REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

VISUAL ASSISTED FORMATION CONTROL OF DISTRIBUTED UAV SWARM

Onur ÖZTÜRK

MASTER OF SCIENCE THESIS
Department of Mechatronics Engineering
Mechatronics Engineering Program

Advisor
Assoc. Prof. Dr. Aydın YEŞİLDİREK

January, 2021

REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

VISUAL ASSISTED FORMATION CONTROL OF DISTRIBUTED UAV SWARM

A thesis submitted by Onur ÖZTÜRK in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 07.01.2021 in Department of Mechatronics Engineering, Mechatronics Engineering Program.

Assoc. Prof. Dr. Aydın YEŞİLDİREK
Yildiz Technical University
Advisor

Approved By the Examining Committee

Assoc. Prof. Dr. Aydın YEŞİLDİREK, Advisor
Yildiz Technical University _____

Prof. Dr. Fikret ÇALIŞKAN, Member
İstanbul Technical University _____

Assoc. Prof. Dr Mehmet Selçuk ARSLAN, Member
Yildiz Technical University _____

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of Visual Assisted Formation Control of Distributed UAV Swarm supervised by my supervisor, Assoc. Prof. Dr. Aydın YEŞİLDİREK. In the case of a discovery of false statement, I am to acknowledge any legal consequence.

Onur ÖZTÜRK

Signature

This study was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) Grant No: 2210.

Dedicated to my wife and family

ACKNOWLEDGEMENTS

I present my respect and thanks to,

My esteemed professor and my thesis advisor Assoc. Prof. Dr. Aydın YEŞİLDİREK, for his guidance and insight throughout my master's education and thesis work,

My beloved wife Elif Meltem Aslan ÖZTÜRK, for her love and support in this difficult process,

My esteemed professor Dr. Erdem ARSLAN, who always supported my academic and private life since my undergraduate education,

My dear family Vildan and Yüksel ÖZTÜRK, who raised me and always supported me.

Onur ÖZTÜRK

TABLE OF CONTENTS

LIST OF SYMBOLS

$\vec{a}$	Acceleration vector.
K	Kalman gain.
n	Measurement noise vector.
$\vec{z}$	Measurement vector.
1_n	n×n identity matrix.
0_n	n×n zero matrix.
C_n	nth camera.
$\vec{p}$	Position vector.
P	Process covariance matrix.
Q	Process noise covariance matrix.
w	Process noise vector.
P_n	Projection of point P on nth camera.
R	Rotation matrix.
λ	Scale factor.
R	Sensor noise covariance matrix.
x	State vector.
A	System matrix.
$\vec{v}$	Velocity vector.

LIST OF ABBREVIATIONS

EKF	Extended Kalman Filter
GPS	Gloval Positioning System
LIDAR	Laser imaging, detection, and ranging
UAV	Unmanned Aeiral Vehicle

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Visual Assisted Formation Control of Distributed UAV Swarm

Onur ÖZTÜRK

Department of Mechatronics Engineering
Master of Science Thesis

Advisor: Assoc. Prof. Dr. Aydın YEŞİLDİREK

In recent years, unmanned aerial vehicles have been used effectively in many military and civilian areas. Because of it provides many advantages areas of surveillance, imaging, mapping, search and rescue etc. It has become an almost indispensable part of them.

The use of more than one unmanned aerial vehicle in the performance of the tasks saves time and cost compared to a single vehicle, while allowing the completion of tasks that cannot be done with a single vehicle. For example, in missions where a certain area is required to be imaging or mapping, a single drone takes longer to scan the entire area than multiple drones. Since the airtime of small unmanned aerial vehicles is relatively short, a battery replacement will interrupt the task and increase this time even more. Using large-sized unmanned aerial vehicles that can stay in the air for a longer time can make the same task too costly. The use of unmanned aerial vehicles swarms allows different tasks to be performed by using methods such as triangulation or back azimuth that cannot be done with a single vehicle.

Success of swarming often depends on preserving swarm formation. It is critical for all swarm members to maintain their position in the formation to successfully perform certain tasks, such as triangulation. In order to achieve this, each swarm member must precisely determine its relative position with respect to other agents.

In this study a swarm of UAVs has been created by using method which relative position of UAVs determined by using only a monocular camera and IMU sensor, without

GPS, VICON, LIDAR etc. which are external dependent or high-cost systems. UAVs are estimates the relative position between them by intersecting areas in the camera images and IMU acceleration measurements they send to each other. In formation acquisition, flocking and formation maneuver simulations, the swarm agents managed to create a swarm by staying within %5 error zone.

Keywords: UAV swarm, sensor fusion, extended kalman filter

ÖZET

Dağınık İHA'ların Görsel Destekli Formasyon Kontrolü

Onur ÖZTÜRK

Mekaronik Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Doç. Dr. Aydın YEŞİLDİREK

Son yıllarda insansız hava araçları birçok askeri ve sivil alanda etkin olarak kullanılmaktadır. Gözetleme, görüntüleme, haritalama, arama kurtarma vb. alanlarda birçok avantaj sağlaması nedeniyle bu alanların neredeyse vazgeçilmez bir parçası haline gelmiştir.

Görevlerin yerine getirilmesinde birden fazla insansız hava aracının kullanılması tek bir araca göre zaman ve maliyet açısından kazanç sağlarken, tek araçla yapılamayacak görevlerin başarılmasına olanak sağlamaktadır.Örneğin belirli bir alanın görüntülenmesi veya haritalanması istenen görevlerde tek bir insansız hava aracının alanın tamamını taraması birden fazla araca göre daha uzun sürmektedir. Küçük boyutlu insansız hava araçlarının havada kalma süreleri nispeten kısa olduğu için batarya değişimi görevi kesintiye uğratıp bu süreyi daha da arttıracaktır. Daha uzun süre havada kalabilen büyük boyutlu insansız hava aracı kullanmak ise aynı görevi fazla maliyetli hale getirebilir. İnsansız hava araçlarının sürü halinde kullanılması tek bir araçla yapılamayacak nirengi veya geriden kestirme gibi yöntemlerin kullanılarak farklı görevlerin yerine getirilmesine imkan sağlar.

Sürü olma başarası genellikle sürü diziliminin korunmasına bağlıdır. Tüm sürü bireylerinin dizilim içerisindeki yerlerini korumaları, geriden kestirme gibi belirli görevlerin başarılı bir şekilde yerine getirilmesi için kritik öneme sahiptir. Bunu başarabilmek için ise her sürü bireyinin diğer bireylere göre bağıl pozisyonunu hassas bir şekilde belirlemesi gerekmektedir.

Bu çalışmada insansız hava araçlarında konum belirlemek için kullanılan

yöntemlerden olan GPS, VICON, LIDAR vb. dışa bağımlı veya yüksek maliyetleri sensörlere ihtiyaç duymaksızın yalnızca monoküler bir kamera ve IMU yardımı ile İHA'lar arasındaki bağıl konum hesaplanarak İHA sürüsü oluşturulmuştur. İHA'lar birbilerine gönderdikleri kamera görüntülerindeki kesişen alanlar ve IMU ivme ölçümleri ile birbirleri arasındaki bağıl konumu tahmin ederler. Gerçekleştirilen formasyon oluşturma, sürü halinde uçuş ve formasyon manevrası simulasyonlarında sürü bireyleri %5 hata bölgesi içinde kalarak sürü oluşturmayı başarmışlardır.

Anahtar Kelimeler: Sürü İHA, sensör füzyonu, extended kalman filtresi

1
INTRODUCTION

1.1 Literature Review

Global positioning system GPS which is used to determine the position cannot provide the desired sensitivity in some cases and cannot be used in indoor areas.Odometry applications have been developed as an alternative to GPS. Odometry is the estimation of the robot's position by continuously measuring the changes in the robot's motion with various sensors in small time intervals. The predicted position is relative to the starting position of the robot, not to a global reference. For example, using the number of turns of the wheel of the vehicle[1], the position relative to the starting point can be found. In the odometry application that Zhang and Singh[2] created using the lidar sensor, they also mapped the areas passed and determined the position of the robot.

In addition the method which camera motion is calculated with the previous frames is called visual odometry. The features in the images are detected with feature detection algorithms [3–9] and matched with the features in the previous image. Nister et al. In their work [10], they determined the relative motion of the camera by detecting the features in each frame with the Harris Corner Detector [3] and matching the features in the previous frame. In this way, the route can be found by following the camera movement. Davidson and Reid [11], Klein and Murray [12] calculated the position of the camera by mapping the region as well as calculating the camera movement. In their recent study, Mur-Artal and Montiel [13] closed the loop when they passed a point again and corrected the accumulated errors.

One dimension is lost because cameras project 3D space to 2D space.Therefore, the actual lengths cannot be calculated because the scale is not known in visual odometries using a single camera. If two cameras are used, the distance between the cameras must also be large to measure relatively long distances. In recent years, a pair of cameras and IMU sensors have been used to calculate the scale that cannot be obtained using a single camera. Weiss and Siegwart [14] calculated the metric scale using the IMU sensor to find the camera position using the previous frames.Visual odometries

developed by using camera and IMU sensor have been used in studies with land vehicles [15] and air vehicles[16–18].

For robotic swarms, the correct determination of each individual's position within the swarm is critical to the preserving the formation.Different sensors and methods are used to achieve this. Rivard et al.'s[19] study in 2008 determined the relative position between robots by placing transmitters and receivers that emit ultrasonic sound waves and detect them on each robot, using the duration of the sound waves in the air. F. Roberts et al. [20], In their study in 2012, placed infrared receivers and transmitters around flying robots and calculated the relative position between robots with the differences in the strength of the signals. Oliveira et al.[21] Calculated the relative positions of the robots from differences in RSSI power in the communication channel in their 2014 study.

Merino et al. [22] conducted the first study in 2006, which determined the relative positions of two unmanned aerial vehicles from intersecting fields of view. In the study, each of the unmanned aircraft calculates its relative position using the intersecting areas on its own image and the image taken from the other. However, since only one camera was used, the height information was obtained by a laser sensor. Achtelik et al. [23] using a single camera and IMU in each unmanned aircraft to determine the position of the vehicles in relative to each other from the intersection of the images in real time. This method has only recently been used in cameras and IMUs [24, 25] and is increasingly gaining popularity.

1.2 Objective of the Thesis

It is aimed to calculate the relative position of each individual unmanned aerial vehicle by comparing the image taken from the other aget with the image taken from its own sub camera. With these camera measurements and taken IMU measurements are fused to determine the relative position between agents. In this way, each individual will maintains its position relative to its connected agents and all swarm member will be able to move with the desired formation. By choosing an agent as leader, all swarm can be controlled together without deterioration in formation.

1.3 Hypothesis

The main contribution of the study compared to other studies [23–25] is that it performs the tasks by creating a formation and preserving it. With the help of images taken from other swarm members using only the camera and the IMU pair, each

individual will be able to maintain its position in the formation with a formation control algorithm. Each agent runs its own formation control algorithm to maintain its position in the swarm without any dependence of central control unit.

2
BACKGROUND

In this section, the models and methods used in the thesis will be mentioned. After giving the quadrotor model and controller used in the study, the methods by which image measurements are obtained will be briefly mentioned. Next, the basics of Kalman Filter and Extended Kalman Filter adapted to nonlinear systems will be shown.

2.1 Quadrotor Dynamics

The quadrotor model and controller used in simulation were obtained from the study of Mishra [26].

We will begin with Newton Euler Equation which combines Euler's two laws of motion for a rigid body to a single matrix form equation. This equation consist of six coupled second-order differential equations for the position of the center of mass and for the angular orientation of the rigid body. Newton Euler Equation shows that change in the angular momentum about the center of mass is equal to the total moment of all the forces about the center of mass [27].

Newton Euler Equation for quadrotor as follows:

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} 1_3 m & 0_3 \\ 0_3 & I_3 \end{bmatrix} \begin{bmatrix} a \\ \alpha \end{bmatrix} + \begin{bmatrix} 0 \\ \omega \times I_3 \omega \end{bmatrix} \tag{2.1}$$

Where:
$\boldsymbol{F}$: Total force applied to the quadrotor.
$\boldsymbol{\tau}$: Total torque.
$\boldsymbol{m}$: Quadrotor mass.
$\boldsymbol{I}$: Moment of inertia.
$\boldsymbol{\omega}$: Angular velocity.
$\boldsymbol{a}$: Linear acceleration.

$\boldsymbol{\alpha}$: Angular acceleration.

The rotation matrix $\boldsymbol{R}$ transforms the body coordinates to the world cooridantes. The rotation matrix represents a rotation by an angle about a fixed axis that lies along the unit vector.

$$ {}^{W}R_{B} = \begin{bmatrix} c(\psi)c(\theta) - s(\phi)s(\psi)s(\theta) & -c(\phi)s(\psi) & c(\psi)s(\theta) + c(\theta)s(\phi)s(\psi) \\ c(\theta)s(\psi) + s(\phi)c(\psi)s(\theta) & c(\phi)c(\psi) & s(\psi)s(\theta) - c(\theta)s(\phi)c(\psi) \\ -c(\phi)s(\theta) & s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{2.2} $$

Where :
$\boldsymbol{\phi}$: Roll angle (Angle of rotation about x axis).
$\boldsymbol{\theta}$: Pitch angle (Angle of rotation about y axis).
$\boldsymbol{\psi}$: Yaw angle (Angle of rotation about z axis).
$c(\cdot)$ and $s(\cdot)$ stands for *cos(·)* and *sin(·)*.

