Object Recognition, Pose Estimation and Tracking of a Rigid Object with Planar Surface

Ramkumar Natarajan **Siddharthan Rajasekaran** **Sri Ramana Sekharan** **Vishaal Kabilan**

Abstract

In this project we recognize and estimate the pose of a Rubik's Pyramid. We model the object using a 3D-mesh, recognize it using OpponentSURF, estimate the pose using EPnP and then track the object using Linear Kalman filter. We discuss the theory behind our approach, its performance and its limitations. In this project our focus is not general object recognition, instead we exploit the structure of our object and tune our algorithm based on it.

1. Introduction

The task of object recognition from an image or motion picture can be accomplished for many applications by identifying an unknown object as a member of a set of well known objects. In our project we deal with rigid Rubik's pyramid object whose model is known beforehand.

Many object recognition methods use a set of abstract characterizations in order to efficiently compare and represent the object. Such characterizations are canonically defined by measurable object feature extracted from various types of images and any previous knowledge available. Similarity between characterizations is interpreted as similarity between the objects themselves. Thus the ability of a given algorithm to describe the object uniquely from the available information determines the effectiveness of the for the given application.

2. Problem Statement

Our object is a pyramid with four faces each of which is shown in Fig. 1. Each of these faces has a single color namely green, blue, orange, gold. Each face of the pyramid is joint by a number of smaller triangles. So the smooth color on every side has interleaved triangular patterns in white. Since this is a well-defined regular object, we can accurately model the object. We use the 3D modelling approach followed by recognition using feature detection. We then estimate the pose using EPnP and the track using kalman filter.

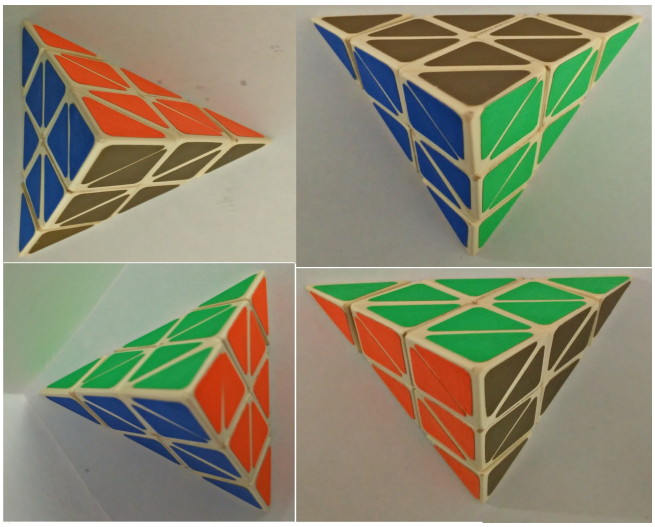

Figure 1. Different Poses of Object:The object to be detected in different poses shown here. From top-left clockwise: pose1, pose2, pose3, pose4

3. Our Approach

We explain the theory behind each component of our approach here and how it applies to our object.

3.1. Model Registration

Model registration is the essential part of our method. We first register the model of the given object, then compare the image frame captured online with our model to classify the features observed as inliers and outliers. (Pi, 2015) To do this we do model

RBE/CS 549: Computer Vision. Project work in partial fulfillment of the requirements for course evaluation.

registration offline before the detection stage. We have two major steps in model registration. First, we define the 3-D mesh of the object. A 3-D mesh is a set of triangles that lay on the surface of the object. We choose triangles over other polygons because 1) we are directed towards recognizing a rigid body with planar surface 2) computing area of a triangle is easier than computing area of other polygons. The latter is required for area based sampling which we'll explain in the following sections.

Apart from the 3-D mesh, the model consists of n-unique feature points that characterize the object. We will be referring to these as correspondence points. In our case we pick the corners of pyramid and the middle points of each edge as correspondence points. This is because the corner points have 3 colors around it in a specific order. For example, point $P4$ in Fig.3.1.2 has blue,green and gold colors in anticlockwise direction. Also, we need four non-coplanar control points on the boundary of the object to span through any interior point of the object. This is important to preserve the constraints (5. $\sum_{j=1}^{4} \alpha_{ij} = 1$) of the barycentric coordinates of all the other correspondence points(Pi, 2015). However we need 6 unique points apart from the control points to find the camera parameters. We choose 6 points on the mid point of the 6 edges of the pyramid.

