EKFMobileSLAM

A mobile Simultaneous Location and Mapping (SLAM) platform

University of Arizona
ECE 573
February 2015 (SP 15)

Team D.R.Y.SLAM
Anthony Rodriguez
Shuai Yuan
Shreenivaas Devarajan

Summary

Simultaneous Location and Mapping (SLAM) is the computational problem of constructing a virtual representation of an environment based upon sensory input gathered while in the environment. SLAM is a problem with many challenges. While most of these challenges are generally considered to be solved, commercial releases of these capabilities are rare due to the relative infancy of the field. The SLAM approach takes into account probabilistic calculations so as to estimate the location. The location is estimated based on the observations and the method employs conditional probability estimates to arrive at the estimate. The key feature of SLAM is to extract the landmark (object) and state estimation. From the landmark extracted, the state is updated and the landmark is extracted. The SLAM process is further augmented by the use of a filter known as Extended Kalman Filter which is described in the next section in detail.

Table of Contents

Table of Figures

Table of Tables

2. Project Overview

In our project, we employ hypothesis testing using the Extended Kalman Filter and Random Sample Consensus (RANSAC) in order to perform real time using improvements. The same concept was shown in Civera et. Al[1] using the same methodology. To demonstrate the concept, the algorithm (termed EKFMonoSLAM) was implemented in MATLAB's interpretive scripting language to run on a pre-recorded video sequence.

The primary goal of this project is to implement the 1-point RANSAC method as a standalone program that may be used on an embedded device. To demonstrate the success of this method and show 1-point RANSAC's ability to operate in a system with limited processing power, the algorithm will be run on a mobile device with access to an onboard camera, among other sensors. Given the success of this experiment we intend to pursue two other ventures: (1) Show how the addition of accelerometer and gyroscope data to EKFMonoSLAM's Extended Kalman Filter (EKF) affects the program, and/or (2) demonstrate the ability of this algorithm to capture, process and display location data simultaneously in real-time. The mobile device platform intended for this project is an Apple iPhone 6 (2014). Our main goal in this project is to use the mobile device in place of a sophisticated Robot to perform EKFMonoSLAM and show that the application can utilize the iPhone's Accelerometer and Gyroscope to perform accurate landmark extraction.

2. Requirements

The general requirements of the application are shown in table 1:

Table 1: Requirements

1. The app integrates Accelerometer and Gyroscope data to initialize EKF. (Difficult rating: 6)
   1) Accelerometer and Gyroscope reflects actual movement in the environment.
   2) The rate that the data generated synchronizes with the rate that app generates video.
   3) EKFMonoSLAM algorithm takes advantage of saved data.
2. The program implements EKF to estimate and update the matrix of location. (Difficult rating: 10)
   1) EKFMonoSLAM algorithm utilizes a recorded video to estimate and update.
   2) EKFMonoSLAM algorithm is able to process location data in real time.
3. The program extracts landmark features and saves them as a series of lines. (Difficult rating: 5)
   1) EKFMonoSLAM algorithm can decode landmark features of the saved video correctly.
   2) EKFMonoSLAM algorithm is still accurate when it extracts lines from a real time vedio.
4. The program stores processed landmarks as point cloud. (Difficult rating: 5)
   1) Detect landmarks using 1 Point RANSAC algorithm with a discrete series of reference images.
   2) Detect corresponding features in successive images using 1 Point RANSAC algorithm.
5. The app records image data as video from camera. (Difficult rating: 4)
6. The app displays the gathered point cloud graphically. (Difficult rating: 6)
   1) EKFMonoSLAM algorithm can gather point cloud form an saved video.
   2) EKFMonoSLAM algorithm process real time point cloud as exactly as the saved video.
7. The app displays the processed playback in real time. (Difficult rating: 9)

In general, meeting of requirements (1), (2), (3), (4) & (6) is considered successful with respect to the project goals ("B" criteria), while meeting of either goal (5) or (7) is considered to be our "A" criteria. The "Difficulty Rating" metric roughly corresponds to the number of weeks that are expected for a single individual to accomplish the task. Since our development team consists of three individuals, the total amount of "development-weeks" divided by the size of our team is equal to 15, which is slightly longer than our allotted development time for the project. However, since not all requirements must be met for the "A" criteria, these initial requirements appear reasonable.

It is worth noting that requirements (2) and (4) are part of the EKFMonoSLAM program and are considered the main dependencies for the project. Because of this, these requirements will be the first to be implemented and tested (see timeline in Section 5 for more information).

3. iPhone integration and the View-Controller architecture

3.1 Summary

The View-Controller Figure 1a consists of the main App Delegate, Primary view and three subviews that interface to a Model Adapter. The three subviews/subcontrollers are responsible for the three operating states of the UI: *Recording*, *Processing* and *Displaying* data. The Controller-Model adapter allows the EKFMonoSLAM model to operate as a standalone C program: The Model is then agnostic to the source of the data, allowing the system to operate the same whether integrated into a desktop program or operating within the embedded mobile system. The state diagram showing how these interact is shown in Figure 1b.

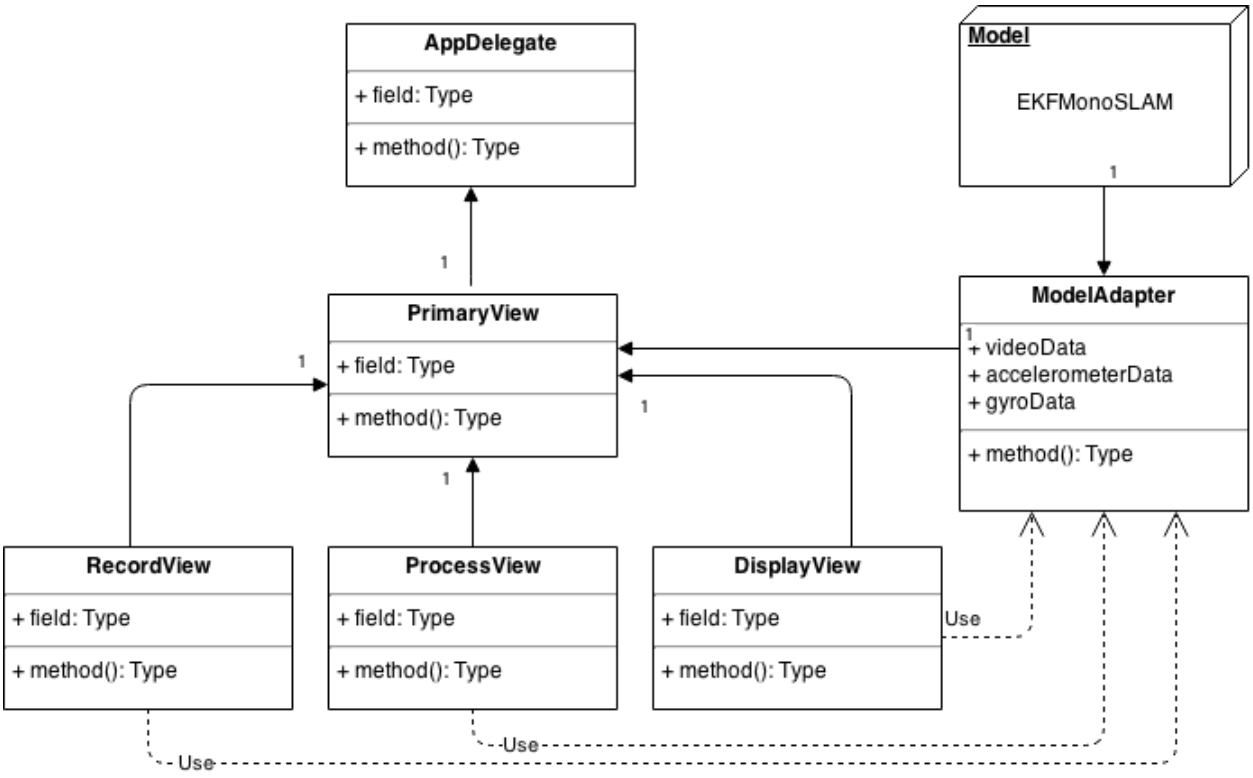

Figure 1a: The EKFMonoSLAM standalone program interfaces with an Adapter that is either present in the physical system or simulated for standalone testing

