Implementation of Vision Processing Tasks on a Smart Camera Platform Using Neighborhood Processors

Dingyi Hong

Follow this and additional works at: http://digitalcommons.unl.edu/elecengtheses

Part of the Computer Engineering Commons, and the Other Electrical and Computer Engineering Commons


http://digitalcommons.unl.edu/elecengtheses/97
IMPLEMENTATION OF VISION PROCESSING TASKS ON A SMART CAMERA PLATFORM USING NEIGHBORHOOD PROCESSORS

by

Dingyi Hong

A THESIS

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfilment of Requirements
For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professors Michael Hoffman and Sina Balkir

Lincoln, Nebraska
July, 2018
IMPLEMENTATION OF VISION PROCESSING TASKS ON A SMART CAMERA PLATFORM USING NEIGHBORHOOD PROCESSORS

Dingyi Hong, M.S.
University of Nebraska, 2018

Advisors: Michael Hoffman and Sina Balkir

Two applications of vision processing on a smart camera platform using neighborhood processors are presented. A smart camera application of real-time trajectory calculation for tracking objects and predicting their path of motion by using a single intelligent camera or a dual camera system are designed, simulated and implemented. Another application of object recognition with a key points algorithm for detecting simple shaped objects and locating the camera position within a 3-D coordinate system via imaging by using artificial neural network is applied to provide quick and accurate processing on a neighborhood-level parallel processor or other limited vision chip. For all applications an emphasis has been placed on simplifying algorithmic requirements to allow implementation on processors with limited computational resources.
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Michael Hoffman for mentoring me and encouraging me during my time at the University. His teaching, encouragement and patience made my graduate experience enjoyable and fruitful. I would like to thank Dr. Sina Balkir for his teaching in integrated circuit area. I appreciate Dr. Jerry L. Hudgins for being my committee member. I would also like to thank Dr. Mark Bauer for his friendly help and his guidance in my research career. I learned a great deal from senior graduate students in the department, specifically Mahir Gharzai and Joseph Schmitz who not only led me in the begining of this research, but also shared their helpful experiences. Finally, I am grateful to my family who loves and supports me unconditionally.
# Contents

**Contents**

**List of Figures**

**List of Tables**

1 **Introduction**

1.1 Smart Camera Overview

1.2 Neighborhood-level Parallel Processing Camera

1.3 Smart Camera Applications

1.4 Thesis Organization

2 **Trajectory Calculation and Prediction**

2.1 Single Camera Model

2.1.1 Trajectory Calculation

2.1.2 Trajectory Prediction

2.1.3 Simulation

2.1.4 Results

2.2 Dual Camera System

2.2.1 Regular Model and Vector-based Algorithm
2.2.2 Similarity Triangle-based Algorithm .................................. 18
2.2.3 Lookup Table Algorithm ............................................... 19
2.2.4 Improved Lookup Table Algorithm ................................. 23
2.2.5 Complexity Analysis ..................................................... 25
2.2.6 MATLAB Simulation ..................................................... 27
2.2.7 Results ................................................................. 29

3 Fast Scale, Rotation and Shift Invariant Recognition Application 31

3.1 Overview ................................................................. 31
3.2 Recognition Algorithm Analysis ....................................... 32
  3.2.1 Convolutional Neural Network .................................... 32
  3.2.2 Cross-correlation-based Algorithm ............................... 33
  3.2.3 Scale and Rotation Invariant Analysis ......................... 34
3.3 Key Points Algorithm .................................................. 34
  3.3.1 Basic Algorithm Introduction .................................... 34
  3.3.2 Additional Processing, Multi-object Detection ............... 35
  3.3.3 Object Comparison .................................................. 37
  3.3.4 Simulation and Results ........................................... 39
3.4 Recognition Algorithm Application .................................. 40
  3.4.1 Application Introduction ......................................... 41
  3.4.2 Neural Network Analysis ......................................... 43
  3.4.3 Side Lengths Model ............................................... 46
  3.4.4 Vertices Model ..................................................... 47
  3.4.5 Spherical Model ................................................... 50
  3.4.6 Implementation and Results .................................... 51

4 Conclusion and Future Work 55
List of Figures

11 Block diagram of the neighborhood-level parallel processing architecture. Data movement, global program control, and global analog-to-digital signals shown. [1] ......................... 3

12 Camera test setup. Mounted to tripod with 3D printed holster. Using f/1.0 lens and USB 2.0 communications. [1] ................. 3

21 Position vs. time for second order motion ....................... 8

22 Predicted trajectory for linear and non-linear object ............. 10

23 Linear trajectory prediction for a light traveling in a straight line(top, middle row) and making a turn(bottom row) with inverted color. Green: past path, Red: predicted path. [2] ......................... 11