Angular velocity components of the robot in the body frame defined as:

$$ {}^{W}\omega_{B} = p\mathbf{x_B} + q\mathbf{y_B} + r\mathbf{z_B} \tag{2.3} $$

Where **p**,**q** and **r** are angular velocities. These angular velocities are related to the derivatives of roll, pitch and yaw angles with following equation:

$$ \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & -cos(\phi)sin(\theta) \\ 0 & 1 & sin(\phi) \\ sin\theta & 0 & cos(\phi)cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.4} $$

Linear motion equation of the quadrotor in the world frame $x_W y_W z_W$:

$$ m\ddot{l} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + {}^{W}R_{B} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{2.5} $$

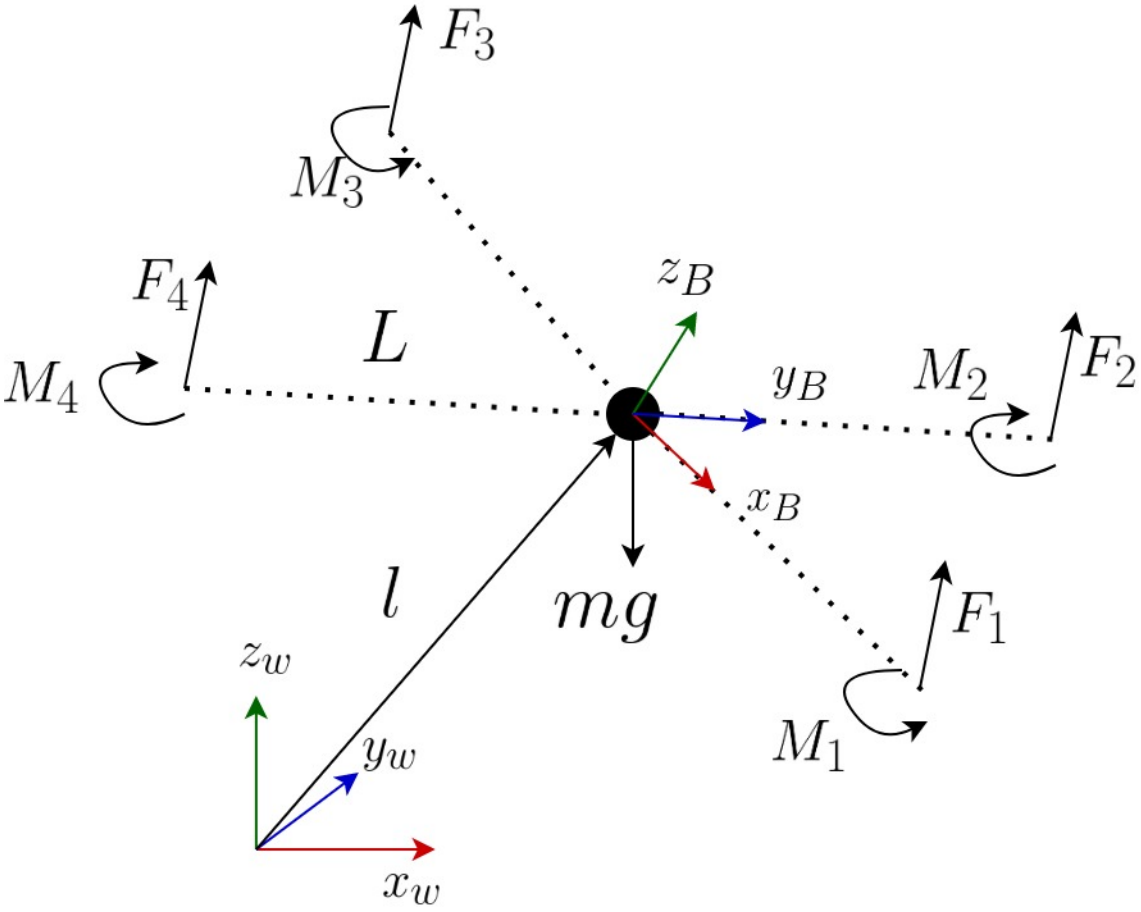

Figure 2.1 Free body diagram of quadrotor

Angular motion equation of the quadrotor in the body frame $x_B y_B z_B$:

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.6}$$

2.2 Quadrotor Control

PD controller is used for quadrotor control. Dynamic equations for the quadrotor are:

$$m\ddot{l} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ u_1 \end{bmatrix} \tag{2.7}$$

Where $\mathbf{u_1}$ is the thrust input.

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = u_2 - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.8}$$

Where $\mathbf{u_2}$ is the moment input. From (2.4)

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ sin(\theta)tan(\phi) & 1 & -cos(\theta)tan(\phi) \\ -\frac{sin(\theta)}{cos(\phi)} & 0 & \frac{cos(\theta)}{cos(\phi)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.9}$$

Desired trajectory defined as

$$l_T = \begin{bmatrix} x_{des} \\ y_{des} \\ z_{des} \\ \psi_{des} \end{bmatrix} \tag{2.10}$$

If we define position error as:

$$e_p = l_T - l \tag{2.11}$$

Similarly, velocity error as:

$$e_v = \dot{l}_T - \dot{l} \tag{2.12}$$

Sufficient condition $k > 0$ to guarantee that error goes exponentially to zero is:

$$\ddot{l}_T - \ddot{l}_c + k_{d,l} e_v + k_{p,l} e_p = 0 \tag{2.13}$$

Where $\ddot{\mathbf{r}}_\mathbf{c}$ is the command acceleration, calculated by the PD controller. If we assume approximations below, to linearize the dynamics at the hover configuration:

$$u_1 \approx mg$$
$$\theta \approx 0$$
$$\phi \approx 0$$
$$\psi \approx \psi_0$$
$$u_2 \approx 0$$
$$p \approx 0$$
$$q \approx 0$$
$$r \approx 0$$

We get following equations

$$\phi_c = \frac{1}{g}(\ddot{l}_{1,c} sin(\psi_{des}) - \ddot{l}_{2,c} cos(\psi_{des})) \tag{2.14}$$

$$\theta_c = \frac{1}{g}(\ddot{l}_{1,c} cos(\psi_{des}) - \ddot{l}_{2,c} sin(\psi_{des})) \tag{2.15}$$

$$\psi_c = \psi_{des} \tag{2.16}$$

$$\ddot{l}_{3,c} = \ddot{l}_{3,des} + k_{d,3}(\dot{l}_{3,des} - \dot{l}_3) + k_{p,3}(l_{3,des} - l_3) \tag{2.17}$$

Control laws can be written as

$$u_1 = m(g + \ddot{l}_{3,c}) \tag{2.18}$$

$$u_2 = I \begin{bmatrix} k_{p,\phi}(\phi_{des} - \phi) + k_{d,\phi}(p_{des} - p) \\ k_{p,\theta}(\theta_{des} - \theta) + k_{d,\theta}(q_{des} - q) \\ k_{p,\psi}(\psi_{des} - \psi) + k_{d,\psi}(r_{des} - r) \end{bmatrix} \tag{2.19}$$

2.3 3D Reconstruction of Cameras

In this section, some methods[28] used to determine the relative positions of two cameras with intersecting field of view will be discussed.

Many visual odometry and SLAM applications have been made by calculating the relative positions of the camera from the intersecting field of view. In visual odometry applications, the camera movement is calculated by comparing the previous image of the moving camera with the current image. Thus, the movement and position of the agent to which the camera is attached is determined [10, 29–31]. In SLAM applications, while 3D mapping, the position of the camera according to the starting position is also determined [11, 32, 33]. In this study, the method is used to calculate the relative positions of multiple cameras connected to multiple agents in real time, instead of calculating the change in the motion of the camera connected to an agent.

2.3.1 Relative Positions and Orientations of Two Cameras

Suppose there are 2 cameras C_0 and C_1 seeing the same P point from different positions and angles. See Figure 2.2

If we draw beams from P to cameras C_0 and C_1, the projection of the P point on the camera planes will be the points p_0 and p_1. These two cameras and the point P form the C_0C_1P epipolar plane. The line connecting the two cameras is called C_0C_1 base

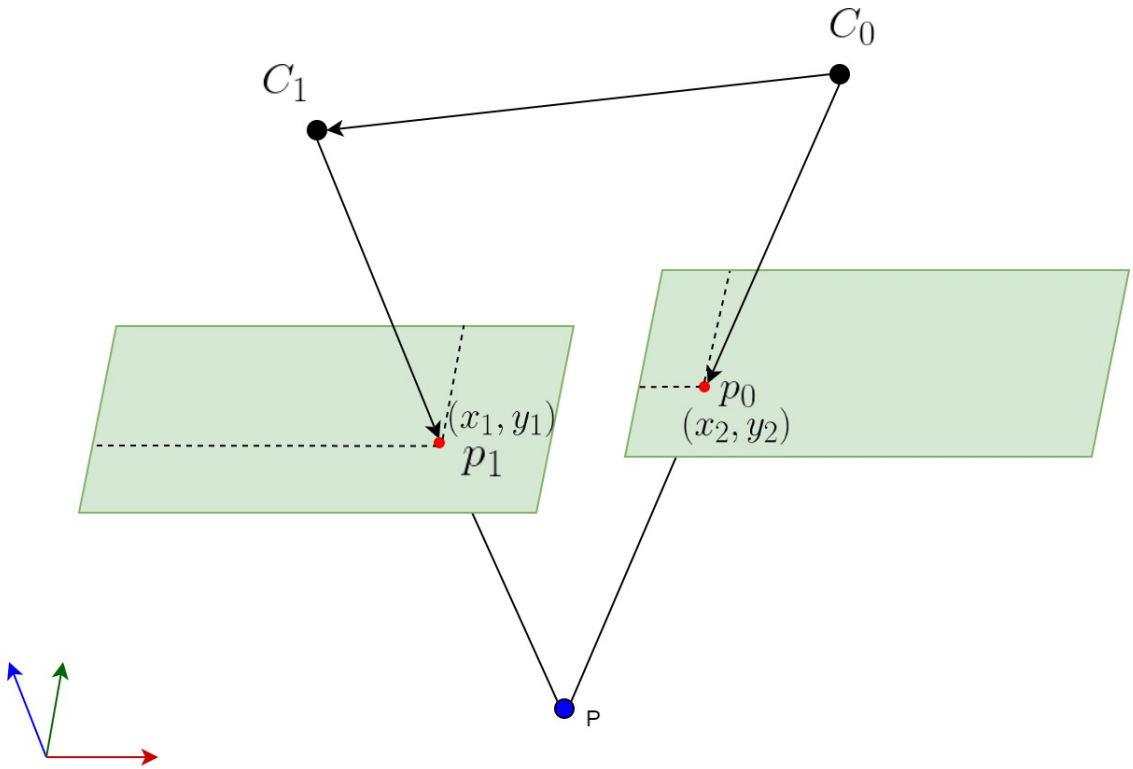

Figure 2.2 Epipolar geometry

line. In epipolar geometry, the vectors $\vec{C_0C_1}$, $\vec{C_0p_0}$ and $\vec{C_1p_1}$ are on the same plane. Based on this, we can write the following relation:

$$\vec{C_0p_0} \cdot (\vec{C_0C_1} \times \vec{C_1p_1}) = 0 \tag{2.20}$$

One could remember that cross product of two coplanar vectors results perpendicular vector to them. If we express the points p_0 ve p_1 as vectors:

p_0 is defined in the reference system C_0

$$p_0 = \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \tag{2.21}$$

p_1 is defined in the reference system C_1

$$p_1 = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{2.22}$$

If we want to define the point p_1 in reference system C_0, we can use the relation Rp_1.

So (2.20) can be written as follows:

$$p_0 \cdot (t \times Rp_1) = 0 \tag{2.23}$$

With t:translation matrix from C_0 to C_1 and R: rotation matrix from C_0 to C_1.
$t \times R_{p_1}$ can be represented as $[t]_\times Rp_1$. Therefore (2.23) can be rewritten as:

$$p_0^T [t]_\times Rp_1 = 0 \tag{2.24}$$

Where $[t]_\times$ is skew-symmetric matrix that:

$$[t]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{2.25}$$

Let define $E = [t]_\times R$ called essential matrix which is a 3×3 matrix. Then (2.24) becomes:

$$p_0^T E p_1 = 0 \tag{2.26}$$

Essential matrix relates the image of a point in camera C_0 to its image in camera C_1, given a translation and rotation. Our goal is to find the essential matrix elements and get the translation matrix **t** and rotation matrix **R**.

Essential matrix is a 3×3 matrix with 9 unknowns. So we need at least 8 equations to calculate the essential matrix [34]. These equations are obtained from known points on two camera planes.

If we expand (2.26):

$$\begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = 0 \tag{2.27}$$

Expanding (2.27) again :

$$E_{11}x_0x_1 + E_{12}x_0y_1 + E_{13}x_0 + E_{21}y_0x_1 + E_{22}y_0y_1 + E_{13}y_0 + E_{31}x_1 + E_{32}y_1 + E_{33} = 0 \tag{2.28}$$

Finally, if it is written in the matrix form:

$$\begin{bmatrix} x_0x_1 & x_0y_1 & x_0 & y_0x_1 & y_0y_1 & y_0 & x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} E_{11} \\ E_{12} \\ E_{13} \\ E_{21} \\ E_{22} \\ E_{23} \\ E_{31} \\ E_{32} \\ E_{33} \end{bmatrix} = 0 \tag{2.29}$$

And having multiple known points enable us to obtain matrix equation $Ax = 0$ to solve this equation, the A matrix must consist of at least 8 rows.
We will use Singular Value Decomposition to find solution:

$$A = UDV^T \tag{2.30}$$

The solution x is the column of V corresponding to the only the null singular value of A. This is the rightmost column of V.

One should note that the translation matrix obtained here is a unit vector.

2.3.2 Finding Points on Images

Since we cannot manually select the points that need to be determined in order to calculate the essential matrix in automated systems, we need methods to extract them from the picture.Camera relative attitude maybe extracted from the essential matrix. To obtain such common points maybe achived through special points. These methods are called feature extraction. Features are interesting parts of images and generally they are edges and corners. A good feature detection algorithm should not be affected by rotation, scale and illumination changes. Some of the most recent and popular algorithms will be mentioning in the following.

In Figure 2.3, the features in the images taken from the sub-cameras of two UAVs in different orientations and positions were extracted and matched.

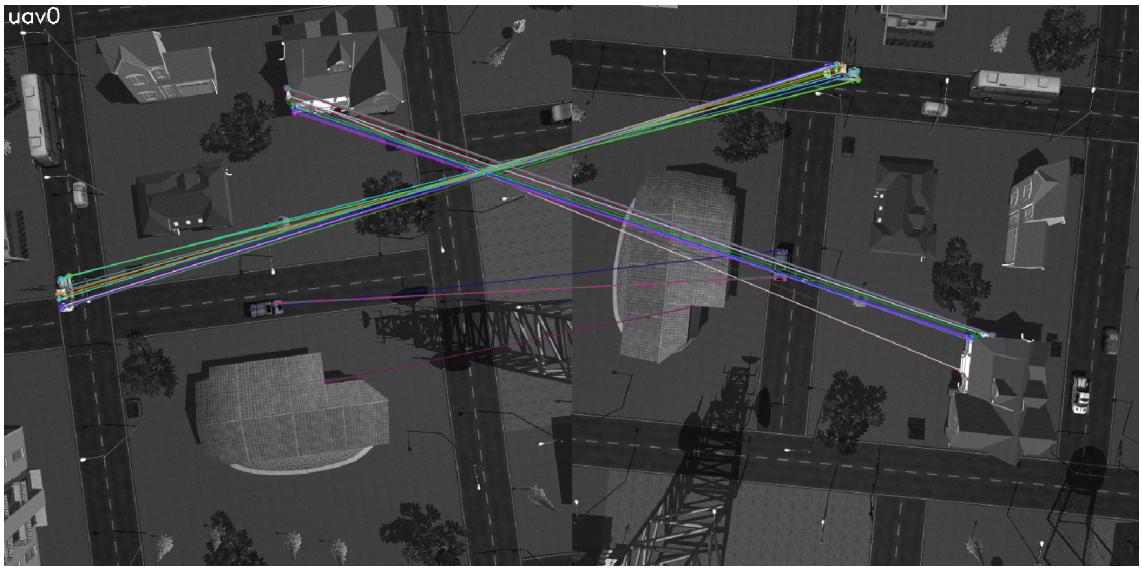

Figure 2.3 Features are extracted by ORB and matched with Brute-Force algorithm

2.3.3 Harris Corner Detector

Harris Corner Detector as it is said, a mathematical method to find the corners in images. It was introduced by Chris Harris and Mike Stephens in 1988[3]. Harris Corner Detector popularity comes from its independence of scale, rotation and illumination variations.

