VISION-BASED UAV POSE ESTIMATION

A Thesis Presented
by
Zahra Mohajerani

to
The Department of Electrical and Computer Engineering

In partial fulfillment of the requirements
for the degree of

Master of Science
in
Electrical Engineering

Northeastern University
Boston, Massachusetts
08/2008
I would like to express my gratitude to all who supported me and helped me throughout this study. I thank Ehsan Nourbakhsh, for his invaluable assistance, and for all his helps, with the Linux and programming in C. I am also grateful to Fei Xiong, who helped me run the experiments over and over again. I thank Ali Nouri, for his guidance, his encouragement, and all his helps which showed me how to think like a computer scientist, not an engineer. And finally, I thank my parents for always having faith and belief in me.
# Contents

<table>
<thead>
<tr>
<th>Acknowledgements</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Abstract</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Introduction</strong></td>
<td>2</td>
</tr>
<tr>
<td>2.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Problem Description and Approach</td>
<td>2</td>
</tr>
<tr>
<td>2.2.1 Unmanned Aircraft Vehicles (UAV)</td>
<td>3</td>
</tr>
<tr>
<td>2.2.2 Pose</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Pose Estimation - Related Works</td>
<td>4</td>
</tr>
<tr>
<td>2.3.1 Single Camera and Moire Pattern</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2 Two Ground Cameras</td>
<td>7</td>
</tr>
<tr>
<td>2.3.3 Two Cameras: One Ground And One Onboard</td>
<td>8</td>
</tr>
<tr>
<td>2.3.4 Other Approaches</td>
<td>8</td>
</tr>
<tr>
<td><strong>3 Essential Preliminaries</strong></td>
<td>12</td>
</tr>
<tr>
<td>3.1 Markov Sequence</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Conditional Mean and Variance in Gaussian Distribution</td>
<td>13</td>
</tr>
<tr>
<td>3.3 EKF</td>
<td>13</td>
</tr>
<tr>
<td>3.4 Monte Carlo Simulation</td>
<td>15</td>
</tr>
<tr>
<td>3.5 Bayes Theorem</td>
<td>16</td>
</tr>
<tr>
<td>3.5.1 DBN</td>
<td>16</td>
</tr>
<tr>
<td>3.6 Shi and Tomasi</td>
<td>18</td>
</tr>
<tr>
<td>3.7 Spatial Relationships</td>
<td>19</td>
</tr>
<tr>
<td>3.7.1 AT</td>
<td>19</td>
</tr>
<tr>
<td>3.7.2 Coordinate Frame Relationships</td>
<td>20</td>
</tr>
<tr>
<td>3.7.3 Stochastic Map</td>
<td>23</td>
</tr>
<tr>
<td>3.7.4 Updating The Map</td>
<td>26</td>
</tr>
</tbody>
</table>
List of Tables

4.1 SLAM Notations .................................................. 31
4.2 EKF-SLAM Notations ............................................ 34
## List of Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>6 Degree Of Freedom (6DOF) of an aircraft</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>A moiré pattern generated by superposition of two layers consisting of</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>parallel lines; the lines of the revealing layer are parallel to the lines of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the base layer—courtesy of [15]</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Moiré Target used by Tournier for UAV pose estimation</td>
<td>6</td>
</tr>
<tr>
<td>2.4</td>
<td>Calculation of Altitude and Yaw angle</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Using the color markers under the quadrotor and 2 cameras onboard</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>and on the ground, pose of the UAV can be computed</td>
<td></td>
</tr>
<tr>
<td>2.6</td>
<td>Using Accelerometers in wii and iphone for motion analysis</td>
<td>10</td>
</tr>
<tr>
<td>2.7</td>
<td>GUI view of MonoSLAM software</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>DBN for the SLAM problem. $U_i$: control input applied at time $(i − 1)$ to</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>derive the system to state $X(i)$, $Z(i, j)$: observation of $L_j$ made at time $i$.</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>A sequence of approximate transformations. line: AT, ellipse: AT’s uncertainty, solid: gained by movement, dashed: gained by observation.</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Compounding, Merging and Reversal operations. line: AT, ellipse:</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>AT’s uncertainty, solid: gained by movement, dashed: gained by observation,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$X_{i,j}$: AT from frame $i$ to frame $j$</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>map of 3D world, each feature represented as a 6DOF vector (3 DOF position and 3 DOF orientation)</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>updating the map</td>
<td>26</td>
</tr>
</tbody>
</table>
4.1 1: start the prediction step, 2: estimate new location, 3: add process noise, 4: start the measurement step, measure features, 5: update positions and uncertainties.

4.2 The convergence in landmark uncertainty. The plot shows a time history of standard deviations of a set of landmark locations. A landmark is initially observed with uncertainty inherited from the robot location and observation. Over time, the standard deviations reduce monotonically to a lower bound. New landmarks are acquired during motion.

4.3 Before and after closing the loop, this figure is from [19].

5.1 The evolution of depth probability density from uniform distribution to Gaussian, points are uniformly distributed over the image patch’s normal line from 0.5 to 5m. Courtesy of [12].

5.2 The camera poses in 2 different viewpoints looking at the same image patch with normal n. Courtesy of [12].

5.3 Camera Motion Model.

5.4 Vicon camera.

5.5 Vicon markers, from [3].

5.6 The output of MonoSLAM for a sample sequence, with 4 startup features. The map is built gradually by updating the visible features in each frame. Blue marks correspond to failed matches, red marks correspond to successful matches and yellow marks correspond to unused features in current frame.

5.7 MonoSLAM Error in X, Y and Z Dimension.

5.8 UAV orientation, MonoSLAM and Vicon outputs.

5.9 Comparing the output trajectories in 2D space, from Vicon and MonoSLAM.
7.1  a: existing map of the room; 10 points stars are the features, darker color corresponds to high quality features and lighter colors corresponds to low quality ones. 4 points pink stars are the selected landmarks. b: updated map, selecting 2 other landmarks by only taking into account the quality of features. b: updated map, selecting features by quality and distance from the existing features in the map.
Chapter 1

Abstract

The popularity of Unmanned Aerial Vehicles (UAVs) has increased dramatically in the past few decades. The main challenging task for UAVs operation is to find their position and orientation accurately and continuously during the flight, using the onboard sensors (camera in our case). There are lots of different approaches for UAV pose estimation in literature now. Several different approach and their problems in real-life application were studied in this research. We chose the real time single camera SLAM method [12], because our robot was equipped with an onboard camera, and MonoSLAM had proved its reliability in previous localization experiments[12,10,9,11]. In our work, we obtained the ground truth by the Vicon cameras. The experimental results suggest that this method can be reliably used in the control of indoor UAV navigation.
Chapter 2

Introduction

In our work, we consider the problem of pose estimation in a helicopter, when the sensory information comes only from a single mounted camera. We show that a previously studied algorithm called MonoSLAM [12] can be successfully used to solve this task by checking the ground truth with Vicon cameras.

2.1 Motivation

The popularity of Unmanned Aerial Vehicles (UAVs) has increased dramatically in the past few decades. The main challenging task for UAVs operation is to find their position and orientation accurately and continuously during the flight, using the onboard sensors.

2.2 Problem Description and Approach

In order to control the UAV during any mission, we need to know its position and orientation to take the proper decision in each moment. In the cases where a complete map of the environment is known ahead of time, this extra information can be used to get leverage on solving the pose estimation problem. However, in most of the real-life applications, the map is not available \textit{a priori}. This imposes additional complexities into the localization problem. Here we can redefine the localization problem as the
2.2. PROBLEM DESCRIPTION AND APPROACH

ego-motion estimation of a moving robot inside the world and build the map of the world at the same time and update it dynamically. This problem is known as The Simultaneous Localization and Mapping (SLAM) problem in the robotics community. As robot’s updating sensor we chose the camera mounted on a helicopter and used the MonoSLAM program of Andrew Davison to implement the real time pose of the UAV. Finally we checked the ground truth using Vicon cameras. The experiments showed that the calculated pose by this algorithm is reliable enough to be used in the UAV controlling step.