24 Non-linear trajectory prediction in the positive x-axis for a light accelerating, then decelerating and hitting an inflection point with inverted color. Green: past path, Red: predicted path. [2] ......................... 12


26 Dual camera system for 3-D tracking. Black: linear motion, Green: non-linear motion ......................... 14
27 Dual camera system planar model in xy plane. Red rectangle: camera1, Black rectangle: camera2, Red dot: tracking object, Red arrows: vectors from cameras to object. ........................................... 15

28 Pinhole model in 3-D space. $o_i$: frame center, $o$: pinhole, $p$: the middle point of the bottom side of first pixel on the left of the frame center $o_i$, $o_s$: center of scene, $s$: $p$ projection in the scene plane ......................... 16

29 A simulated camera output for a moving object. Blue dots: tracked object centers for continuous frames, $c1h$: horizontal position in frame $-$ half of frame length, $c1v$: vertical position in frame $-$ half of frame height ... 18

210 A similarity method model in xy plane. Red rectangle: camera1, Black rectangle: camera2, Red dot: tracking object, Red arrows: vectors from cameras to object. $a1$, $a2$, $b1$, $b2$: distances between object and cameras in x or y directions ......................................................... 20

211 Simulated camera views in camera1 and camera2. Object is moving from left to right in both views. Blue dots: tracked object centers, Red stars: tracked object centers in both camera1 and camera2 views for calculation 22

212 Object trajectory and positions in xy plane with $10 \times 10$ resolution. Camera1 points towards to the positive direction of y axis and camera2 points towards to the positive direction of x axis. ......................... 23

Simulated camera views in camera1 and camera2. Object is moving from left to right in both views. fdx: horizontal pixel difference between object and center of NP for camera 1, fdy: horizontal pixel difference between object and center of NP for camera 2, Blue dots: tracked object centers.

A simulation of object trajectory calculation with 10 × 10 scale-down resolution. Camera1 points towards to the positive direction of y axis and camera2 points towards to the positive direction of x axis. Green line: motion trajectory, Red circle: calculated object position with LUT algorithm, Blue cross: calculated object position with vector-based algorithm.

RMS error analyzing in different directions for each sample by using regular lookup table, vector-based, improved LUT with dx comparison, improved LUT with f dx comparison algorithm.

A projected acute triangular object in a simulated frame. Colored arrows and numbers present next-edge-pixel searching priorities based on left-up-right-down to its original direction. Dark red pixels: detected key points, Dark gray pixels: tracked edges.

A projected acute triangular object in a simulated frame. Black arrows: trajectory of edge detecting tracer, Dark red pixels: detected key points, Yellow pixel: special key point.

A projected obtuse triangular object in a simulated frame. Black arrows: trajectory of edge detecting tracer, Dark red pixels: detected key points.

Two projected triangular objects are in a simulated frame. Black arrows: object detecting tracer, Dark red arrows: edge detecting tracer, Dark gray pixels: marked detected edge.
35 Simulated recognitions of triangular objects with key points algorithm.
Light gray pixels: detected object edges, White pixels: detected key points

36 Drone camera models for bottom view camera and front pitch-adjustable camera.

37 A simulated scene with a 5 : 12 : 13 right triangle for imaging simulation.

38 Simulated images for right triangle with different camera positions and
directions in 100 × 100 pixel frames.

39 Simulated images for right triangle with different camera positions and
directions in 100 × 100 pixel frames.

310 Neural network structure.

311 Neural network data training model. The position of camera \( p(x, y, z) \) is
random where \(-500 < x < 1500, -500 < y < 1500\) and \(200 < z < 3200\) (mm)

312 Neural network training regression for unordered side lengths inputs.

313 Neural network training performance for randomly unordered side lengths inputs. Blue: target positions, Red: neural network outputs, Yellow: errors

314 Simulated images for highlighted vertex triangle with different camera
positions and directions in 100 × 100 pixel frames.

315 Neural network training performance for randomly vertex position inputs.
Blue: target positions, Red: neural network outputs, Yellow: errors

316 Neural network training performance in spherical coordinate system. Blue:
target positions, Red: neural network outputs, Yellow: errors

317 Neural network training performance for final implementation model. Blue:
target positions, Red: neural network outputs, Yellow: errors
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Operation and space complexity</td>
<td>27</td>
</tr>
<tr>
<td>31</td>
<td>Root mean squared error for different ANNs in each direction</td>
<td>53</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Smart Camera Overview

A CCD or CMOS image sensor chip is commonly utilized for standard digital camera systems. CMOS chips provide lower power consumption and are inexpensive compared to CCD technologies. The flexibility of CMOS technology allows additional circuits to be placed in an image sensor chip. Thus, a camera with an image chip which produces not only image capture but also in-chip processing abilities is considered a smart or intelligent camera.