In Figure 2.4 1, 2, 3 squares are called kernels which are used to determine corners by moving them all directions. If we move 1. kernel on the flat surface any direction there will be no gradient changes. So it can't be specify any interesting point on the image and can't repeatedly refer to same point. Also if we move 2. kernel along the line there will be no gradient change and it is not useful. If we move 3. kernel any direction on the image there will be significant gradient change occur. Harris Corner Detector scans the image this way to determine corners.

This operation can be written mathematically as

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y,v) - I(x,y)]^2 \tag{2.31}$$

Where
$\mathbf{E(u,v)}$: Differences produced by shifting (u,v).
$\mathbf{u}$: Kernel displacement in x direction.
$\mathbf{v}$: Kernel displacement in y direction.
$\mathbf{w(x,y)}$: Kernel at (x,y) position.
$\mathbf{I(x,y)}$: Intensity of the image.

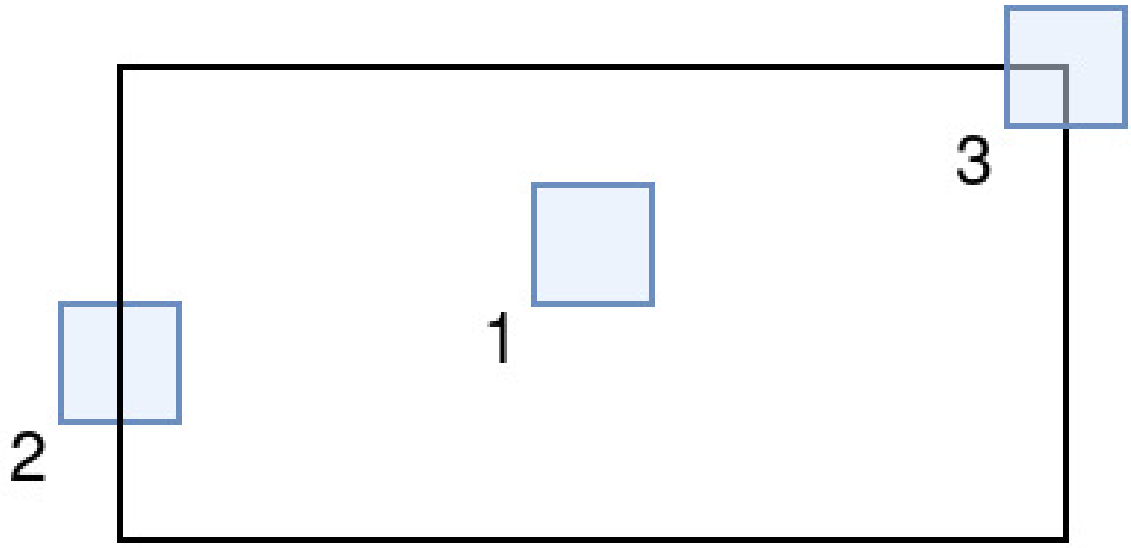

Figure 2.4 Harris Corner Detection

To find corners we should maximize the function $E(x,y)$. This is done by maximizing the term

$$\sum_{x,y}[I(x+u,y,v)-I(x,y)]^2 \tag{2.32}$$

To do that we will be expanding the term above with Taylor Series and ignore the higher order terms

$$E(u,v) \approx \sum_{x,y}[I(x,y)+uI_x+vI_y-I(x,y)]^2 \tag{2.33}$$

With I_x and I_y are gradient in x and y. If we expand square

$$E(u,v) \approx \sum_{x,y} u^2I_x^2+2uvI_xI_y+v^2I_y^2 \tag{2.34}$$

We will write above equation matrix form

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \sum \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.35}$$

Now, we define A for simplicity as

$$A = \sum w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \tag{2.36}$$

(2.35) becomes

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} A \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.37}$$

For each kernel we will find the eigenvalues of matrix A

$$\begin{aligned} R &= \det A - k(\text{trace}A)^2 \\ \det A &= \lambda_1 \lambda_2 \\ \text{trace}A &= \lambda_1 + \lambda_2 \end{aligned}$$

Using eigenvalues λ_1, λ_2 and R we can decide whether kernel in a region of corner, edge or flat. If both λ_1 and λ_2 are small kernel is in a flat region. λ_1 or λ_2 significantly large kernel is in an edge. If λ_1 and λ_2 has large values kernel is in a corner.

2.3.4 SIFT (Scale-Invariant Feature Transform)

The algorithm proposed by D.Lowe [5] in 2004 which extract keypoints from image and computes its descriptors. To do that, scale-space is generating with a function $L(x, y, \sigma)$. Scale-space consist of different scales of the original image. Each scaled image progressively blured with Gaussian Blur operator. For each pixel following operation applied

$$L(x, y, \sigma) = G(x, y, \sigma) \circledast I(x, y) \tag{2.38}$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2.39}$$

Where
G : Gaussian Blur operator.
I : Image itself.
σ : Scale factor.

Difference of Gaussians (DoG) is generating using these scaled and blurred images differences. A representation of DoG can be seen below

After generating DoG all images are scanned to find local extrema. A pixel is compared with 8 pixels in its neighbourhood and 9 pixels in the previous and next scale. If this pixel is local extrema, it means it is possible keypoint. The keypoints found are eliminated by a threshold value and the best matching ones remain. In order not to be affected by orientation changes, orientation is assigned to keypoints. For this, 36

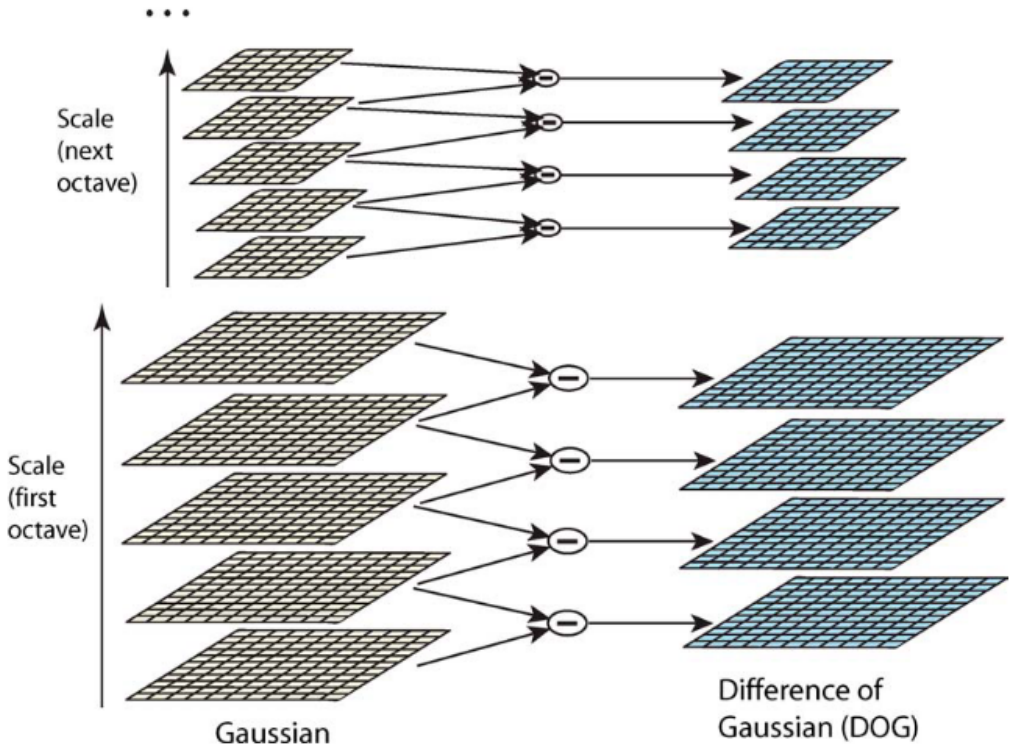

Figure 2.5 Difference of Gaussians[5]

bins orientation histogram covering 360 degrees around the keypoint is created. The highest value and over % 80 values in the created histogram are using to calculate the orientation. Descriptors for keypoints will be calculated after assigning the position, scale and orientation of each keypoint extracted from the image. By creating a 16x16 window around each keypoint, this window is divided into 16 sub-blocks of 4x4 size. All subblocks have an orientation histogram of 8 bin. Thus, 128 bin values are obtained.

2.3.5 ORB (Oriented FAST and Rotated BRIEF)

Oriented FAST and Rotated BRIEF algorithm proposed by Ethan Rublee in 2011 [9]. It is alternative to SIFT or SUFT which are patented and not free to use for everyone. ORB uses FAST[35] for keypoint detector and BRIEF[8] as descriptor. There are modifications were made on these algorithms to improve performance and to make it orientation invariance.

To achieve scale invariance, the ORB algorithm uses an image pyramid with a reduced resolution at each level of the original image. It determines keypoints on each scale using the FAST algorithm. Then it determines orientations by detecting intensity changes using the intensity centroid.

BRIEF algorithm converts keypoints detected with FAST into binary vector form. The BRIEF algorithm selects a random pair of pixels, called a patch, around the keypoint. One of the pixels is taken from Gaussian Distribution with standard deviation sigma around the keypoint.The other is taken in the same way as two sigma. If the first selected pixel is bigger than the other, a value of 1 is assigned.

Since there is no orientation in the BRIEF algorithm, ORB offers the rBRIEF method. ORB scans all patches for this, and creates the vector T by testing each patch according to their distance to mean of 0.5. It then looks for all possible tests with high variance and means close to 0.5.

2.3.6 Matching Detected Features

After the features are extracted from the images, these features that are in the same position in the two images must be matched. Thus, points can be detected and the relative position between cameras can be calculated as in the Section 2.3.1.

2.3.7 Brute-Force Matcher

The simple Brute-Force Matcher algorithm tries to determine the features that are the same in two images by calculating the distance between features. For this, it calculates the distance between all the features in the second image with a distance calculation function for all the features extracted from the first image. This distance function can be calculated with the following methods:

Sum of Squared Differences (SSD):

$$d(f_i, f_j) = \sum_{k=1}^{n} (f_{i,k} - f_{j,k})^2 \tag{2.40}$$

Sum of Absolute Differences (SAD):

$$d(f_i, f_j) = \sum_{k=1}^{n} |f_{i,k} - f_{j,k}|^2 \tag{2.41}$$

Here, the distance between the feature f_i in the first image and all the features in the second image is calculated and the feature f_j with the minimum distance between them is matched. In case the feature f_i in the first image is not found in the second image, a σ threshold value is defined. Matching is made if the distance between features is less than $sigma$ threshold.

2.3.8 FLANN Based Matcher

FLANN proposed by Muja and Lowe in their paper Fast Approximate Nearest Neighbours With Automatic Algorithm Configuration[36]. FLANN is a library that contains many algorithms. This library automatically chooses the best algorithm and optimum parameters according to the data provided. This library which developed for large datasets and high-size features, gives better results than Brute-Force Matcher in large datasets.

2.4 Kalman Filter

Kalman filter used in many systems [37] allows to predict unmeasurable states from inaccurate and uncertain noisy measurements. It is an algorithm consists set of equations that allow the process to predict future states form the previous state. The algorithm consists of two steps and these steps are repeated continuously. In the Prediction step, the future state of the system is estimated together with the model and uncertainty of the system. When each measurement arrives, in the second step which is correction, the states are predicted in the previous step are updated using weighted average. Here the weight depends on the accuracy of the model or measurement. In this section, Kalman Filter will be briefly mentioned. Later, extended Kalman filter, which is adapted to nonlinear systems, will be mentioned. One can found more information about kalman filter in [38]

2.4.1 System Model

The system estimating with Kalman Filter is defined by the following linear difference equation:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.42}$$

Where

$\mathbf{x} \in \mathbb{R}^a$: State vector.
$\mathbf{u} \in \mathbb{R}^p$: Input vector.
$\mathbf{w}$: Process noise.
$\mathbf{A}$: $a \times a$ System matrix.
$\mathbf{B}$: $a \times c$ Input matrix.

The expression (2.42) represents the model of the system. Kalman Filter uses this

model to make predictions in the next step. In practice it is not possible to know the w system noise. This noise indicates the change of the system over time due to external factors.

The measurement equation is as follows

$$z_k = Hx_k + n_k \tag{2.43}$$

Where

$\mathbf{z} \in \mathbb{R}^b$: Measurement vector.
$\mathbf{H}$: $b \times a$ Measurement matrix.
$\mathbf{n}$: Measurement noise.

The random variables w_k and n_k assumed to be zero-mean Gaussian with the covariances Q and R as

$$p(w) \sim \mathcal{N}(0, Q) \tag{2.44}$$

$$p(n) \sim \mathcal{N}(0, R) \tag{2.45}$$

Where

$\mathbf{Q}$: Process noise covariance.
$\mathbf{R}$: Measurement noise covariance.

We use process covariance matrix $\mathbf{Q}$ because estimation process is not exact. In practice, predicted states and real states may not be equal. This situation is caused by noise and other external factors. Measurement covariance matrix $\mathbf{R}$ represents errors in the measurement. There may be a difference between the measured states and the actual states when making measurements. This situation is caused by uncertainties in the sensors and external factors.

We will call some variables *priori* i.e $\hat{x}^-$ with "super minus" and *posteriori* i.e $\hat{x}$ without "super minus" and $\hat{x}$ with "hat" means estimated variable.
We define P_k error covariance matrix at time step k as

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \tag{2.46}$$

2.4.2 Prediction Step

In this step *priori* state vector and error covariance matrix will be estimated. System model (2.42) will be used to estimate the *priori* state vector. The *priori* state vector is the prediction made using only the system model before the actual prediction made in the correction step.*priori* state is calculated as follows:

$$\hat{x}_k^- = Ax_{k-1} + Bu_{k-1} \tag{2.47}$$

The *priori* error covariance matrix representing errors in state estimation will be used to calculate Kalman gain K_k in the correction step. *priori* error covariance matrix is calculated as follows:

$$P_k^- = AP_{k-1}A^T + Q \tag{2.48}$$

Prediction step will be executed until measurement arrives.

2.4.3 Correction step

This step begin with calculating the Kalman Gain K_k :

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.49}$$

After calculating the Kalman gain, *posteriori* state is updated

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.50}$$

The Kalman gain actually weights the term $(z_k - H\hat{x}_k^-)$ known as *innovation* or *measurement residual* in (2.50).