We hand code the 3-D coordinates of these points and store it alongside the 3-D mesh. We find the points that are visible for different poses of the object. Our object has only four such poses (with different set of correspondence points visible) and hence this method works well with few definitions of these different poses. For example, in pose 1 3.1.2 we see all the points i.e. P1,P2,P3,P4,P5,P6,P7,P8,P9,P10 given the perspective. Once we have such definition for the vertices and edges we have sufficient information for associating 2D correspondence points detected using features to 3D coordinates and hence find the pose of the camera.

3.1.1. Barycentric Coordinates

The 3-D mesh of an object is a set of triangles. We store the mesh as a collection of three vertices (for each triangle) which stay connected to each other in the mesh. To convert the mesh of an object to point cloud we need to uniformly sample points from the mesh all over its surface area. We randomly sample a triangle $\triangle_i$ from the mesh with a weight proportional to its area A_i calculated from its vertices $\{V_1^i, V_2^i, V_3^i\}$ (stored alongside mesh definition). $A_i = \frac{1}{2}. \mid\mid (V_1^i - V_3^i) \times (V_1^i - V_2^i) \mid\mid$. We now have to sample a point from this triangle with uniform distribution. Barycentric coordinates (**?**) provide a perfect framework for uniform sampling. Barycentric coordinates of any n dimensional point can be represented as an m dimensional point when it is on an m dimensional simplex. Since we use triangles ($m = 3$) to represent the mesh, the barycentric coordinates of the point ($\vec{X^i}$) sampled from this triangle will be three dimensional.

$$\vec{X'} = \begin{bmatrix} x_1^i \\ x_2^i \\ x_3^i \end{bmatrix} \xleftarrow{sample} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \mid u+v+w = 1, 0 \leq u, v, w \leq 1 \tag{1}$$

u,v,w is all the space spanned by $\vec{X'}$. To get a uniform distribution over this space we choose two random numbers, $r_1 and r_2$ between 0 and 1. We then assign,

$x_1^i,\ x_2^i,\ \&\ x_3^i$ using

$$x_1^i = 1 - \sqrt{r_1}$$

$$x_2^i = \sqrt{r_1}(1 - r_2)$$

$$x_3^i = 1 - x_1^i - x_2^i$$

3.1.2. Moller-Trumbore algorithm

Any point P inside triangle $\triangle_i$ can be represented as an intersection of a parametric ray given by $P = O + t.D$, where O is the origin of the ray, D is the direction and t is the distance of the point P from the origin O. Inversely, the barycentric coordinates of the intersection of the ray with the triangle can be found using

$$[-D\ (V_2^i - V_1^i)\ (V_3^i - V_1^i)] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = (O - A)$$

It is easy to find the mesh 3-D of rigid bodies with planar surface. Given our object (triangular faces), which is a collection of only four faces and can be exactly represented using triangles without any approximation, we pick the farthest three points on the surface of the object such that the triangle obtained by joining these three points is exactly the surface area of the object lying under this triangular section. In Fig. 3.1.2 we have shown the mesh using blue lines. We choose triangles (for mesh) on a face such that all the correspondence points on that face get selected at least once. For any detected feature point f we use k-nearest neighbors to classify the point as an inlier or

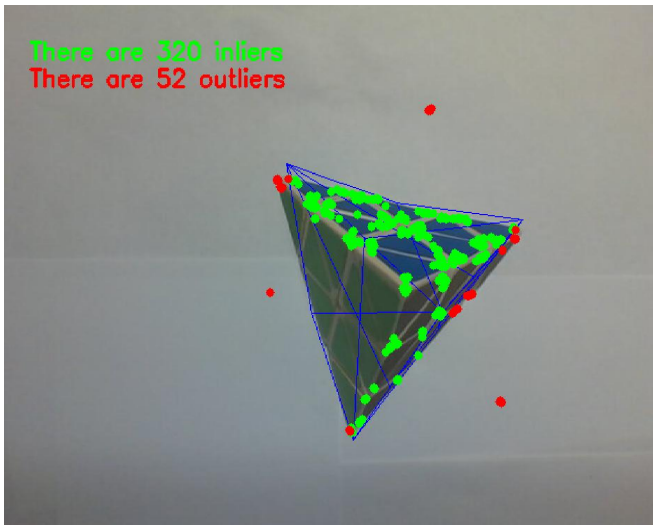

Figure 2. 3D Mesh:The mesh is shown using blue lines. The inliers are shown in green and the outliers in red