2.2.1 Unmanned Aircraft Vehicles (UAV)

Among all aircrafts, helicopters are are particularly interesting in that they possess unique characteristics. Helicopters, unlike fixed wings aircrafts, can land and take off vertically without the needs for a runway, because their rotors’ blades revolve in the air, which lets the helicopters to move vertically without the need to move forward. Another interesting characteristic of helicopters is that they can hover for an extended period of time, and that they can handle low airspeed conditions. Because of these unique properties, helicopters can be used to accomplish tasks that no other aircrafts can.

Today, helicopters are used for transportation, construction, fire fighting, search and rescue, and lots of different tasks that require the special capabilities mentioned above\textsuperscript{[4]}. UAVs are aircrafts that work without pilots. They can be used in situations, where a human’s life would be at risk, like fire fighting, search and rescue in unknown fields, and all missions which are too dangerous for manned aircrafts[wiki]. The focus of this research is to provide a way of controlling the unmanned quadrotor, with the hope of taking advantages of the good characteristics of both UAVs and helicopters.
2.2.2 Pose

Degrees of freedom (DOF) are the set of independent displacement of the object in the space that completely define the position of the object or system during its movement. 6DOF or pose refers to motions in 3-dimensional space, three of these degrees are related to angular position (rotation around perpendicular axes/attitude: $(\phi, \theta, \psi)$) which are pitch, roll and yaw; and the other three are the translation in three perpendicular axes (position:$(x, y, z)$): moving backward/forward (Longitudinal), up/down (Vertical) and left/right(Lateral). Figure 2.1 illustrates these 6DOF for an airplane.

2.3 Pose Estimation - Related Works

The first step to successfully use UAV is to find its position and attitude (pose) in the 3D space accurately and fast. This information is very critical for controlling

\[\text{from NASA: } \text{http://www.aviationsystemsdivision.arc.nasa.gov/}\]
the UAV. Because the main use of UAVs is in unknown environments for search and rescue, or for surveillance in cluttered environments, in which the global positioning system (GPS) usually doesn’t work, lots of research has been recently done to find the pose using other devices like camera or laser range finder and other sensors. A brief summary of these approaches and their positive and negative properties is presented in this section.

2.3.1 Single Camera and Moire Pattern

![Moiré Pattern](image)

Figure 2.2: A moire pattern generated by superposition of two layers consisting of parallel lines; the lines of the revealing layer are parallel to the lines of the base layer—courtesy of [15].

Glenn Tournier [36], used a single camera onboard as a quadrotor sensor for pose estimating. He used the property of Moire effect to estimate the quadrotor pose by making a target with two different types of Morie patterns in two perpendicular sides and 4 colored LEDs on each corner as indicated in Figure 2.3. The Morie pattern in
the target is generated by placing two similar-structured grid on top of each other by a small distance. Tournier computed the "altitude" of the camera by applying the principle of similar triangles on the LED around the target, the "yaw" by computing how parallel the lines $\overline{EB}$ and $\overline{DC}$ are to the vertical, and the rest four parameters by using the moire effect appearing on the target due to the camera movement.

**Disadvantages**

Although this approach showed good results in [16] and [36], there are some disadvantages to it if used in outdoor environments or in other real-life approaches:

1. Moire effects are sensitive to change in illumination, and perform poorly in outdoor environments. Because the primary UAV usage in our research is surveillance, search and rescue, firefighting and etc, and these all occur in outside environments, Moire pattern is not a good choice.

2. The pose estimation here is highly dependent to the target, and if the target is not visible to the camera for a short period of time, the UAV will lose its control due to the lack of sufficient knowledge about its pose. Also if we want to fly...
in a higher altitude than the room ceiling in outdoor environments, the target and its grating will become too small and its tracking will become impossible (Unless we significantly increase the target size). So, finding the pose while tracking a fixed target is not a good idea for outdoor environment use either.

### 2.3.2 Two Ground Cameras

Andrew NG used 2 ground stereo pair cameras and a machine-learning approach for helicopter pose estimation and flight control. His projects were performed by a real helicopter flying in outdoor environments and showed very good results (the experiment movies are available in [his website]{2}). Because of using two ground cameras, this approach can not be used in unknown strategic environments for surveillance tasks.

---

{2}http://heli.stanford.edu/
2.3.3 Two Cameras: One Ground And One Onboard

Erdinc Altug and Camillo Taylor [5] also used 2 cameras. They placed a pan/tilt/zoom camera on the ground and another one on the quadrotor, looking downward to the ground camera. Their research goal was to find the quadrotor position and attitude with respect to the ground camera rather than finding the quadrotor pose by itself. In order to find the camera pose, some colored markers with different colors and known radius (2.5cm in their experiment) were attached under each rotor, on the center of the camera and also on the top of the ground camera as illustrated in Figure 2.6. By tracking the markers and computing their areas, the relative pose to the camera could be computed.

Disadvantages

1. In order to use in real-life situations, the markers must be very big to be recognizable from a far distance, and this makes the approach unusable in critical situations like surveillance.

2. There is no other way to find the quadrotor pose when the ground camera is out of sight of the quadrotor camera. This makes the approach very sensitive to any possible occlusion in outdoor environments.

3. Having the ground camera is not possible for some missions like fire fighting or search and rescue.

2.3.4 Other Approaches

Beside cameras, it is possible to use other sensors to equip the system with extra useful information in order to increase the localization precision. Using Gyro to find the attitude is a great choice when the weight limit is not an issue. Three accelerometers in three perpendicular axes can also be used to estimate position [Wii]. Finally, laser
range finders have also been used to improve the performance of pose estimation.

Since the quadrotor we planned to use in this research was equipped with an onboard camera, and since the weight limit did not let us to have extra sensors on board, we chose the single camera approach, and used MonoSLAM \[12\]. A brief sketch of MonoSLAM is presented here and in Chapter 4 the algorithm is explained in details.

**MonoSLAM**

MonoSLAM is a camera based approach of SLAM, first introduced by Andrew Davison \[10\]. Before this method was introduced, other sensory information such as laser range finders were used in SLAM.

The camera mounted on the Robot captured a series of pictures from the environment as it moves around, and detects the high-quality features inside them. These image patches are then used in the regular SLAM method as discussed in \[14\] and \[8\]. Unlike the others, MonoSLAM relies more on localization rather than finding the precise map of the room. This method stores just enough number of features to find
the robot’s position, thus creating a very sparse map (only around 100 total features), so the output is a very sparse map and the robot’s pose during its motion.

To decrease the uncertainty in the pose, Davison chose to initialize the map in the first frame with four known features (by giving their appearances and true locations), and also he gave the starting pose of the camera. Hence, both the map and pose data series start with known data, which help the system have a sense of depth at the start point.

**Disadvantages**

1. If we want to use MonoSLAM in real time (30Hz for example), we cannot have more than 100 features in the map, which is enough for pose estimation inside a small room but not in bigger indoor or outdoor environments.

2. The other setback of this approach is its fragility against moving objects. The key point of the algorithm is repeatable observation, and if features move inside the scene, reobservation of them can result in closing incorrect loops and unreasonable localization. Moreover, as a result of this problem, it’s almost
impossible to use the robot for tracking (a moving) object inside the scene.

We study possible ways to overcome these disadvantages in Chapter 7, and believe this approach is currently the best one in the literature for our purpose.
Chapter 3

Essential Preliminaries

In this chapter, we review the fundamental concepts and techniques, which are necessary for understanding the MonoSLAM algorithm. We demonstrate some probability estimation techniques, Markov sequence, Monte Carlo simulation, Bayes theorem, Dynamic Bayesian Network (DBN) and also the fundamental foundation of spatial relationships.

3.1 Markov Sequence

Papoulis[27] defined Markov process as a random process, in which its upcoming probabilities are only dependent on their most recent values. In other words, a stochastic process \( x(t) \) is Markovian if it has the following properties for all \( n \) and \( t_1 < t_2 \ldots < t_n \):

\[
P(x(t_n) \leq x_n|x(t_{n-1}), \ldots, x(t_1)) = P(x(t_n) \leq x_n|x(t_{n-1}))
\]

This is equivalent to:

\[
P(x(t_n) \leq x_n|x(t), \forall t \leq t_{n-1}) = P(x(t_n) \leq x_n|x(t_{n-1}))
\]

(\[27\], p.535).