Based on different processor structures of CMOS vision chips, processing elements can be mainly described as pixel-level, column-level or chip-level. For pixel-level processing, there is a processing element (PE) in each pixel which produces the highest processing speed. This type of chip structure performs low-level operations with adjacent pixel data such as edge detection or morphological operations \[3\]. However, pixel-level processing is inflexible for programming and can be difficult for high-level algorithm implementation. Unlike pixel-level processing, row or column-level processing distributes a processor for each column or row in the frame which allows
processors to quickly access pixel data in their row or column. With this structure, image statistics computation such as histograms and centroids, and image transformations including mirroring and rotations have relatively high processing speed but processing flexibility was still limited [4]. Chip-level processing allows all pixel access for a single processor which provides a significant programming flexibility such in [5]. Since chip-level processing has to read each pixel value sequentially, the overall operation speed is limited, especially for high resolution frames. Some hybrid smart camera architectures combine with different type of processors to maintain high processing speed but increase programming flexibility [6].

1.2 Neighborhood-level Parallel Processing

Camera

For different processor architectures, overall performance is limited by combinations of programmability, power consumption and operation speed. A new category of neighborhood-level parallel processor arrays has been designed and implemented to optimize these drawbacks. Each neighborhood processor (NP) contains $8 \times 8$ photodiode sensors and their pixel ADCs, memory and an arithmetic logic unit (ALU) to provide more flexibility without sacrificing operation speed [7]. Also, pixel values or processed data can be efficiently transmitted between any adjacent NPs.

The neighborhood-level parallel processing architecture is constructed by multiple NPs as shown in Figure 11. For the current version of the neighborhood-level parallel processing camera, $8 \times 10$ NPs are embedded to generate a $64 \times 80$ resolution vision chip. Since the number of pixels for each processor is fixed, the overall processing speed does not increase when the resolution is increasing. The design, testing and
Figure 11: Block diagram of the neighborhood-level parallel processing architecture. Data movement, global program control, and global analog-to-digital signals shown. [1]

demonstration of this vision chip are described in [1] and [8]. Also, the neighborhood parallel intelligent camera and its testbed is shown in Figure 12.

Figure 12: Camera test setup. Mounted to tripod with 3D printed holster. Using f/1.0 lens and USB 2.0 communications. [1]
1.3 Smart Camera Applications

To implement simple and efficient algorithms on the neighborhood-level parallel processing camera, assembly language with syntax similar to Intel and ARM assembly are used for programming. Commands and instructions for neighborhood processors are represented in [7]. A variety of processing tasks have been implemented on the vision chip including imaging, edge detection, filtering, 2-D discrete cosine transform, statistics gathering, and object tracking. Based on highly efficient algorithms and parallel processing features, these applications achieve relatively high frame rates compared with other vision chips [1].

In order to further explore the potential of neighborhood processing, more complicated algorithms are applied in this work including trajectory calculation and object recognition. For real-time trajectory calculation, the algorithm requires efficiency for high frame per second processing. The complexity of the recognition algorithm has been limited to satisfy the hardware performance of neighborhood processors. Meanwhile, some applications require multithread algorithm optimization for parallel processors to improve performance. Since both trajectory calculation and recognition algorithms are quite complex for directly implementation on neighborhood parallel intelligent cameras, the algorithms have been simulated in the MATLAB environment to obtain capability and assess feasibility before testing on the actual vision chip.

1.4 Thesis Organization

Chapter 1 introduces the reader to the background of smart camera and neighborhood processors. Chapter 2 describes the trajectory calculation and prediction algorithm in this work, including both single processor or dual camera system implementation.
Chapter 3 introduces a fast scale, rotation and shift invariant recognition algorithm and its application by using artificial neural networks on UAVs. Chapter 4 includes future work and conclusions of the research.
Chapter 2

Trajectory Calculation and Prediction

2.1 Single Camera Model

An intelligent camera that produces high frame rates and processing speeds can be used for trajectory calculation and prediction. The single camera model allows for tracking and predicting object’s trajectory within the frame.