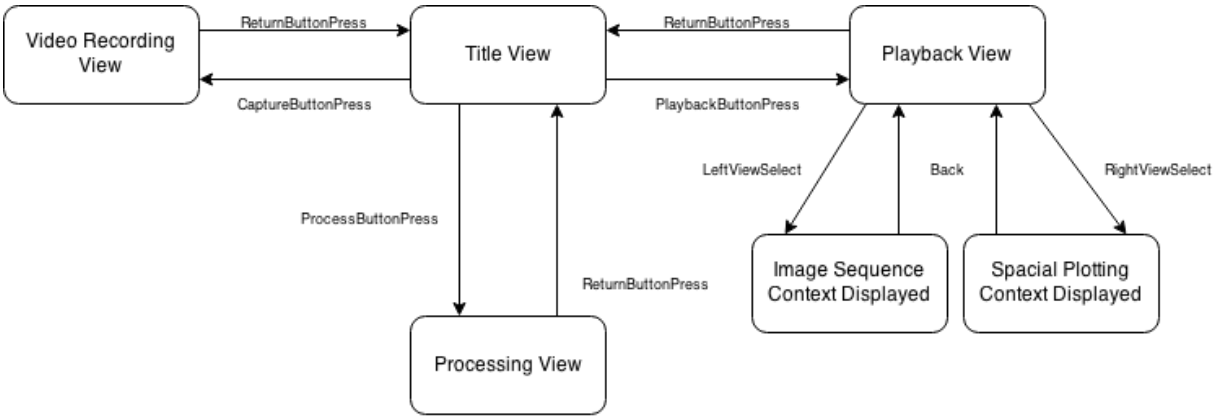

Figure 1b: State diagram showing how the UI should be navigated

3.2 RecordViewController and Sensor Integration

The *RecordViewController* class is responsible for the gathering and synchronization of data gathered by the device itself. If in the future real-time calculation is accomplished, this view will also show the landmarks as they are extracted. In the case of monocular input as the only data source the task of gathering data becomes trivial as the only goal is to record successive frames. However, integration of accelerometer and gyroscope data for the EKF demands synchronization of the data vector for each specific time-step.

In Civera et. al. [1], the authors show successful validation for their SLAM algorithm in real-time with low resolution at 30fps. Fortunately, iOS OpenCV and Core frameworks provided for developers allow setting of framerate, grayscale and resolution parameters for video capture. In addition, the Apple CoreMotion framework allows polling of 3-axis accelerometer and gyroscope data at any discrete time interval. This will allow the video frame matrix and the vector of sensor parameters to be indexed by time interval during recording.

Because duplication of a working, offline EKFMonoSLAM copy in C is the first important milestone of the project, the RecordViewController is also responsible for creating a log of the data gathered during the recording process for exporting. In this case, the easiest method of accomplishing this appears to be saving a grayscale video and a corresponding ASCII data file and sending via the built-in iOS email client. An alternative solution involves transmitting the data via TCP/IP to a server dedicated to processing these files. This process is described further in Section 5.

3.3 ProcessViewController

The *Processing* UI state is responsible for initiating the MonoSLAM computational process and as such is the most computation-heavy portion of the application. For the final product, this view may potentially depreciate and be assimilated into the RecordViewController in the case that the system is able to perform in real-time, a decision that will be made during the intermediate verification phase. Prior to that situation, however, the ProcessViewController will be responsible for launching the MonoSLAM process and providing the user some progress feedback during processing.

In order to avoid hang-ups and locking in the UI, it is necessary to implement multithreading and poll the processing thread's status while the program is running. Shared memory located within the ModelAdapter class will be used to enforce concurrency between the threads and allow the application UI to inform the user of progress updates while the algorithm runs. This process is shown in Figure 2.

The ultimate goal of processing is to produce a simple Comma-Separated Value (CSV) ASCII file which can be read by the DisplayViewController(Section 3.4) facilitating the rendering of all mapping data produced by the algorithm.

3.4 DisplayViewController

The *DisplayViewController* will host the view outlets that display the processed SLAM results to the user. The minimum goal for this project is to create an approximation of EKFMonoSLAM's MATLAB figure outputs (Figure 3). Because of the limited screen space on the mobile device, however, an alternate approach is needed to channel the program's results without any loss of critical information.