If (2.49) is examined, the measurement error covariance matrix R approaches zero while your kalman gain weighs more on *measurement residual*. Thus, the estimator will be more confident in the incoming measurements.

$$\lim_{R_k \to 0} K_k = H^{-1} \tag{2.51}$$

On the other hand, *priori* estimated error covariance P_k^- approaching zero while your

kalman gain K will weighs less the *measurement residual*.

$$\lim_{P_k^- \to 0} K_k = 0 \tag{2.52}$$

Finally *posterierirori* error covariance P_k is updated in the correction step.

$$P_k = (I - K_k H) P_k^- \tag{2.53}$$

2.5 Extended Kalman Filter

While Kalman filter enables the estimation in linear systems, extended Kalman filter is used in systems where the system or measurement is nonlinear. For this, the system or measurement function is linearized at the current state of the system. Since EKF will be applied to nonlinear systems, instead of linear system and measurement in (2.42) and (2.43), the nonlinear system function is as follows:

$$x_k = f(x_{k-1}, w_{k-1}) \tag{2.54}$$

And nonlinear measurement function is:

$$z_k = h(x_k, n_k) \tag{2.55}$$

Since it is not possible to know of w_k and n_k noise values at all time steps in practice, we will use following approximations:

$$\tilde{x}_k = f(\hat{x}_{k-1}, 0) \tag{2.56}$$

and

$$\tilde{z}_k = h(\tilde{x}_k, 0) \tag{2.57}$$

To linearize the system function we will expand the function $f(.)$ in Taylor Series about $\tilde{x}_{k-1}$

$$f(x_{k-1}) \equiv f(\hat{x}_{k-1}) + A(x_{k-1} - \hat{x}_{k-1}) + \text{H.O.T} \tag{2.58}$$

where A is the jacobian of function $f(.)$ with respect to x which is

$$A = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_a} \\ \vdots & & & \vdots \\ \frac{\partial f_a}{\partial x_1} & \frac{\partial f_a}{\partial x_2} & \cdots & \frac{\partial f_a}{\partial x_a} \end{bmatrix} \tag{2.59}$$

and W is jacobian of $f(.)$ with respect to w

$$W = \frac{\partial f}{\partial w} = \begin{bmatrix} \frac{\partial f_1}{\partial w_1} & \frac{\partial f_1}{\partial w_2} & \cdots & \frac{\partial f_1}{\partial w_i} \\ \vdots & & & \vdots \\ \frac{\partial f_i}{\partial w_1} & \frac{\partial f_i}{\partial w_2} & \cdots & \frac{\partial f_i}{\partial w_i} \end{bmatrix} \tag{2.60}$$

If we neglect the higher order terms (H.O.T), the final equation becomes

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + W_{w_{k-1}} \tag{2.61}$$

Next we will linearize the measurement function $h(.)$ by expanding Tylor Series about $\tilde{x}_k$

$$h(x_k) \equiv \tilde{z}_k + H(x_k - \hat{x}_{x-1}) + V_{v_k} + \text{H.O.T} \tag{2.62}$$

where H is the jacobian of function $h(.)$ with respect to x which is

$$H = \frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_a} \\ \vdots & & & \vdots \\ \frac{\partial h_n}{\partial x_1} & \frac{\partial h_a}{\partial x_2} & \cdots & \frac{\partial h_a}{\partial x_a} \end{bmatrix} \tag{2.63}$$

and V is jacobian of $f(.)$ with respect to v

$$V = \frac{\partial f}{\partial v} = \begin{bmatrix} \frac{\partial f_1}{\partial v_1} & \frac{\partial f_1}{\partial v_2} & \cdots & \frac{\partial f_1}{\partial v_j} \\ \vdots & & & \vdots \\ \frac{\partial f_j}{\partial v_1} & \frac{\partial f_j}{\partial v_2} & \cdots & \frac{\partial f_j}{\partial v_j} \end{bmatrix} \tag{2.64}$$

Neglecting higher order terms (H.O.T) resulting following equation

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + V_{v_k} \tag{2.65}$$

Similar to Section 2.4 prediction and correction steps will be executed.

2.5.1 Prediction Step

Using (2.56) we will estimate the *priori* state and *priori* error covariance matrix

$$\hat{x}_k^- = f(\hat{x}_{k-1}, 0) \tag{2.66}$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{2.67}$$

2.5.2 Correction Step

As Section 2.4 in correction step we will calculate the kalman gain

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \tag{2.68}$$

Then update the estimated state while weighting the *measurement residual* with calculated kalman gain K_k

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-, 0)) \tag{2.69}$$

Finally update the process error covariance matrix

$$P_k = (I - K_k H_k) P_k^- \tag{2.70}$$

To maintain the symmetry of P_k is preferred for computational considerations.

3
RELATIVE POSITION ESTIMATION

In order to create a swarm and maintain formation, the positions of unmanned aerial vehicles in central or decentralized controllers must be known relative to each other or to a fixed reference. GPS, which provides positioning with respect to a fixed reference, is widely used in unmanned aerial vehicles. However, it is insufficient for some applications due to the low refresh rate of GPS and low accuracy in motion. Systems such as VICON, on the other hand, allow working in a very limited area. In this section, the proposed method for determining the relative positions of swarm members consisting of unmanned aerial vehicles will be mentioned by fusing only IMU and camera sensors.

3.1 Problem Statement

Agents sends camera images and IMU measurements to each other to calculate their position relative to other agents to which they are connected. The relative position between the two cameras is calculated from the intersecting areas of the image taken from its own image and its connected agents. However, since cameras projects 3 dimensions to 2 dimensions, 1 dimension, i.e. depth information, is lost. Therefore, the relative position from image measurements is a unit vector. The scale factor must be known to determine the actual position. IMU measurements are used to determine the scale factor. Scale ambiguity problem which vision measurement suffered eliminated by the IMU measurements and error accumulation problem arising from IMU measurements is eliminated by image measurements. This way, the two sensors eliminate each other's weaknesses and allow accurate position estimation.

An example of the system is shown in Figure 3.1. Here, agents i, j and k take IMU measurements and camera images of the agents they are following. All agents send their own accelerometer sensor measurements and camera data to the agents they are connected to in the formation. Using the intersecting field of view and relative acceleration, agents calculate the distance between the agents to which they are

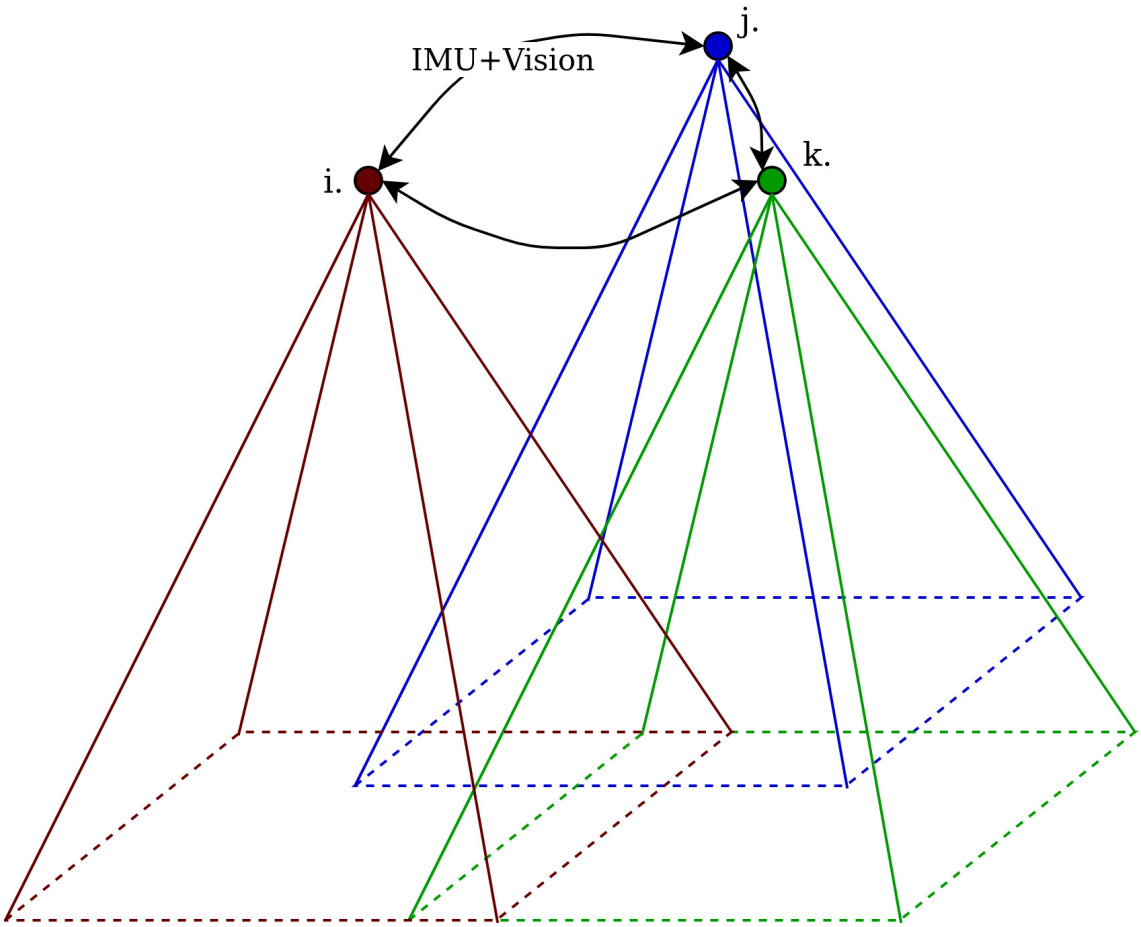

Figure 3.1 System representation

connected.

The swarm formation can take many different forms than the simple representation in Figure 3.1. Apart from the formation formed only in 2 dimensional plane, different formations such as 3 dimensional sphere, cube, pyramid can be easily created as long as the image areas of interconnected agents intersect. As the flight height of the swarm increases, the view areas will increase, so larger volumes of swarm can be created.

3.2 System Model

We will use a discrete time kinematic model to define the system. Scale factor λ added to the model for absolute scale estimation[39]. The scale factor λ will be continuously updated by Kalman Filter with incoming measurements.

We define the system model consisting of the relative position, velocity, acceleration

and scale factor lambda of the agents as follows:

$$x_k = \begin{bmatrix} \vec{p}_k \\ \vec{v}_k \\ \vec{a}_k \\ \lambda_k \end{bmatrix} = \begin{bmatrix} \mathbf{1_3} & T\mathbf{1_3} & \frac{T^2}{2}\mathbf{1_3} & 0 \\ \mathbf{0_3} & \mathbf{1_3} & T\mathbf{I_3} & 0 \\ \mathbf{0_3} & \mathbf{0_3} & \mathbf{1_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{p}_{k-1} \\ \vec{v}_{k-1} \\ \vec{a}_{k-1} \\ \lambda_{k-1} \end{bmatrix} \tag{3.1}$$

The following notation will be used in the rest of the work:

$\mathbf{1_n}$: n×n Identity matrix
$\mathbf{0_n}$: n×n Zero matrix

System states are:

${}^i_j\vec{p}_k$: i. agent position relative to j. agent.
${}^i_j\vec{v}_k$: i. agent velocity relative to j. agent.
${}^i_j\vec{a}_k = \vec{a}_j - \vec{a}_i$: i. agent acceleration relative to j. agent.
${}^i_j\lambda_k$:Scale factor of i. and j. agents.

To simplify the representation, $\mathbf{x_k}$ will be used instead of ${}^{\mathbf{i}}_{\mathbf{j}}\mathbf{x_k}$.

Nonlinear difference equations of the system:

$$x_k = f(x_{k-1}, w_{k-1}) \tag{3.2}$$

$$z_k = h(x_k, n_k) \tag{3.3}$$

Where:

$\mathbf{x} \in \mathbb{R}^{\mathbf{10\times 1}}$: State vector.
$\mathbf{w} \in \mathbb{R}^{\mathbf{10\times 1}}$: Process noise vector.
$\mathbf{z} \in \mathbb{R}^{\mathbf{3\times 1}}$: Measurement vector.
$\mathbf{n} \in \mathbb{R}^{\mathbf{3\times 1}}$: Measurement noise vector.

Agents keep in their memory (3.2) and (3.3) state and measurement vectors for each

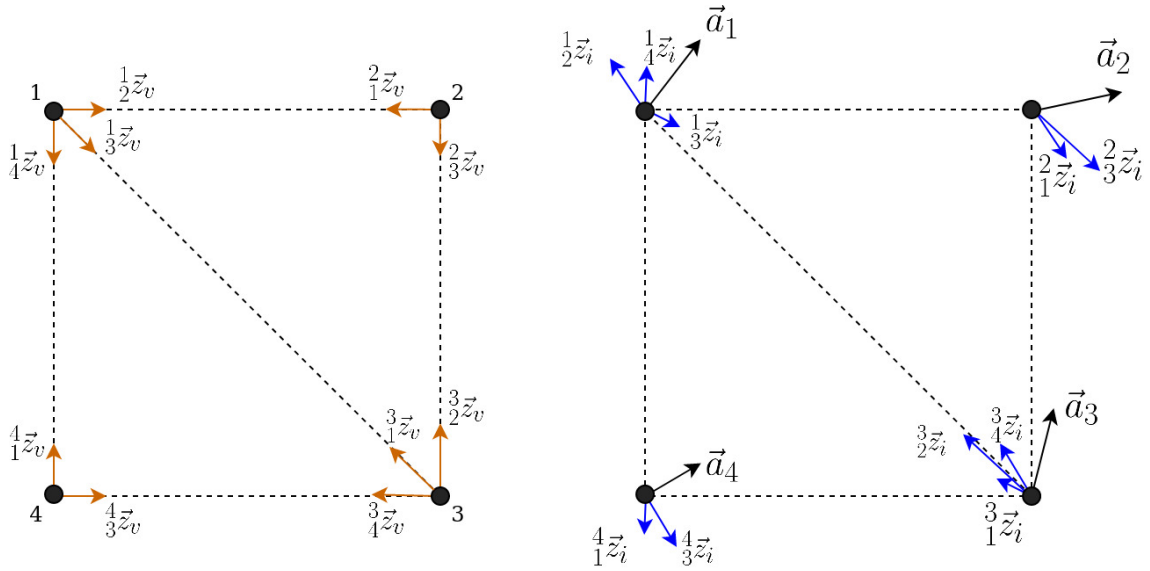

Figure 3.2 Agents state and measurement vectors

agent they connected. For example, if we examine the Figure 3.2, agent 1 keeps ${}^{1}_{2}\vec{x}$, ${}^{1}_{3}\vec{x}$, ${}^{1}_{4}\vec{x}$ systems in its memory for each agent in the set of its connected agents $\{2,3,4\}$ and continuously update the measurements ${}^{1}_{2}\vec{z}_v$, ${}^{1}_{3}\vec{z}_v$,${}^{1}_{4}\vec{z}_v$, ${}^{1}_{2}\vec{z}_i$, ${}^{1}_{3}\vec{z}_i$, ${}^{1}_{4}\vec{z}_i$.

3.3 Vision Measurements

Image measurements are obtained from the intersecting field of view of the two agents. To simplify models and calculations, the camera will be assumed to be at the center of agents.