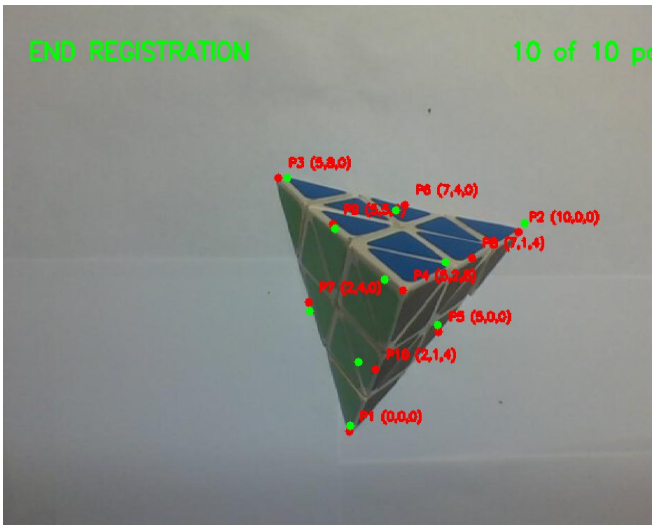

Figure 3. Correspondence points: The user marked correspondence points using mouse is given in red color. The correspondence points detected by feature matching algorithm is given in green.

outlier(Pi, 2015). If the point lies near the point-cloud sampled from the mesh, we classify it as an inlier. Fig. 3.1.2 shows the number of inliers (green) and the number of outliers (red).

In Fig.3.1.2 we can see the manually registered correspondence points marked in red while that detected by our feature matching algorithm are marked in green.

After we find the inliers we still have some features that distort the orientation calculations. We filter these features using left/bottom-right/top constraint and the neighbor ratio check. That in the Fig.registration if point $P_2 \Rightarrow P_3$ then $P_4 \Rightarrow P_5$ ('$\Rightarrow$' is read 'is right/top side of'). Mathematically,

$$(\vec{P_2} - \vec{P_3}).(\vec{P_4} - \vec{P_5}) \geq 0$$

We check the distance between matches in order to remove repeated elements. We keep matches with ratio of distance of first nearest neighbor to the second less than a threshold δ.

3.2. Camera Calibration

Object recognition is a tract of the broader problem of pose estimation. To get the position and orientation of the object, we formalize the problem into a PnP problem. This is because using Direct Linear Transform (DLT) was computationally expensive and slow. Thus we calibrate the intrinsic parameters. The camera projection matrix is given as:

$$M = K[RT] \tag{2}$$

where K is the intrinsic(inner) and R,T are the extrinsic(outer) camera parameters.

Out of the various techniques used for camera calibration, we used the 2D plane based calibration. Techniques in this method requires to observe a planar pattern shown at a few different orientations. Unlike Tsai's technique[v1],the knowledge of the plane motion is not necessary. In our experiment we used the checkerboard pattern. The pattern was first printed and attached to planar wall 3.2. Then a few images of the model plane were taken under different orientations by moving the camera. Then the feature points in the images are detected. Then the five intrinsic parameters are estimated by the closed form solution discussed in [v2]. This was followed by estimating the coefficients of radial distortion by solving the least squares

$$k = (D^T D)^{-1} D^T d \tag{3}$$

Finally all the parameters are refined,including lens distortion parameters by minimizing

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \|m_{ij} - m(A, k_1, k_2, R_i, t_i, M_j)\|^2 \tag{4}$$

Once the intrinsic parameters like the focal length and principal point were obtained, they were then used for pose estimation.

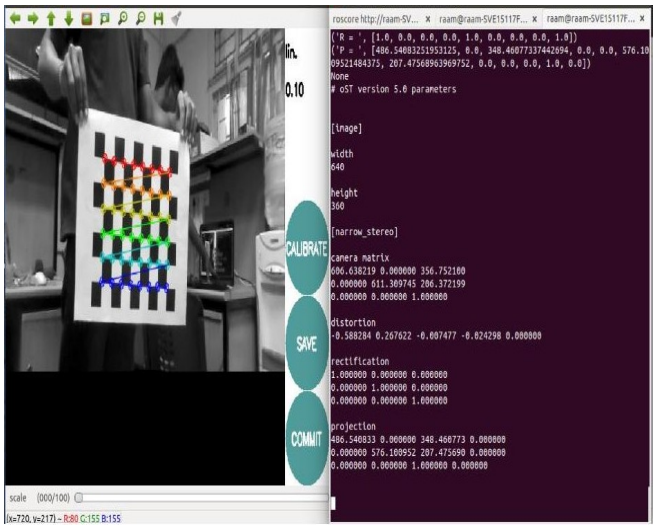

Figure 4. Camera calibration : On the right (output) the 'camera matrix' shows the scaling factor and camera offset during calibration