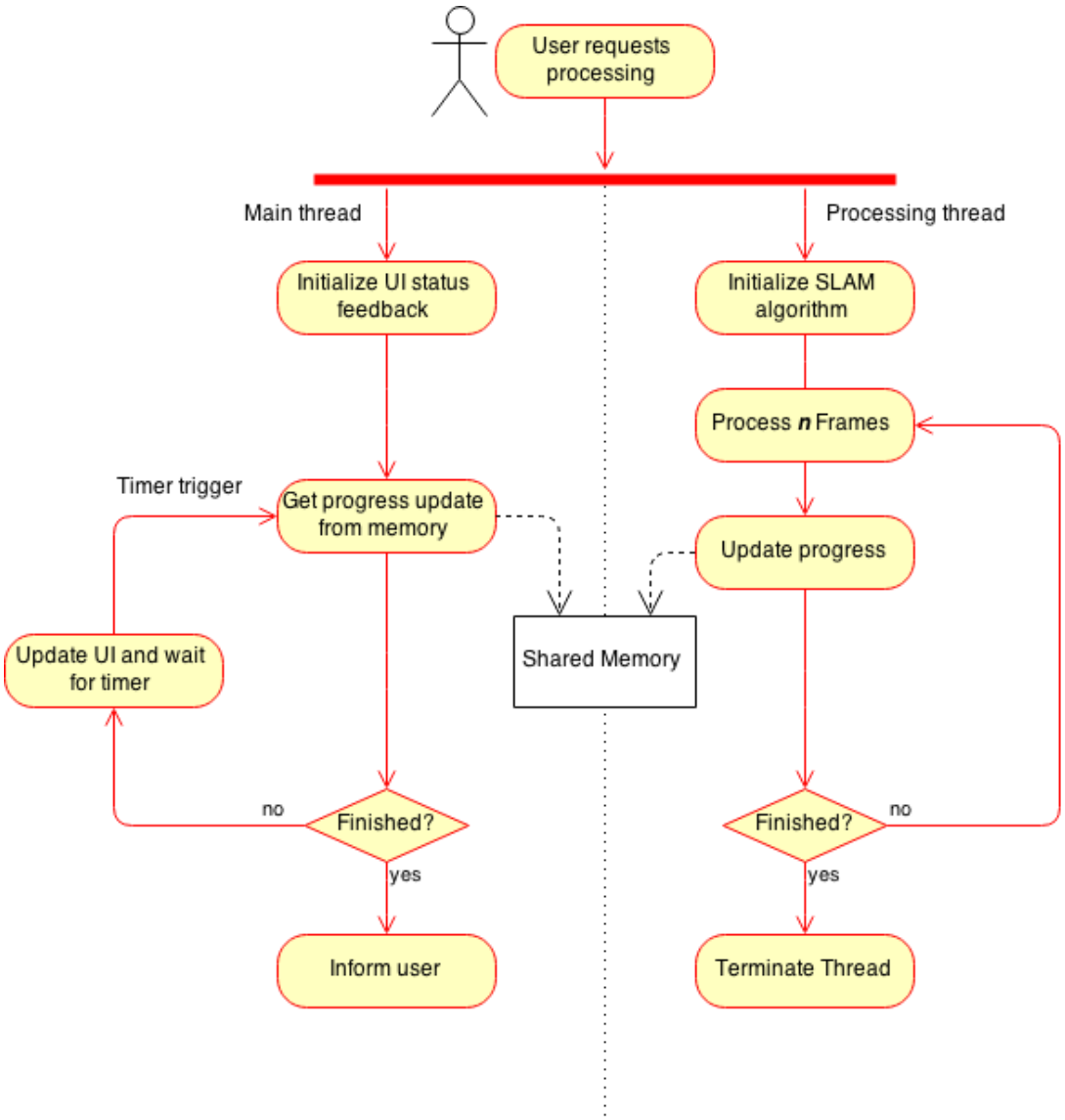

Figure 2: In order to produce a usable interface while the SLAM algorithm is running, the process must be partitioned into separate threads

The CSV file described in Section 5 is read by the *DisplayViewController* to facilitate plotting. The candidate frameworks for producing these plots include OpenGL and Apple's Core-Plot framework. At the time of this writing, the development plan focuses on utilizing OpenGL both for video overlay (Figure 3 left) as well as plotting in 3D- or 2D-space (Figure 3 right). The video overlay shows the landmarks tracked by the algorithm and the plot shows where the 1-Point RANSAC algorithm places those points in relation to the phone. **These are the main deliverable features of the application.** Because data is collected at a rate of 30Hz and because the target platform (at least in the case of Apple's iPhone 6) uses separate GPU processes for rendering, no rendering lag is anticipated for approximately real-time playback. However, reducing framerate for playback to a range of 10-15Hz would be only a matter of aesthetic appearance and would not cause any loss of critical information.

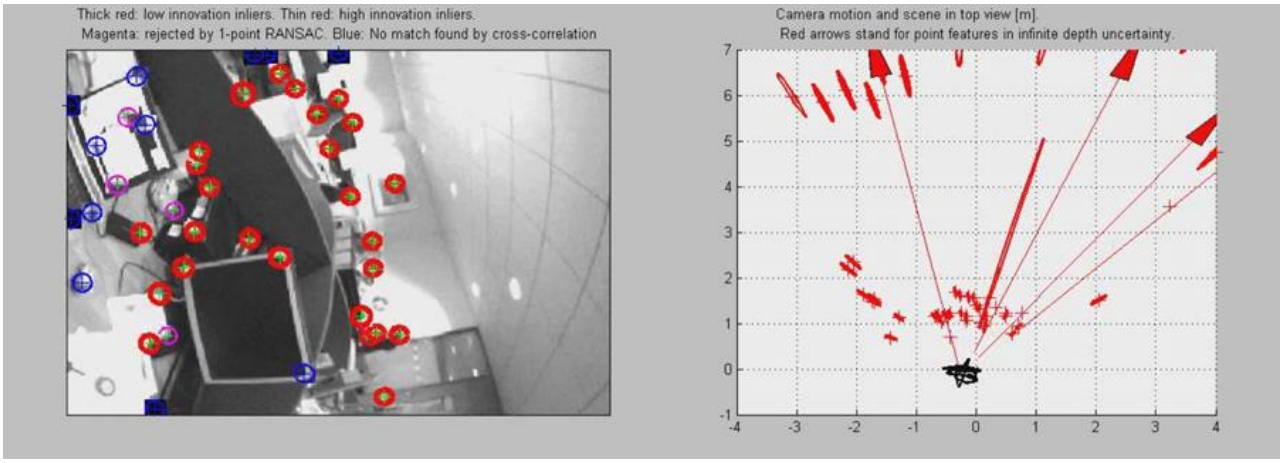

Figure 3: EKFMonoSLAM MATLAB figures. Landmarks are superimposed over image data on the left, while corresponding mapped locations are plotted on the right.

This ViewController is to be designed with the intention of preserving the viewing sequence shown in Figure 3. To accomplish this task, the plan is to independently develop two separate OpenGL UI contexts: One for each sub-image in Figure 3. One intuitive method for accomplishing this is to display the two views side-by-side with the ability to maximize either context in the View upon receiving a tap gesture by the user. The concept is illustrated in Figure 4.

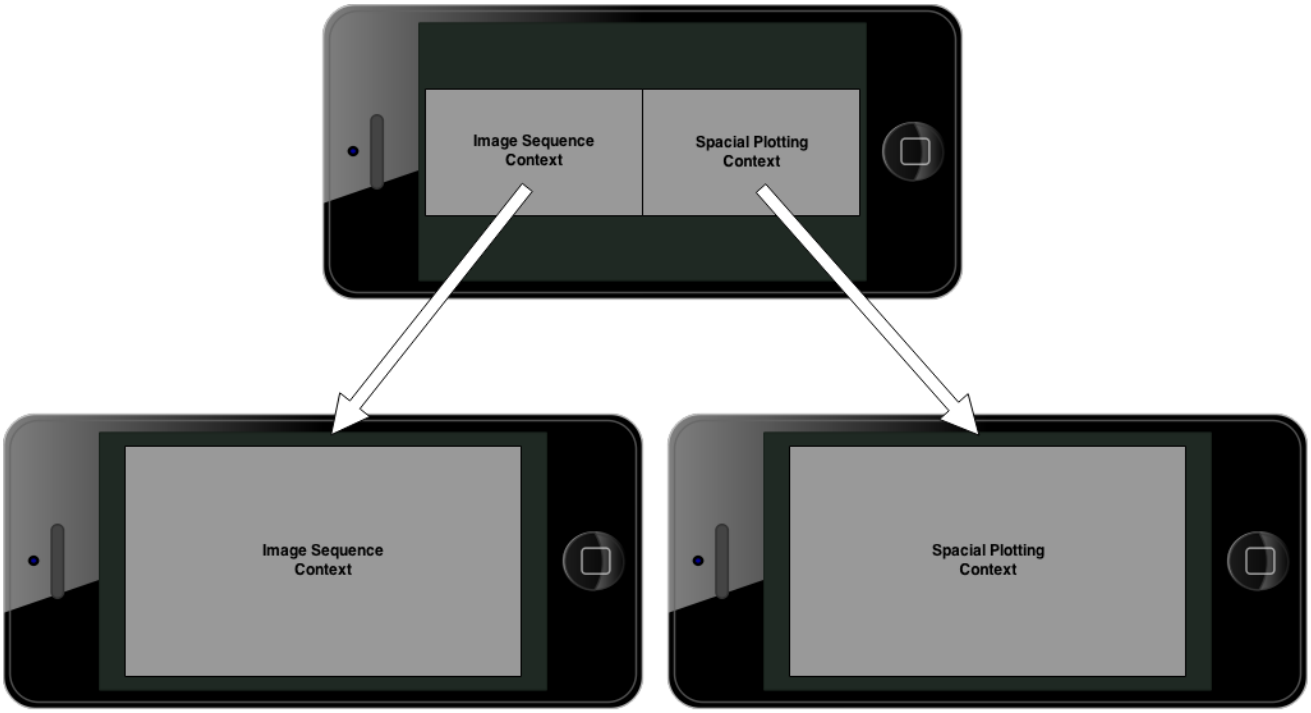

Figure 4: Playback sequences may be displayed side-by-side or individually at the request of the user