To maintain high frame rate, the algorithm is inherently limited in complexity with a premium for efficiency. For different embedded systems or intelligent cameras, memory (including program ROM and RAM) may affect algorithm implementation. It is difficult or inefficient to realize large sized arrays or hash or look up tables on limited storage devices. For embedded processors, fewer assembly operations can increase the complexity and length of the code. The neighborhood-level parallel parallel vision chip is designed with basic operations including ADD, SUB, AND, OR and bit shifts. When designing or implementing algorithms for the processor, it is necessary to avoid or reduce multiplications, divisions, or other non-built-in
operations. In this manner, complex computations or operations may need to be replaced with linear approximation formulas or lower complexity functions.

2.1.1 Trajectory Calculation

For a moving object in a static background, its trajectory can be calculated by observing differences between two successive frames. Several pre-processing steps are required for object tracking such as imaging, correlated double-sampling, and frame differencing. For different embedded systems or intelligent cameras, pre-processing code can be written to fit hardware features for efficient and fast performance. The hybrid nature of the parallel processors can run imaging, double-sampling and frame differencing code by all NP’s in parallel. After frame differencing, a moving object in a frame can be detected or highlighted by comparing each pixel value with a threshold. For most regular objects, center positions can be calculated by averaging edge pixel index for both x or y axes, requiring four extra registers to store the top, bottom, left and right pixel indecies. The averaging operation adds up top and bottom or left and right indices, then right bit shifts to get the result. Two 8-bit registers and the equation \((a + b)/2\) are used in this operation since neighborhood vision chip has \(80 \times 64\) resolution. For a higher resolution imaging chip, the equation \(a + (b - a)/2\) or larger registers can be used to avoid overflow.

2.1.2 Trajectory Prediction

Uniform linear motion with constant velocity or zero acceleration is one of the most common motions for trajectory prediction. A uniform linear motion model can be implemented in trajectory prediction for bullets or other high velocity and less gravity affected objects. For a linearly moving object, unit velocity can be detected and stored
by observing changes in the positional data. Predicted positions can be calculated by adding one (or multiple) recorded unit velocity \times frame period.

For a moving object with acceleration, the previous method ignores the effect of acceleration which can generate a more significant error in prediction. This prediction can be improved by using extra storage for differences of velocity which represent unit acceleration. With this concept, predicted position can be calculated by adding current position with unit velocity and unit acceleration. For a fixed time period \Delta t, v represents displacement from \(x(k - 2)\) to \(x(k - 1)\) and \(a\) represents difference of the displacement for \(x(k)\) as shown in Figure 21.

![Figure 21: Position vs. time for second order motion](image)

The predicted position for next frame \(x(k + 1)\) with constant frame per second
can be calculated by equations shown below:

\[ v = x(k - 1) - x(k - 2) \]  \hspace{1cm} (2.1a)
\[ a = x(k) - x(k - 1) - v \]  \hspace{1cm} (2.1b)
\[ x(k + 1) = x(k) + v + 2a \]  \hspace{1cm} (2.1c)

For variable acceleration motion with few changes of acceleration, predicted position can be approximately calculated by using latest updated acceleration. In most cases, the trajectory of motions, such as circular motion and non-ideal parabolic movement trajectory with less variance of the acceleration magnitude direction, can be calculated by this previous method with little error. However, this method produces major predicted error if the acceleration changes significantly. For example, consider a bouncing ball motion. In this case, the object velocity direction is reversed when it hits the ground. Predicted position maintains parabolic movement trajectory even though the object has changed the direction of travel. In this case, the hitting point can be detected by finding the difference between current and last calculated acceleration. An acceleration threshold determines specific point such as hitting point for the bouncing ball motion. After locating the point where the acceleration significantly changes, an additional step in the algorithm can be implemented for the specific motion. Alternately, the predicted position can be ignored and the calculation restarted after the hitting point to reduce the effect of the error.

2.1.3 Simulation

MATLAB simulations are shown in Figure 22. For the uniform linear motion model, an object path is randomly generated in a 64 × 80 grid of pixels which simulates the intelligent camera frame resolution. The line shows the actual trajectory and dots
correspond to object centers for each frame in initial trajectory graph. Predicted object centers are shown in predicted graph based on initial object centers. Since the storage and calculation on the intelligent camera are based on unsigned 8-bit integer, initial centers and predicted centers of MATLAB model are rounded to fit the simulated frame. The figure shows that the predicted centers follow the trajectory path with a small rounding error. For the non-linear bouncing object model, an acceleration threshold is used for detecting contact points. Calculated centers with accelerations that change significantly are considered as dummy points and are not used in the plot.