These equations signify the fact that the current state value has all the information
3.2 CONDITIONAL MEAN AND VARIANCE IN GAUSSIAN DISTRIBUTION

we need and the future value of such a variable is independent of its past. So, the process with this property is very computationally efficient.
Markov sequences also have the following property:

\[ p(x(t + 1)) = \int p(x(t + 1)|x(t))p(x(t))dx(t), \]

which is used in SLAM prediction step as explained in Chapter 4.

3.2 Conditional Mean and Variance in Gaussian Distribution

For jointly Gaussian random vectors \( y \) and \( z \), the conditional distribution of \( y \) given \( z \) is also Gaussian with the following moments [33].

Conditional mean:

\[ E[y|z] = E[y] + C_{yz}C_z^{-1}(z - E[z]) \]

Conditional variance:

\[ C_{yy|z} = C_{yy} - C_{yz}C_z^{-1}C_{zy}, \]

Where:

\[ C_{yz} = E[(y - E[y])(z - E[z])^T] \]

3.3 EKF

Unlike the regular Kalman Filter, which tries to estimate the state of a controlled linear process, EKF (Extended Kalman Filter) addresses the problem of estimating the state of nonlinear processes, where the state transition and/or observation models are nonlinear functions. EKF linearizes these two nonlinear functions about their mean vector and covariance matrix, similar to Taylor series [17].
Assume the process to be estimated is:

\[ x_k = f(x_{k-1}, u_k, w_{k-1}), \]

and the measurement is:

\[ z_k = h(x_k, v_k), \]

where \( w_k \) and \( v_k \) represent the process and measurement gaussian zero-mean noise with the given covariance, \( x_k \) and \( z_k \) represent the state and observation at time \( k \) respectively, \( f \) is the process model and \( h \) is the measurement model.

EKF algorithm consists of two parts of time update and measurement update. The complete set of equations is given below.

**EKF time update(prediction) equations:**

1. Project the state ahead:
   \[ \hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \]

2. Project the error covariance ahead:
   \[ P_k^- = A_k P_{k-1} A_k^T + W_k Q_k W_k^T \]

   Where \( A_k \) and \( W_k \) are the process jacobians at step \( k \), and \( Q_k \) is the process noise covariance at step \( k \).

**EKF measurement update(correction) equations:**

1. Compute the Kalman gain:
   \[ K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \]

2. Update estimate with measurement \( z_k \):
   \[ \hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k, 0)) \]
(3) Update the error covariance $P_k$:

$$P_k = (I - K_k H_k) P_k^-$$

Where $H_k$ and $V_k$ are the measurement Jacobians at step $k$, and $R_k$ is the measurement noise covariance at step $k$.

### 3.4 Monte Carlo Simulation

The Monte Carlo method was invented by S. Ulam and N. Metropolis in 1949 [38], for solving problems using random numbers and probability distributions. This technique is usually used when we want to simulate physical and mathematical systems when their system models are nonlinear, have lots of uncertain parameters, or are too complicated to solve analytically [2].

The Monte Carlo simulation iteratively samples from the probability distribution we defined for the uncertain input variables and uses these samples to feed the system and evaluate the result.

In order to do this, we must try to find the probability distribution that matches our input data as close as possible, and also expresses our knowledge about these data precisely.

Monte Carlo simulation algorithm has the following steps [3]:

1. **Step 1**: model the system.
2. **Step 2**: find the input probability distribution, generate data from it.
3. **Step 3**: run the system with this data set and check the result.
4. **Step 4**: iterate step 1 to 3 for $n$ times.
5. **Step 5**: dissect the result.
Monte Carlo sampling gives improved estimates of the output, as the input sampling continues.

### 3.5 Bayes Theorem

In the probability theory, Bayes’ theorem simplifies the mathematical formula used for calculating conditional probabilities, which makes the simple connection between the conditional and marginal probabilities of two random variables or events [30].

\[
P(A|B) = \frac{P(B|A)P(A)}{P(B)},
\]

where:

- \( P(A|B) \): **posterior**, the probability of the state of nature being \( A \) given that feature value \( B \) has been measured
- \( P(B|A) \): **likelihood**, the likelihood of \( B \) with respect to \( A \)
- \( P(A) \): **prior**, probability of the state of nature being \( A \) before any observation

The Bayes formula is often used to compute posterior probabilities given observations (prior) as below:

\[
\text{posterior} = (\text{likelihood} \times \text{prior}) / \text{normalizing constant}.
\]

#### 3.5.1 DBN

Dynamic Bayesian Network [24] is a specific type of BN which was built to solve the stochastic process cases and represents sequences of variables (usually time series). DBNs are directed graph models of these kind of processes. HMM (Hidden Markov Model) is a simple example of DBN [20].

BN (Bayesian or belief network), is a simple tool for representing and reasoning about uncertainty and belongs to the graphical model family (GM). It visualizes how different variables are dependent to each other, represents what we know about an uncertain domain and helps to simplify the complex problems. Bayesian network B is an
3.5. BAYES THEOREM

Figure 3.1: DBN for the SLAM problem. $U_i$: control input applied at time $(i - 1)$ to derive the system to state $X(i)$, $Z(i, j)$: observation of $L_j$ made at time $i$.

acyclic graph of the form $B = (G, \Theta)$ which corresponds to the joint probability distribution over a set of random variables, where $G$ is a DAG (Directed Acyclic Graph) with nodes: $X_1, ..., X_n$ and $\Theta$ represents the network parameters $\theta_{X_i|\pi_i} = P_B(X_i|\pi_i)$ where $\pi_i$ represents the parent nodes. $B$ defines the unique joint probability distribution given below:

$$P_B(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P_B(X_i|\pi_i) = \prod_{i=1}^{n} \theta_{X_i|\pi_i}$$
The Bayesian Networks can be made following these 3 steps:

Step 1: adding variables we want to be included in the network as nodes $X_i$

Step 2: adding the links from the parent ($\pi_i$) to child. Each link represents a relationship between two nodes, note that the graph must remain acyclic

Step 3: for each node $X$ defining a probability table containing $P(X|\pi)$ for all combination of its parents

Figure 3.1 illustrates the DBN of SLAM problem.

3.6 Shi and Tomasi

Shi and Tomasi introduced another feature tracking algorithm that obtains the features of an image at time $t + \tau$ based on features of image at time $t$ [31]. This method works by moving every point in the earlier image by an appropriate displacement vector $\delta = Dx + d$, where $d$ is the features’ translation vector and $D$ is the deformation matrix defined by:

$$D = \begin{pmatrix}
  d_x x & d_x y \\
  d_y x & d_y y
\end{pmatrix}$$

Denote the current image by $I(x, t)$ and the upcoming image by $I(Ax + d, t + \tau t)$, where $A = I_{2x2} + \delta$. Shi and Tomasi’s idea was to find an affine transformation that minimizes the dissimilarity, $(\epsilon)$, between these two images:

$$\epsilon = \int \int_W [I(Ax + d) - I(x)]^2 dx,$$

and $W$ is a window around the feature in the current image, where this residual is being computed. In the simplest case, $D$ is equal to the identity matrix and $d$ the
only important quantity is the translation vector $d$. By setting the derivative of $\epsilon$ with respect to $d$ equal to zero and linearizing the system by truncated Taylor expansion, the following linear system will be obtained:

$$Gd = \epsilon$$

$$G = \sum W \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}, \quad e = \sum W \begin{pmatrix} I_x \\ I_y \end{pmatrix}$$

The method finds the good features to track by computing the eigenvalues of the matrix $G$, ($\lambda_1$ and $\lambda_2$), and checking them against a predetermined threshold, ($\lambda$), that is if $\min(\lambda_1, \lambda_2) > \lambda_{[35]}$.