4. EKFMonoSLAM and 1-Point RANSAC

4.1 System Summary

The signal flow diagram in Figure 5 shows how objects, both hardware and software, interact with each other. The environment is accessible by iPhone camera. Accelerometer and gyroscope delivers the data collected from iPhone camera to data storage device. Then EKF filter initializes with the information about state vector and covariance vector given by storage device. For each step of processing, the EKF filter picks up old state and covariance from storage device and sends back new ones to it. The EKF filter provides prior information to RANSAC algorithm which is used to extract and update landmarks. Then the landmarks are stored in the hardware. After image processing, it will show up on the screen of iPhone.

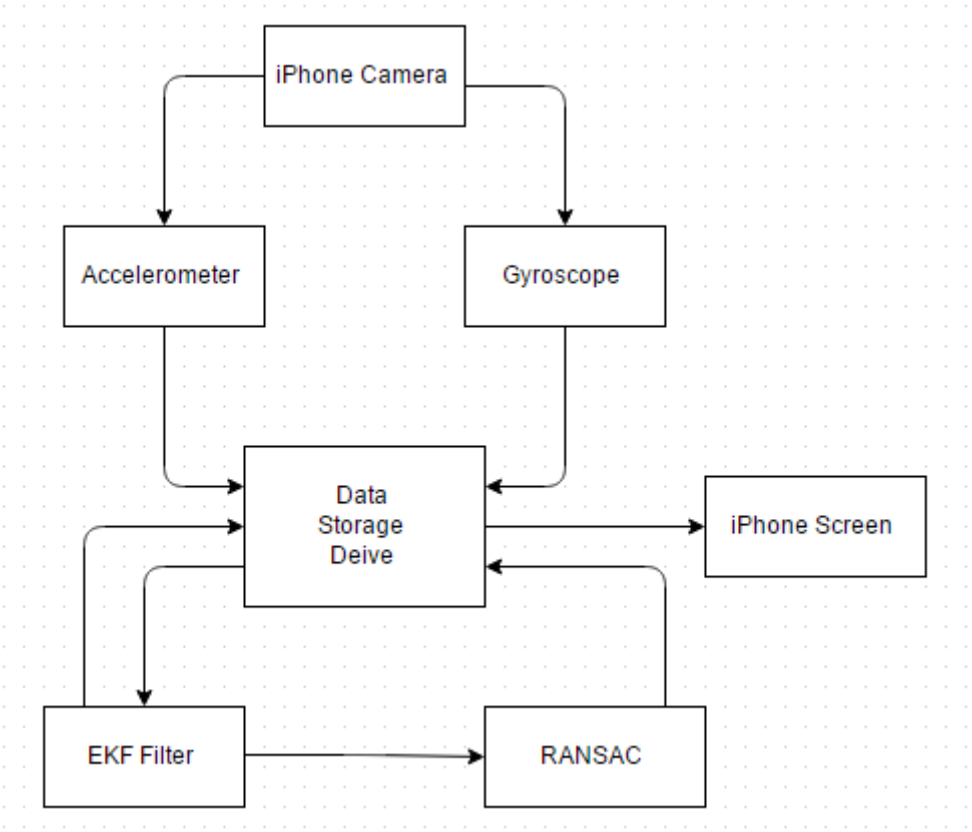

Figure 5: The EKF/RANSAC processes share a common storage device with the hardware system. Instead of interacting directly with the sensors, data is stored to and read from the storage device with the assumption that the data is ready for use.

4.2 1-Point RANSAC Hypothesis Testing

Figure 5 explains the relationship between multiple operations. In order to initialize the program, the information about state, covariance, camera and image is required to input. State and Covariance are used to give initial value of EKF filter, while camera and image are essential for map management and search matches. RANSAC hypotheses are constructed by prior knowledge provided by search matches. The following LI filter help decide inliers from sampling point, and HI filter re-estimates inliers for a more accurate result. Once iPhone users stop recording a video, the mapping of recorded environment will display. Otherwise, the RANSAC algorithm will be repeated.