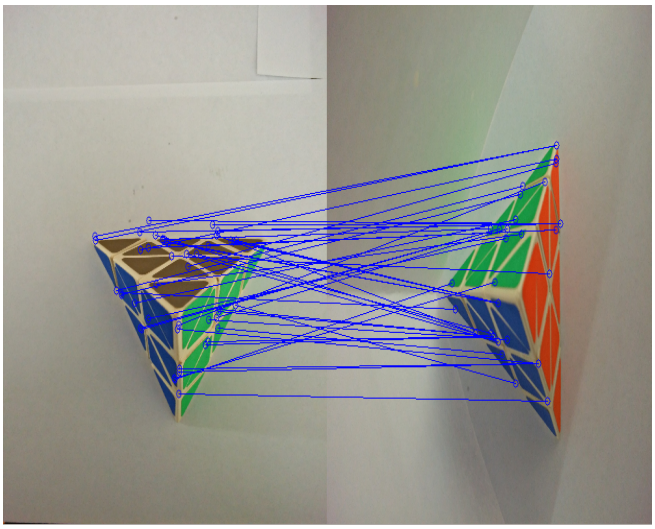

Figure 5. Failure of SURF: The image shows color side mismatch. Keypoints from blue side are matched erroneously with orange/green and vice versa.

3.3. Feature descriptors and Matching

To do object recognition we resort to feature based techniques. We tried SURF feature descriptors for matching. Since, SURF works only for gray scale we tried a color-based variant of SURF known as OpponentSURF. After extracting the descriptors we match them using FLANN matcher.

3.3.1. Speeded Up Robust Features

The SURF includes SURF keypoint detection algorithm that is robust to change in scale, rotation and translation and the SURF descriptors(Bay et al., 2006). Given an image the interest points are first detected using hessian corner detector. Since the detected corner points need to be scale invariant, scale space extrema detection is done using Difference of gaussians(DoG) approach. In SURF the gaussians are approximated by square-shaped filters. Once potential keypoint locations are found, they are refined to get more accurate results. Keypoints thus localized are robust to change in scale. These keypoints are then assigned an orientation, so as to make them invariant to rotation. The next step is to describe the keypoint with as much unique information as possible.To do that SURF, centers a 20x20 square window on every keypoints, and oriented along the direction of assigned orientation. It is divided into smaller 4x4 sub-blocks and for each sub-blocks their HAAR wavelet response is recorded. The above process results in a 128 length vector for every keypoint. This 128 length vector is known as the SURF descriptor.

We used SURF algorithm for feature matching on our pyramid object. It resulted in very bad performance 3.3.1. The primary reason for this behavior, is the lack of distinguishing features for different colored sides in the gray scale image of the object. Gray scale image does not contain enough information to distinguish between different colored regions. Taking a look at the object, we see that there are a small set of features in the object, that repeats itself on every colored side. Since gray scale image is not able to distinguish colors, there is a high probability of occurrence of wrong matching pairs from one colored side to a differently colored side. Failure to incorporate color information results in poor performance.

3.3.2. Opponent SURF

In order to incorporate color information, we use a new variant of SURF known as OpponentSURF (Van De Sande et al., 2010). The opponent color space (O_1, O_2, O_3) of an RGB image is defined by the following expression.

$$\begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix} = \begin{bmatrix} \frac{R-G}{\sqrt{2}} \\ \frac{R+G-2B}{\sqrt{6}} \\ \frac{R+G+B}{\sqrt{3}} \end{bmatrix}$$

The O_1 and O_2 channels carry color information and the intensity information will cancel out due to the subtraction in their calculation. The O_3 channel carry intensity information, since we average the three