## 3.7 Spatial Relationships

### 3.7.1 AT

Smith [29] defines AT (uncertain or Approximate Transformation) as an estimate between two different coordinate frames, companying with a covariance matrix which shows how much uncertainty this estimate has.

Each AT represent an observation (or other kind of sensing) in a relative motion. Smith used $\text{AT}(A) = [x, y, \theta]$ (where $\theta$ is the rotation about z-axis), to express the relative location of landmark A to the world reference frame, and $\text{AT}(AB)$ to express the relative location of landmark A coordinate frame to landmark B's.

In Figure 3.2 (from [29]), Smith and Cheeseman used an arrow (form the reference frame to the landmark) to represent the AT and an ellipse representing the two dimensional Gaussian distribution) around the landmark to show its uncertainty.
Figure 3.2: A sequence of approximate transformations. line: AT, ellipse: AT’s uncertainty, solid: gained by movement, dashed: gained by observation.

3.7.2 Coordinate Frame Relationships

Having different ATs in different coordinate frames, we need to estimate the relative AT between these frames. To do so, we must use some operations to combine information from the middle steps and produce a single AT([29], [28]).

We can write any relationship as:

\[ y = g(x) \]
3.7. SPATIAL RELATIONSHIPS

Where $g()$ is a general function consisting of a desired combination of Compound, Merge and Reversal operations. The estimate of the first two moments two moments, i.e. mean and covariance matrix, of this function is given by:

\[
\hat{y} \approx g(\hat{x}) \\
C(y) \approx G_x C(x) G_x^T
\]

Where $G_x \equiv \frac{\partial f(x)}{\partial x}(\hat{x})$

**Compounding:**

Compounding operation allows us to recursively compress a chain of subsequent ATs and get a single AT. By using this operation, in each step (combining two AT) the uncertainty of the resultant AT with respect to the two ATs it is made from will increase. This is similar to moving with closed eyes in an unknown environment. With each step you take, the uncertainty about where you are with respect to where you started increases.

Compounding formulas for a pair are given by:

\[
x_{ik} = x_{ij} \oplus x_{jk} = \begin{bmatrix}
x_{jk} \cos(\theta_{ij}) - y_{jk} \sin(\theta_{ij}) + x_{ij} \\
x_{jk} \sin(\theta_{ij}) + y_{jk} \cos(\theta_{ij}) + x_{ij} \\
\theta_{ij} + \theta_{jk}
\end{bmatrix}, \quad x_m = \begin{bmatrix} x_m \\ y_m \\ \theta_m \end{bmatrix},
\]

compounded mean, is:

\[
\hat{x}_{ik} \approx \hat{x}_{ij} \oplus \hat{x}_{jk}
\]

and compounded covariance matrix is:

\[
C(x_{ik}) = J_\oplus \begin{bmatrix}
C(x_{ij}) & C(x_{ij}, x_{jk}) \\
C(x_{jk}, x_{ij}) & C(x_{jk})
\end{bmatrix} J_\oplus^T,
\]
where Jacobian of the compounding operation is:

\[
J_{\oplus} = \left( \begin{array}{ccc}
1 & 0 & -(y_{ik} - y_{ij}) \\
0 & 1 & (x_{ik} - x_{ij}) \\
0 & 0 & 1
\end{array} \right) \cos(\theta_{ij}) - \sin(\theta_{ij}) 0 \\
\left( \begin{array}{ccc}
0 & 1 & \sin(\theta_{ij}) \\
0 & 0 & \cos(\theta_{ij}) \\
0 & 0 & 1
\end{array} \right)
\]

\[
= \begin{bmatrix} J_1_{\oplus} & J_{2_{\oplus}} \end{bmatrix}
\]

Merging: $\otimes$

The merging operation is used to combine parallel ATs between the same pair of coordinate frames. These parallel ATs occur as a result of having more than one source of data: sensors which collect data and also UAV’s odometry data after each movement. So the sensor result and the final approximation (computed with $\oplus$ operation) give us two different pieces of information about the same landmark, and combining these two will decrease the uncertainty.

Merging a pair of transformations $X_1$ and $X_2$ is showed by:

\[
X_3 = X_1 \otimes X_2.
\]

In order to find the merged mean and covariance matrix, the Kalman gain factor is computed as:

\[
K = C_1 \times [C_1 + C_2]^{-1},
\]

using this gain, we can compute merged covariance matrix:

\[
C_3 = C_1 - K \times C_1,
\]

and merged mean:

\[
\hat{X}_3 = \hat{X}_1 + K \times (\hat{X}_2 - \hat{X} - 1),
\]

where:

$C_1$, $C_2$ and $\hat{X}_1$, $\hat{X}_2$ are covariance matrices and mean vectors of the two ATs we want to merge respectively, and $C_3$ and $\hat{X}_3$ are covariance matrix and mean vector of the
resulting merged pair.

**Reversal:** $\ominus$

For merging and compounding, all ATs must be in the right direction. Otherwise, using the reversal operation we can correct the direction (sense) of the AT $X_{ij} = (x_{ij}, y_{ij}, \theta_{ij})$ which is in the wrong direction.

Reversal formulas:

$$X_{ji} = \ominus X_{ij} = \begin{bmatrix} -x_{ij} \cos(\theta_{ij}) - y_{ij} \sin(\theta_{ij}) \\ x_{ij} \cos(\theta_{ij}) - y_{ij} \sin(\theta_{ij}) \\ -\theta_{ij} \end{bmatrix}$$

Reversal mean:

$$\hat{x}_{ji} \approx \ominus \hat{x}_{ij}$$

Reversal covariance matrix:

$$C(X_{ji}) = J_\ominus C(X_{ij}) J_\ominus^T$$

Where the jacobian of the reversal operation is:

$$J_\ominus \equiv \frac{\partial X_{ji}}{\partial X_{ij}} = \begin{bmatrix} -\cos(\theta_{ij}) & \sin(\theta_{ij}) & y_{ji} \\ \sin(\theta_{ij}) & -\cos(\theta_{ij}) & -x_{ji} \\ 0 & 0 & -1 \end{bmatrix}.$$
Bayesian Super Resolution technique [32]. Learning these properties are useful especially when examining the observed landmark to find out if it has been seen before or is a new one. To know the environment completely, UAV needs to know each patch’s pose too. The most popular way for this aim is the EKF method, where the world is represented as a set of features and each feature is the pose of a single patch mentioned above. EKF tracked a mean state vector which consists of the UAV pose followed by the pose of each feature respectively [33]. Figure 3.4 illustrates a simple example for the world map.
3.7. SPATIAL RELATIONSHIPS

Because we do not know exactly where the landmarks and UAV are, we must show the objects in a stochastic map which includes uncertain spatial relationships, their uncertainties and a scale to show the inner-dependencies between these uncertainties. The uncertain spatial relationship can be represented as a probability distribution around its spatial variable, and because this distribution is unknown, we can model it by just estimating its first two moments (mean $\hat{x}$ and covariance matrix $C(x)$) and use this normal distribution to approximate the real distribution.

For the 2D case map is made with the following 2 steps algorithm, assuming no sensor is provided, and the world is 2D:

step 1: Adding the first relation to the empty map
\[
\hat{X} = \hat{X}_1 = \begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{\phi}
\end{bmatrix},
C(X) = \begin{bmatrix}
\sigma_x^2 & \sigma_{xy} & \sigma_{x\phi} \\
\sigma_{xy} & \sigma_y^2 & \sigma_{y\phi} \\
\sigma_{x\phi} & \sigma_{y\phi} & \sigma_x^2
\end{bmatrix}
\]

step 2: Adding a new object, one at a time:

\[
\hat{X}' = \begin{bmatrix}
\hat{x} \\
\hat{x}_n
\end{bmatrix},
C(X') = \begin{bmatrix}
C(X) & C(X, X_n) \\
C(X_n, X) & C(X_n)
\end{bmatrix}
\]

### 3.7.4 Updating The Map

The map can change if one of the followings situations happen:

1. the world itself changes (note: UAV is part of the world, if it moves the world will change, and because we assumed landmarks are stationary in our research, the world can change only by UAV movement).
2. our knowledge about the world changes, for example by observing a landmark again and more precisely (loop closure problem).