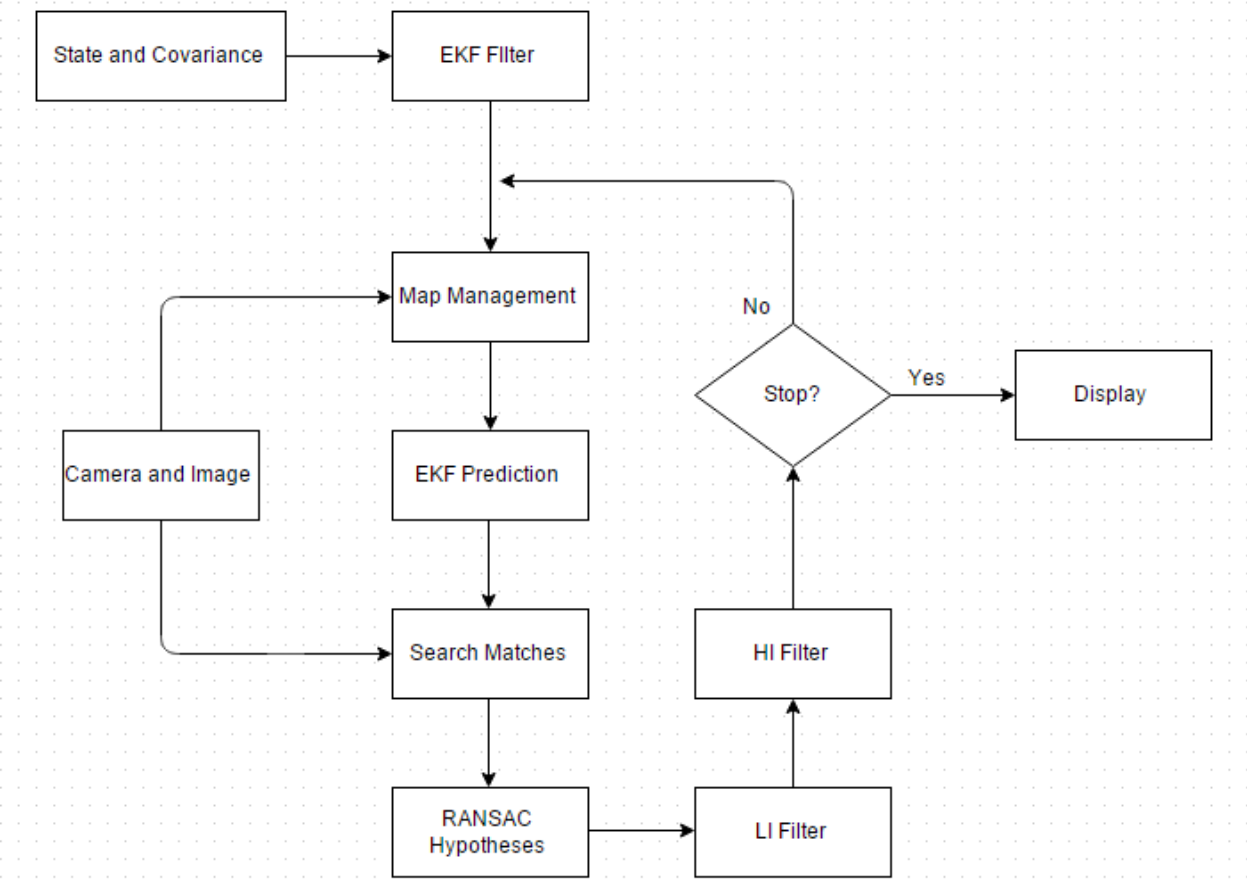

Figure 6: General overview of 1-point RANSAC iterative process

Random Sample Consensus (RANSAC) is an algorithm to extract lines from a sensor to use as landmarks. It works by constructing model hypothesis from 2 points of random minimal data subsets and evaluating their validity from the support of the whole data. Based on traditional RANSAC algorithm, 1-Point RANSAC algorithm is an improved technique that takes advantage of prior information given by the Extended Kalman Filter and then construct hypotheses with 1 point, which leads to a reduced sample size of random hypotheses.

Generally, 1-Point RANSAC algorithm is divided into four steps: 1) collecting information from EKF prediction and individually compatible matching points; 2) proposing a certain number of hypotheses and select the most supported one; 3) distinguishing inliers and outliers among sampling points; 4) rescuing possible inliers from outliers and re-estimating model.

In the first step, the most important prior knowledge is obtained from the estimation on state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k|k-1}}$ and covariance vector $\boldsymbol{P}_{\boldsymbol{k|k-1}}$ provided by the EKF. The predicted state vector $\boldsymbol{x}_{\boldsymbol{k|k-1}}$ is well fitted by Gaussian distribution with parameters $\hat{\boldsymbol{h}}_{\boldsymbol{i}}$ and $\boldsymbol{S}_{\boldsymbol{i}}$, where $\hat{\boldsymbol{h}}_{\boldsymbol{i}}$ and $\boldsymbol{S}_{\boldsymbol{i}}$ are calculated according to $\hat{\boldsymbol{x}}_{\boldsymbol{k|k-1}}$ and $\boldsymbol{P}_{\boldsymbol{k|k-1}}$ respectively. Individually compatible match is the vector $\boldsymbol{z}_{\boldsymbol{i}}$ which include the points inside 99% probability region defined by such model. Furthermore, $\boldsymbol{z}$ is created to save individually compatible matches for each update.

Algorithm 1 1-Point RANSAC EKF

INPUT: $\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}$ {EKF estimate at step $k-1$}
  th {Threshold for low-innovation points. In this paper, $th = 2\sigma\_pixels$}
OUTPUT: $\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}$ {EKF estimate at step k}

{A. EKF prediction and individually compatible matches}
$[\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}] = EKF\_prediction(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}, \mathbf{u})$
$[\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1}] = measurement\_prediction(\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$
$\mathbf{z}^{IC} = search\_IC\_matches(\hat{\mathbf{h}}_{k|k-1}, \mathbf{S}_{k|k-1})$

{B. 1-Point hypotheses generation and evaluation}
$\mathbf{z}^{li\_inliers} = [\,]$
$n_{hyp} = 1000$ {Initial value, will be updated in the loop}
for $i = 0$ to n_{hyp} **do**
  $\mathbf{z}_i = select\_random\_match(\mathbf{z}^{IC})$
  $\hat{\mathbf{x}}_i = EKF\_state\_update(\mathbf{z}_i, \hat{\mathbf{x}}_{k|k-1})$ {Notice: only state update; NO covariance update}
  $\hat{\mathbf{h}}_i = predict\_all\_measurements(\hat{\mathbf{x}}_i)$
  $\mathbf{z}_i^{th} = find\_matches\_below\_a\_threshold(\mathbf{z}^{IC}, \hat{\mathbf{h}}_i, th)$
  if $size(\mathbf{z}_i^{th}) > size(\mathbf{z}^{li\_inliers})$ **then**
    $\mathbf{z}^{li\_inliers} = \mathbf{z}_i^{th}$
    $\epsilon = 1 - \frac{size(\mathbf{z}^{li\_inliers})}{size(\mathbf{z}^{IC})}$
    $n_{hyp} = \frac{log(1-p)}{log(1-(1-\epsilon))}$
  end if
end for

{C. Partial EKF update using low-innovation inliers}
$[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{li\_inliers}, \hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1})$

{D. Partial EKF update using high-innovation inliers}
$\mathbf{z}^{hi\_inliers} = [\,]$
for every match $\mathbf{z}^j$ above a threshold th **do**
  $[\hat{\mathbf{h}}^j, \mathbf{S}^j] = point\_j\_prediction\_and\_covariance(\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, j)$
  $\nu^j = \mathbf{z}^j - \hat{\mathbf{h}}^j$
  if $\nu^{j\top} \mathbf{S}^{j-1} \nu^j < \chi^2_{2,0.01}$ **then**
    $\mathbf{z}^{hi\_inliers} = add\_match\_j\_to\_inliers(\mathbf{z}^{hi\_inliers}, \mathbf{z}^j)$ {If individually compatible, add to inliers}
  end if
end for
if $size(\mathbf{z}^{hi\_inliers}) > 0$ **then**
  $[\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}] = EKF\_update(\mathbf{z}^{hi\_inliers}, \hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k})$
end if

In the second step, the number of hypotheses N is set as a constant initially. Later, it will be updated according to

$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon))}$$

where

$$\varepsilon = 1 - \frac{size(\boldsymbol{z_i})}{size(\boldsymbol{z})}$$

Then, for each hypothesis, a single sampling point is generated by a random value belonging to the vector $\boldsymbol{z_i}$. A point is said to support this hypothesis if it is below a threshold within the sampling point. The selected point is decided by the hypothesis with the maximum amount of supporting points. This

method of reduction on N is the mechanism that allows 1-point RANSAC to operate much faster than standard RANSAC and allows this algorithm to become a candidate for real-time SLAM.

In the third step, those supporting points are classified as low-innovation inliers in the third step. The new stated vector $\hat{\boldsymbol{x}}_{\boldsymbol{k|k}}$ and the covariance vector $\boldsymbol{P}_{\boldsymbol{k|k}}$ are updated according to the previous state and covariance and low-innovation inliers.

In the last step, model is estimated again. Because the points which don't support the selected hypothesis still have the possibility to be inliers. If this is the case, they will be claimed as high-innovation inliers. Mathematically, the unsupported points below a specific threshold are classified as high-innovation inliers. Again, the new state vector $\hat{\boldsymbol{x}}_{\boldsymbol{k|k}}$ and the covariance vector $\boldsymbol{P}_{\boldsymbol{k|k}}$ are calculated, but at this step, high-innovation inliers are considered.