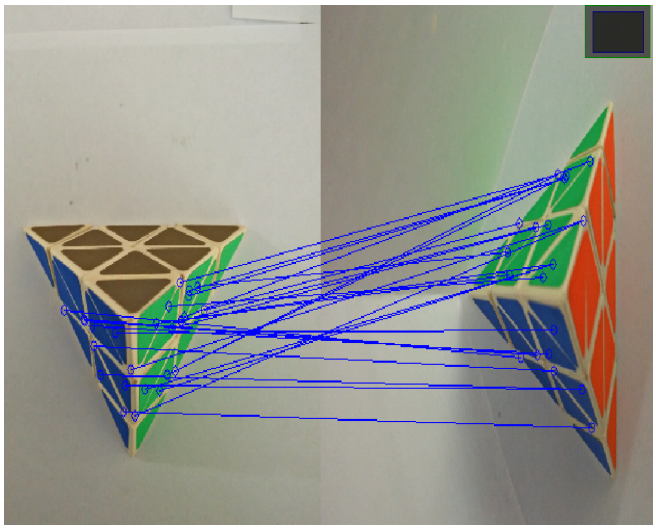

Figure 6. OpponentSURF: The image shows that same colored side matches on both images due to the incorporation of color information in OpponentSURF

channels. In OpponentSURF method, the keypoints are obtained from the same stage as the SURF algorithm. The difference is that instead of taking SURF descriptor around the keypoints in the gray scale image, OpponentSURF uses each of the opponent color channels. So for a given key-point, we obtain three SURF descriptors of length 128 corresponding to three color channels. The OpponentSURF descriptor is the concatenation of the three SURF descriptors and so it is a 3x128 length vector.

OpponentSURF was able to find robust representations for the object, that gives good matching performance due to the incorporation of color information 3.3.2.

3.3.3. FLANN matching

We have a database of images with stored feature descriptors. When we have a new image, we extract features from that. We need to match these features on the incoming image with that of the features stored in the database. This process is called feature matching. We use FLANN matcher for this operation.

FLANN stands for Fast Library for Approximate Nearest Neighbors. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features(Muja & Lowe, 2014). It uses kNN algorithm to find the closest match. There is a problem in this approach, in that it results in poor matches. To alleviate that problem, ratio test is done. In this method, the first and the second best match is taken and if the the ratio between them is less than a threshold it is taken as an authentic match. We took the ratio to be .85 as suggested by Lowe(Lowe, 2004).

$(F_A^i, F_B^j) \quad : \quad j \quad = \quad \min_k d(opp(F_A^i), opp(F_B^i)) \quad < \quad \delta \min_{k \neq j} d(opp(F_A^i), opp(F_B^i))$

In the above expression, $opp(F_A^i)$ indicates the opponent SIFT descriptors of feature point F_A.

3.4. Pose Estimation

Given the internal parameters, we find the external parameters for estimating the pose of the camera. Without the internal parameters or the correspondence points we found the camera parameters (both intrinsic and extrinsic) using Direct Linear Transform (DLT)(Hartley & Zisserman, 2003). DLT requires only one world coordinate (equivalent to one correspondence point) to find these parameters. The algorithm turned out to be slow and inaccurate for a fairly simple 4 dimensional simplex (object). Hence we choose calibrating the intrinsic parameters and hand-coding the coordinates of 6 correspondence and 4 control points over DLT.

We use EPnP algorithm to reduce the time complexity of iterating over n-correspondence points. We can represent any point inside the object with respect to camera (p_i^c) as a linear combination (α_{ij} barycentric coordinates) of the four control points c_j available to us.

$$p_i^C = \sum_{j=1}^{4} \alpha_{ij} * c_j^C \tag{5}$$

The coordinates in image plane are given by,

$$p_i^I = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} * \sum_{j=1}^{4} \alpha_{ij} * c_j^I \tag{6}$$

We can rewrite the above equation as $Mx = 0$ where M is 2n+12 parametric matrix. Assuming that the distance between two control points in the world coordinates and the camera coordinates are same, we find the constants of linear combination ($\mathbf{x} = \sum_i \beta_i v_i$) of eigen vectors of 12×12 parametric matrix ($M^T M$) for finding the coordinates of the correspondence points as

$$\beta = \frac{\sum_{i,j \in 1:4} || \vec{v_i} - \vec{v_j} || \cdot || c_i - c_j ||}{\sum_{i,j \in 1:4} || \vec{v_i} - \vec{v_j} ||^2}$$

where $\vec{v}$s are the eigen vectors of M. And hence we find $x = \sum_i \beta_i$. The advantage of this method is that we convert any n- correspondence points into

an unknown of constant 12- dimensions. Though we have (including the control points) only 10 correspondence points, this method gives user the flexibility to choose more correspondence points. This was helpful in forming the mesh around the detected object in case of extreme environments(extremely cluttered or ill-illuminated i.e strong deviation, too high or too low, from the reference image). One such instance is shown in Fig.cluttered.

