Project manuscript No.
(ENPM667 Project 1)

Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM

ENPM667 Midterm Project Report

Harsh Kakashaniya · Rohitkrishna Nambiar

Submitted: 19th November 2018

Abstract The fusion of inertial and visual data is widely used to improve an objects pose estimation. However, this type of fusion is rarely used to estimate further unknowns in the visual framework. In this paper we present and compare two different approaches to estimate the unknown scale parameter in a monocular SLAM framework. Directly linked to the scale is the estimation of the objects absolute velocity and position in 3D. The first approach is a spline fitting task adapted from Jung and Taylor and the second is an extended Kalman filter. Both methods have been simulated offline on arbitrary camera paths to analyze their behavior and the quality of the resulting scale estimation. We then embedded an online multi rate extended Kalman filter in the Parallel Tracking and Mapping (PTAM) algorithm of Klein and Murray together with an inertial sensor. In this inertial/monocular SLAM framework, we show a real time, robust and fast converging scale estimation. Our approach does not depend on known patterns in the vision part nor a complex temporal synchronization between the visual and inertial sensor.

Keywords IMU vision fusion · Absolute scale · Monocular SLAM · Kalman filter

Harsh Kakashaniya
UID: 116311236
E-mail: harshbk@umd.edu

Rohitkrishna Nambiar
UID: 115507944
E-mail: rohit517@umd.edu

Contents

List of Figures

1 Introduction

This is a report of the paper titled *Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM* [19]. On-board pose estimation is very useful for a variety of applications ranging from autonomous robotics to augmented reality. Pose estimation can be done using different sensors such as cameras and inertial measurement units (IMU). Simultaneous localization and mapping (SLAM) using a monocular camera can be used to get the trajectory of the camera and a map of the environment. But monocular cameras suffer from scale ambiguity causing the overall trajectory to drift making it unusable in real time. Using a stereo camera helps solve the scale ambiguity. IMUs on the other hand can be used to get the trajectory traveled by integrating the acceleration measurements over time. But this leads to highly inaccurate trajectory estimates. The estimate of the scale factor is essential to fuse both these measurements. This fusion helps us determine the unknown scale factor λ.

In this paper [19], two methods are presented for scale estimation. The first one is a spline fitting method by Jung and Taylor [11]. The second is a multi rate Extended Kalman Filter(EKF). Both the approach have been simulated in MATLAB.

This report is organized as follows. Section 2 goes over the Camera and IMU setup, Section 3 covers the camera and image formation process, Section 5 goes over the scale estimation problem with monocular camera, Section 4 explains working of inertial measurement units, Section 6 outlines the spline fitting method with results, Section 7 covers Extended Kalman Filter and Section 8 explains visual odometry, SLAM along with filter based and key-frame based methods.

2 Hardware Setup

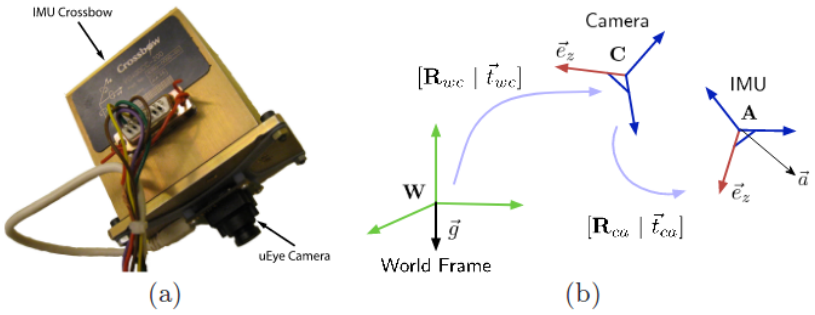


Fig. 1 **a.** Camera/IMU setup. **b.** Frame transformations between IMU, Camera and World frame.

This section goes through the hardware setup used in the research paper. USB uEye UI-122xLE fisheye camera was used as the vision input. The camera has a resolution of 752480 and a frame rate up to 87fps. Motion blur was minimized due to the high dynamic range and global shutter of the camera. VG400CC-200 solid state gyro which includes a tri-axial gyroscope and tri-axial accelerometer was used. It has an output frequency of 75 Hz with an input range of $\pm$10g and $< 1.25mg$ resolution. The IMU outputs acceleration around its 3-axis along with yaw, pitch and roll.

3 Camera and Image formation process

To understand how a camera perceives the environment, it is necessary we understand the image formation process. For us humans, we see things when a light originating from a light source is reflected on an object and enters our eyes. A camera acts very much similar to the human eye. The earliest and first model of an optical camera is the pinhole camera which is a simple and highly accurate representation of our eye model. This is the simplest device to form an image of a 3D scene on a 2D surface. As seen in figure 1 rays of light enter the pinhole and forms an inverted image of the object. This is called perspective projection. As the image formed in the image place is inverted, we consider a virtual plane in front of the pinhole that acts as the image plane.

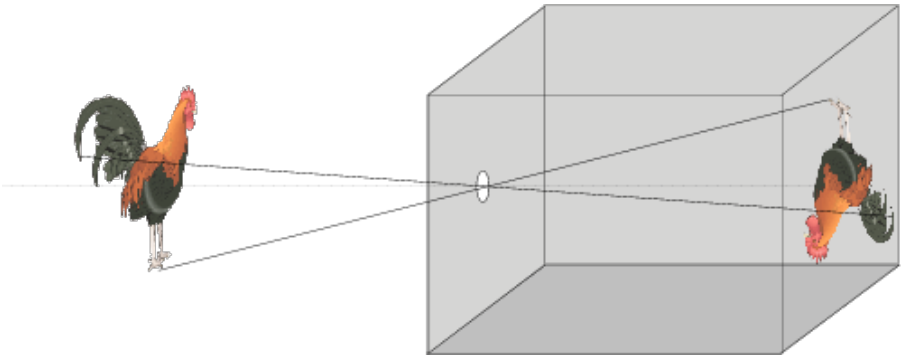


Fig. 2 Pinhole Image formation [25]

Although the pinhole model is quite accurate, modern cameras have lenses. They help gather more light from a source leading to higher quality sharper images. Ignoring the internal diffraction, we assume thin lens equations for the imaging process. Every camera has a region of depths over which the scene is sharp. Modern cameras have variable apertures which help in focusing on objects at varying distances in the scene.

Every camera has an intrinsic, extrinsic and distortion parameters that are important to understand for every application. These parameters are used to

correct for lens distortion, measure objects in physical world and to determine the location of the camera in the real world. Camera calibration is the process of estimating these parameters specific to a given camera and application setup. Figure 2 demonstrates how the a real world 3D coordinate is related to a 2D pixel coordinate using the parameters mentioned above. The intrinsic camera matrix **K** is given by

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

where (f_x, f_y) is the focal length, (c_x, c_y) is the optical center and s is skew factor. The extrinsic camera matrix **[R—t]** is given by

$$R = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix} t = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{2}$$

where R is the rotation matrix and t is the translation vector between the camera center and world co-ordinates.

The camera matrix P is given by

$$P = K[Rt] \tag{3}$$

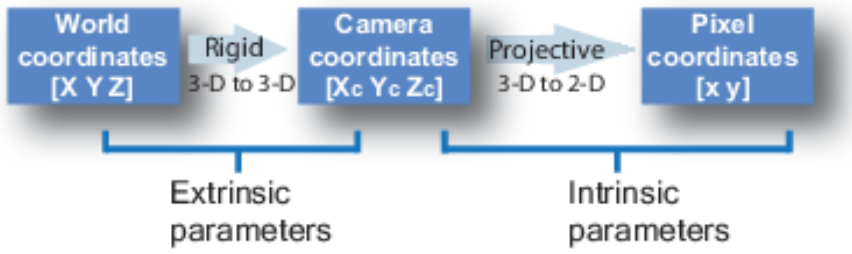


Fig. 3 World co-ordinates to pixel co-ordinates [17]

The extrinsic parameters relate the rotation R and translation t of the camera origin at the optical center to the world frame. The intrinsic parameters consists of the focal length given by f_x and f_y, the optical center given by c_x and c_y and the skew coefficient s. The distortion parameters consists of radial and tangential distortion along the x and y direction respectively. The number of coefficients are decided based on the lens in consideration. Camera calibration techniques presented in [26, 2, 9] are widely used in commercial and open source camera calibration tool boxes.