Finally, all extracted lines can be calculated by all state vectors and covariance vectors. Next they will be plotted in the iOS application through use of a CSV data file. In addition, in order to accelerate the progress of programming, these algorithms will be optimized in further steps. Some parallel processing methods, such as message passing and shared memory, may be introduced depending on testing results described in Section 5.

4.3 Extended Kalman Filter

The Extended Kalman Filter (EKF) takes the values returned by the Accelerometer and Gyroscope and interprets these values in order to calculate certain probability values like State and Covariance estimates. These estimates allow us to calculate near accurately the parameters of interest in our Project. The values are created at start time for the Extended Kalman Filter and are constantly updated whenever we process an image. The primary purpose of the EKF is to reduce uncertainty in the values which the application displays to the user. The main components which interact with each other is shown in Figure 7.

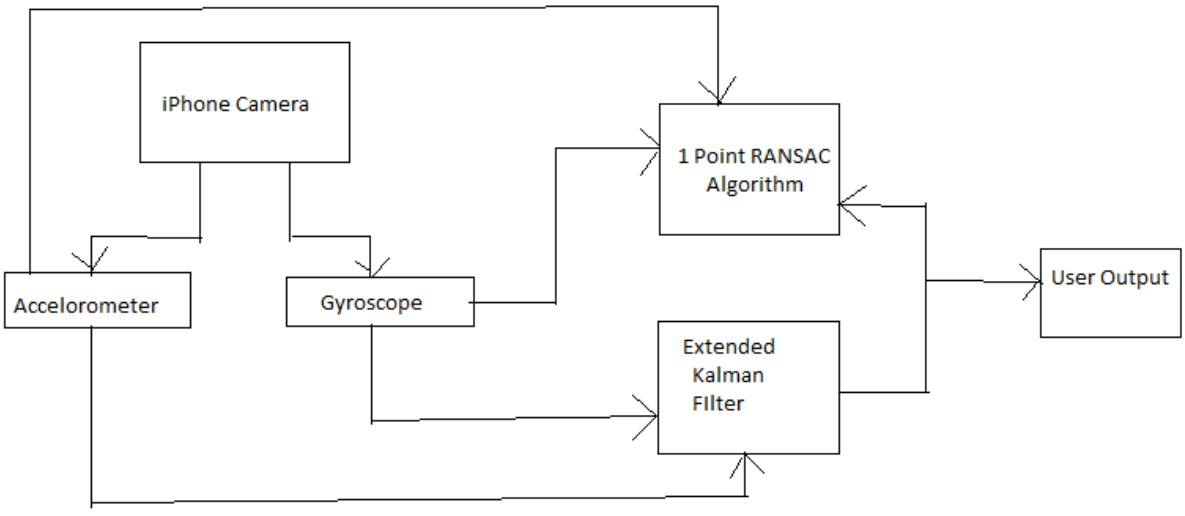

Figure 7: Diagram depicting interaction of EKF-critical components

From the diagram, we show the different components of the iPhone device which interact with each other in order to calculate and display the result. The 1- Point RANSAC Algorithm uses the values provided by the Accelerometer and Gyroscope in order to start the estimation of the number of inliers. The number of inliers is an important metric in order for the Extended Kalman Filter (EKF) to initialize the State and Covariance estimates and update them after receiving the number of inliers. The EKF then calculates the necessary probability and state values and uses the calculated values to better estimate the parameters specific to the object in observation.

Given the state of the parameter (for example, Velocity) at time t-1, the following general expression can provide us the state of the estimate that we are calculating,

$$\tilde{x}t|t\text{-}1 = f(\tilde{x}t\text{-}1|t\text{-}1, u_{k-1}) \rightarrow \text{State Estimate}$$

We also estimate the Covariance in the initial stages as shown below.

$$P_{t|t-1} = F_{t-1}P_{k-1}\,F_{t-1}^{T} + Q_{t-1} \rightarrow \text{Covariance Estimate}$$

In the above expression, the F_{t-1} is the Jacobian Expression for the State Transition Matrix from one time instance to another. The term F_{t-1}^{T} is the transpose of the State Transition Matrix, x is the state vector and u is a control input vector. The above expressions are the values initialized at the start of the application. Once the video processing begins, the recorded video is passed to the Extended Kalman Filter input, which in turn takes the values and updates the State and Covariance Estimates which are calculated as shown above. With the updated values, the state and covariance values are updated according to the equations given below:

The updated State Estimate can be given as,

$$\tilde{x}t|t = \tilde{x}t|t\text{-}1 + K_t\tilde{y}_t$$

Similarly, the Covariance update when the values are obtained from the target is given as shown below,

$$P_{t|t} = (I - K_tH_t)P_{t|t-1}$$

where $K_t = P_{t|t-1}H_t^{T}S_k^{-1}$ and $S_t = H_tP_{t|t-1}H_t^{T} + R_t$ and H_t is the Jacobian which can be calculated.

In this way, we utilize the camera of the iPhone to provide video which is then in conjunction with the maps to specify the values of a target object. The coordinates of the target object is extracted using the Accelerometer and Gyroscope and are fed as input to an Extended Kalman Filter which predicts the state and covariance matrix, reducing the ambiguity in the estimated parameter space of the target object.

In our project, we utilize the device sensors that are available in-built in the device like Accelerometer and Gyroscope. These sensors capture the parameters of the object that are of interest and store a vector as shown below.

For the Accelerometer, a vector of the x, y and z coordinates are stored of the target object as shown below. x_acc and y_acc are the exact readings calculated by the Accelerometer. However, they

may be presence of spurious noise sources which can affect the accelerometer accuracy as shown in the mathematical expressions:

$$x\_rec = x\_acc \pm n_a$$

$$y\_rec = y\_acc \pm n_b$$

where n_a and n_b are the noise distortion which can affect the estimate of the accelerometer. Similarly, we observe the effect of noise over the accuracy of the Gyroscope as shown by the following equations. If x_gyro and y_gyro are the values of the rate of rotation in rad/s, then we observe the following values as shown below.

$$x\_rec\_gyro = x\_gyro + n_c$$

$$y\_rec\_gyro = y\_gyro + n_d$$

The 1-Point Random Sample Consensus (RANSAC) method provides an estimate of the inliers which are the x and y values from the vector of data that are collected and provides an estimate $z^{hi\_inliers}$ which is then used to reject spurious noise that may have affected the quality of correct reception. The state estimate is calculated from the EKF as shown below.

$$\tilde{x}t|t\text{-}1 = f(\tilde{x}t\text{-}1|t\text{-}1, u_{k\text{-}1})$$

The covariance estimate is initialized as shown below. These two values allow the 1 Point RANSAC to estimate the $z^{hi\_inliers}$.

$$P_{t|t\text{-}1} = F_{t\text{-}1}P_{k\text{-}1}\,F_{t\text{-}1}{}^{T} + Q_{t\text{-}1}$$

The 1 Point RANSAC algorithm iteratively checks for each value in the vector of data given by accelerometer and gyroscope and rejects all values that lie outside the covariance and state estimates and stores only the values that lie within the $z^{hi\_inliers}$. This way, the EKF and the RANSAC algorithm refines the values that are taken from the sensors and helps in reducing the effects of noise. More importantly, the efficiency of the system is better when the accelerometer and gyroscope sensors are used.