To find the pose of the camera we calculate p_i^I from the image. Then we pick one of the control points as 0. With this as reference we compute p_j^C for the remaining points. We compare the p_j^W with p_j^C and hence find R and T by comparing the initial and final frames of the object. We use the screw motions formula to estimate the amount of rotation.

$$\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} = e^{[S].\theta}$$

where S is the screw (a vector) around which we rotate and θ is the amount of rotation. Using S and θ we can estimate R and T **??**

3.5. Object Tracking

The objective of object tracking is estimate the trajectory of an object in the image plane as it moves around a scene. (Roach, 1980a) and (Roach, 1980b) explain few of the basic approaches of tracking using several points of object correspondence from image sequence. However object tracking in our case boils down to tracking the 3-D mesh of the object alone. We use Kalman Filter to estimate the position and orientation of the 3-D mesh. Another key advantage of Kalman filtering is to keep track of the object pose in those cases where the number of inliers is lower than a given threshold.

3.5.1. Kalman Filter as a Object tracker.

The Kalman filter is a recursive method to estimate the state of the object in the process of tracking (Broida & Chellappa, 1986). We use linear kalman filter to estimate the transnational and rotational position, velocity and acceleration. The successive states $s_t \in \mathbb{R}^n$ of a discrete-time controlled process are assumed to evolve according to a dynamics model written as follows,

$$s_t = Fs_{t-1} + w_t$$

where F is the state transition matrix. For the purpose of tracking the object the state vector consists of six defining parameters of the 3-D mesh's pose along with their translational and rotational velocities and acceleration.

The state transition model F is given by the following matrix. The model computes the values of state components at time t given the estimate at time $t-1$, where the duration between successive time steps is Δt.

$$F = \begin{smallmatrix}
1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{1}{2}\Delta t^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{smallmatrix}$$

As we have already mentioned, measuring the camera pose is equivalent to estimating the real-time orientation of the object (Horn et al., 1988). Our implementation of linear kalman filter assumes that the measurement z_t of camera pose at any time t is related to state vector by a linear measurement model.

$$z_t = \mathcal{H}s_t + v_t$$

where $\mathcal{H}$ is the linear measurement model. w_t and v_t are the process and measurement noise respectively which are assumed to be normally distributed with zero mean. The measurement model $\mathcal{H}$ which can observe the 6 defining parameters of mesh from the state vector is given below,

$$\mathcal{H} = \begin{smallmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{smallmatrix}$$

The state of the kalman filter is represented as follows,

$$s_t = \left(x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, \psi, \theta, \phi, \dot{\psi}, \dot{\theta}, \dot{\phi}, \ddot{\psi}, \ddot{\theta}, \ddot{\phi}\right)$$

where (.) represents the velocity component and (..) represents the acceleration component of any parameter. The six primary parameters of the camera pose are shown in fig 7,

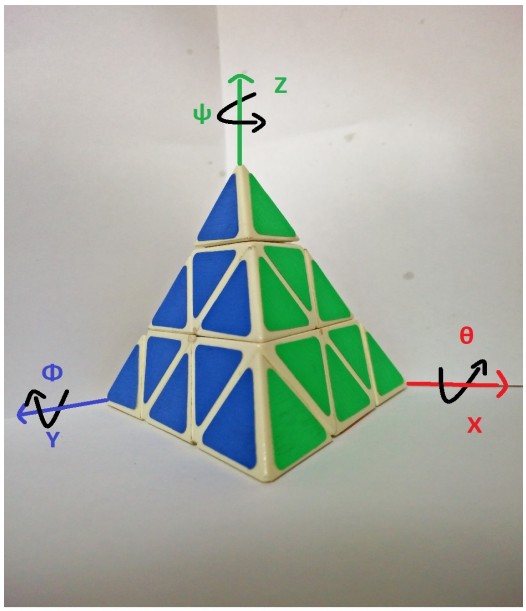

Figure 7. Image showing reference axis and pose angles used for implementing Kalman filter.