4 Inertial Measurement Units

4.1 Introduction

Inertial measurement units (IMU) are self-contained systems that measure linear and angular motion of an object or vehicle. Measurements are summed over a time period to determine the instantaneous position, velocity, orientation, and direction of movement.The IMU is comprised of at least two dedicated sensors, one or more linear accelerometers and one or more gyroscopes or angular accelerometers. An optional magnetometer may be integrated into the unit to calibrate against orientation drift.

Fig. 4 IMU Physical model [7]

4.2 Working

IMUs operate by use of reference data, bias values from an initial starting point, and calculate changes to these values using its integrated sensors. A central processing unit calculates directional information; position, speed, orientation, and direction of movement, at a given time in space using the IMU. The sensors suffer from orientation drift as they calculate these variables using a process known as dead-reckoning and are subject to accumulative errors.Calibration parameters can be stored in the memory of an IMU and are automatically reflected in the measurement data. In-situ calibrations may also be accomplished by use of a magnetometer or GPS unit to correct for orientation drift and improve the accuracy of the directional information.

Here we can see there is a weight less ball. And we are considering forward acceleration of 1 g hence ball is exerting inertial force on the box in -X direction as it is principle of action and reaction. So here we can learn that

$$Acceleration = \frac{force\ exerted}{mass\ of\ ball}$$

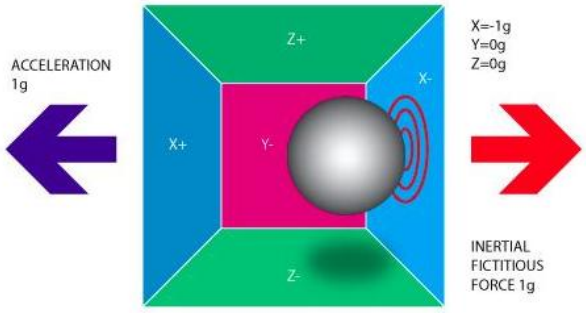


Fig. 5 Working of IMU model [7]

Hence computing load on -X face of wall will tell us about the acceleration. similarly if we have a ground on one side it will exert a force of g in lower direction which will be constant. There will also be conditions where forces will be applied on two sides combined. So its magnitude will give is the angle of tilt and accelerations in any angular directions.

So if we consider enclosed box and a ball we can measure all 3 acceleration in X,Y,Z due to all side enclosed box. We can use sine and cosine functions to represent angles with different axis. So if we we take Angular accleration is differentiation of velocity where

$$Angularvelocity_x = \frac{\theta_{x2} - \theta_{x1}}{T_2 - T_1}$$

So with equations and forces in all direction we can compute all 3 axis linear accelerations and all 3 angular accelerations(Gyro accelerations).

5 What's wrong with monocular camera?

As the name suggests a monocular camera is a camera with a single imaging sensor. In section X. we say that given a perfectly calibrated camera ie. with the intrinsic and extrinsic parameters known, we can measure size of objects in the real world. This however is not true for a single image taken from a monocular camera. When 3D objects are projected onto the image plane (2D), the depth information stored in the z-axis is lost. This can be seen in the example below.

As seen in the Fig 6, irrespective of the location of the object in the real world, the size of the object in the image plane remains the same. Mathematically, this can be shown as

$$\frac{y}{f} = \frac{y_1}{z_1 + f} = \frac{y_2}{z_2 + f} \tag{4}$$

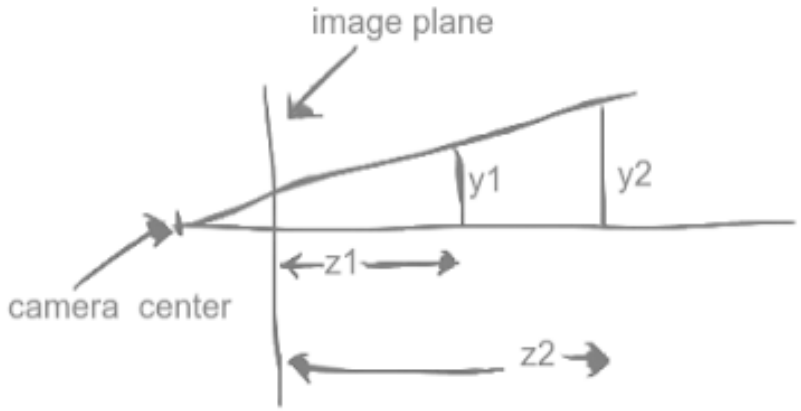


Fig. 6 Monocular camera scale problem

Where f is the focal length of the camera. The title of the paper mentions scale estimation. In the context of SLAM which will be covered in the next section, scale is a term used to denote the factor by which the computed trajectory by a SLAM algorithm needs to be multiplied/scaled to make it equal to the ground truth. For example, if the measured distance is 2m, whereas the ground truth is 4m, the scale value would be 2. This is an area of active research and the community has come up with different techniques to solve this problem. As the title suggests, we would be talking about the fusion of IMU and Vision for solving this challenge.

6 Spline Fitting Method

In this paper there are 2 methods of fusing IMU data and Monocular camera. First of which is spline fitting using scale factor with the data of IMU and Monocular camera. Depending on the reliability of the data from both sources to estimate distance.So here we did simulation of whole system with arbitrary value of data considering linear moment of robot with noisy data. These simulations can be seen in result section. With the matlab code. So initially theory behind curve fitting is explained to have a algebraic background of types of curve fitting and their applications.We use spline in curve fitting because we get discrete data from sensors and camera but tracing spline helps for interpolation and extrapolation to have continuous high probability data of path.

Spline is fitted on the data points which is output of sensor in our case. Spline fitting also helps in reducing error due to noise. For Improving accuracy we can increase number of spline but this will also result into increase in computation. So we have to trade off between accuracy and computation.

Spline fitting is useful to give smooth curve .N degree spline may has N-1 curves in its shape. So if spline has X3 term that means it can have 2 curves in its spline.Ideal method to cover n points in a plan is by tracing nth order spline which will result into zero error. So spline will pass through all the points and will have a general equation as

$$A_0 + A_1X + A_2X^2 + A_3X^3 + ... + A_nX^n = Y \tag{5}$$

This problem then converts in simple Ax = B form. we can get values of n terms by plugging in n points values.

Where

$$A = \begin{pmatrix} 1 & X_1 & X_1^2 & \cdots & X_1^n \\ 1 & X_2 & X_2^2 & \cdots & X_2^n \\ 1 & X_3 & X_3^2 & \cdots & X_3^n \\ : & : & : & \cdots & : \\ 1 & X_n & X_n^2 & \cdots & X_n^n \end{pmatrix} B = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ : \\ A_n \end{pmatrix} C = \begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \\ : \\ Y_n \end{pmatrix} \tag{6}$$

But for practical applications we use other methods and do not plot n degree spline as it is computationally expensive .Secondly if there is more noise in system it will give wrong results. For practical application there are 3 kinds of spline fitting according to the application:-

6.1 Types of curve fitting

6.1.1 Maximum error

This is the method where spline is fit in a way where the point with maximum error is reduced and curve move towards the outlier point. If there exist in order to reduce maximum error of a point. This on the other hand results into

$$E(f) = |max(f(xi) - yi)|where i goes from 1 to left n \tag{7}$$

Actually due to one outlier here line shifted in order to reduce maximum error. So this is not reliable process but is easy to compute.

6.1.2 Average error

This method is used to minimize average and fit the spline with minimum error conditions.

$$E(f) = \frac{1}{n} * \Sigma|f(x_k) - Y_k| \tag{8}$$

Actually due to one outlier here line shifted in order to reduce maximum error. So this is not reliable process but is easy to compute.