We have estimated the world map by its first two moments, so in order to change
the map we must change and update these two moments, Figure 3.5 (from [28]) shows the change in map due to change of the world and adding new observations (measurements).

Smith et al. assumed in [28] that the new observations are applied at discrete time $k$, and their effect is immediate. They showed the moments before the observation by $x_k^-$ and $C(x_k^-)$ and right after it by $x_k^+$ and $C(x_k^+)$. By making a new observation, our knowledge about the world will be increased and this will decrease the uncertainty of the observed landmark and all the world respectively (direct result of merging operation).

For finding $x_k^+$ and $C(x_k^+)$ mathematically, we must model our sensor (camera in our case) somehow to describe the effect of the sensor on mapping the spatial variables ($x_i$: 2D image patches) into the sensor variables ($z_i$: pose of each landmarks). Camera model is defined as function $h$, and because the measurement is noisy we add an independent gaussian zero mean noise $v$ with a given covariance to it:

$$z = h(x) + v$$

The estimated map after this observation, using the formula in Section 3.2 is:

$$\hat{x}|z = \hat{x} + C(x,z)C(z)^{-1}(z - \hat{z})$$

$$C(x|z) = C(x) - C(x,z)C(z)^{-1}C(x,z)$$

where $x_k^+ = x|z$ and $x_k^- = \hat{x}$

If the above covariance matrices $C(x)$ and $C(x, z)$ are to be replaced by their approximations, then we would get the Kalman Filter equations given below:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k[z_k - h_k(\hat{x}_k^-)]$$

$$C(\hat{x}_k^+) = C(\hat{x}_k^-) - K_kH_xC(\hat{x}_k^-)$$

$$K_k = C(\hat{x}_k^-)H_x/6t[H_xC(\hat{x}_k^-)H_x^T + C(v)_k]^{-1}$$
Between the two steps of updating the sensor (i.e time $k - 1$ and $k$), UAV can move. Assume the UAV is the $R$th relationship in the map, so its current estimated location is defined by $x_R$, future location (after movement) by $x'_R$ and the relative motion it makes to get to this place by $y_R$, note that this relative motion is uncertain. So, using the compounding formula given in Section 3.5.2 and assuming $x_R$ and $y_R$ have independent errors, the new state vector and covariance matrix can be inferred as follows:

\[
x'_R = x_R \oplus y_R
\]
\[
\hat{x}'_R \approx \hat{x}_R \oplus \hat{y}_R
\]
\[
C(x'_R) \approx J_{1\oplus}C(x_R)J_{1\oplus}^T + J_{2\oplus}C(y_R)J_{2\oplus}^T
\]
\[
C(x'_R, x_i) \approx J_{1\oplus}C(x_R, x_i)
\]

and the new map will become:

\[
\hat{x}' = \begin{bmatrix}
\hat{x}'_R \\
\hat{x}_R
\end{bmatrix}, \quad C(x'_R) = \begin{bmatrix}
C(x', x'_R) \\
C(x'_R, x) & C(x'_R)
\end{bmatrix}
\]
Chapter 4

SLAM

4.1 Abstract

Simultaneous localization and mapping (SLAM) is the problem of being in an unknown location in an unknown environment and simultaneously learning the environment (mapping) and determining the location accurately and repeatedly (localization) using this map. So, in order to solve the slam problem, we must answer the following questions:

1. Where am I?

Localization is the act of providing the map to UAV, where UAV is required to find its location with respect to the objects in the given map.

2. What does the world around me look like?

Mapping on the other hand is when the UAV location is known, and we ask it to find all the object’s locations in the environment surrounding it (create the sparse map of the room) with respect to its own location.

The main problem here is, to map a feature in the world map UAV needs to know its own location; and to find its own location, UAV needs the accurate map of the
Figure 4.1: 1: start the prediction step, 2: estimate new location, 3: add process noise, 4: start the measurement step, measure features, 5: update positions and uncertainties.

world! And this is a bootstrapping problem.
So, Slam is the act of the UAV building a map of the environment around it and using this map to find its location at the same time.

4.2 Implementation

SLAM is considered as a solved problem now ([14] and [8]), and can estimate the trajectory of the sensor (which is located on the robot, and hence has the same
4.2. IMPLEMENTATION

Table 4.1: SLAM Notations

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>state vector of UAV, including its location and orientation (pose) at time $k$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{0:k} = {X_{0:k-1}, x_k}$</td>
<td>history of UAV pose, up to and including time $k$.</td>
</tr>
<tr>
<td>$u_k$</td>
<td>the control input (applied at time $k-1$) which derived UAV to the state $x_k$ (at time $k$).</td>
</tr>
<tr>
<td>$U_{0:k} = {U_{0:k-1}, u_k}$</td>
<td>history of input controls, up to and including time $k$.</td>
</tr>
<tr>
<td>$z_{ik}$</td>
<td>observation (sensor input) of the $i$th landmark at time $k$.</td>
</tr>
<tr>
<td>$z_k = {z_{1k}, \ldots}$</td>
<td>observation of all the landmarks at time $k$.</td>
</tr>
<tr>
<td>$Z_{0:k} = {Z_{0:k-1}, z_k}$</td>
<td>history of observed landmarks, up to and including time $k$.</td>
</tr>
<tr>
<td>$m_i$</td>
<td>estimated location of the $i$th landmark, we assumed all the landmarks are stationary (their true locations don’t change with time).</td>
</tr>
<tr>
<td>$m = {m_1, m_2, \ldots, m_n}$</td>
<td>Set of all landmarks (Notice that the map parameters do not have a time subscript as they are modelled as stationary)</td>
</tr>
</tbody>
</table>

trajectory as the robot) and locations of all the landmarks in the environment without any prior information.

4.2.1 Notations

In order to solve the SLAM problem Bailey and Durrant-Whyte [14] introduced the notations showed in Table 4.1.

4.2.2 Observation Model

$$P(z_k|x_k, m)$$

The observation model describes the effect of the observation. It computes the probability of making the $k$th observation of the environment ($z_k$), assuming we have the location of the UAV ($x_k$) and the map of the room ($m$), and describes how the uncertainty can be reduced. If we precisely know the true map, the new observations will not change it, so in
CHAPTER 4. SLAM

that case making observations is independent from the map of the world.

4.2.3 Motion Model

\[ P(x_k|x_{k-1}, u_k) \]

The motion model describes the effect of the control input on state transition and calculates how the uncertainty increases. Motion model is function of the previous state \( x_{k-1} \) and the control input which derived us to the current state \( u_k \). Since the probability of \( x_k \) is just dependent on \( x_{k-1} \), this model is Markov and we can use its properties (as mentioned in Section 3.1) later in the SLAM algorithm.

4.3 Probabilistic SLAM

Hugh and Bailey defined the SLAM problem as the computation of the joint posterior probability \( P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \) or simply \( P(x_k, m) \) for the UAV state \( x_p \) and map \( m \), based on all the observation and control inputs for all time \( t \) in [14]. To do so, they introduced the following two-steps recursive algorithm, in which the first step is the time update and the second one is the measurement-update.

4.3.1 Step one: Prediction (time-update)

When UAV moves, we need to estimate its new position and the uncertainty of its location (knowing that this uncertainty is monotonically increasing). The motion model used in this step is:

\[
P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \times P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1}
\]

4.3.2 Step two: Correction (measurement-update)

The aim of this step is adding new feature to the map and remeasure the previously added features. When UAV observes a feature which was previously in the map, it needs to update the system using Bayes Theorem and Markov property as follows:

\[
P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})}
\]
This formula is a function of the vehicle model \( P(x_k|x_{k-1}, u_k) \) and observation model \( P(z_k|x_k, m) \).

And when UAV observes a new feature (which is not in the map), using the inverse of observation equation \( (P(m_i|x_k, m)) \) its initial position will be estimated.