4.4 Software Structure

As Figure 8 shows, five classes are proposed for the 1-Point RANSAC Algorithm, including EKF_Filter, Search_Matches, RANSAC_Hypotheses, Low_Innovation_Inliers and High_Innovation_Inliers. The EFF_Filter is a sub-class of Search_Matches and generates initial values of state vector and covariance vector for Search Matches. With these parameters, Search_Matches will predict a new state vector and search all individually compatible matching points. Next, in the RANSAC_Hypothses class, Hypotheses_Filter selects one hypothses with the most voting points based on matching points and the calculated number of hypotheses. Finally, Low_Innovation_Inliers and High_Innovation Inliers are two classes that share similar variables and methods. Both of them need previous state vector and covariance vector to calculate. But Low_Innovation_Inliers utilizes a LI_Filter those voting points to update EKF data, while High_Innovation_Inliers re-estimates taking possible inliers into account. Therefore, the latter class inherits from the former one but replace LI_Filter with HI_Filter.

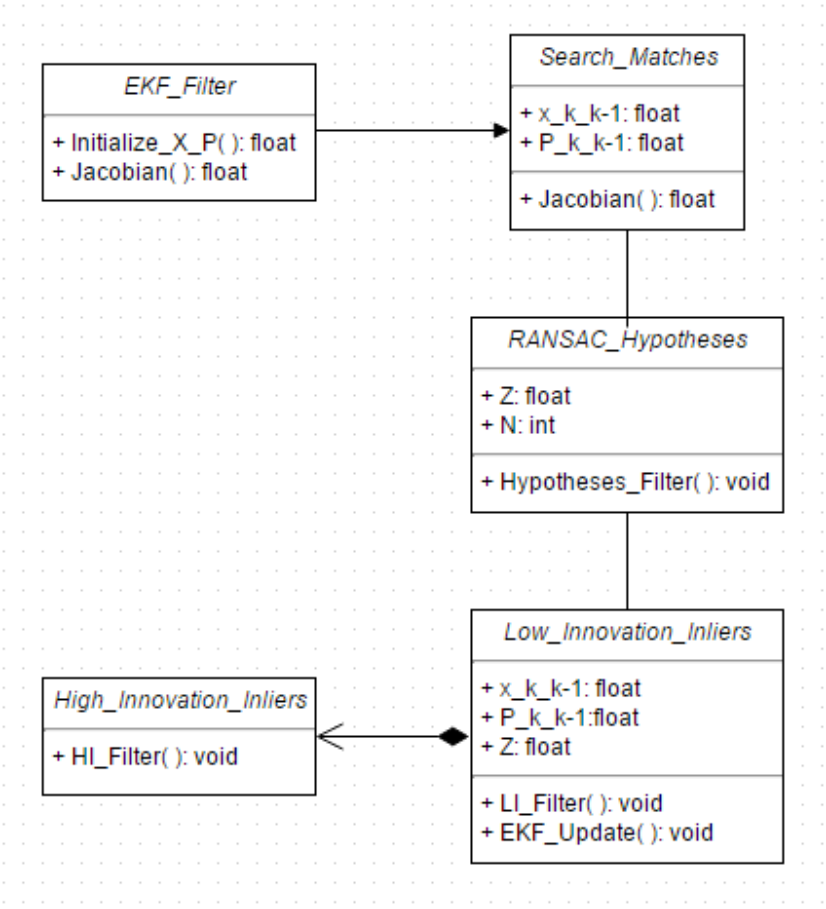

Figure 8: Basic proposed classes for 1-Point RANSAC algorithm

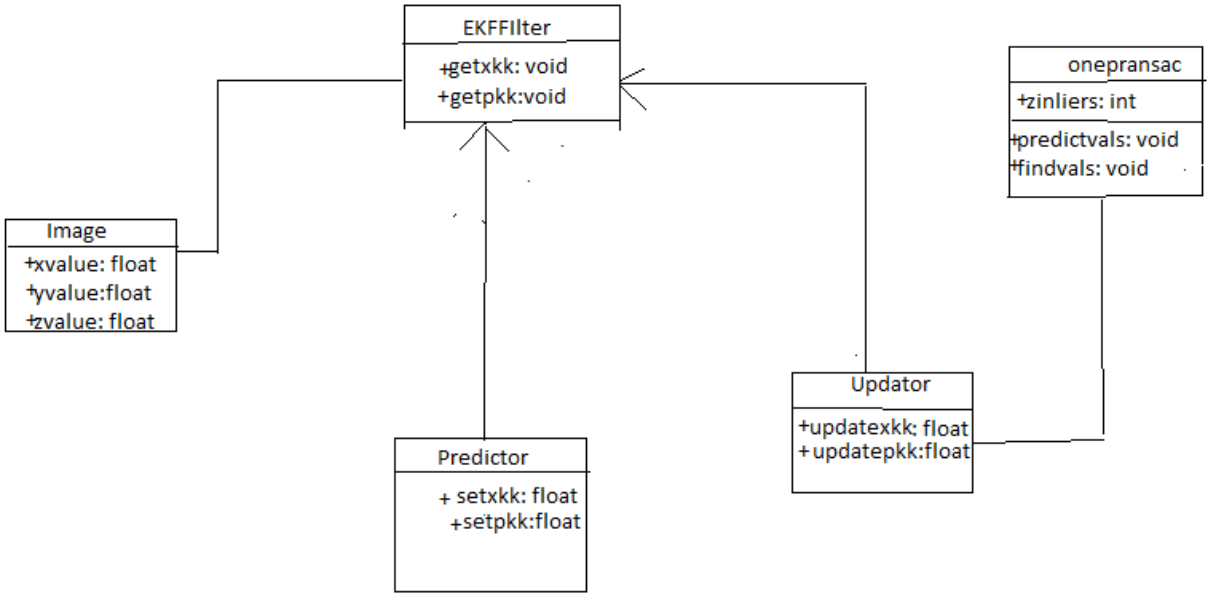

Figure 9: Class Diagram depicting the interaction between EKF-related classes

Figure 9 illustrates the relationship between the different EKF classes and how they interact with each other. The Image class is the image of the target object that the device captures at a point in time. The Image class consists of parameters like the x, y and z coordinates of the image. These parameters are passed to the EKFFilter class which derives these values and has the methods, 'getxkk' and 'getykk', which takes these values and stores it. The Predictor class derives these values and is the class where the initial prediction of the State and Covariance Estimates are calculated. These are the initial predictions for these two parameters and these parameters may change after the analysis from the 1-Point Random Sampling Consensus (RANSAC) method. The Updator class may update the state and covariance estimates based on the number of inliers that are found using the RANSAC method. The number of inliers refers to data points that lie within a confidence interval under the 1 point RANSAC algorithm. The updated values for the State and Covariance estimates are then stored. The above class diagram best explains the functions of these classes.

The opepransac class returns the z*inliers which returns the data value for the number of inliers according to the 1-Point RANSAC algorithm. This value is used by the Updator class to update the values based on the number of inliers calculated by the algorithm.

5. Testing

5.1 Summary

The general process of development if planned to occur in three stages. Because of the mathematical nature of the problem, lower level testing will be accomplished as outlined in Sections 5.3-5.5 between the project milestones described below:

1. EKFMonoSLAM will be implemented as a standalone desktop program in a language that may be compiled on the iOS platform. The goal is to write most of this functionality in C, although C++ and objective-C may be used for their respective strengths. While this program is being developed, the necessary iOS functionality outlined in Section 3 will be prepared as well.