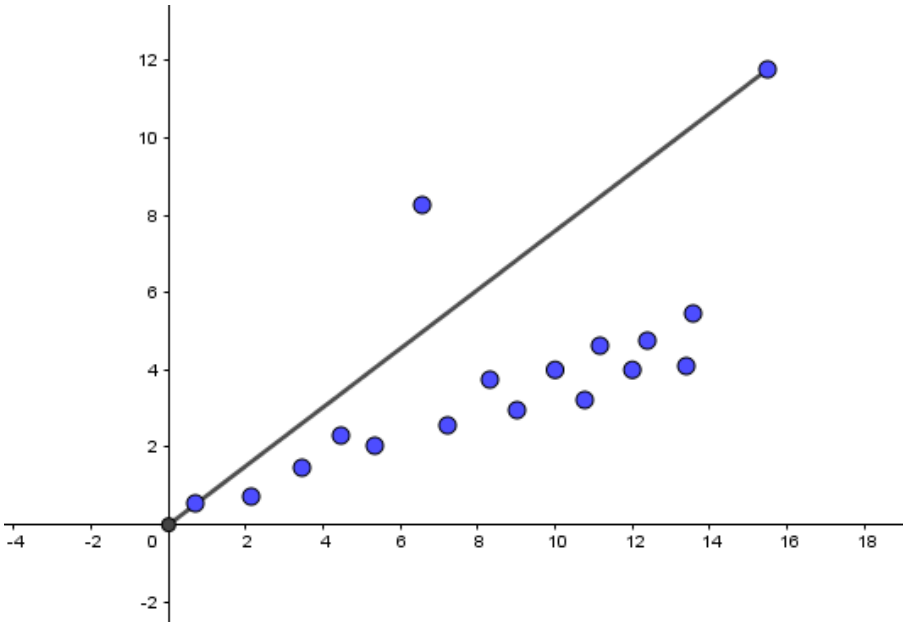


Fig. 7 Curve Fitting with Maximum error

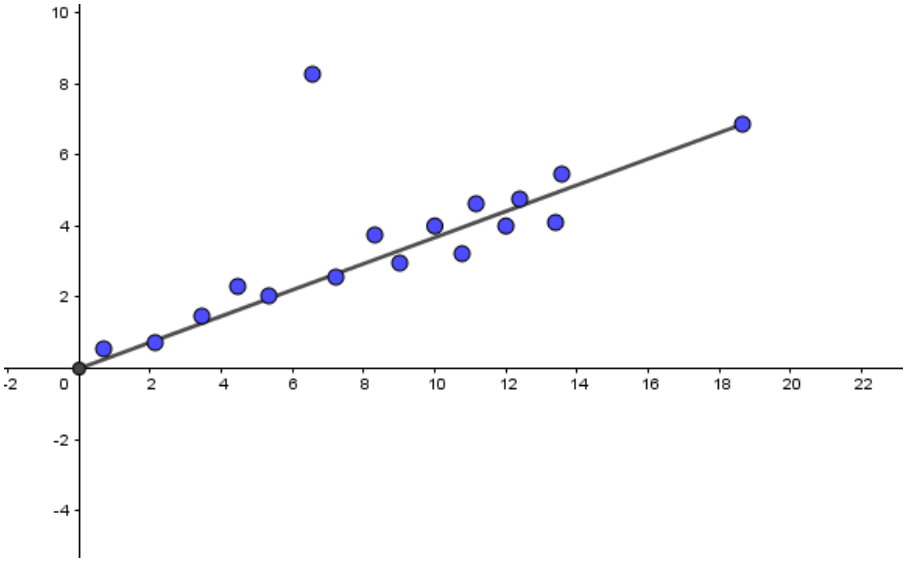


Fig. 8 Curve Fitting with Average error

6.1.3 RMS (Least square method)

This is the most efficient method to minimize the error this is also called as least square method. Hence this is mostly used in curve fitting. We have also created an algorithm with the help of same type of curve fitting.

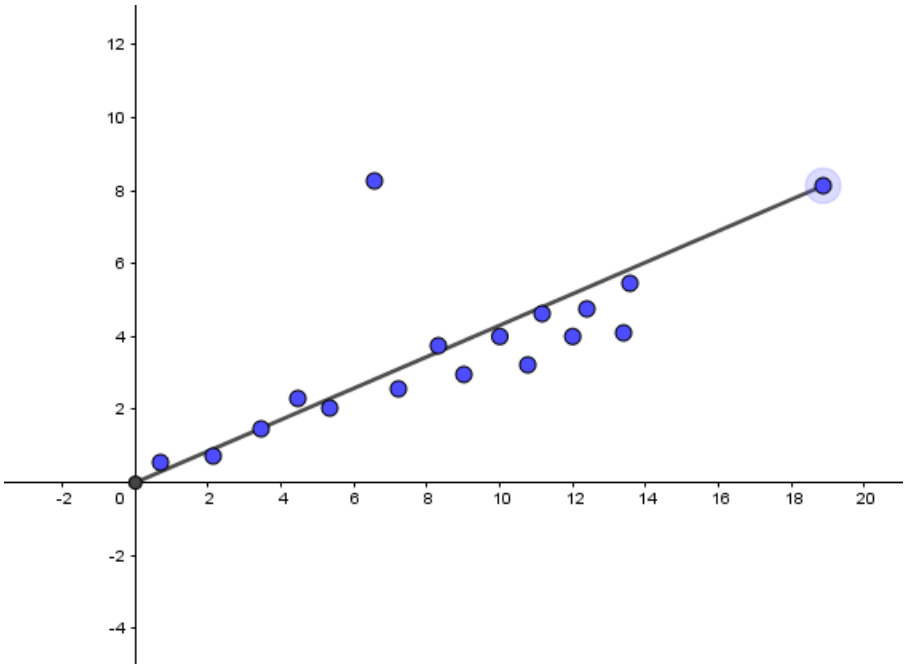


Fig. 9 Curve Fitting with RMS error

$$E(f) = \{\frac{1}{n} * \Sigma |f(x_k) - Y_k|\}^{\frac{1}{2}} \tag{9}$$

6.2 IMU Reading

As we dont have real IMU lets simulate virtual readings for this we took X, Y, Z as a function on time

$$X_1 = A_y t^2 + B_y t + C_x \tag{10}$$

$$Y_1 = A_y t^2 + B_y t + C_y \tag{11}$$

$$Z_1 = A_z t^2 + B_z t + C_z \tag{12}$$

With this treatment we will add some random noise to all the terms so that the data can be similar to what we obtain form IMU.

$$noise = A * (rand(1, n) - 0.5) \tag{13}$$

So our noise will range from -A/2 to A/2. As function is rand() so it will change for every point.

$$X = X_1 + noise \tag{14}$$

$$Y = Y_1 + noise \tag{15}$$

$$Z = Z_1 + noise \tag{16}$$

We also took some random points of camera pose. And provided our system with data set. Proof of Least square method in our case: Lets consider optimization of X first.

From equation (10) we have

Here A, B, C are known We have data set of X with respect to t To calculate error in RMS we have

$$Error_x = \{\sum_{i=1}^{n}(|X_i - (A_x t_i^2 + B_x t_i + C_x)|)^2\}^{\frac{1}{2}} \tag{17}$$

When we differentiate some quantity and equate it to zero it goes to either minima or maxima And if double derivative is positive it goes to minima.

So in this case if we take partial derivatives of error with respect to Ax , Bx , Cx

$$\frac{\partial Error_x}{\partial A_x} = \{\sum_{i=1}^{n} 2(|X_i - (A_x t_i^2 + B_x t_i + C_x)|)\}(-t_i^2) \tag{18}$$

$$\sum_{i=1}^{n}(|-t_i^2 X_i + (A_x t_i^4 + B_x t_i^3 + C_x t_i^2)|) = 0 \tag{19}$$

$$\frac{\partial Error_x}{\partial B_x} = \{\sum_{i=1}^{n} 2(|X_i - (A_x t_i^2 + B_x t_i + C_x)|)\}(-t_i) \tag{20}$$

$$\sum_{i=1}^{n}(|-t_i X_i + (A_x t_i^3 + B_x t_i^2 + C_x t_i^1)|) = 0 \tag{21}$$

$$\frac{\partial Error_x}{\partial C_x} = \{\sum_{i=1}^{n} 2(|X_i - (A_x t_i^2 + B_x t_i + C_x)|)\}(-1) \tag{22}$$

$$\sum_{i=1}^{n}(|-1X_i + (A_x t_i^2 + B_x t_i + C_x)|) = 0 \tag{23}$$

So we can compute matrix in such a way that the problem changes to Ax=B

Where,

$$A = \begin{pmatrix} \Sigma t^4 & \Sigma t^3 & \Sigma t^2 \\ \Sigma t^3 & \Sigma t^2 & \Sigma t \\ \Sigma t^2 & \Sigma t & \Sigma 1 \end{pmatrix} B = \begin{pmatrix} A_x \\ B_x \\ C_x \end{pmatrix} C = \begin{pmatrix} \Sigma X_i t^2 \\ \Sigma X_i t \\ \Sigma X_i \end{pmatrix} \tag{24}$$

