data at high rates but this data usually contains a lot of noise. Simply sampling the depth of a single pixel and using its value implicitly will lead to poor results since the noise fluctuations usually span more than about a 5 pixel radius.

The rigid body translation was represented as a vector in 3D space and the rotation was represented as a unit quaternion. Unit quaternions are ideal for this approach since their values can be computed directly from vector operations. Additionally, they only require 4 variables to represent the rotation (which has 3 degrees of freedom) compared to using a rotation matrix which needs 9 variables.

Raw odometry estimates were produced by performing vector operations on the camera and scene models for the previous and latest frames.

10.1 Limitations

Visual Features. The algorithm relies heavily on the presence of numerous visual features in the colour images. This is typically the case in the images of real world scenes. However there do exist scenes that have little or no visual features. Imagine a room with all walls painted white. This blank room would contain little or no visual features for the algorithm to perform matching on. Since the scene model relies on detected visual features, a model of the scene cannot be formed and the estimation would fail. Another scenario is when the camera is facing a blank wall at very close distance. This would register as a blank image matrix with no features even if the rest of the room contains many visual features, and the estimation would fail.

Maximum Speed. The maximum estimation rate produced was about 3 frames per second. At this estimation rate, the robotic platform will be expected to move low speeds less than 1 meter per second. Increasing the platform's speed also increases motion blur in the images. The amount of motion blur depends on the camera's internal design, especially the shutter speed. A large amount of motion blur makes it more difficult to detect visual features in the colour images and therefore reduces the maximum features that can be used per scene section.

Moving Objects in the Scene. One scenario that is not covered by this approach is the presence of moving objects in the scene. The scene was modeled as a single rigid body which is fixed in the world coordinate reference. The model created does not account for moving objects within the camera's field of view. This means that a practical implementation can only work in undisturbed environments where the robot is the only mobile entity.

10.2 Future Work

The algorithm can be possibly improved in several ways:

Feature Detector. This method has been implemented in the C++ programming language using the

OpenCV library (OpenCV, 2014). A SURF feature detector (Bay et al, 2006) was used in the feature detection step thanks to OpenCVs built in SurfFeatureDetector. Possible perfomance improvements may be achieved by experimenting with other feature detectors such as SIFT (Lowe, 2004), CENSURE (Agrawal et al, 2008), Harris (Harris and Stephens, 1988), Shi-Tomasi (Shi and Tomasi, 1994), and FAST (Rosten and Drummond, 2006).

Loop Closure. The drift problem exists since motion is estimated for many increments and errors are accumulated over time. There are some good attempts to minimise drift from visual odometry alone such as loop closure. If a robot moves around in some environment and ends up in its initial position and orientation, it may recognise the initial position and directly apply the odometry estimates from the initial position. This is known as loop closure and may be used to reduce drift substantially.

Loop closure was not applied in this algorithm since the problem of knowing if the robot has reached its initial destinations falls out of the scope of this work. Future work that extends or is related to this algorithm could be equipped with a loop closure mechanism to further improve the estimates where a loop is involved.

References

Agrawal M, Konolige K, Blas M (2008) Censure: Center surround extremas for realtime feature detection and matching. In: *Proc. European Conf. Computer Vision*, pp 102–115

Bay H, Tuytelaars T, Van Gool L (2006) Surf: Speeded up robust features. In: *Proc. ECCV*, pp 404–417

Engel J, Sturm J, Cremers D (2012) Camera-based navigation of a low-cost quadrocopter. In: *Proc. Intelligent Robots and Systems (IROS)*, pp 2815 – 2821

Harris C, Stephens M (1988) A combined corner and edge detector. In: *Proc. of Fourth Alvey Vision Conference*, pp 147–151

Huang AS, Bachrach A, Henry P, Krainin M, Maturana D, Fox D, Roy N (2011) Visual odometry and mapping for autonomous flight using an rgb-d camera. In: *Int. Symposium on Robotics Research (ISRR)*

Konolige K, Agrawal M, Sola J (2007) Large scale visual odometry for rough terrain. In: *In Proc. International Symposium on Robotics Research*

Lowe DG (2004) Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60:91 – 110

Microsoft (2014) Microsoft developer network. URL http://msdn.microsoft.com/en-us/library/jj131033.aspx, [Online; accessed Mar-2014]

Moravec HP (1980) Obstacle avoidance and navigation in the real world by a seeing robot rover.

Nister D, Naroditsky O, Bergen J (2004) Visual odometry. In: *Proc. of the 2004 IEEE Computer Society Conference*, pp 652–659

OpenCV (2014) Opencv. URL http://opencv.org/, [Online; accessed Mar-2014]

Rosten E, Drummond T (2006) Machine learning for high-speed corner detection. In: *Proc. of the 9th European conference on Computer Vision*, pp 430–443

Scaramuzza D, Fraundorfer F (2011) Visual odometry part i: The first 30 years and fundamentals. *IEEE Robotics & Automation Magazine* pp 80–92

Shi J, Tomasi C (1994) Good features to track. In: *Proc. of Computer Vision and Pattern Recognition Conference*, pp 593–600

Sturm J, Burgard W, Cremers D (2012) Evaluating egomotion and structure-from-motion approaches using the tum rgb-d benchmark. Tech. rep.

TUM (2014) Computer vision group. URL http://vision.in.tum.de/data/datasets/rgbd-dataset, [Online; accessed Mar-2014]

Weiss S, Achtelik MW, Lynen S, Chli M, Siegwart R (2012) Real-time onboard visual inertial state estimation and self-calibration of mavs in unknown environments. In: *Proc. IEEE International Conference on Robotics and Automation*, pp 957 – 964