![Graphs showing initial and predicted trajectories for linear and non-linear objects](image)

Figure 22: Predicted trajectory for linear and non-linear object
2.1.4 Results

To implement the algorithm on neighborhood-level parallel processing vision chip, the program is converted to assembly language. It is necessary to add another comparison with two variables before subtraction to handle unsigned integers. Also, pre-processing, such as imaging and double-sampling, are part of the program. Due to the limitation of intelligent camera memory capacity, 512 instructions can be used for single program. Thus, it is difficult to implement both X and Y direction non-linear trajectory prediction under this chip version. Future neighborhood-parallel vision chips can be constructed with a larger memory space.

Figure 23: Linear trajectory prediction for a light traveling in a straight line (top, middle row) and making a turn (bottom row) with inverted color. Green: past path, Red: predicted path. [2]

An LED light source is used to provide a moving object for testing with the
neighborhood-level parallel processor. Figure 23 shows a first order linear trajectory prediction for both X and Y direction. A linear trajectory prediction for either X or Y direction and a non-linear trajectory prediction for another direction algorithm is implemented on this camera. Figure 24 shows positive X direction non-linear prediction algorithm with a light source is accelerating towards the positive X direction and suddenly changes direction to the negative X direction. The algorithm predicted trajectory in the positive X direction with an increasing acceleration and detected the inflection point accurately. The algorithm stops calculations after the object begins moving in the negative X direction.

Figure 24: Non-linear trajectory prediction in the positive x-axis for a light accelerating, then decelerating and hitting an inflection point with inverted color. Green: past path, Red: predicted path. [2]

A light source with positive Y direction acceleration is shown in Figure 25. The source changes its direction to positive Y direction with increasing acceleration. The algorithm accurately predicts increasing acceleration in Y axis.
2.2 Dual Camera System

Based on the single camera model, the intelligent camera calculates and predicts trajectory at the frame level, then highlights predicted positions in the same frame. In some cases, real-time frame level trajectory calculation and prediction are limited or not adequate to provide the actual position or velocity in 3-D space. To calculate the real 3-D space trajectory or positions, extra information is required beyond the single camera model.

For a known simple shaped object, the ratio of area between the actual object and the object in the frame can be used to calculate the distance between object and camera lens in space. However, for any unknown complex shaped objects, this method is not functional or generates significant error overall. A dual camera system is implemented which produces two projected objects in two planes and calculates object position in 3-D space.
2.2.1 Regular Model and Vector-based Algorithm

A dual camera system for trajectory calculation or prediction in 3-D space is shown in Figure 26. The angle between two non-parallel cameras is fixed at 90 degrees in this system. The positions for both two cameras are adjustable and determine the detectable area of the system.

Figure 26: Dual camera system for 3-D tracking. Black: linear motion, Green: non-linear motion

Figure 27 shows the camera locations, camera view sights and detectable area of dual camera system in the xy plane. In the Figure 27 $c_2(x_2, y_2, z_2)$ and $c_1(x_1, y_1, z_1)$ are the locations of camera2 and camera1. $p(x, y, z)$ is the object location in 3-D space. $v_2(a_2, b_2, c_2)$ and $v_1(a_1, b_1, c_1)$ are vectors from cameras to intersection $p(x, y, z)$. The concept of the dual camera system is based on two vectors calculated from two camera frames to get the intersection point position which is the object location in 3-D space.
Figure 27: Dual camera system planar model in xy plane. Red rectangle: camera1, Black rectangle: camera2, Red dot: tracking object, Red arrows: vectors from cameras to object.

The intersection can be calculated by these equations which are shown below:

\[
\begin{align*}
\begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + u \times \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \\
\begin{bmatrix} x \\ y \\ z \end{bmatrix} &= \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + v \times \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}
\end{align*}
\]

(2.2a) (2.2b)

For equations 2.2, unit vector variables \(u\) or \(v\) can be solved by combining two
equations. After that, the intersection \( p(x, y, z) \) can be calculated by using either equation. Overall, it is necessary to get vectors from cameras to object \( v2(a2, b2, c2) \) and \( v1(a1, b1, c1) \) first. A simulated camera model is generated to approximately calculate the vectors. In most cases, object distance is much larger than focal length and the imaging lens model can be replaced by pin hole model. Thus, there is a linear relationship between the object to the center of its plane vertical to camera in 3-D space and the object in frame to the frame center. To simplify these equations, a pin hole model is used and shown in Figure 28.