From this equation we can calculate X matrix.Similarly,We can get X matrix of Y and Z axis.

$$A = \begin{pmatrix} \Sigma t^4 & \Sigma t^3 & \Sigma t^2 \\ \Sigma t^3 & \Sigma t^2 & \Sigma t \\ \Sigma t^2 & \Sigma t & \Sigma 1 \end{pmatrix} B = \begin{pmatrix} A_y \\ B_y \\ C_y \end{pmatrix} C = \begin{pmatrix} \Sigma Y_i t^2 \\ \Sigma Y_i t \\ \Sigma Y_i \end{pmatrix} \tag{25}$$

$$A = \begin{pmatrix} \Sigma t^4 & \Sigma t^3 & \Sigma t^2 \\ \Sigma t^3 & \Sigma t^2 & \Sigma t \\ \Sigma t^2 & \Sigma t & \Sigma 1 \end{pmatrix} B = \begin{pmatrix} A_z \\ B_z \\ C_z \end{pmatrix} C = \begin{pmatrix} \Sigma Z_i t^2 \\ \Sigma Z_i t \\ \Sigma Z_i \end{pmatrix} \tag{26}$$

By this treatment we converted given points into equation with least square method here we used condition of

$$min \begin{pmatrix} A_x t^2 + B_x t + C_x - \lambda_i X_c \\ A_y t^2 + B_y t + C_y - \lambda_i Y_c \\ A_z t^2 + B_z t + C_z - \lambda_i Z_c \end{pmatrix}^2 \tag{27}$$

Here in this formula we have different value of for different spline according to accuracy of camera data and IMU reading.

6.3 Results

In our case we simulated the results and found following outputs. Plotting the second order curve for given data of IMU with just least square method. We get the following graph

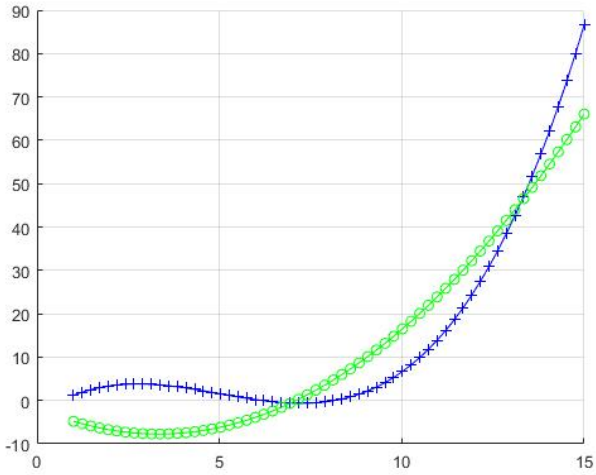


Fig. 10 In this case Blue is actual curve traced by IMU this are points with noise.And green is plotted curve. With least square method.

Then according to the paper we fused data of camera and IMU so we get following graph.

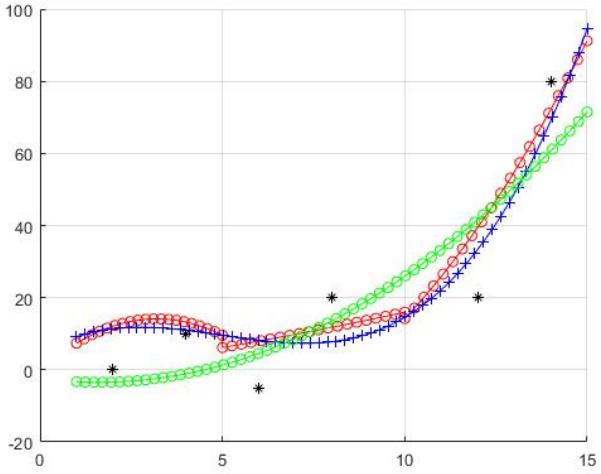


Fig. 11 Multiple spline fit

In Figure 11 we have 3 curved fitted as shown as red which gives better results and it also uses data of camera shown with black dots. So here according to the above formula scale factor of different section is different it is according to the quality of reading by camera. For example, in our case

$$\lambda_1 = 0.5, \lambda_2 = 0.5, \lambda_3 = 0.1 \tag{28}$$

In 3rd section camera readings were not accurate so taken into smaller scalar value. This camera data further improved the result. And we get minimum error and good fit. With the given spline fitting method.

So if we look into different of error with traditional Least square method with Jung and Taylor method. It gives good results. Almost we get the same curve.

The graph in Figure 12 has error comparison with and without Jung and Taylor method of split curve fitting and scaling camera input. So it is better to follow the given first method in the paper. It results into considerable reduction in error which is clearly seen in graph.

With the same approach we can optimize and plot spline in 3 Dimension so when we use this algorithm for 3D space we get the following result where we get an second order optimized curve. we also included considerable noise and tested our algorithm.

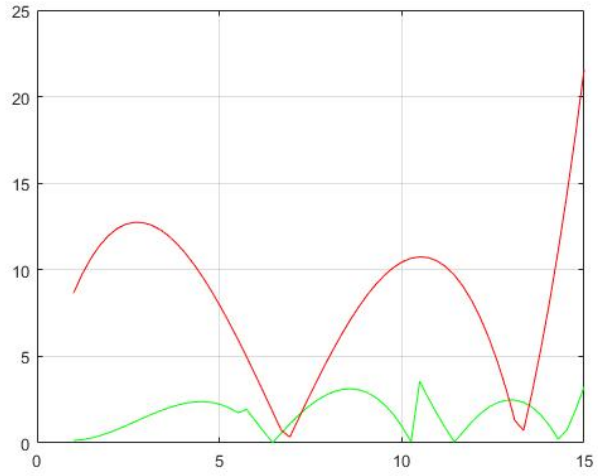


Fig. 12 **a**. Red curve for method with one curve and least square method. **b**. Green curve is using the given method by Jung and Taylor.

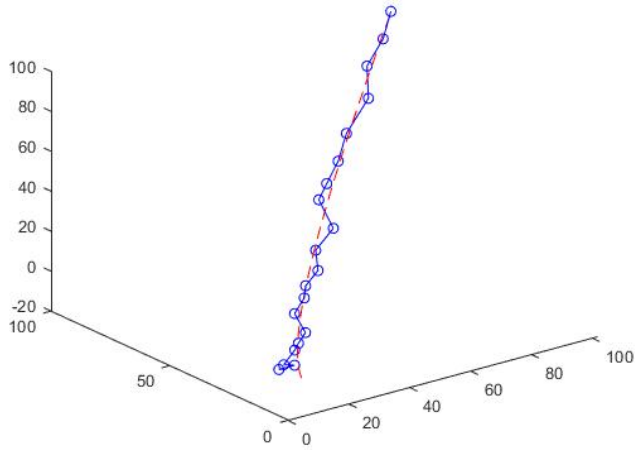


Fig. 13 Actual 3D path with noise vs Optimized spline fit

7 Extended Kalman Filter

In estimation theory, the extended Kalman filter (EKF) is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. In the case of well defined transition models, the EKF has been considered to estimate the state. in this system we have a state and an

observer these are given by equation. $X = AX + BU + W_d$ and $Y = CX + W_n$ In this W_d is disturbance and W_n is the noise by sensor. So Kalman filter is used judges between data be disturbance and noise.

If $W_d >> W_n$ that means disturbance amount is higher than noise hence state estimation can rely more on sensor data for mapping than on previous state. And sensor is more reliable.

if $W_n >> W_d$ that means noise amount is higher than disturbance hence state estimation can rely more on previous plot data for mapping than on sensor. And state is more reliable than sensor.

So according in paper we learned that non-linear system is defined as.