### 4.4 EKF-SLAM

EKF-SLAM is the most popular solution to the SLAM problem. It uses a linearized Gaussian probability distribution model to approximate the posterior as a Gaussian distribution, where the Mean \( (\mu_k) \) contains the state vector (which is the location of the robot and map of the environment), and covariance matrix \( (\Sigma_k) \) estimates the uncertainties and demonstrates how the elements of state vector are correlated to each other.

A posteriori state estimate equation represents the mean of the state distribution:

\[
\mu_k = \begin{bmatrix} \hat{x}_{k|k} \\ \hat{m}_k \end{bmatrix} = E \left[ \begin{bmatrix} x_k \\ m \end{bmatrix} | Z_{0:k} \right],
\]

and a posteriori estimate error covariance equation represents the variance of the state distribution:

\[
\Sigma_k = P_{k|k} = \begin{bmatrix} P_{xx} & P_{xm} \\ P_{xm}^T & P_{mm} \end{bmatrix} = \begin{bmatrix} x_k - \hat{x}_k \\ m - \hat{m}_k \end{bmatrix} \begin{bmatrix} x_k - \hat{x}_k \\ m - \hat{m}_k \end{bmatrix}^T | Z_{0:k}
\]

Hence, the joint probability distribution which was previously used can be defined as the following Normal equation, knowing the additive process and measurement noises are white:

\[
P(x_k, m| Z_{0:k}, U_{0:k}, x_0) \leftrightarrow N(\mu_k, \Sigma_k)
\]

Defining the posterior as this normal distribution, we can apply EKF to compute the mean and covariance matrix of the joint posterior distribution \( P(x_k, m| Z_{0:k}, U_{0:k}, x_0) \).
Table 4.2: EKF-SLAM Notations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>motion model</td>
</tr>
<tr>
<td>( h )</td>
<td>measurement model</td>
</tr>
<tr>
<td>( w_k = N(0, Q_k) )</td>
<td>additive, zero mean, uncorrelated Gaussian motion disturbances with covariance ( Q_k )</td>
</tr>
<tr>
<td>( v_k = N(0, R_k) )</td>
<td>additive, zero mean, uncorrelated Gaussian observation errors with covariance ( R_k )</td>
</tr>
</tbody>
</table>

**Motion model**

The motion model describes change of robot’s state with time as:

\[
P(x_k|x_{k-1}, u_k) \Leftrightarrow x_k = f(x_{k-1}, u_k) + w_k
\]

**Observation model**

Observation model predicts the measurement value given robot’s state as:

\[
P(z_k|x_k, m) \Leftrightarrow z_k = h(x_k, m) + v_k
\]

where the notations are explained in table 4.2.

**4.4.1 Time Update(Predict)**

The purpose of this step is to update the vector state, as the landmarks or the robot itself may move during the learning process. So, there is no need to compute the time update if the landmarks and robot are stationary and do not move. In our case all the landmarks are stationary and just the robot can move, and when it moves using this step we can find its new position and also the uncertainty of its location, noting that positional uncertainty always increases. Time update performs via the following two steps:

step 1: project the state ahead:

\[
\hat{x}_{k|k-1} = f(\hat{x}_{k|k-1}, u_k)
\]
4.4. EKF-SLAM

step 2: project the error covariance ahead:

\[ P_{xx,k|k-1} = \nabla f P_{xx,k|k-1} \nabla f^T + Q_k \]

### 4.4.2 Observation Update (Correct)

This step aims to add new features to map and remeasure the existing features by observing them again:

step 1: update the estimate with observation \( z_k \):

\[
\begin{bmatrix}
\hat{x}_{k|k} \\
\hat{m}_k
\end{bmatrix} = \begin{bmatrix}
\hat{x}_{k|k-1} \hat{m}_{k-1}
\end{bmatrix} + W_k [z_k - h(\hat{x}_{k|k-1}, \hat{m}_{k-1})]
\]

step 2: update the error covariance:

\[ P_{k|k} = P_{k|k-1} - W_k S_k W_k^T, \]

where the Kalman gain is:

\[ S_k = \nabla h P_{k|k-1} \nabla h^T + R_k \]

\[ W_k = P_{k|k-1} \nabla h^T S_k^{-1}, \]

and \( \nabla h \) is the Jacobian of \( h \) computed at \( \hat{x}_{k|k-1} \) and \( \hat{m}_{k-1} \).

### 4.4.3 Advantages and Disadvantages

EKF-SLAM solution inherits many of the standard EKF benefits and problems, the most important advantages of this method are its simplicity and the proven experience of working well in practice.

The problems of EKF-SLAM mostly arise because firstly EKF-SLAM represents the estimate of state by the mean and its uncertainty by a covariance matrix and because we get
these two moments by using the linearization they are approximate and not exactly the true moments of the distribution. And secondly we assumed the map distribution is uni-modal gaussian and so it can be represented by its first two moments but in fact this distribution is not gaussian [9].

Consistency

Bailey et al. claimed in [9] the most important source of inconsistency of EKF-SLAM is due to the heading uncertainty (variance). They examined their hypothesis in different scenarios (moving and stationary vehicle) and with different heading uncertainty and concluded that with a relatively small heading uncertainty (less than 1.7 degree), inconsistency is time dependent and its effect can be weaken by adding some stabilising noise. But for Large variance, failure occurs after a few updates and so it can not be prevented by adding noise or any other method.

Julier and Uhlmann, on the other hand, showed in [37] the map built by the full covariance SLAM algorithm is always inconsistent, even in the case of stationary vehicle and no process (motion) noise. And the reason people working in this area didn’t mention it in the previous studies was the fact that this inconsistency appears after about thousands of updates, and the works that have been performed in the past had only simulated for the first few hundred steps in which the divergence is not noticeable.

So, inconsistency becomes evident when either the ground truth is available or the loop becomes big (e.g. greater than 100m), and it can even have more effect on divergence than computational complexity problem [26].

Convergence

Huang and Dissanayake showed the error of robot heading has an important role in limit or lower bound of the uncertainty of the estimation of the landmark locations, and so has a significant effect on convergence analysis of EKF-SLAM. They stated and proved the following theorems for the convergence of 2D EKF-SLAM in [13], and also for finding the limit of covariance matrices and limit of inconsistency:

**Theorem 1:** The determinant of any submatrix of the map covariance matrix decreases monotonically as successive observation are made.
**Theorem 2:** If robot keeps stationary and observe a new landmark many times, the robot uncertainty keeps unchanged.

**Theorem 3:** If robot keeps stationary and observes two new landmarks many times, the robot uncertainty keeps unchanged.

**Theorem 4:** In EKF SLAM, if the robot keeps stationary at point A and observing a new landmark $k$ times, the inconsistency may occur due to the fact that Jacobians are evaluated at different estimation values. When $k \to \infty$, the inconsistency may cause the robot orientation error to be reduced to zero.

Whyte and Bailey showed in [14] the map will converge if the map covariance matrix and all landmark’s pair submatrices monotonically converged to zero. The distinct landmark’s variances inherit the initial uncertainty of robot position and observations and converge to the lower bound monotonically as shown in Figure 4.2 from [14].

![Figure 4.2: The convergence in landmark uncertainty. The plot shows a time history of standard deviations of a set of landmark locations. A landmark is initially observed with uncertainty inherited from the robot location and observation. Over time, the standard deviations reduce monotonically to a lower bound. New landmarks are acquired during motion.](image-url)
In short, they claimed the Convergence and consistency can only be guaranteed in the linear case.

**Loop Closure**

As the UAV moves inside the environment and explores the map around it, the localization error will increase, therefore the uncertainty in UAV position will increase because of this accumulated error. As a direct result of this problem, the map will diverge gradually. To overcome the divergence, UAV must be able to recognize the positions which it was in before and use their new observations to correct the map estimate and reduce the uncertainty of the map and its pose. This process in SLAM is known as the loop closure step. Gutmann and Konolige [18] illustrated the loop closing result in Figure[]. Since the map will become larger after adding new landmarks in each iteration, and also, since in online processing we need each step to be computed in a little time to have near real time execution, all the computations must be performed with little cost. Hence closing the loop must be performed with little computational cost, and in ideal case it does not need a separate algorithm.