In order to obtain image measurements, first, features are extracted from images taken from the sub cameras of both agents using one of or similar methods mentioned in Sections 2.3.3, 2.3.4, 2.3.5. Features extracted from both images are matched using one of or similar methods mentioned in Sections 2.3.7,2.3.8. Then we recover the relative position between cameras by calculating homograpy and essential matrix as mentioned in 2.3. It should be noted that the relative position between cameras obtained in this way is a unit vector and indicates only the direction. This can be seen on $\vec{z}_v$ measurement vectors on the left side of the Figure 3.2.

3.4 Relative Position Estimation With Extended Kalman Filter

In order to determine the relative positions in true scale, we will use Extended Kalman Filter [38], which is an adaptation of the famous Kalman Filter [37] for nonlinear systems. In Section 3.4 brief explanation and equations of Extended Kalman Filter can be found.Although our system function (3.2) that we will use in EKF is a linear

model, our measurement function (3.3) is nonlinear.

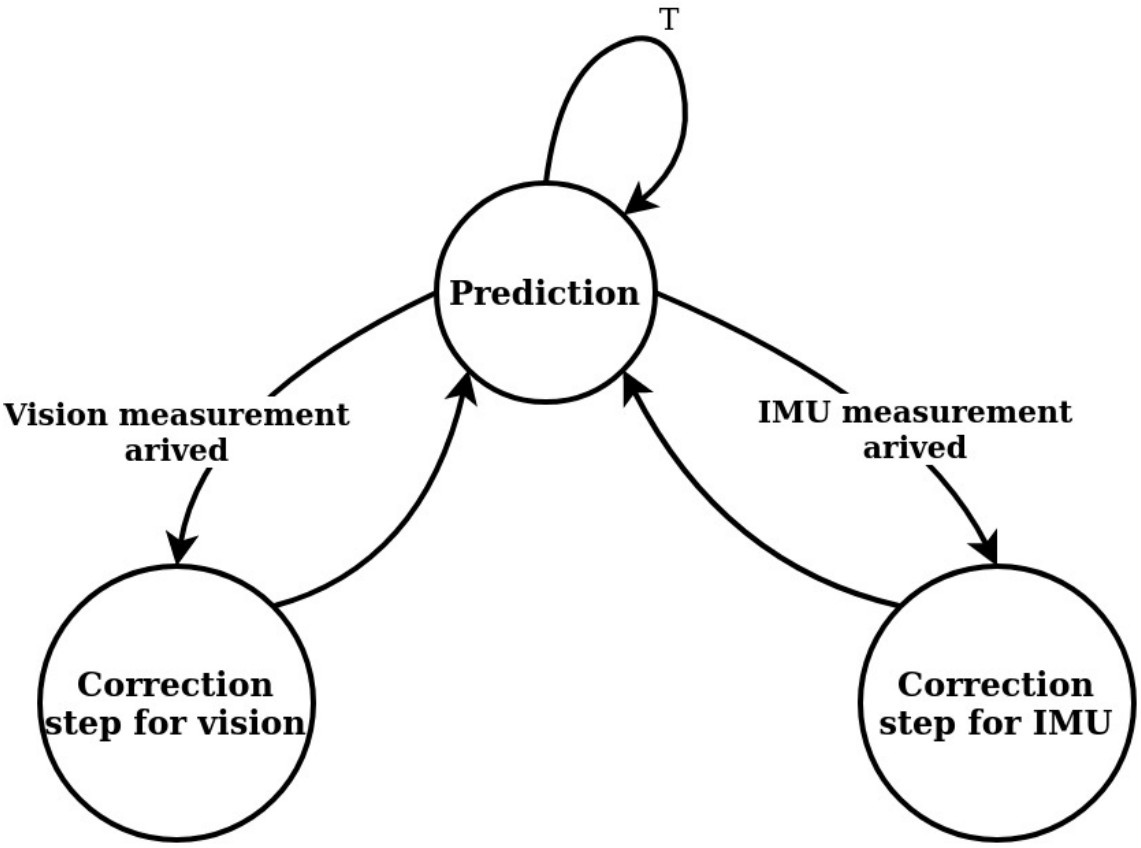

Figure 3.3 State diagram for Extended Kalman Filter

Each agent performs the steps in Figure 3.3 to determine its position relative to the agents to which it is connected with the Extended Kalman Filter. The agent predicts the states for next time step with the model (3.1) every T time. The prediction process continues until the vision or IMU measurement arrives. When any of the vision or IMU measurements arrive, predicted state is updated and the prediction process continues.

3.4.1 Prediction of States

In prediction step, using (3.1) the states of the system predicted at time step k using states at time step $k-1$. This step is repeated continuously until any of the vision or IMU measurements arrive.

Since the model is linear, the state prediction is made as follows:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, 0) \tag{3.4}$$

After the state is predicted, the error covariance matrix is calculated.

Error covariance matrix:

$$P_k^- = A_k P_{k-1} A_k^T + WQW^T \tag{3.5}$$

Where;
$\mathbf{W} = \frac{\partial f}{\partial w}$
$\mathbf{Q}$: Process noise covariance matrix
When calculating P_k^-, W and Q matrices are taken as constant.

Matlab implementation for prediction step :

```
function predict(T)
        % Priori state prediction
    x_ = [x(1:3)+x(4:6)*T+T^2/2*x(7:9);
                  x(4:6)+T*x(7:9);
                  x(7:9);
                  x(10)];

        % System matrix
    A = [eye(3) T*eye(3) T^2/2*eye(3) zeros(3,1);
         zeros(3) eye(3) T*eye(3) zeros(3,1);
         zeros(3,6) eye(3) zeros(3,1);
         zeros(1,9) 1];

        % Priori error covariance matrix
    P_ = A*P*A' + W*Q*.W';
end
```

3.4.2 Correction Step

Correction step will be calculated separately for each vision measurement and acceleration measurement. IMU measurement comes with higher rate than the vision, when the vision measurement arrives, the errors accumulated in the IMU measurements are corrected and the scale factor is updated.

For example, if we examine the position change of agent **j** relative to agent **i** from time step k to $k+5$, $\hat{\vec{p}}$ is updated with the IMU measurement $\vec{z}_i$ received each step, this noisy measurement causes an accumulation of errors up to time step $k+5$. The measurement of $\vec{z}_v$ from the vision at $k+5$ corrects these accumulated errors. If we examine it from the vision side, all of the incoming $\vec{z}_v$ measurements come in the form of a unit vector due to the use of a monocular camera and the scale factor λ must be known in order to determine the actual $\hat{\vec{p}}$ length. The λ is updated at time step $k+5$

because the incoming IMU measurements give the actual dimensions.

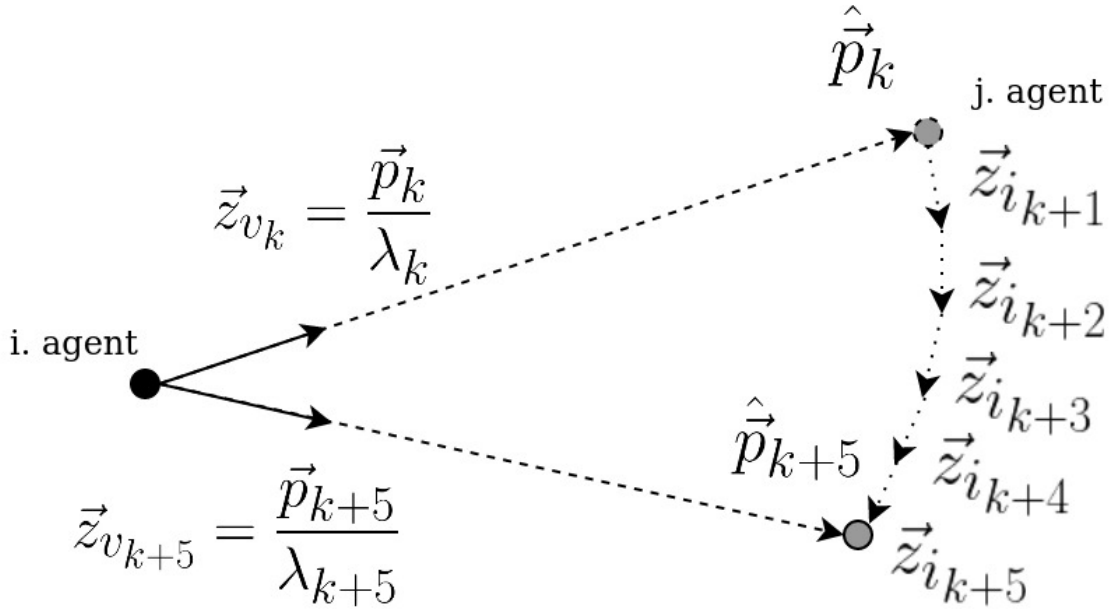

Figure 3.4 Measurement vectors

3.4.3 Correction Step for Vision Measurements

The nonlinear measurement function z_v will be used since the values from the vision measurements will come as an unscaled unit vector:

$$\hat{z}_{v_k} = h_v(\hat{\vec{x}}_k^{\,-}) \tag{3.6}$$

$$\hat{z}_{v_k} = \left[\tfrac{1}{\lambda}\mathbf{1_3}\right]\hat{\vec{p}}_k^{\,-} \tag{3.7}$$

To get the $\mathbf{H_v}$ matrix, we calculate the jacobian of the $\mathbf{h_v}$ function with respect to $\mathbf{x}$:

$$H_{v_k} = \frac{\partial h_v}{\partial x} = \left[\tfrac{1}{\lambda}\mathbf{1_3} \quad \mathbf{0_3} \quad \mathbf{0_3} - \tfrac{p}{\lambda^2}\right] \tag{3.8}$$

The kalman gain $\mathbf{K_v}$ is calculated for the image measurement:

$$K_{v_k} = P_k^- H_{v_k}^T (H_{v_k} P_k^- H_{v_k}^T + R_v)^{-1} \tag{3.9}$$

Where $R_v \in \mathbb{R}^{3\times3}$ is the image measurement covariance matrix. In the next, the

predicted states are updated:

$$\hat{x}_k = \hat{x}_k^- + K_{v_k}(z_{v_k} - h_v(\hat{\vec{x}}_k^-)) \tag{3.10}$$

Then the process covariance matrix is updated:

$$P_k = (\mathbf{I_{10}} - K_{v_k} H_{v_k}) P_k^- \tag{3.11}$$

Matlab implementation of correction step for vision measurements :

```
function update_vision(vision_measurements)
    Hv = [1/x_(10) 0 0 0 0 0 0 0 0 -x_(1)/x_(10)^2;
                  0 1/x_(10) 0 0 0 0 0 0 0 -x_(2)/x_(10)^2;
                  0 0 1/x_(10) 0 0 0 0 0 0 -x_(3)/x_(10)^2];

        % Update kalman gain
    Kv = P_*Hv'/(Hv*P_*Hv' + Rv);

    estimate = [x_(1)/x_(10);x_(2)/this.x_(10);x_(3)/x_(10)];

        % Make state correction
    correction = Kv * (vision_measurements - estimate);
    x = x_ + correction;

        % Update error covariance matrix
    P = (eye(10) - Kv*Hv)*P_;
end
```

3.4.4 Correction Step for IMU Measurements

h_i will be a linear function since the acceleration measurement directly gives the position vector $\vec{x}$. Thus H_i is:

$$H_i = \begin{bmatrix} \mathbf{0_3} & \mathbf{0_3} & \mathbf{1_3} & \mathbf{0_3} \end{bmatrix} \tag{3.12}$$

Kalman gain is calculated for this step:

$$K_{i_k} = P_k^- H_i^T (H_i P_k^- H_i^T + R_i)^{-1} \tag{3.13}$$

Where $R_i \in \mathbb{R}^{3\times3}$ is the acceleration measurement covariance matrix. In the next step, the states are updated:

$$\hat{x}_k = \hat{x}_k^- + K_{i_k}(z_{i_k} - H_i \hat{x}_k^-) \tag{3.14}$$

Process covariance is updated,

$$P_k = (\mathbf{1_{10}} - K_{i_k} H_i) P_k^- \tag{3.15}$$

In Figure 3.5, the state diagram of the equations can be seen for each connected agent.

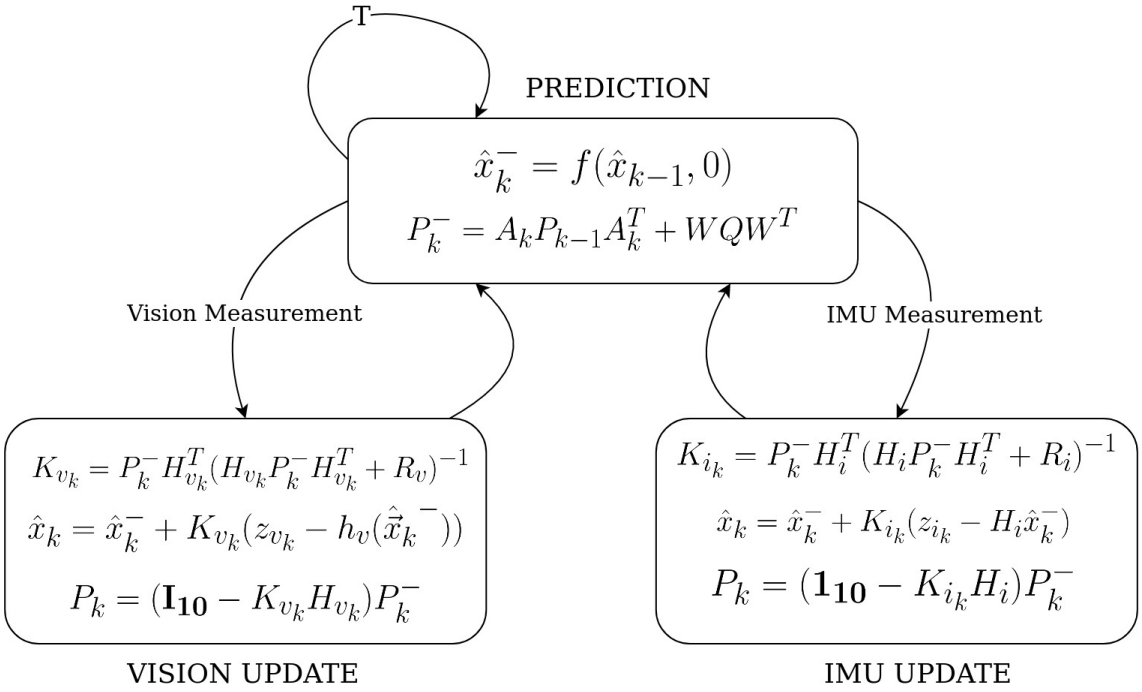

Figure 3.5 Kalman state diagram with equations

Matlab implementation of correction step for IMU measurements :

```
function update_imu(self_accelerationMeasurement,
    other_accelerationMeasurement)
    Hi = [zeros(3,6) eye(3) zeros(3,1)];

        % Calculate kalman gain
    Ki = P_*Hi'/(Hi*P_*Hi' + Ri);

        % Make state correction
    correction = Ki*(other_accelerationMeasurement -
        self_accelerationMeasurement - Hi*x_);
    x = x_+ correction;

        % Update error covariance matrix
    P = (eye(10) - Ki*Hi)*P_;
end
```

4
FORMATION CONTROL

The method described in the Section 3 has been used to create UAV swarms and maintain their position in the formation. With the formation to be created by the Graph Theorem, 3 different swarm movements have been studied. This section will show the methods used to create and maintain the formation.