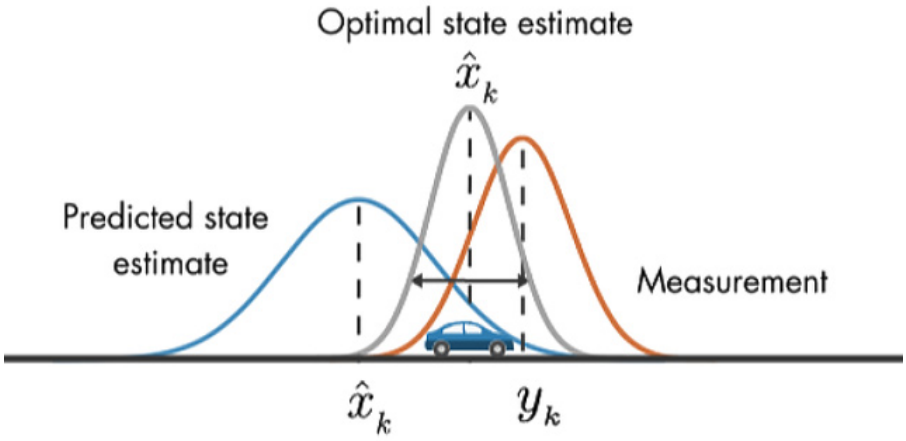


Fig. 14 Kalman filter state estimation [1]

Algorithm 1 Kalman filter algorithm

1: Kalman Filter $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2: $\mu_t = A_t\mu_{t-1} + B_t u_t$
3: $\Sigma_t = A_t\Sigma_{t-1}A_t^T + R_t$
4: $K_t = \Sigma_t C_t(C_t^T\Sigma_t C_t^T + Q_t)^{-1}$
5: $\mu_t = \mu_t + K_t(z_t - C_t\mu_t)$
6: $\Sigma_t = (I - K_t C_t)\Sigma_t$
7: return μ_t, Σ_t

So now lets take non-linear state equation as follows

$$\vec{z}_{k+1} = \vec{f}_k(\vec{z}_k) + v_k \tag{29}$$

$$\begin{pmatrix} \overrightarrow{x}_{k+1} \\ \overrightarrow{v}_{k+1} \\ \overrightarrow{a}_{k+1} \\ \overrightarrow{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} I_3 & \frac{T}{\lambda} I_3 & \frac{T^2}{2\lambda} I_3 & 0 \\ 0 & I_3 & T I_3 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \overrightarrow{x}_k \\ \overrightarrow{v}_k \\ \overrightarrow{a}_k \\ \overrightarrow{\lambda}_k \end{pmatrix} \tag{30}$$

where $x_k + 1$ is the position without scale of the IMU/Camera and v_{k+1} , $a_k + 1$ are the velocity and acceleration of the IMU/Camera in metric unit [m]. $_k$ is the gaussian process noise. Main reason behind extended kalman filter working is when ever transition takes places between matices resultant output remains Gaussian and have defined mean and co-variance so in out treatment we can only do those transformation where we dont loose condition of system being Gaussian. For example, Linear curve when transformed is Gaussian but if the function is non linear it is resultant does not result into Gaussian so whole point of using Kalman filter is killed. So in Extended Kalman filter we linearize the system. Every vector in z_k is resolved in the world frame W. Note that we do not include the orientation information in the model nor use it as a measurement in order. to keep the algorithm simple and fast. On each acceleration measurement we do the conversion from the inertial to the world frame by using a zero order hold of the unfiltered attitude measurement returned by the visual SLAM framework. As we work in a middle size environment with enough loops we assume negligible drift win the SLAM map and assume thus highly accurate attitude estimation from the visual SLAM framework. The model in its linearized form yields,

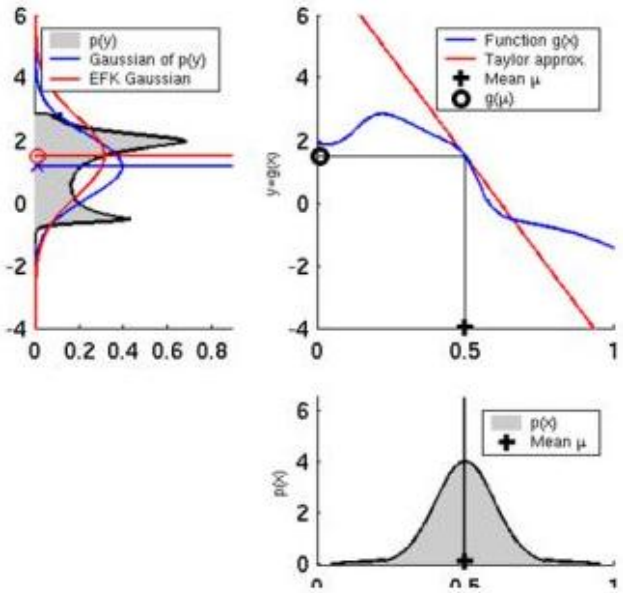


Fig. 15 Non-Linear system [1]

$$F_k = \begin{pmatrix} I_3 & \frac{T}{\lambda} I_3 & \frac{T^2}{2\lambda} I_3 & -\frac{T}{\lambda^2} I_3 - \frac{T^2}{2\lambda^2} I_3 \\ 0 & I_3 & TI_3 & 0 \\ 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{31}$$

For fusion implementation we consider the measurements in different observation vectors. For a multi rate filter, as it is in our case, the literature suggests two solutions. One would be using a (higher order) hold to synchronize the different measurements. Another is to weight the uncertainty of the measurement according to its temporal occurrence. We claim no certainty at all if no measurement is available (i.e. the measurement noise variance is infinite). Thus the update equations simplify to improve results. A more complex weighting function (i.e. exponential decay in time) could also be applied, however, at the cost of speed. The measurement updates for the vision and the IMU yields (V and I denotes Vision and IMU)

The innovation done by authors in for the vision part is,

$$K_{V,k} = P_k^- H_{V,k}^T (H_{V,k} P_k H_{V,k}^T + R_v)^{-1} \tag{32}$$

$$\overrightarrow{z}_k = \overrightarrow{z}_k + K_{V,k}(\overrightarrow{x}_{SLAM} - H_{V,k} \overrightarrow{z}_k) \tag{33}$$

$$P_k = (I - K_{V,k} H_{V,k}) P_k^- \tag{34}$$

The innovation for by authors in the IMU part is,

$$K_{I,k} = P_k^- H_{I,k}^T (H_{I,k} P_k H_{I,k}^T + R_I)^{-1} \tag{35}$$

$$\overrightarrow{z}_k = \overrightarrow{z}_k + K_{I,k}(\overrightarrow{x}_{IMU} - H_{I,k} \overrightarrow{z}_k) \tag{36}$$

$$P_k = (I - K_{I,k} H_{I,k}) P_k^- \tag{37}$$

The two matrices R_I , R_V are the noise covariance matrices for the vision and IMU measurement inputs x_{SLAM} , a_{IMU} which are resolved in the world frame W. The vector x_{SLAM} is the position without scale obtained from the vision algorithm(SLAM). The IMU measurement a_{IMU} needs special attention, because significant errors arise in the conversion from the raw IMU output.

$$\overrightarrow{a}_w = R_{wc} R_{ca} (\overrightarrow{a}_a - \overrightarrow{b}) - \overrightarrow{g}_w \tag{38}$$

8 Simultaneous Localization and Mapping

In this section, we cover SLAM, and the different concepts related to SLAM such as Visual Odometry. We then explore the two different approaches to SLAM mentioned in the paper. We briefly cover filter based SLAM called MonoSLAM [6] and a key-frame based SLAM called Parallel Tracking and Mapping (PTAM) [13]. A comprehensive analysis of filtering based SLAM is given in [22]. Simultaneous Localization and Mapping (SLAM) is a technique for estimating the motion of the robot and reconstructing the map/structure of the unknown environment. SLAM using only visual information only is

specifically referred to as visual SLAM (vSLAM). The SLAM problem can be stated as follows:

How can a body navigate in a previously unknown environment while constantly building and updating a map of its workspace using onboard sensors only? **[16]**

Mathematically, this can be expressed as [21]
Input:

$$Control\ input:\ u_{1:T} = u_1, u_2, ..., u_T \tag{39}$$

$$Observations:\ z_{1:T} = z_1, z_2, ..., z_T \tag{40}$$

Output:

$$Map\ of\ Environment:\ m \tag{41}$$

$$Path\ of\ robot:\ x_{0:T} = x_0, x_1, ..., x_T \tag{42}$$

From the problem statement above, we notice that the robot has no a priori knowledge of the workspace or environment that it is in. This makes SLAM a very challenging problem in probabilistic robotics. This is also referred to as a chicken-egg problem in that we need to map the environment to get an accurate pose, but at the same time we also need an accurate pose to build a correct map. Therefore, it is an iterative process of estimating pose and building a map simultaneously. Accurate pose estimation is critical for many applications in computer vision, autonomous robotics and augmented reality. A survey of SLAM implementations is given in [3,4,23]. The basic modules for a vSLAM system are

- Initialization
- Tracking
- Mapping
- Relocalization
- Global Map optimization

According to [20], the relationship between vSLAM and VO can be represented as follows

$vSLAM = VO + global\ map\ optimization$

From Fig 16 we see that a typical SLAM system consists of a front-end and a back-end. The front-end is responsible for feature extraction while the back-end is responsible for MAP estimation. Both the methods (MonoSLAM & PTAM) are feature based implementations.