EKF based SLAM needs a separate algorithm to identify loops and closing the loop requires a very expensive computation of $O(N^2)$ as shown in [23]. So, loop closure remains one of the main unsolved issues with EKF-SLAM.

![Figure 4.1: Before and after closing a loop. Courtesy of [34].](image)

![Figure 4.3: Before and after closing the loop, this figure is from [19](image)
4.4. EKF-SLAM

Data Association

Data association, is the process of relating the observed features through the sensor (camera in our case) to the features in the existing map. If we associate these observations incorrectly, the filter update will be based on incorrect data, and if this incorrect association happens for more and more features, the EKF filter will diverge and can not be corrected [25]. Incorrect data association can occur if we use an inaccurate or imprecise model for the sensor, if the measurements (observations) are too noisy, or simply when the features are not stationary in the environment as we previously assumed and can move. Kalman filter based SLAM can not cope with the incorrect data association problem and will diverge.

Computational Complexity

The most computational cost of EKF-SLAM is because of computing the Kalman gain in the observation update step. Due to computing the inverse matrix its cost is of the order of $O(N^2)$. Hence, EKF update procedure needs a quadratic number of operation and its computational complexity is $O(N^2)$ where $N$ is the number of features in map. So, EKF-SLAM can not be used in the case of big maps (i.e. big $N$).

Uni-modal Gaussian

Because the map is assumed to have "Uni-modal Gaussian Probability" distribution in all the cases, missions like the one in Figure ?? can not be completed. For solving this problem the EKF approach must be replaced with other methods.

Nonlinearity

In EKF, nonlinear functions are approximated with their linear approximation. So the filter can diverge if the function’s nonlinearity is large.
Chapter 5

Experiments

The main focus of this research is to find the correct pose of an autonomous-flying quad rotor in a room, and to use this information to better control the device. The quad rotor we used was equipped with a single onboard camera, and MonoSLAM was used as the pose finder, with the onboard camera as sensor. MonoSLAM is described in Section 5.1. Davison’s C++ library was used to implement MonoSLAM\(^1\). Ground truth of quad rotor pose was checked by VICON cameras installed in robust control lab; a brief overview of VICON is presented in Section 5.2.

In order to simulate the flight of the UAV, we used a handheld camera. The result of the experiments are presented in Section 5.3.

5.1 MonoSLAM

This section presents the result of Davison’s work on the MonoSLAM problem \([12]\). In MonoSLAM, the main aim of the process is finding the robot position (localization) rather than building the complete map of the room, and the robot will add just as much feature to the map as it is necessary to find its position in the unknown world around it. So, the output map is a sparse set of high quality landmarks including around 100 features for a limited space in the room. Davison assumed the scene is rigid and hence the features inside the scene are stationary.

\(^1\)The latest version of this software is available at his website: http://www.doc.ic.ac.uk/ ajd/software.html
The camera state vector $\hat{x}_v$ is a 13 parameter vector consisting of a 3-element vector $r^W$ representing the 3D metric position, a 4-element vector $q^{RW}$ representing the orientation quaternion, and 3-element vectors $v^W$ and $w^R$ representing velocity and angular velocity respectively. The camera is modelled as a rigid body with its pose completely described by its translation and rotation:

$$\hat{x}_v = \begin{bmatrix} r^W \\ q^{RW} \\ v^W \\ w^R \end{bmatrix},$$

where $W$ indicates the "world frame" and $R$ illustrates the "robot frame".

Map of the world $\hat{x}$, consists of camera state vector($\hat{x}_v$) and feature’s position ($\hat{y}_i$). The uncertainty of this map is showed by the covariance matrix $P$:

$$\hat{x} = \begin{bmatrix} \hat{x}_v \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{bmatrix}, P = \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & \cdots \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & \cdots \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The map is made by the image patches with their corresponding feature positions (mean) and an elliptical shape around each feature to show their uncertainty bound (standard deviation) in 3D space.

If the map had $N$ features, the size of the map would have become $O(N^2)$ because of the size of matrix $P$, and the MonoSLAM complexity is also $O(N^2)$. Because of the real time processing, for 30Hz implementation, the possible number of features in map is around 100 features, and Davison showed in [11] 100 well-chosen features are sufficient to span the limited volume of the room where the experiment is taking place in.

As mentioned in the last chapter, close features (which can be observed at the same time) are related to each other with a good approximation, but the group position has a large
uncertainty. Which means the elliptical bounds mentioned above are correlated and this legitimizes the non-zero off diagonal elements of covariance matrix.

MonoSLAM is based on repeatable observation of features from different viewpoints. Various subsets of features at different depths will be covisible as the camera moves freely inside the limited 3D space and different size loop closure will be occurred; so knowing the detailed correlation between the map’s subsets is very important for solving the loop closure problem, Davision et al. showed that among all the known methods, ”Standard single, full covariance EKF approach” is the most computationally feasible method which can achieve the goal of ”long term, repeatable localization within restricted volumes”.

Davison and Maury used large image patches(11x11 pixels) as the long-term landmarks, and used the Shi and Tomasi feature detection approach for finding the features. Instead of using the normal template matching for recognizing the features after reobservation, they assumed each landmark is placed on a locally planar surface and approximated this surface’s normal with the vector oriented from the feature to the camera. Because the real orientation is unknown at initialization, each feature will be stored as oriented planar texture after being fully initialized. In order to detect the initialized feature from a different viewpoint when the camera pose changes, they projected the image patch to the predicted place where they expected to find the feature (using the new camera 3D position and orientation) and made the new template and compared it with the existing image patch in each location until a peak is found 5.1.

In order to update the normal of each feature’s plane in 3D map, as the camera moves and the new estimate of its pose becomes available, the warped version of the initial image patch will be computed using the following formula:

$$H = CR[n^T x_p I - tn^T]C^{-1},$$

where: $C$= camera’s calibration matrix (perspective projection),
$R$= camera rotation matrix,
$t$= camera translation matrix,
$n$= surface normal,
$x_p$= projection of patch’s cente,
Figure 5.1: The evolution of depth probability density from uniform distribution to Gaussian, points are uniformly distributed over the image patch’s normal line from 0.5 to 5m. Courtesy of [12].

Figure 5.2: The camera poses in 2 different viewpoints looking at the same image patch with normal n. Courtesy of [12].

$I = 3 \times 3$ identity matrix,
which corresponds to its predicted appearance from the current viewpoint. The difference
between this predicted amount and the measurement is used to update the feature’s surface normal direction. Davison assumed in [12] the feature’s plane normal direction is just weakly correlated with camera and feature positions, and therefore maintains a separate 2-parameter EKF for estimating the normal direction for each feature.

To aid the system in its computation to assign a precise scale to the estimated map and to help it to estimate the camera motion with a good approximation in the starting frames, the map is initialized with 4 features (corners of a letter sized paper for example) with known position and known appearance. Also the initial position of the camera in the first frame (the viewpoint from which the initialized features are stored) is known.

Gathering several measurements from different viewpoints is required for estimating the feature’s depth. To find the depth, a semi-infinite 3D line is defined after initializing the feature, starting from the camera position and passing through the feature point and going to infinity. The true location of the feature lies on this line and the Gaussian uncertainty of this ray is showed by an elliptical shape around it. The representation of this line is showed by the vector $y_{pi}$ as follows, where $r_i$ is the position of the camera and $\hat{h}_i^w$ is a vector which describes its orientation:

$$y_{pi} = \begin{pmatrix} r_i^W \\ \hat{h}_i^W \end{pmatrix}$$

To realize if the recognized feature is a new one (so it can be initialized) or an existing one (so the knowledge about its depth can be updated), a set of uniformly distributed depth hypothesis will be defined along this semi-infinite line, the image patch (stored in initialization step) will be projected in each discrete place to make new image templates, and the observed feature will be compared with them using 2D template matching technique. The location of the hypothesis corresponding to feature $y_i$ at depth $\lambda$ has the form $y_{\lambda i} = r_i^W + \lambda \hat{h}_i^W$, and its Gaussian uncertainty is showed with an ellipse around this point. Therefore, at the beginning, for each feature we have a set of ellipses uniformly distributed along a line with their centers lying on that line. After reobserving a feature and finding its likelihood to be which one of these hypothesis along the line, the probability of each hypothesis will be recalculated until the correct depth is found.