2. The standalone program will be integrated into the iPhone platform. In this step, pre-recorded video and sensor data from the iPhone will be used as inputs to the EKFMonoSLAM prototype and the necessary output information will be sent internally for plotting.

3. Once a usable approximation of the EKFMonoSLAM program is fully functioning on the mobile device, one or both of two avenues will be explored based upon alpha test results:

3a. Frames will be dynamically processed during capture to facilitate real-time plotting.

3b. Accelerometer and Gyroscope data will be introduced to the EKF

Corresponding verification is as follows; the project may not progress to the next milestone the following conditions are met for each milestone:

1. The original EKFMonoSLAM MATLAB program and the project prototype will both be configured to produce the same CSV data file for plotting. Discrepancies between the two programs may then be found by comparing these two files to one another. The milestone is met when these discrepancies are within a suitable threshold (to be determined).

2. The csv-based metric technique described above is applied to the program versions running on (a) a desktop application and (b) the iOS application. In this step, processed videos must be produced by the iPhone. This milestone is met if the deviation is within a certain limit.

3a. Proposed framerate is compared to a measured time for computation of a single iteration of the SLAM algorithm. Playback should appear smooth and the computational time is found to be lower than the theoretical time between frames.
3b. Actual measurements will be taken between physical objects and compared before and after the new EKF strategies. The experiment is successful if computational time is lowered, predictions become more accurate or the device may be moved in irregular motion while preserving accuracy.

The general mapping of tests to the requirements shown in Table 1 is shown below:

Test-Requirements mapping

1. The standalone program will be integrated into the iPhone platform. Pre-recorded sensor data from the iPhone will be used as inputs to the EKFMonoSLAM prototype and the necessary output information will be sent internally for plotting.
2. The location recorded location matrix reflects the actual locations of iPhone accurately. The matrix should have the same track as actual movement.
3. The length and degree of each extracted line should be within a specific tolerance compared with the actual scene, which can be measured by comparing extracted landmarks with real pictures.
4. The original EKFMonoSLAM MATLAB program and the project prototype will both be configured to produce the same CSV data file for plotting. Discrepancies between the two programs may then be found by comparing these two files to one another. The milestone is met when these discrepancies are within a suitable threshold.
5. The CSV-based metric technique described above is applied to the program versions running on (a) a desktop application and (b) the iOS application. In this step, processed videos must be produced by the iPhone. This milestone is met if the deviation is within a certain limit.
6. Use similar testing method as step 3. But in this step, compare the point cloud with real pictures recorder by iPhone camera. The error should be below a certain small value.
7. Actual measurements will be taken between physical objects and compared before and after the new EKF strategies. The experiment is successful if computational time is lowered, predictions become more accurate or the device may be moved in irregular motion while preserving accuracy.

5.2 Timeline

Responsibilities and deadlines are outlined in this section, color coded by developer:
Anthony Rodriguez
Shreenivas Devarajan
Shuai Yuan

Expected milestones are as follows:

3/13: Alpha release:
Anthony: All views for iOS application are implemented and transitions work as expected. Basic plotting is shown for a fabricated "dummy" CSV file.
Shuai: All classes and functions have been defined for 1-pt RANSAC
Shreenivaas: Image data may be accessed by EKFMonoSLAM program, even if processing is not accomplished. All classes and functions have been defined for EKF

3/27: Standalone EKFMonoSLAM program works with image data provided by authors of EKFMonoSLAM. Both 1-pt RANSAC and the EKF deviations from MATLAB implementation are documented.
Shuai: RANSAC deviations from MATLAB implementation are noted
Shreenivaas: EKF deviations from MATLAB implementation are noted
Anthony: CSV file produced by algorithm can be read and displayed by iOS client.

4/3: Testing items #1 and #2 described in section 5.1 have been accomplished and quantified results have been documented. The data is used to determine which portion(s) of item #3 to implement.

4/17: Beta release: Items #1 and #2 described in Section 5.1 are fully operational. Any implemented portion of item #3 has been tested and verification results have been documented.

5/6: Project Presentations. Final release version packaged.

5.3 1-Point RANSAC Testing

1-Point RANSAC algorithm may cause time-delay on mapping and localization, which will reduce the validation of this application. **Unit testing** is necessary to count how long the process would last. Code need to be optimized until the product meets the standard of a real-time app.

Extended Kalman Filter plays an important role in the 1-Point RANSAC algorithm. Therefore, **integration testing** is introduced to test how well they interacts with each other.

A real-time app is also necessary on the requirement level. In the end, **verification testing** is designed to inspect its processing period.

5.4 Extended Kalman Filter

a) The usage of the Extended Kalman Filter (EKF) does bring slight errors. The error bounds shall be measured in due course of time as we design the application. To test that the application estimate error is within a certain boundary, we shall test the output data against the data calculated probabilistically to determine the deviation from the Mathematical estimate. This test may require performing unit testing of the EKF component of the code.

b) The usage of the Accelerometer and Gyroscope shall improve the efficiency of the implementation as near accurate values are returned from these sensors. We plan to test the accuracy of the state estimate based on the MATLAB implementation which exists. This way we can test the improvement in state estimation by the usage of these sensors along with the Extended Kalman Filter. We need to set up the Accelerometer and Gyroscope to extract the coordinates before the accuracy can be verified.

5.5 iOS Application Framework

The critical verification items in this system are (1) the data delivery method and (2) the plotting entity.

Verification on item (1) can be performed by demanding incoming data be written to a data file that may be parsed in a tool such as MATLAB. This system is considered to be operating properly if the frames in the video capture match the length of the data vectors for the accelerometer/gyro readings.

Item (2) is accomplished by plotting from a "dummy" CSV file as described in section 5.2.

6. Development Team

In our project, Shreenivaas shall be performing the code design and development for the Extended Kalman Filter Algorithm. Shreenivaas will handle the code development and testing for the EKF component. He has taken special interest in the concept of using Extended Kalman Filter and its implementation in a device to improve accuracy. Besides, he has experience in working with C# platform as part of his previous work experience.

Shuai will mainly focus on landmark extraction and update. Shuai is the most suitable member of our group to perform this part. Because in another class, digital signal processing, he has already learned several algorithms about landmark extraction and update, including standard RANSAC, which will inspire him to our specific algorithm – 1-Point RANSAC. The assignment of Shuai will include 1) learning and designing 1-Point RANSAC algorithm, 2) developing and optimizing relevant programming, and 3) testing the 1-Point RANSAC algorithm.

Anthony will focus on integration of the SLAM algorithm into the iOS device and ensuring that the systems are verified to work together as intended. Because the tasks allotted to Shreenivaas and Shuai require extensive research and development time, Anthony's time will be devoted to ensuring the system properly furnishes all input data and handles output data correctly while performing checks to ensure that the system behavior of the SLAM algorithm is correct. Anthony has significant experience in verification of real-time systems as well as visual APIs such as OpenGL and OpenCV.

7. Licensing

EKFMonoSLAM is licensed under GNU GPL, which allows the public to conduct these experiments with their research. In the spirit of open research, this application is developed under the same license.

References

1. Javier Civera, Oscar G. Grasa, Andrew J. Davison and J. M. M. Montiel: 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry , Journal of Field Robotics, 2010

2. Javier Civera, Andrew J. Davison, J. M. M. Montiel: Inverse Depth Parametrization for Monocular SLAM , IEEE Transactions on Robotics, 2008