4.1 Formation Properties

Cube-shaped formation consists of 9 agents. 8 agents are located at the corners of the cube and 1 agent is located in the center of the cube. One edge of the cube is 20 meters long. The formation and the connection between agents can be seen in Figure 4.1.

According to graph theory $G = (V, E) \in \mathbb{R}^3$, $n = 9$, $l = 26$ Graphs are defined by vertices and edges as $G(V, E)$ where V is a set of all vertices in the graph and E is the set of all the edges connecting the vertices by the pairs of $E = \{(v_i, v_j) : i, j \in V\}$. Any graph topology may be defined by $\{G(V, E), n, l\}$ with n the number of vertices and l the number of edges.
Set of nodes:

$$V = 1, 2, 3, 4, 5, 6, 7, 8, 9$$

Set of edges:

$$\begin{aligned} E = \{&(1,2),(1,4),(1,5),(1,6),(1,8),(1,9),(2,3),(2,4),(2,6),\\ &(2,9),(3,4),(3,6),(3,7),(3,8),(3,9),(4,8),(4,9),(5,6),\\ &(5,7),(5,8),(5,9),(6,7),(6,9),(7,8),(7,9),(8,9)\} \end{aligned}$$

The set of edges shows agents connected together. Desired distances:

$$d = \{20, 20, 20, 20\sqrt{2}, 20\sqrt{2}, 17.32, 20, 20\sqrt{2},$$
$$20, 17.32, 20, 20\sqrt{2}, 20, 20\sqrt{2}, 17.32, 20, 17.32, 20,$$
$$20\sqrt{2}, 20, 17.32, 20, 17.32, 20, 17.32, 17.32\}$$

The desired distances show the distances to be maintained in the connection between the agents given in the set E.

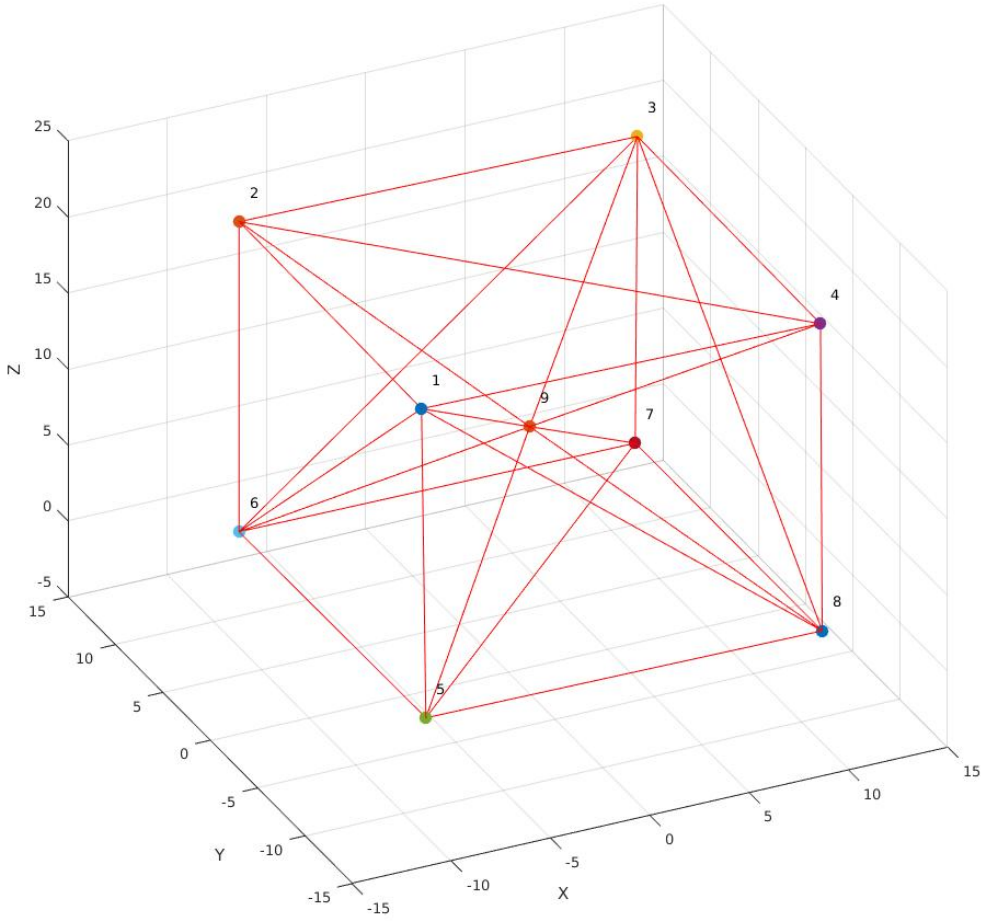

Figure 4.1 Swarm formation

The first order differential equation to be used according to the control rule proposed in Queiroz [40]:

$$\dot{x}_i = u_i \qquad i = 1, ..9, \tag{4.1}$$

Where x_i is position and u_i is the control input for the agent i.

The control input u_i for 3 different swarm movements are calculated in different ways. Here, without a central controller, each agent calculates its control input only based on the relative position of the agents to which it is connected. Thus, it can form a swarm and move as a swarm without any external dependency.

4.2 Formation Acquisition

Here, agents will start at a random distance from their location and try to maintain their position. Control rule for formation acquisition:

$$u = u_a := -k_v R^T(\vec{p})z \tag{4.2}$$

Accordingly, the equation to be used by each agent:

$$u_i = -k_v \sum_{j \in N_i(E)} {}^i_j\vec{p}^i_j z, \qquad i = 1,..9, \tag{4.3}$$

It was applied to the swarm consists of 9 agents. Each agent determines its position in the swarm without central control.
Where:

$k_v > 0$: Control gain
${}^i_j e = \|{}^i_j\vec{p}\| - {}^i_j d$
${}^i_j z = {}^i_j e({}^i_j e + 2{}^i_j d)$
$R(\vec{p}) = \frac{1}{2}\frac{\partial \phi(p)}{\partial p}$: Rigidity matrix. Where $\phi(p) = [..., \|{}^j_i\vec{p}\|^2, ...] \quad (i,j) \in E$.

Matlab implementation of formation acquisition is as follows:

```
function formationAcquisition
    u = zeros(1,3);
    for j = 1:9
                % Check is agent j connected
        if ~isempty(find(connected_agents == j,1))
                        % Get estimated distances from
                           estimator
            pij = norm(poseEstimator(j).x(1:3));
                        % Find error between estimated and
                           desired distances
            e = pij - formation_distances(j);
            z = e*(e+2*formation_distances(j));
                        % Calculate control input
            u = u +kv*poseEstimator(j).x(1:3)'*z;
        end
    end
    set_input(u);
end
```

4.3 Formation Maneuvering

They performs the determined maneuver while maintaining the swarm formation. In this method, the agent 9 from the center of the swarm will be the leader. The herd will maneuver while maintaining the linear velocity of $v(t)$ and the angular velocity of $\omega(t)$. Control rule for formation maneuver:

$$u = u_a + v_d \tag{4.4}$$

Accordingly, the equation each agent will use:

$$u_i = -k_v \sum_{j \in N_i(E)} {}^{i}_{j}\vec{p}^{\,i}_{j}z + v(t) + \omega(t) \times \vec{p}_{in}, \qquad i = 1,..9, \tag{4.5}$$

p_{in}: Distance of i. agent to the leader agent n.
$n = 9$
k_v and ${}^{i}_{j}z$ Defined in Section 4.2.

Matlab implementation of formation maneuvering is as follows:

```
function formationManeuvering
    u = zeros(1,3);
    maneuverVelocity = [1 cos(0.01*this.t) 0];
    for j = 1:9
                % Check is agent j connected
        if ~isempty(find(connected_agents == j,1))
                        % Get estimated distances from
                           estimator
            pij = norm(poseEstimator(i).x(1:3));
                        % Find error between estimated and
                           desired distances
            e = qij - formation_distances(i);
            % Calculate control input
            z = e*(e+2*formation_distances(i));
            u = u +kv*poseEstimator(i).x(1:3)'*z;
        end
    end
        % Add maneuver velocity
    u = u + maneuverVelocity + cross(maneuverAngularVelocity,
       poseEstimator(9).x(1:3)');
    set_input(u);
end
```

4.4 Flocking

Control rule for constant speed flight while maintaining swarm formation:

$$u = u_a + \hat{v} \tag{4.6}$$

$$\dot{\hat{v}}_i = \alpha \sum_{j \in N_i(E)} ({}^j_i\hat{v}) - \alpha b_i(\hat{v}_i - v_0), \qquad i = 1,..9, \tag{4.7}$$

Here,

$$b_i = \begin{cases} 1, & \text{If } i \in V_0 \\ 0, & \text{Otherwise,} \end{cases}$$

$\hat{v}$: Agent velocity.

V_0 : The set of agents that can access flight speed information,

v_0 : Flight speed,
$\alpha > 0$: Observer gain.

Matlab implementation of formation maneuvering is as follows:

```
function flockingConstantVelocity
    u = zeros(1,3);
    v_sum = zeros(1,3);
    for i = 1:9
                % Check is agent j connected
        if ~isempty(find(connected_agents == i,1))
                        % Get estimated distances from
                            estimator
            pij = norm(poseEstimator(i).x(1:3));
                        % Find error between estimated and
                            desired distances
            e = pij - formation_distances(i);
            z = e*(e+2*this.formation_distances(i));
                        % Calculate control input
            u = u +kv*poseEstimator(i).x(1:3)'*z;
                        % Sum of agents relative velocities
            v_sum = v_sum + alpha*poseEstimator(i).x(4:6)';
        end
    end
    vdot = v_sum - alpha*(velocity - flockingVelocity);
    vhat = vhat + vdot*dT;
    set_input(u+vhat);
end
```

5
SIMULATION

The proposed method was simulated in Matlab environment. A quadrotor model with PD controller [26] in Simulink environment was used for each agent.

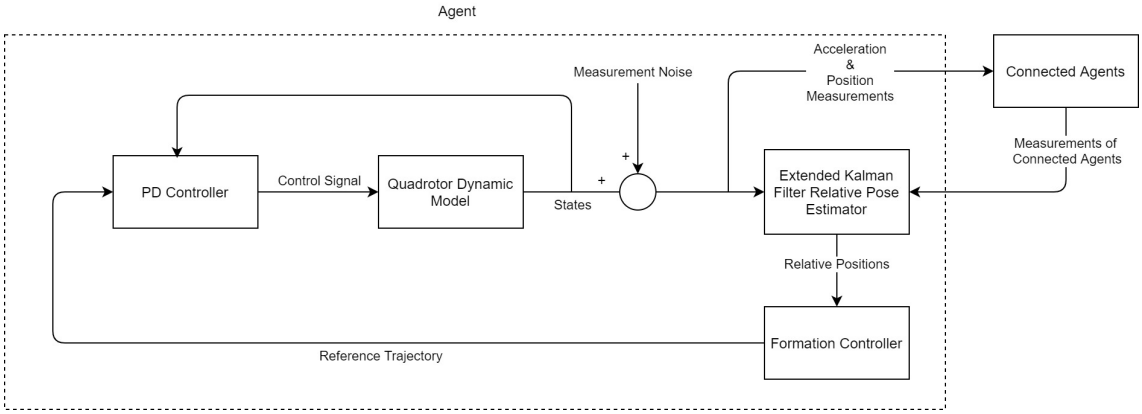

Figure 5.1 Simulation data flow

The data flow in the simulation can be seen in Figure 5.1. Each agent receives noisy measurements from the quadrotor model and measurements of other agents. It calculates the relative position with Extended Kalman Filter and then sends the reference trajectory to the PD controller with the formation controller. Each agent adds measurement noise to the output of its model and shares it with the connected agents and uses it to simulate its own IMU and vision measurement. Agents use simulated images and IMU measurements from others to estimate relative position as shown in Section 3.4. After the relative position is calculated, it gives the reference trajectory input to the simulink model to keep the position in the swarm with the control rules shown in Section 4.

In Figure 5.2, the quadrotor model defined for each agent in Simulink environment can be seen.

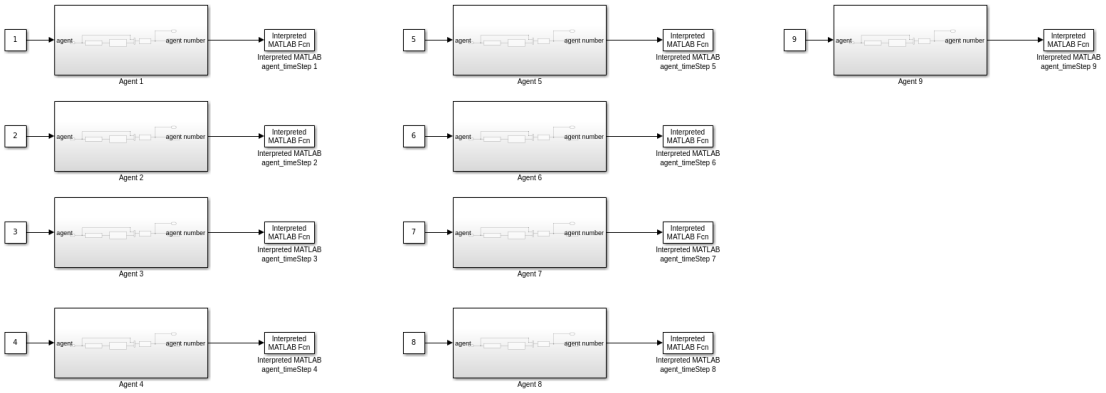

Figure 5.2 Agent models in Simulink

5.1 Simulating Quadrotors

The quadrotor model and thr PD controller mentioned in Section 2.1 is simulated in Simulink. PD controller takes the reference position as input and keeps the quadrotor model at this reference position. See Figure 5.1. The quadrotor model and controller used in simulation were obtained from the study of Mishra[26].

5.2 Simulating Measurements

To simulate IMU measurements, unbiased random white noise $\mathcal{N}(\mu, \sigma^2)$ was added to the acceleration output of the quadrotor model. Similarly, in order to obtain vision measurements, white noise was added to the relative position between the agents and normalized. The equation from which the vision measurements are obtained is as follows:

$$\vec{x}^{\,j}_{i_{meas}} = \|x_{j_{true}} - x_{i_{true}} + rand()\| \tag{5.1}$$

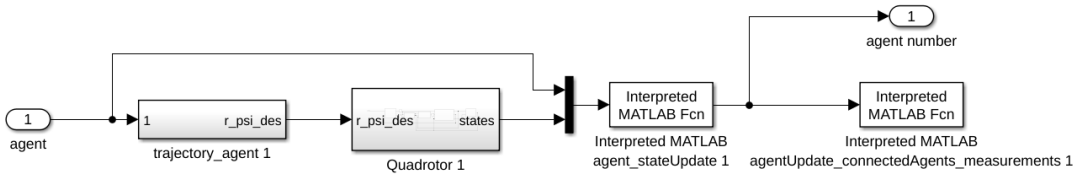

Figure 5.3 Agents updates own and connected agents measurements each step

In Figure 5.3, random noise is added to the acceleration and vision output of the simulink model of the agent to obtain its own measurement values. These measurements are then sent to other agents connected to the agent.