The robot must decide when a new feature can be added to the map and when the
existing feature can be deleted from it while it is flying inside the room and grabbing new frames. Davison showed it is sufficient to have around 12 visible features in each frame for accurate localization. A new feature will be added if the number of visible features become less than this threshold and will be deleted from the map after more than 50% failure of detecting it in a predetermined number of frames. Since the image patches are relatively large, it is very rare to have a poor match between a true feature and an occlusion due to their appearance similarity.

5.1.1 Motion

As mentioned in Chapter 3, the SLAM algorithm consists of two steps: prediction and update. For MonoSLAM algorithm these two steps are as follows:

1) prediction step:
Using the standard pinhole model for the camera, the location \((u, v)\) we expect to find the feature can be found from:

\[
h_i = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{k_u h_i}{h_R L_z} u_0 - f k_u h_i \\ \frac{k_v h_i}{h_R L_z} v_0 - f k_v h_i \end{pmatrix},
\]

where \((f, k_u, k_v, u_0, v_0)\) are camera calibration parameters and \(h^R_L\) is the expected feature position relative to camera in the robot’s frame:

\[
h^R_L = R^{RW}(y^W_i - r^W).
\]

2) update step:
The camera motion model which Davison used in this project is constant velocity, constant angular velocity, assuming the unknown accelerations occur with a Gaussian profile cause an impulse of velocity \((V^W)\) and angular velocity \((\Omega^R)\) in each step:

\[
n = \begin{pmatrix} V^W \\ \Omega^W \end{pmatrix} = \begin{pmatrix} a^W \Delta t \\ \alpha^R \Delta t \end{pmatrix}
\]
Using the above motion model, the new state estimate update $f_v(x_v, u)$ and its uncertainty $Q_v$ can be computed as follows:

$$
f_v = \begin{pmatrix}
    r_{new}^W \\
    q_{new}^{WR} \\
    v_{new}^W \\
    w_{new}^R
\end{pmatrix}
= \begin{pmatrix}
    r^W + (v^W + V^W)\Delta t \\
    q^{WR} \times q((w^R + \Omega^R)\Delta t) \\
    v^W + V^W \\
    w^R + \Omega^R
\end{pmatrix},
$$

$$
Q_v = \frac{\partial f_v}{\partial n} P_n \frac{\partial f_v}{\partial n}^T,
$$

where $q((w^R + \Omega^R)\Delta t)$ is the quaternion defined by the rotation vector $(w^R + \Omega^R)\Delta t$, $n$ is the noise vector and $P_n$ is its diagonal covariance matrix representing the uncorrelated noise between all state's components. The motion uncertainty growth rate is determined by the size of $P_n$, the covariance of noise vector; and smoothness of the motion is defined by its parameters value. Small $P_n$ corresponds to a smooth motion with small acceleration (which can not fulfill the rapid movement), high $P_n$ on the other hand can cope with this kind of motion but needs us to make lots of good measurement in each step.
5.2 Vicon

Vicon system is designed for optical motion capturing by tracking the infrared reflective markers attached on the object of interest, which reflect the infrared light emitted by a ring of infrared LEDs around the lens of each camera.

Vicon systems consists of 4 main parts:

1. **MX Cameras**: system includes 8 high resolution high speed cameras installed around the room. Each camera outputs raw IR point values reflected by the markers to the ultranet system as shown in Figure 5.4.

![Vicon Camera](image)

Figure 5.4: Vicon camera.

2. **Ultranet MX**: which links the MX cameras and the host computer.

3. **Host computer**.

4. **Infrared reflective markers**: which are available in 4 different sizes as shown in Figure 5.5. The key point in using the Vicon is placing these markers somehow to give us the opportunity to predict the complete motion from their positions. Because the camera is a rigid body without any joint, the model we used for it is a simple box, with a marker on the center of each side as showed in Figure 5.6.

By fitting the box model into camera motion, its exact pose will be found for checking the ground truth of the MonoSLAM algorithm output.
5.3 Experimental Results

We used "Panasonic PV-GS500" to run the experiment and "Vicon cameras" to check the ground truth of MonoSLAM program. Camera calibration was performed using the standard software and calibration grid [1] and the following parameters were obtained:

Focal length: $f_k = f_v = 941.5$ pixels;
Principal point: $(u_0, v_0) = (426.0, 239.5)$;
Distortion factor: $K_1 = 0.001$;
Image size: $853 \times 480$.

The experiments were done inside the robust control lab, and data sequence were collected while camera was looking downward to the floor and moving in a restricted volume inside the room.

In our setup, several challenges made the task harder. The problem here was that usually there is not enough object on the floor, and the features are mostly tile corners which are very similar to each other and can lead the system to incorrect loop closure. And also because we wanted to move the camera inside the space which is covered by Vicon’s camera, it was not possible to move the camera very high, so using the objects located on the desk and tables was not possible either. We made a cluttered space full of objects on the floor to help the system.

Camera localization was performed using Davison’s MonoSLAM open source library, and the ground truth was checked by Vicon Cameras. The output of MonoSLAM for some discrete times is shown in Figure 5.6. Each figure consists of the map of the room and the detected features for the corresponding frames. Camera trajectory for a sample run is shown in Figure 5.9 and the errors in x and y dimensions are shown in Figure 5.7.

The output of Vicon cameras has also an error with the average standard deviation of $(0.0194, 0.0154, 0.0261)$ in $x$, $y$ and $z$ dimensions respectively in this experiment, therefore the error illustrated in Plot 5.7 is not only due to MonoSLAM program. We can see the
difference between the output of MonoSLAM and output of Vicon has a big peak at the beginning and get less as the program continued.
Figure 5.6: The output of MonoSLAM for a sample sequence, with 4 startup features. The map is built gradually by updating the visible features in each frame. Blue marks correspond to failed matches, red marks correspond to successful matches and yellow marks correspond to unused features in current frame.
5.3. EXPERIMENTAL RESULTS

Figure 5.7: MonoSLAM Error in X, Y and Z Dimension.

(a) Robot localization error in "x" dimension. (b) Robot localization error in "y" dimension. (c) Robot localization error in "z" dimension.

Figure 5.8: UAV orientation, MonoSLAM and Vicon outputs

(a) Orientation in x-y plane (b) Orientation in y-z plane (c) Orientation in z-x plane
Figure 5.9: Comparing the output trajectories in 2D space, from Vicon and MonoSLAM.
Chapter 6

Conclusions

In this research, we studied different approaches for finding the pose of the flying quadrotor inside the room. The disadvantages of each approach in real life situations were explained and finally the Davison’s MonoSLAM approach was chosen for the localization step in "Vision-Based UAV Control" research.
This was the starting point for our UAV controlling approach, and we showed MonoSLAM can generate the reliable result to be used in control step.
Chapter 7

Discussion and Future Works

7.1 Discussion

7.1.1 Which Feature to Add

Davison relied on finding high quality landmarks, and added a new one when the number of observable features in each frame was below a predefined threshold (e.g 12).

It is possible to have the most high quality features, placed close to each other in an environment. This will cause having most of the landmarks overcrowded in some part of the map and no information in some other parts. Davison’s assumption for having 12 landmarks in each frame will help a lot to solve this problem, but it’s better to find features scattered around the room equally. This can also help us to run MonoSLAM in bigger environments.

The reasonable solution is when we want to add a new landmark to our map, not look for them in a defined space around the existing ones. This will prevent the program from finding close high quality features (which can not add that much useful information to the localization problem).

7.1.2 Moving Objects

When we use the SLAM algorithm for localization, after building the complete map, occlusion, acceleration and any unexpected movement can be handled by the system and they can not affect the pose estimation process.
7.2. Future works

The next step of this research is unmanned controlling of the quadrotor inside the room, and also we *will* try to note the solutions mentioned previously in discussion part to improve the UAV localization results.
Bibliography