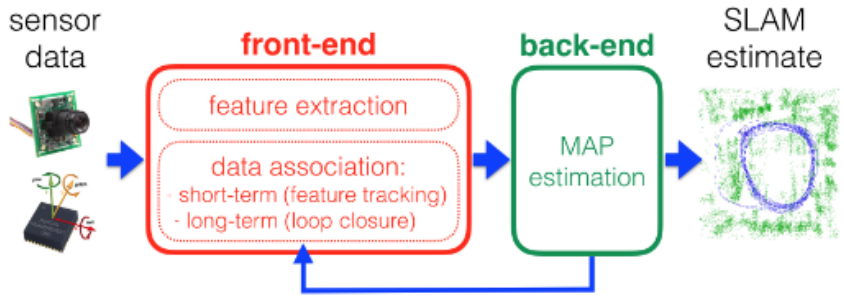


Fig. 16 SLAM architecture [4]

8.1 Visual Odometry

Odometry is the technique of estimating the position of a robot over time using sensors such as cameras, wheel encoders or any sensor measuring relative movement. Compared to SLAM which maintains a global consistent map, visual odometry (VO) maintains a local consistent map optimized over the last n frames. A generalized VO pipeline can be summarized as follows

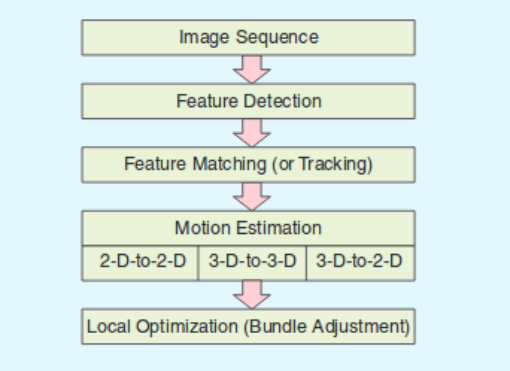


Fig. 17 Generalized Visual Odometry Pipeline [20]

Different algorithms exist due to the different type of cameras available such as Stereo, Mono, RGB-D [20,8]. As we use a monocular setup in our research paper, we would be going through the visual odometry algorithm from 2D to 2D correspondences for a monocular sensor. The algorithm is summarized as follows [20].

Algorithm 2 Monocular Visual Odometry

1: Capture a new image frame I_k
2: Extract and match features between frames I_k and I_{k-1}
3: Compute the essential matrix E from the matched feature points
4: Decompose E into R_k and t_k to form T_k
5: Choose the correct T_k matrix and scale t_k accordingly from scale factor
6: Compute $C_k = C_{k-1} * T_k$
7: Goto step 1.

Since we are using a monocular sensor, we always compare the image I_k captured at time instant k with the image I_{k-1} capture at time instant $k-1$. In a stereo setup, we would compare images from the left and right part of the stereo. Once an image is captured, we compute the features for image I_k and I_{k-1}. In the feature detection step, the image is searched for key-points called features which are likely to be present in successive images. Features can be defined as image patterns that differ from its immediate neighborhood in terms of intensity, color and texture. There exist a variety of feature detectors in literature which vary in performance and properties. Some of the properties of feature detectors are rotation, scale and affine invariant, repeatability, localization accuracy, robustness and efficiency. Heitanen et al. present a comparison of feature detectors and descriptors for object class matching in [10]. Once we detect features, we then need to compute a descriptor such that features detected in two images can be compared with each other. The easiest approach would be to form a patch of pixels and compare using a sum of squared distances (SSD) or normalized cross correlation (NCC) metric. Such descriptors are not invariant to any of the above mentioned properties and perform poorly in practical applications. One of the popular descriptors called SIFT [15] uses gradient orientations as its descriptors. This forms a 128-element descriptor that is invariant to most of the above mentioned properties such as rotation, scale and illumination which makes it widely applicable in practical real-time applications. For extracting features from two images there are two paths with one being to detect and match features independently in two frames and the other being to track features in subsequent frames an example of which is a KLT Tracker [24]. There are various methods employed to match features accurately such a RANSAC which stands for Random Sample Consensus and is used to remove outliers among feature matches. Once we detect and match features in I_k and I_{k-1}, we calculate the essential matrix E which describes the geometric relationship between two images. We can use Nisters five point algorithm [18] or Longuet-Higgins eight point algorithm [14] to get the essential matrix E. Once we get the essential matrix E, we decompose it to extract the rotation and translation parts. Four different solutions are obtained of which the correct pair can be found out by triangulation. The solutions are given as

$$R = U(\pm W^T)V^T \tag{43}$$

$$t = U(\pm W^T)SV^T \tag{44}$$

where R is the rotation matrix and t is the translation vector.

8.2 MonoSLAM

MonoSLAM was developed by Davison et al. [5,6]. It uses EKF as an estimator. In EKFSLAM, the state $\mathbf{x}$ space is represented by the robot state $\hat{x_\nu}$ along with the position of the different landmarks $\hat{y_i}$ that the robot observes. The state vector along with the covariance matrix can be represented as follows

$$\hat{x} = \begin{pmatrix} \hat{x_\nu} \\ \hat{y_1} \\ \hat{y_2} \\ : \end{pmatrix} P = \begin{pmatrix} P_{xx} & P_{xy1} & P_{xy2} & \cdots \\ P_{y1x} & P_{y1y1} & P_{y1y2} & \cdots \\ P_{y2x} & P_{y2y1} & P_{y1y2} & \cdots \\ : & : & : & \cdots \end{pmatrix} \tag{45}$$

The covariance matrix is used to represent the first order uncertainty. As the robot moves around, new features are added to the state vector. It is important to note that the robot initialization is done by using a known object. The prediction and correction steps are performed as mentioned in the EKF algorithm in section 7. One of the problems with this method is that the state vector becomes large when the number of land marks or key-points tracked increases. This increases the computation time.

8.3 Parallel Tracking and Mapping (PTAM)

PTAM is a SLAM system specifically designed to track a hand-held camera in a small AR workspace. For real-time operation, PTAM splits the tracking and mapping threads into two separate tasks running in parallel. The tracking thread robustly tracks hand-held motion and the mapping thread produces a 3D map of the features. Although developed specific to AR applications, the concept of running operations on multiple threads was incorporated into future SLAM algorithms.

The map consists of a collection of key-points/features that are located in a world frame W. They are represented in the homogeneous form. The map also contains key-frames which a instances of the frames taken from the hand-held camera. Each point is stored with a source key-frame. Unlike earlier methods which process each and every frame, this method only processes frames when there is sufficient information. Thus incremental mapping is replaced with a more computationally expensive batch method called bundle adjustment.

The tracking algorithm can be summarized as follows.

Algorithm 3 PTAM - Tracking Algorithm

1: A new frame is captured and a prior pose estimate is generated from motion model.
2: Map points are projected into the image based on prior pose.
3: Small number of features (50) are searched in the image.
4: The camera pose is updated from the feature matches.
5: Large number of points is re-projected and searched in the image.
6: Final pose estimate is computed from all matched found.
7: Goto step 1.

PTAM is initialized by a lateral offset movement of the hand-held camera. This can be assumed as a stereo image for starting the mapping. The initial map is constructed using the 5 point algorithm [18] with arbitrary scale. As the camera moves, the key-frames increase from an initial two. Here, we only add key-frames if the tracking quality is good and a minimum of twenty frames has passed from the last keyframe. Bundle adjustemnt is iteratively performed to adjust the map based on a cost function. The mapping algorithm can then be summarized as.