At each time step, the Kalman Filter makes an 'estimation' of the current 3-D mesh's pose parameters called the *apriori* state estimate s_t^-, which is refined by incorporating the measurements to yield the *aposteriori* estimate s_t. s_t^- and its covariance matrix S_t^-, are computed during the prediction stage and can be written as,

$$s_t^- = \mathbf{A}s_{t-1}$$
$$S_t^- = FS_{t-1}F^T + \mathbf{Q}$$

where S_{t-1} is the *aposteriori* estimate error covariance for the previous time step and $\mathbf{Q}$ is the process noise covariance that measures quality of the motion model respect to the reality. Next, the Kalman Filter does a 'measurement update' or correction. The *aposteriori* state estimate s_t and its covariance matrix S_t are now generated by adding the measurements z_t.

$$s_t = s_t^- + \mathcal{K}(z_t - Hs_t^-)$$
$$S_t = S_t^- - \mathcal{K}\mathcal{H}S_t^-$$

In the above equations the term $\mathbf{K}$ is called the Kalman Gain which controls a relation between the filter's use of predicting the parameters of mesh and measurement of camera's pose. $\mathbf{K}$ can be computed from the following equation,

$$\mathcal{K}_t = S_t^-\mathcal{H}^T(\mathcal{H}S_t^-\mathcal{H}^T + R)^{-1}$$

where $\mathbf{Q}$ is the measurement covariance matrix. In our context of tracking, the parameters of pose of the mesh is given by the *apriori* estimate s_t. Hence the measurement prediction can be expressed as,

$$z_t^- = \mathcal{H}s_t^-$$

But this measurement prediction may be uncertain which is represented by covariance matrix $\mathbf{P}$ by propagating the error.

$$P = \mathcal{H}S_t^-\mathcal{H}^T + R$$

4. Experiments and Result

The algorithm was able to successfully track the Rubik's pyramid in real time. Pose estimation was also successful, however in partial occulsion if any of the control points are occulded there was substantial error in pose-estimation. Even without occulsion during motion, simultaneous determination of both pose and velocities was not accurate. We have not incorporated algorithms that heuristically search objects that recover from complete occulsion. Hence during recovery from complete occulsions we search the whole image. This leads to a delay of 0.59 seconds for the recovery of tracking. Because of the low frame rate of 30fps captured by Logitech C270 and also our sampling rate of 10fps, the kalman filter was not able to track approximately 10000 pix/sec in image. Kalman filter considers such velocity as above threshold. When the angle of incidence of light source and the viewing angle was equal and both have a high value, the reflection from the light source distorted the color information.Due to this OpponentSURF was not able to detect feature points.

5. Conclusion and Future Work

Our algorithm was able to successfully track the given object by exploiting its structure but with some limitations. One good direction for future work is to experiment with more robust tracking algorithm other than kalman filter. Also for pose estimation we could use UPnP instead of EPnP since the former doe snot need camera calibration.

6. Acknowledgement

We thank Prof. Mike Gennert for his constant motivation. We also extend our gratitude to Koushik Bal-

asubramiam.

7. Video Demonstration

Please visit the following links for the video showing our implementation.

- **Object recognition and Pose estimation of a pyramid.**

- **Recognition of a Rubik's Pyramid**

References

Bay, Herbert, Tuytelaars, Tinne, and Van Gool, Luc. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pp. 404–417. Springer, 2006.

Broida, Ted J and Chellappa, Rama. Estimation of object motion parameters from noisy images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):90–99, 1986.

Hartley, R. and Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003. ISBN 9780511184512. URL https://books.google.com/books?id=eLbfoQEACAAJ.

Horn, Berthold KP, Hilden, Hugh M, and Negahdaripour, Shahriar. Closed-form solution of absolute orientation using orthonormal matrices. *JOSA A*, 5 (7):1127–1135, 1988.

Lowe, David G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Muja, Marius and Lowe, David G. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2227–2240, 2014.

Pi, Edgar Riba. Implementation of a 3d pose estimation algorithm. *MasterâĂŹs Degree in Automatic Control and Robotics*, 2015.

Roach, John Wilson. Determining the movement of objects from a sequence of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):554–562, 1980a.

Roach, John Wilson. Determining the movement of objects from a sequence of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):554–562, 1980b.

Van De Sande, Koen EA, Gevers, Theo, and Snoek, Cees GM. Evaluating color descriptors for object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9): 1582–1596, 2010.