![Figure 28: Pinhole model in 3-D space.](image)

In the model, \( \alpha \) is the length of frame center \( o_i \) and \( p \) as the middle point of the bottom side of first pixel on the left of the frame center \( o_i \), \( o_s \): center of scene, \( s \): \( p \) projection in the scene plane.
scene plane \( s \), and \( b \) is the distance between pinhole \( o \) and \( o_s \). To find the relationship between scene length \( a \) and object distance \( b \), it can be calculated by using \( \beta/\alpha \). For the adjacent pixel, the ratio \( \beta/\alpha \) also can also be expressed as \( b/a \). Thus, for multiple pixels between \( p \) and \( o_i \), the ratio of scene can be updated by ratio / number of pixels. For the simple pinhole camera or pinhole model, the ratio \( \beta/\alpha \) can be calculated by using hardware parameters. However, for any other modern imaging lens cameras, this ratio requires calibration using real imaging data. By measuring the object size, the distance to the lens, and the counts of pixel in the frame, the ratio can be calculated. After obtaining the ratio value, the previous equations 2.2 can be simplified with the ratio and number of pixels.

The updated intersection equations replaced by ratio and object positions in frame are shown below:

\[
\begin{align*}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} &= \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + u \times \begin{bmatrix} c_1 h \\ \text{ratio} \\ c_1 v \end{bmatrix} \\
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} &= \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + v \times \begin{bmatrix} \text{ratio} \\ c_2 h \\ c_2 v \end{bmatrix}
\end{align*}
\]

(2.3a) (2.3b)

Figure 29 represents an object moving through the camera view with a high speed. The blue dots are continuous tracked centers in multiple frames which follow object trajectory. For a tracked object center, \( c_1 v \) is the value of horizontal position in frame for a tracked object center minus frame center which is the half of the frame length. The \( c_1 h \) is the vertical position in frame for the same object minus half of the frame height. After combining two equations in 2.3 with same intersection \( p(x, y, z) \),
either parameters $u$ or $v$ can be solved. Furthermore, the intersection $p(x, y, z)$ can be calculated by using one of these two parameters.

Figure 29: A simulated camera output for a moving object. Blue dots: tracked object centers for continuous frames, $c1h$: horizontal position in frame – half of frame length, $c1v$: vertical position in frame – half of frame height

2.2.2 Similarity Triangle-based Algorithm

Besides using vectors to locate the intersection, a similarity triangle method can also be implemented to calculate that the position of the intersection. Figure 210 presents similarity method model and the variables that are required in the calculation. By solving equations 2.4, the coordinates of the intersection can be calculated using these solved variables with equation 2.5.
The relationship between distance variables for the similarity triangle model:

\[
a_1 + b_2 = x_1 - x_2 \quad (2.4a)
\]

\[
b_1 - a_2 = y_2 - y_1 \quad (2.4b)
\]

\[
b_1/a_1 = ratio/c_1v \quad (2.4c)
\]

\[
b_2/a_2 = ratio/c_2v \quad (2.4d)
\]

Intersection coordinates obtained by using solved distance variables:

\[
x = x_2 + b_2 \quad (2.5a)
\]

\[
y = y_2 + a_2 \quad (2.5b)
\]

\[
z = (c1h/ratio) \cdot (y - y_1) \quad (2.5c)
\]

### 2.2.3 Lookup Table Algorithm

The previous two algorithms produce accurate intersection coordinates but require more complex computations. To solve equations 2.3, 3 multiplication and 2 division operations are required to obtain parameters \( u \) or \( v \). For coordinate values in \( x \), \( y \) and \( z \) directions, each of them will be calculated with 1 more multiplication operation. For the similarity triangle algorithm in equations 2.4 and 2.5, 2 multiplication and 4 division operations are necessary for solving for the first parameter and each of the rest parameters requires 1 more multiplication operation. For a low level processor or embedded system, multiplication and division operations significantly affect program efficiency especially in real time processing. Thus, to simplify or avoid these operations is essential in the program.

A lookup table array is frequently used to replace runtime computation. For either
Figure 210: A similarity method model in xy plane. Red rectangle: camera1, Black rectangle: camera2, Red dot: tracking object, Red arrows: vectors from cameras to object. a1, a2, b1, b2: distances between object and cameras in x or y directions

The vector-based or the similarity triangle-based algorithm, calculating the object’s position in 3-D space requires both $c1v$, $c1h$, $c2v$ and $c2h$ variables. The lookup table size will be $(nm)^2$ where $n$ and $m$ are resolutions of weight and height. Since the size of lookup table is in proportion to the resolution, a lookup table with all four inputs is inefficient for direct implementation in a storage limited intelligent camera or an embedded system. Based on the equations from previous sections, either $c1h$ or $c2h$ can be used to calculate the object position which improves the size of lookup table to $n^2m$. From the equations $u$ or $v$ can be solved with only two inputs $c1v$ and $c2v$. In addition, a lookup table with two inputs $c1v$ and $c2v$ provides variable
u or v which can be used to calculate the object position in 3-D space with fewer required operations. On the other hand, a lookup table with these two inputs can also give coordinate values in x and y directions directly by using the similarity triangle algorithm with a slightly more complicated calculation to get the value in z direction based on equation 2.5. Even so, the overall size of the lookup table is $n^2$ which is still hard to satisfy with a limited storage.