Algorithm 4 PTAM - Mapping Algorithm

Require: Stereo Initialization
if New Keyframe **then**
  Update key-frame
  Integrate key-frame
  Add new features
else if Locally Converged **then**
  if Globally Converged **then**
    Update data association
  else
    Global bundle adjust
  end if
else
  Locally bundle adjust
end if
Sleep 5ms
Goto step 1.

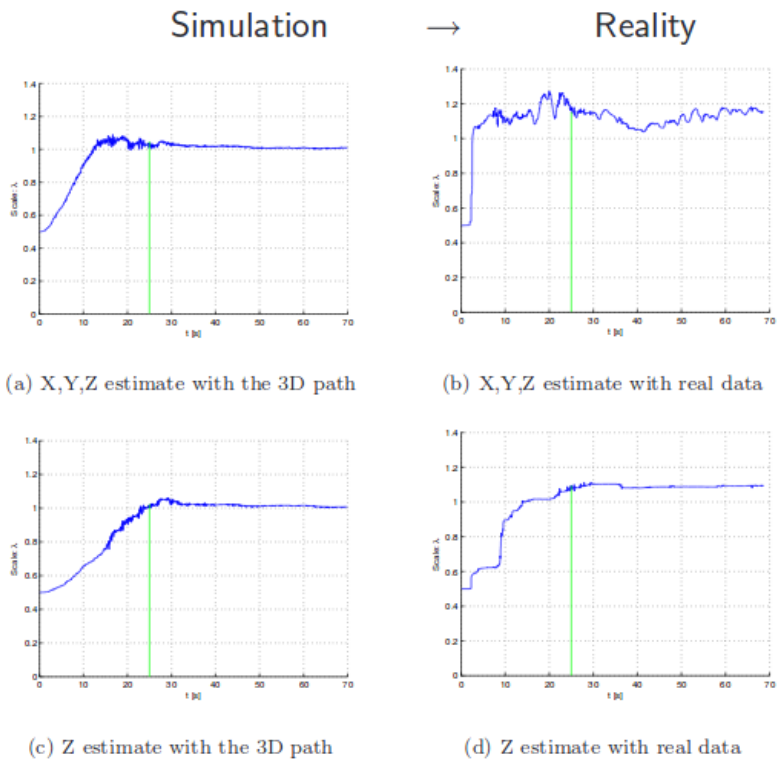
Finally when compared with Filter based SLAM, PTAM can handle thousands of features by splitting the tracking and mapping onto different threads on the CPU. We see that SLAM algorithms built now-a-days have used this philosophy to great success.

8.4 Results

8.4.1 Simulated and Real Data

In the paper, the Kalman filter was simulated offline with simulated data from the spline fitting section. The two inputs to the Kalman filter were the position from the vision sensor and acceleration from the IMU. Three different approaches were taken. The first method is same as Eq. 7, the second method only uses the Z-axis which gives the states as $[\overrightarrow{x}_z, \overrightarrow{v}_z, \overrightarrow{\alpha}_z, \lambda]$ and the third uses only the X and Y-axis.

Figure 18 shows the scale estimation $\lambda(t)$ for a simulated 3D path on left and actual path on right. The standard deviation for the acceleration noise for simulation was chosen same as real data($\sigma_{SLAM} = 0.01, \sigma_{IMU} = 0.2m/s^2$) with initial velocity and acceleration set to zero. Plots in Fig. 18a and Fig. 18c do not differ due to simulating the acceleration from the spline ideally in



(a) X,Y,Z estimate with the 3D path

(b) X,Y,Z estimate with real data

(c) Z estimate with the 3D path

(d) Z estimate with real data

Fig. 18 Offline simulation results with simulated and real data. Fewer directions (X,Y,Z) included in Kalman filter gives us a better estimate of λ. [19]

world frame. The estimate becomes more accurate when we use less number of directions (X,Y,Z). Wrong measurements in acceleration influence λ and make the scale estimation sensitive as seen in Fig. 18b. Hence only using a single axis (Z-axis) gives us the best result.

8.4.2 Online Implementation

For the online implementation, the third setup was incorporated into the PTAM code [12]. As mentioned in section 8.3 PTAM employs two parallel threads called Tracker for tracking and MapMaker for mapping. Two additional threads for IMU and Kalman we added. The IMU thread provides the acceleration measurements. The Kalman thread starts with λ calculated from integrating acceleration values, position from SLAM algorithm and acceleration and velocity set to 0. Values for covariance matrix $\mathbf{Q}$ is time-varied which provides control over the sensitivity of the Kalman filter. For our report we have not incorporated the online implementation and have reported results from the paper.

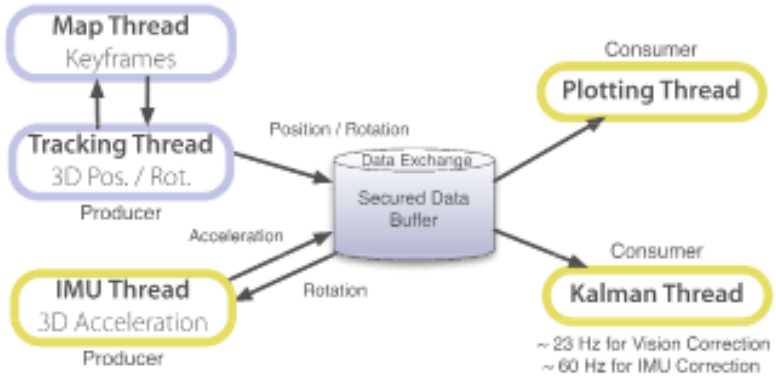


Fig. 19 Online implementation with PTAM work flow.Blue color boxes are original PTAM threads. Yellow boxes are added threads for scale estimation λ. [19]

References

1. Byron Boots. Statistical techniques in robotics, 2015. URL: `https://www.cc.gatech.edu/~bboots3/STR-Spring2015/Lectures/Lecture12/Lecture12_notes.pdf`. Last visited on 11/18/2018.
2. J-Y Bouguet. Camera calibration toolbox for matlab. *http://www. vision. caltech. edu/bouguetj/calib_doc/index. html*, 2004.
3. Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 20:1–1, 2017.
4. Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.
5. Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. page 1403. IEEE, 2003.
6. Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1052–1067, 2007.
7. Starlino Electronics. Imu guide. URL: `http://www.starlino.com/imu_guide.html`. Last visited on 11/18/2018.
8. Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90, 2012.
9. Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106–1112. IEEE, 1997.
10. Antti Hietanen, Jukka Lankinen, Joni-Kristian Kämäräinen, Anders Glent Buch, and Norbert Krüger. A comparison of feature detectors and descriptors for object class matching. *Neurocomputing*, 184:3–12, 2016.
11. S-H Jung and Camillo J Taylor. Camera trajectory estimation using inertial sensor measurements and structure from motion results. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2001.
12. Georg Klein and David Murray. Source code of ptam (parallel tracking and mapping). URL: `http://www.robots.ox.ac.uk/~gk/PTAM/`. Last visited on 11/18/2018.

13. Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.
14. H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133, 1981.
15. David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
16. Roland Siegwart Margarita Chli, Martin Rufli. Lecture notes in autonomous mobile robots, 2017. URL: `https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/asl-dam/documents/lectures/autonomous_mobile_robots/spring-2017/2017%20-%20SLAM%20I%20-%20after%20lecture.pdf`. Last visited on 11/18/2018.
17. MATLAB. What is camera calibration. URL: `https://www.mathworks.com/help/vision/ug/camera-calibration.html`. Last visited on 11/18/2018.
18. David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.
19. Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems*, 61(1-4):287–299, 2011.
20. Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.
21. Cyrill Stachniss. Lecture notes in robot mapping, 2013. URL: `http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam04-ekf-slam.pdf`. Last visited on 11/18/2018.
22. Hauke Strasdata, JMM Montielb, and Andrew J Davisona. Visual slam: Why filter?
23. Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, 2017.
24. Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.
25. Rein van den Boomgaard. Lecture notes image processing and computer vision. URL: `https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/IP/Images/ImageFormation.html`. Last visited on 11/18/2018.
26. Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

9 MATLAB Code

9.1 Spline Fit 2D

```
clear all
clear
n=60; % Points to be considered in curve
X=linspace(1,15,n); % equal spacing of points
noise=(10)*(rand()-0.5);  % noise in data from -1 to 1
Y=(0.1).*X.^3 - (1.5)*X.^2 + (6).*X + noise; % noisy
    output

Camerax=[2 4 6 8 12 14 ]
Cameray=[0 10 -5 20 20 80]
Scale1=0.5
Scale2=0.5
Scale3=0.1
%######################## Without section
    ############################

Xn=linspace(1,15,n);
S4=Xn.^3*transpose(Xn); % X^4 terms sum
S3=Xn.^2*transpose(Xn); % X^3 terms sum
S2=Xn*transpose(Xn); % X^2 terms sum
S1=Xn*ones(n,1); % X terms sum
%   AX=B
An=[S4 S3 S2;
    S3 S2 S1;
    S2 S1 1];

Bn=[Y*transpose(Xn.^2);Y*transpose(Xn);Y*ones(n,1)];
Constant1 = inv(An)*Bn; % by least square method

Ylinen=(Constant1(1,1).*Xn.^2 + Constant1(2,1).*Xn +
    Constant1(3,1)) ;
errorn=Y-Ylinen; % Error term in actual and computed

%###################Section wise ####################

%for first section
X1=linspace(1,5,n/3);

S4=X1.^3*transpose(X1); % X^4 terms sum
S3=X1.^2*transpose(X1); % X^3 terms sum
S2=X1*transpose(X1); % X^2 terms sum
```

```
39 S1=X1*ones(n/3,1); % X terms sum
40
41 %    AX=B
42 A1=[S4  S3  S2;
43     S3  S2  S1;
44     S2  S1   1];
45
46 B1=[Y(1:n/3)*transpose(X1.^2);Y(1:n/3)*transpose(X1);Y
       (1:n/3)*ones(n/3,1)];
47 ConstCamera1=inv([Camerax(1,1)  1;Camerax(1,2)  1])*[
       Camerax(1,1);Camerax(1,2)];
48 Constant1=inv(A1)*B1; % by least square method
49
50 Yline1=(Constant1(1,1).*X1.^2 + Constant1(2,1).*X1 +
       Constant1(3,1))+ Scale1*(ConstCamera1(1,1)*X1+
       ConstCamera1(2,1));
51 error1=Y(1:n/3)-Yline1; % Error term in actual and
       computed
52
53 %
    ##########################################################################
54 %for second section
55 X2=linspace(5,10,n/3);
56
57 S4=X2.^3*transpose(X2); % X^4 terms sum
58 S3=X2.^2*transpose(X2); % X^3 terms sum
59 S2=X2*transpose(X2); % X^2 terms sum
60 S1=X2*ones(n/3,1); % X terms sum
61
62 %    AX=B
63 A2=[S4  S3  S2;
64     S3  S2  S1;
65     S2  S1   1];
66
67 B2=[Y(((n/3)+1):2*n/3)*transpose(X2.^2);Y(((n/3)+1):2*n
       /3)*transpose(X2);Y(((n/3)+1):2*n/3)*ones(n/3,1)];
68 ConstCamera2=inv([Camerax(1,3)  1;Camerax(1,4)  1])*[
       Camerax(1,3);Camerax(1,4)];
69 Constant2=inv(A2)*B2; % by least square method
70
71 Yline2=(Constant2(1,1).*X2.^2 + Constant2(2,1).*X2 +
       Constant2(3,1))+Scale2*(ConstCamera2(1,1)*X1+
       ConstCamera2(2,1));
72 error2=Y(((n/3)+1):2*n/3)-Yline2; % Error term in actual
        and computed
```

```
73
74 %
       ##########################################################################
75
76 %for third section
77 X3=linspace(10,15,n/3);
78
79 S4=X3.^3*transpose(X3); % X^4 terms sum
80 S3=X3.^2*transpose(X3); % X^3 terms sum
81 S2=X3*transpose(X3); % X^2 terms sum
82 S1=X3*ones(n/3,1); % X terms sum
83
84 %    AX=B
85 A3=[S4 S3 S2;
86     S3 S2 S1;
87     S2 S1 1];
88
89 B3=[Y(((2*n/3)+1):n)*transpose(X3.^2);Y(((2*n/3)+1):n)*
       transpose(X3);Y(((2*n/3)+1):n)*ones(n/3,1)];
90 ConstCamera3=inv([Camerax(1,5) 1;Camerax(1,6) 1])*[
       Camerax(1,5);Camerax(1,6)];
91 Constant3=inv(A3)*B3; % by least square method
92
93 Yline3=(Constant3(1,1).*X3.^2 + Constant3(2,1).*X3 +
       Constant3(3,1))+Scale3*(ConstCamera3(1,1)*X1+
       ConstCamera3(2,1));
94 error3=Y(((2*n/3)+1):n)-Yline3; % Error term in actual
       and computed
95
96 %########################## Graph section
       ####################
97
98 error=[error1 error2 error3 ];
99 Yline=[Yline1 Yline2 Yline3];
100 Xf=[X1 X2 X3];
101 figure;
102 hold on
103 plot(Xf,Yline,'-or')
104 plot(X,Y,'+-b',Xn,Ylinen,'-og');grid;
105 plot(Camerax,Cameray,'*k')
106 figure;
107 plot(X,abs(error),'-g',X,abs(errorn),'-r');grid;
```

9.2 Spline Fit 3D

```
%######################## Initialize
    ############################
clear all
clear
n=20 % Points to be considered in curve
t=linspace(0,5,n) % equal spacing of points

%########################## Noise
    ##############################

noisex=5*(rand(1,n)-0.5) %noise in x
noisey=5*(rand(1,n)-0.5) %noise in y
noisez=5*(rand(1,n)-0.5) %noise in z

%########################## Noisy Data
    ##########################

%X=3.*t.^2 + 5.*t + 2 + noisex %noisy data in x
%Y=2.*t.^2 + 7.*t + 2 + noisey %noisy data in y
%Z=3.*t.^2 + 4.*t + 2 + noisez %noisy data in z
%#################### Summation matrix
    ########################

S4=t.^3*transpose(t) % t^4 terms sum
S3=t.^2*transpose(t) % t^3 terms sum
S2=t*transpose(t) % t^2 terms sum
S1=t*ones(n,1) % X terms sum

%################## Least Square matrix
    ########################
%   AX=B
A=[S4 S3 S2;  %A matrix for computing X matrix
    S3 S2 S1;
    S2 S1 1]

Bx=[X*transpose(t.^2);X*transpose(t);Y*ones(n,1)] %B in
    x
By=[Y*transpose(t.^2);Y*transpose(t);Y*ones(n,1)] %B in
    y
Bz=[Z*transpose(t.^2);Z*transpose(t);Z*ones(n,1)] %B in
    z

Constantx=inv(A)*Bx % by least square method x
Constanty=inv(A)*By % by least square method y
Constantz=inv(A)*Bz % by least square method z

```

```
%########### Computing value of new co-ordinates ###########

Ylinex=Constantx(1,1).*t.^2 + Constantx(2,1).*t + Constantx(3,1) %X point
Yliney=Constanty(1,1).*t.^2 + Constanty(2,1).*t + Constanty(3,1) %Y point
Ylinez=Constantz(1,1).*t.^2 + Constantz(2,1).*t + Constantz(3,1) %Z point

errorx= X-Ylinex % error in x
errory= Y-Yliney % error in y
errorz= Z-Ylinez % error in z

error = abs(((errorx).^2 + errory.^2 + errorz.^2).^(0.5)) % final error
%######################## Plots ##############################
figure;
plot3(X,Y,Z,'o-b');grid; %plot of noisy data
hold on
plot3(Ylinex,Yliney,Ylinez,'--r');grid; %plot of least square data
figure;
plot(t,error,'-g');grid; % Error wrt to time
figure;
plot(X,t,'--b')
%########################## Velocity ##########################

Vx=6.*t + 5 %velcity data in x
Vy=4.*t + 7 %velcity data in y
Vz=6.*t + 4 %velcity data in z
figure;
plot(t,Vx,'-g',t,Vy,'-r',t,Vz,'-b');grid;
```