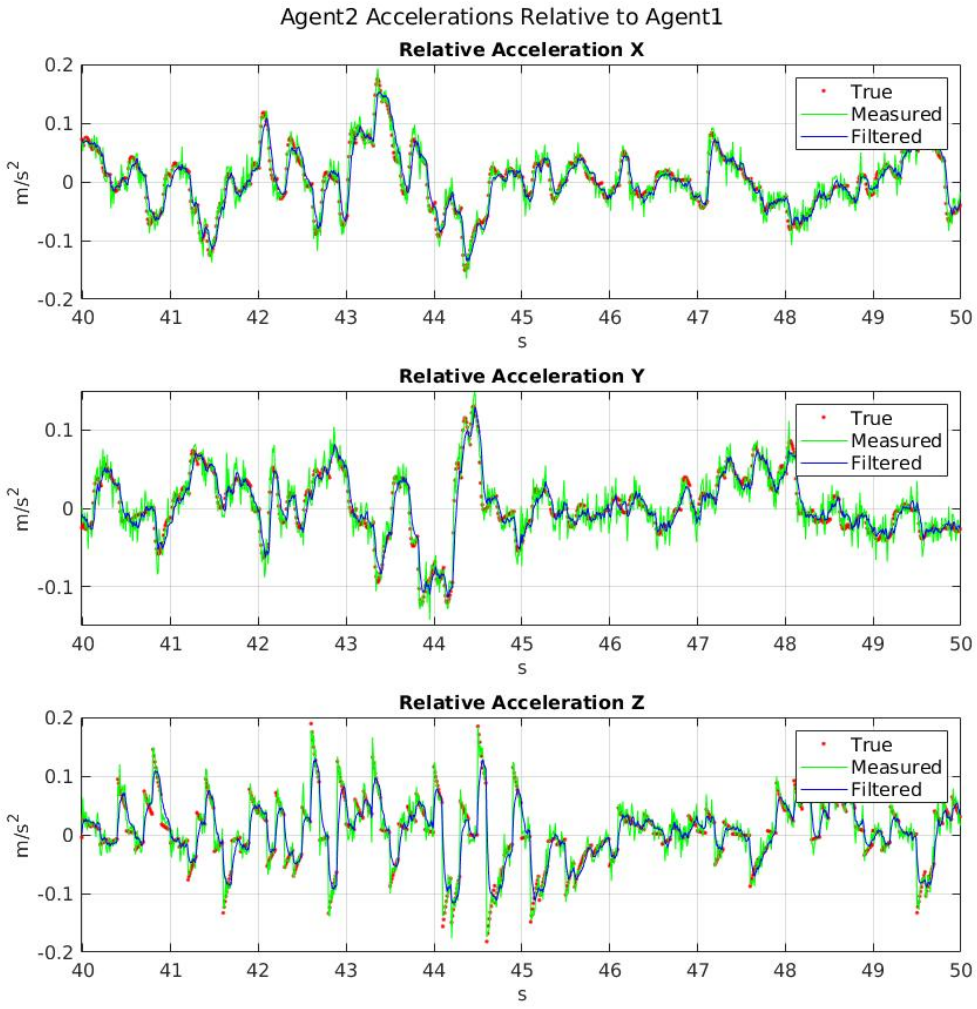

Figure 5.4 True, measured and filtered accelerations of Agent2 relative to Agent1

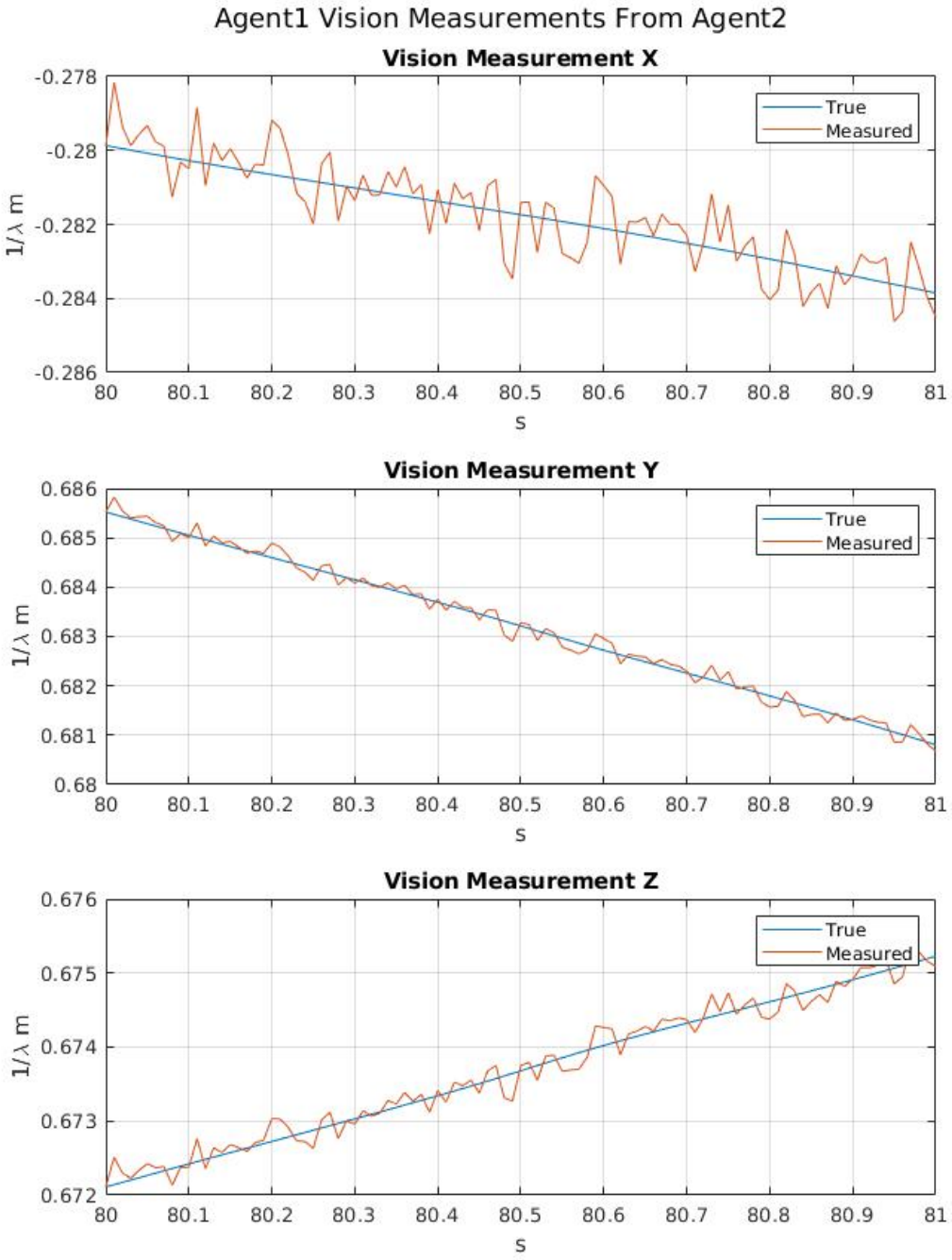

Figure 5.5 True and measured pose measurement of Agent2 relative to Agent1

Figures 5.4 and 5.5 show simulated IMU and image measurements. The measurements values obtained by adding random white noise on the real values are used by Kalman filter.

6
RESULTS AND DISCUSSION

The results were obtained by applying the method suggested in simulations with Matlab to different swarm flight algorithms. In the simulation, which was run for 300 seconds, the agents maintained their position in the formation by staying within a maximum of ±%5 error zones.

6.1 Formation Acquisition Simulation Results

A non-moving swarm was simulated in which agents maintain their desired positions in the formation. Since the relative acceleration between the agents is close to zero, this is the simulation that forces the algorithm the most. The actual distance error between agents in this simulation was below ±%5.

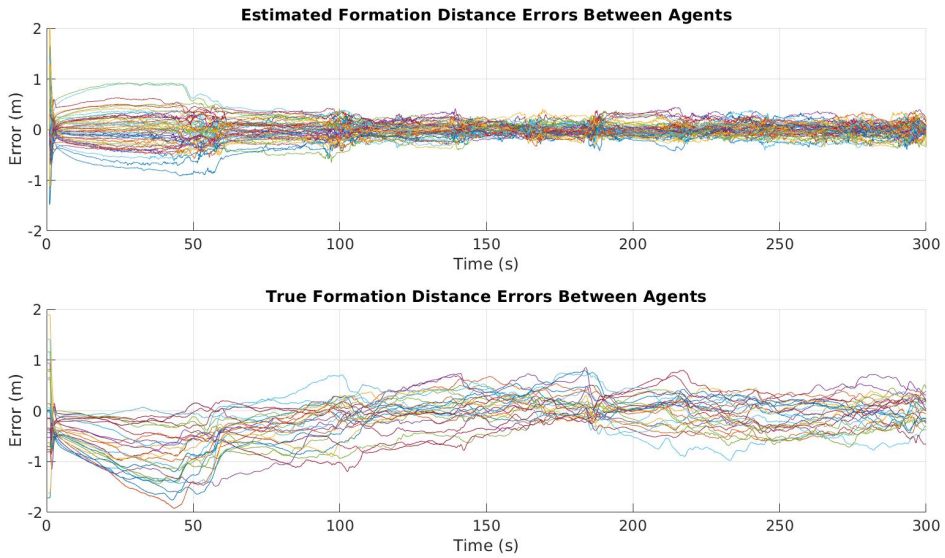

Figure 6.1 Estimated and true formation distance errors between agents on formation acquisition simulation

The position predicted by the Extended Kalman Filter and the actual relative position between agents is shown in Figure 6.2. With the proposed method the extended

Table 6.1 Formation distances (meter) RMSE of agents on formation acquisition simulation

Agents	1	2	3	4	5	6	7	8	9
1	-	0.29	-	0.43	0.35	0.59	-	0.69	0.33
2	0.29	-	0.53	0.65	-	0.51	-	-	0.32
3	-	0.5	-	0.62	-	0.69	0.45	0.38	0.28
4	0.43	0.65	0.62	-	-	-	-	0.24	0.4
5	0.35	-	-	-	-	0.45	0.44	0.22	0.16
6	0.59	0.51	0.69	-	0.45	-	0.58	-	0.55
7	-	-	0.45	-	0.44	0.58	-	0.61	0.41
8	0.69	-	0.38	0.24	0.22	-	0.61	-	0.47
9	0.33	0.32	0.28	0.4	0.16	0.55	0.41	0.47	-

Kalman filter correctly predicted the position with ±%2.5 errors.

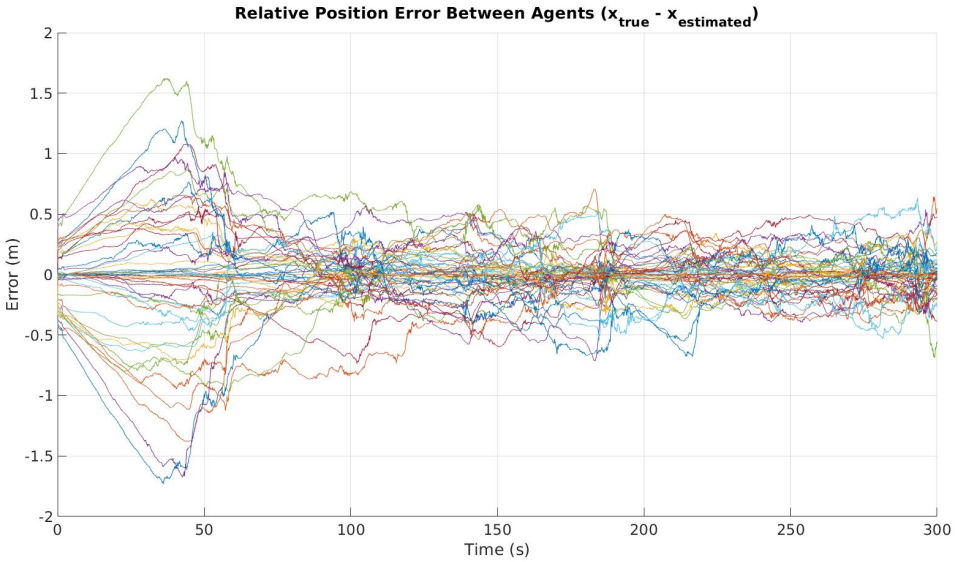

Figure 6.2 Error between true and estimated positions on formation acquisition simulation

In Figure 6.2, although the relative acceleration between agents is very low, the error between the actual relative position and the position estimated by the Kalman filter was below 0.5 m.

Table 6.1 shows the RMS errors of the desired distance between the agents in the columns and the agents which they are connected in the rows.

6.2 Flocking Simulation Results

In this simulation, which is flown as a swarm, the path followed by the agents is shown in Figure 6.3. The flight speed of the swarm is $v_0 = (4, 2, 0)$ m/s, and the set of agents that can access the flight speed information is $V_0 = (1, 2, 3, 4, 5, 6, 7, 8, 9)$.

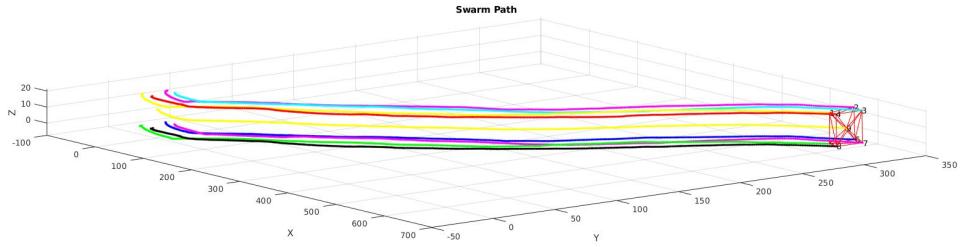

Figure 6.3 Agents paths on flocking simulation

Formation distance errors remained below 1 meter in this flight, where the relative acceleration between agents is still very low.