The size of lookup table can be compressed by reducing all possible inputs, or, in other words, the resolution of the frame. A downscaled simulation frame is used to generate a smaller sized lookup table. A downscaled lookup table model is used for analysing the capability of this algorithm. In the model, an object is moving in the x and y directions but has no velocity in z direction. Thus, two two-input lookup tables are required in this model for calculated object space location in x and y direction. In Figure 211, these simulated $10 \times 10$ pixel grids represent a lower resolution frame. Any detected object will be rounded to the center of its located grid. In addition, each neighborhood processor in the smart camera contains $8 \times 8$ pixels which can directly be the downscaled resolution frame for generating the lookup table. The values of lookup table elements will be calculated as the center positions of NPs and the elements index will be the index of NPs in horizontal direction for both camera 1 and camera 2. If neighborhood processors detect the same object, two NPs in camera 1 and camera 2 will return their horizontal index and get the approximate space position in x and y directions from the lookup table. Admittedly, this approximate position generates significant error because of the scale of lower resolution which is shown in Figure 212.
Figure 211: Simulated camera views in camera1 and camera2. Object is moving from left to right in both views. Blue dots: tracked object centers, Red stars: tracked object centers in both camera1 and camera2 views for calculation.
Figure 212: Object trajectory and positions in xy plane with 10×10 resolution. Camera1 points towards to the positive direction of y axis and camera2 points towards to the positive direction of x axis.

2.2.4 Improved Lookup Table Algorithm

Because the calculated object positions from the reduced resolution lookup table algorithm have glaring random errors, an improved algorithm is implemented by combining the similarity triangle algorithm and the lookup table algorithm to reduce error but maintain the small storage cost with a lower complexity. The concept of the improved lookup table algorithm is to recalculate some data from the lookup table and produce more accurate positions. An improved lookup table model in xy plane is shown in Figure 213. In the previous section, position error is based on distances in x and y directions between the object in frame and central point of it’s located neighborhood processor. Figure 213 presents the actual object location and the lookup table based estimated object location. In this figure, \(dx\) and \(dy\) are distances between lookup table object and target object in x and y directions. Assuming the values of \(dx\) and \(dy\) are known and the lower value determines whether
to use the space position variable in direction x or in direction y from lookup table. In the Figure 213, the lookup table output in y direction is more accurate than in x direction since $dy$ is smaller than $dx$. Thus, the projection in x direction of lookup table output in camera1 to object vector as gray dot can be calculated and it is much closer to actual object position. And a further closer projection in y direction black dot can be computed by using the value in x direction of gray dot. Based on this concept, an approximate projection position produces a better tracking performance compared with using the lookup table only as shown in the figure.

Figure 213: An improved lookup table model in xy plane. Red rectangle: camera1, Black rectangle: camera2, Red dot: tracking object, Red arrows: vectors from cameras to object. Blue dot: LUT output, Blue arrows: vectors for LUT object, Gray dot: projection of blue dot in camera1-object vector, Black dot: projection of gray dot in camera2-object vector
For larger $dx$, keep $y$ value from LUT and calculate new $x$ value by using:

\[ x_{\text{new}} = \frac{y_{\text{lut}} \cdot c1h}{\text{ratio}} \] (2.6a)
\[ y_{\text{new}} = \frac{x_{\text{new}} \cdot c2h}{\text{ratio}} \] (2.6b)
\[ z_{\text{new}} = \frac{x_{\text{new}} \cdot c1v}{\text{ratio}} \] (2.6c)

For larger $dy$, keep $x$ value from LUT and calculate new $y$ value by using:

\[ y_{\text{new}} = \frac{x_{\text{lut}} \cdot c2h}{\text{ratio}} \] (2.7a)
\[ x_{\text{new}} = \frac{y_{\text{new}} \cdot c1h}{\text{ratio}} \] (2.7b)
\[ z_{\text{new}} = \frac{x_{\text{new}} \cdot c1v}{\text{ratio}} \] (2.7c)

To implement this algorithm, it is necessary to compare $dx$ and $dy$ at first without calculating the spatial position. However, equations 2.3 and 2.5 in previous sections provide methods to compute space position with relatively complex calculation. In addition, the comparison of $dx$ and $dy$ is approximately substituted by $fdx \cdot (y_{\text{lut}} - y1)$ and $fdy \cdot (x_{\text{lut}} - x1)$ in frames where $fdx$ and $fdy$ are shown in Figure 214. Approximately, the comparison requires two multiplications for each side which reduces the efficiency of program. Since the equations are in meters, the difference between $fdx$ and $fdy$ is much larger than the difference between $y_{\text{lut}} - y1$ and $x_{\text{lut}} - x1$. Therefore, the comparison can be simplified to compare only $fdx$ and $fdy$ to produce a better computational efficiency.