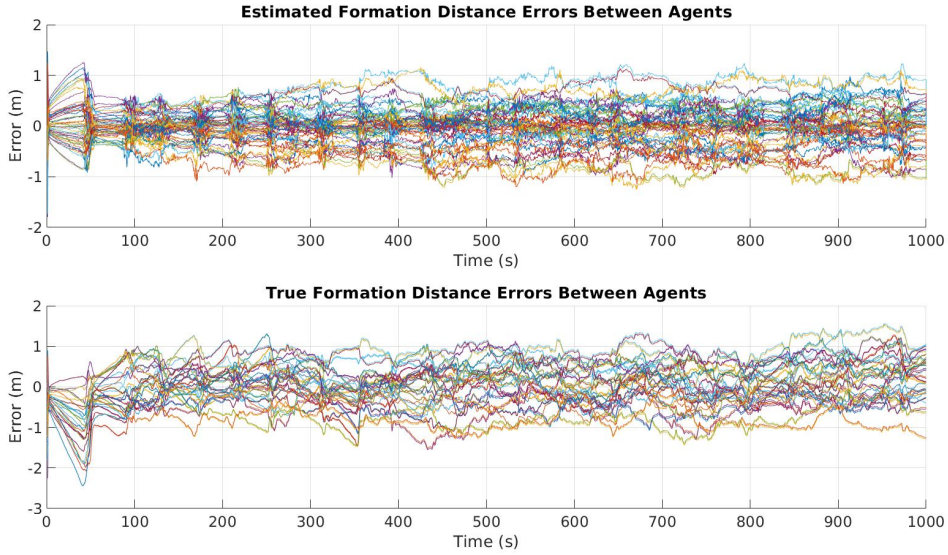

Figure 6.4 Estimated and true formation distance errors between agents on flocking simulation

The error between the actual relative position and the relative position predicted by the Extended Kalman Filter was below 0.5 meters.

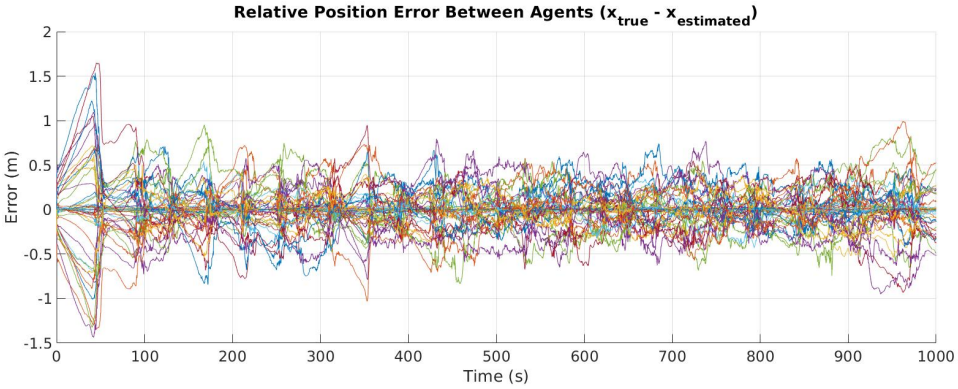

Figure 6.5 Extended kalman filter error on flocking simulation

Table 6.2 Formation distances RMSE of agents on flocking simulation

Agents	1	2	3	4	5	6	7	8	9
1	-	0.41	-	0.47	0.35	0.31	-	0.45	0.51
2	0.41	-	0.35	0.32	-	0.35	-	-	0.34
3	-	0.34	-	0.39	-	0.57	0.43	0.47	0.61
4	0.48	0.32	0.4	-	-	-	-	0.32	0.16
5	0.35	-	-	-	-	0.43	0.76	0.29	0.52
6	0.32	0.35	0.57	-	0.43	-	0.44	-	0.25
7	-	-	0.43	-	0.76	0.45	-	0.62	0.9
8	0.45	-	0.46	0.32	0.28	-	0.62	-	0.45
9	0.49	0.33	0.6	0.16	0.5	0.24	0.88	0.44	-

6.3 Formation Maneuvering Simulation Results

Kalman filter made the best relative position estimation in this simulation since the relative acceleration change between the agents is the highest in this simulation. Thus, the formation errors were the lowest in this simulation. In this simulation, the maneuver speed equations are $v(t) = (1, cos(t), 0)$, $\omega(t) = (0, 0, 0.1)$.

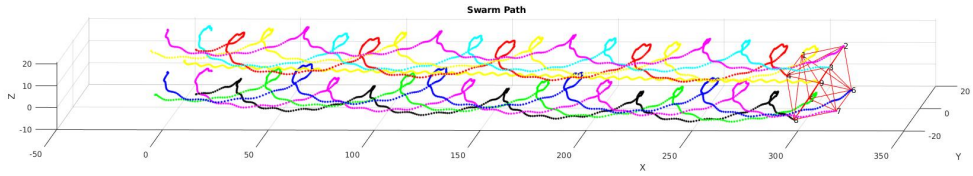

Figure 6.6 Agents paths on formation maneuvering simulation

In Figure 6.6, the path followed by the agents in this simulation can be seen.

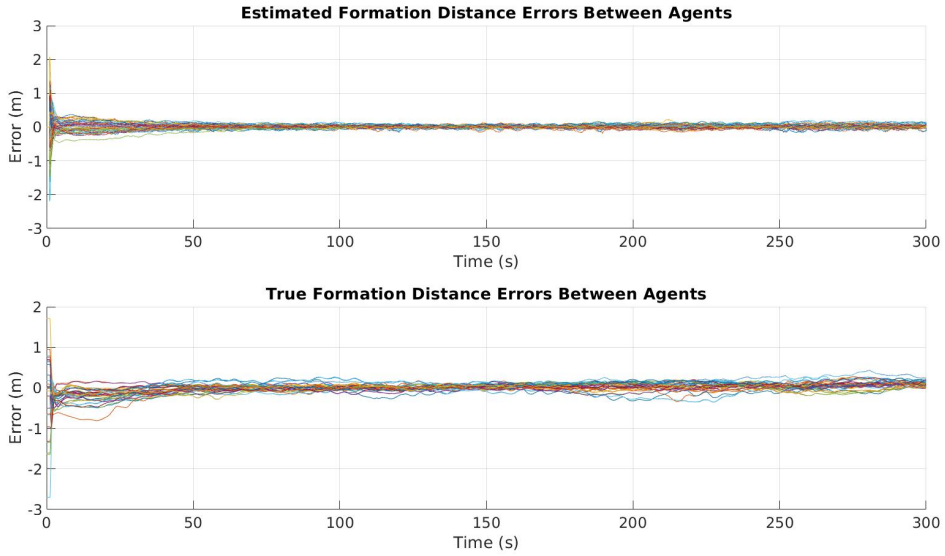

Figure 6.7 Estimated and true formation distance errors between agents on formation maneuvering simulation

As can be seen in Figure 6.7, the error between the position estimated by the Kalman filter and the actual position is below 0.2 meters.

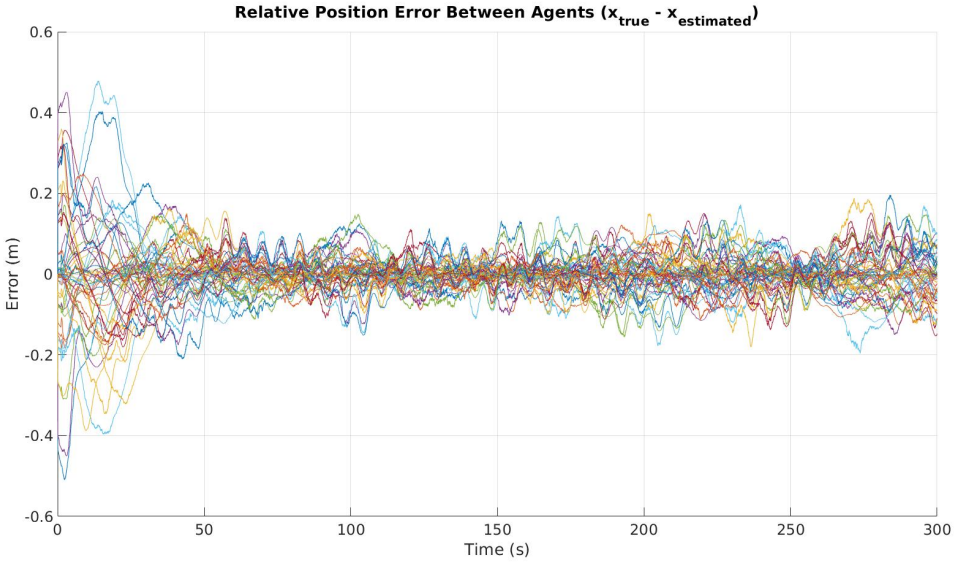

Figure 6.8 Extended kalman filter error on formation maneuvering simulation

Kalman filter correctly predicted the position with ± 0.2 meter.

Table 6.3 Formation distances RMSE of agents

Agents	1	2	3	4	5	6	7	8	9
1	-	0.12	-	0.11	0.23	0.09	-	0.1	0.18
2	0.12	-	0.18	0.13	-	0.15	-	-	0.12
3	-	0.17	-	0.22	-	0.1	0.19	0.17	0.14
4	0.1	0.13	0.23	-	-	-	-	0.2	0.16
5	0.23	-	-	-	-	0.08	0.07	0.1	0.15
6	0.09	0.15	0.1	-	0.09	-	0.1	-	0.1
7	-	-	0.19	-	0.07	0.1	-	0.12	0.08
8	0.1	-	0.17	0.2	0.1	-	0.12	-	0.07
9	0.18	0.12	0.14	0.16	0.15	0.1	0.08	0.07	-

6.4 Discussion

In this study, we have demonstrated that the relative position between agents can be detected only with a monocular camera and IMU sensor, and a swarm can be created without the need for external dependent systems like VICON, GPS etc. In the simulations performed, it was observed that the errors were at an acceptable level while the swarm was standing in a fixed position, flying as a swarm and maneuvering. Unmanned aerial vehicles swarms to be created with the proposed method can perform many tasks such as estimating the location of a radio broadcast with triangulation, mapping or surveillance of large areas in a short time.

REFERENCES

[1] P. Merriaux, Y. Dupuis, P. Vasseur, X. Savatier, "Wheel odometry-based car localization and tracking on vectorial map," in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2014, pp. 1890–1891.

[2] J. Zhang, S. Singh, "Loam: Lidar odometry and mapping in real-time.," in *Robotics: Science and Systems*, vol. 2, 2014.

[3] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.," in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.

[4] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, IEEE, 1994, pp. 593–600.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] H. Bay, T. Tuytelaars, L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, Springer, 2006, pp. 404–417.

[7] E. Rosten, R. Porter, T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 1, pp. 105–119, 2008.

[8] M. Calonder, V. Lepetit, C. Strecha, P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, Springer, 2010, pp. 778–792.

[9] E. Rublee, V. Rabaud, K. Konolige, G. R. Bradski, "Orb: An efficient alternative to sift or surf.," in *ICCV*, Citeseer, vol. 11, 2011, p. 2.

[10] D. Nistér, O. Naroditsky, J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Ieee, vol. 1, 2004, pp. I–I.

[11] A. J. Davison, I. D. Reid, N. D. Molton, O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1052–1067, 2007.

[12] G. Klein, D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, IEEE Computer Society, 2007, pp. 1–10.

[13] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[14] S. Weiss, R. Siegwart, "Real-time metric state estimation for modular vision-inertial systems," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 4531–4537.

[15] M. Li, A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, 2013.

[16] M. Achtelik, M. Achtelik, S. Weiss, R. Siegwart, "Onboard imu and monocular vision based control for mavs in unknown in-and outdoor environments," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 3056–3063.

[17] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, R. Siegwart, "Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments," in *2012 IEEE international conference on robotics and automation*, IEEE, 2012, pp. 957–964.

[18] R. Mur-Artal, J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[19] F. Rivard, J. Bisson, F. Michaud, D. Létourneau, "Ultrasonic relative positioning for multi-robot systems," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 323–328.

[20] J. F. Roberts, T. Stirling, J.-C. Zufferey, D. Floreano, "3-d relative positioning sensor for indoor flying robots," *Autonomous Robots*, vol. 33, no. 1-2, pp. 5–20, 2012.

[21] L. Oliveira, H. Li, L. Almeida, T. E. Abrudan, "Rssi-based relative localisation for mobile robots," *Ad Hoc Networks*, vol. 13, pp. 321–335, 2014.

[22] L. Merino, J. Wiklund, F. Caballero, A. Moe, J. R. M. De Dios, P.-E. Forssen, K. Nordberg, A. Ollero, "Vision-based multi-uav position estimation," *IEEE robotics & automation magazine*, vol. 13, no. 3, pp. 53–62, 2006.

[23] M. W. Achtelik, S. Weiss, M. Chli, F. Dellaerty, R. Siegwart, "Collaborative stereo," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2011, pp. 2242–2248.

[24] I. V. Melnyk, J. A. Hesch, S. I. Roumeliotis, "Cooperative vision-aided inertial navigation using overlapping views," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 936–943.

[25] M. Karrer, M. Agarwal, M. Kamel, R. Siegwart, M. Chli, "Collaborative 6dof relative pose estimation for two uavs with overlapping fields of view," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 6688–6693.

[26] S. Mishra, "Robust control of quadrotors," Ph.D. dissertation, Master thesis, Department of Automatic Control and Robotics, 2017.

[27] J. BillingsleyUniversity, "Mechatronic systems, sensors, and actuators: Fundamentals and modeling," 2009.

[28] R. Hartley, A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[29] P. Corke, D. Strelow, S. Singh, "Omnidirectional visual odometry for a planetary rover," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, vol. 4, 2004, pp. 4007–4012.

[30] J.-P. Tardif, Y. Pavlidis, K. Daniilidis, "Monocular visual odometry in urban environments using an omnidirectional camera," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2008, pp. 2531–2538.

[31] D. Scaramuzza, F. Fraundorfer, R. Siegwart, "Real-time monocular visual odometry for on-road vehicles with 1-point ransac," in *2009 IEEE International conference on robotics and automation*, Ieee, 2009, pp. 4293–4299.

[32] L. A. Clemente, A. J. Davison, I. D. Reid, J. Neira, J. D. Tardós, "Mapping large loops with a single hand-held camera.," in *Robotics: Science and Systems*, vol. 2, 2007.

[33] H. Strasdat, J. Montiel, A. J. Davison, "Scale drift-aware large scale monocular slam," *Robotics: Science and Systems VI*, vol. 2, no. 3, p. 7, 2010.

[34] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[35] E. Rosten, T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, Springer, 2006, pp. 430–443.

[36] M. Muja, D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[38] G. Welch, G. Bishop, *et al.*, *An introduction to the kalman filter*, 1995.

[39] G. Nützi, S. Weiss, D. Scaramuzza, R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 287–299, 2011.

[40] M. de Queiroz, X. Cai, M. Feemster, *Formation Control of Multi-Agent Systems: A Graph Rigidity Approach*. Jan. 2019, ISBN: 9781118887448. DOI: 10.1002/9781118887455.

PUBLICATIONS FROM THE THESIS

Contact Information: onurozturk@gmail.com

Conference Papers

1. O. Ozturk, ”İnsansız hava aracı sürülerinin görsel yardımlı formasyon kontrolü”, presented at 2nd International Eurasian Conference on Science, Engineering and Technology, Gaziantep, Turkey, Oct. 7, 2020.