### 2.2.5 Complexity Analysis

Table 21 presents operation and space complexity for different algorithms where improved LUT($dx$) compares $dx$ and $dy$ and improved LUT($fdx$) compares $fdx$ and
Figure 214: Simulated camera views in camera1 and camera2. Object is moving from
left to right in both views. fdx: horizontal pixel difference between object and center
of NP for camera 1, fdy: horizontal pixel difference between object and center of NP
for camera 2, Blue dots: tracked object centers

f dy. In the table, space complexity represents the size of array or matrix for each
algorithm where m and n are downscaled resolutions. For neighborhood processors
or limited embedded systems, multiplication and division dominate the overall oper-
ational complexity compared with other operations such as addition, subtraction or
comparison. Thus, the numbers of multiplication and division operations are repre-
sented in the table. Since the ratio can be set by adjusting the focal length of the
intelligent camera, division operations in equations 2.6 and 2.7 can be substituted by
bit shift(right shift) if the ratio is in power of 2, further reducing complexity.
Table 21: Operation and space complexity

<table>
<thead>
<tr>
<th>Method</th>
<th>Multi. #</th>
<th>Div. #</th>
<th>Space Complexity</th>
<th>RMS(m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector-based</td>
<td>6</td>
<td>2</td>
<td>$O(1)$</td>
<td>0.0199</td>
</tr>
<tr>
<td>Similarity Triangle</td>
<td>4</td>
<td>4</td>
<td>$O(1)$</td>
<td>0.0176</td>
</tr>
<tr>
<td>Lookup Table</td>
<td>0</td>
<td>0</td>
<td>$O(2mn)$</td>
<td>0.2924</td>
</tr>
<tr>
<td>Improved LUT($dx$)</td>
<td>5</td>
<td>3</td>
<td>$O(2mn)$</td>
<td>0.0431</td>
</tr>
<tr>
<td>Improved LUT($f\ dx$)</td>
<td>3</td>
<td>3</td>
<td>$O(2mn)$</td>
<td>0.0487</td>
</tr>
<tr>
<td>Improved LUT($\text{ratio } 2^n$)</td>
<td>3</td>
<td>0</td>
<td>$O(2mn)$</td>
<td>0.0487</td>
</tr>
</tbody>
</table>

2.2.6 MATLAB Simulation

The dual camera tracking algorithm requires a high speed camera-to-camera communication structure. The current version of neighborhood-level parallel processing camera has to add communication structure or use a separate processing chip to implement these dual camera algorithms. Thus, MATLAB simulations have been used to verify feasibility and performance. To simulate embedded system or neighborhood processor program environment, coding and operations are using Q format to avoid floating point operations. A simulation of object trajectory and position is shown in Figure 215 where each figure is in a different coordinate plane. In the figure, blue crosses represent calculated positions with vector-based algorithm and red circles represent positions with the improved LUT algorithm. Since vector-based algorithm generates most accurate results with pixels rounding error only, calculated positions in each graph mainly follow motion trajectories as green straight lines in the figures. The differences of outputs between vector-based algorithm and the improved LUT algorithm are not obvious in these figures and will be discussed in the next section.
Figure 215: A simulation of object trajectory calculation with $10 \times 10$ scale-down resolution. Camera1 points towards to the positive direction of y axis and camera2 points towards to the positive direction of x axis. Green line: motion trajectory, Red circle: calculated object position with LUT algorithm, Blue cross: calculated object position with vector-based algorithm.


2.2.7 Results

RMS errors for each sample in different directions are shown in Figure 216. In the top two figures, RMS errors of regular lookup table algorithm are significant compared with the other algorithms. For the top figure, average RMS errors of LUT algorithm, vector-based algorithm, improved LUT algorithms with \( dx \) and \( f dx \) comparison are also shown in 21. Since the error of regular LUT algorithm is an order of magnitude larger than others, comparisons of the three other algorithms are represented in the bottom three figures. The most accurate vector-based algorithm generates only half of the error compared with other two improved LUT algorithms. Two improved LUT algorithms have several identical positions but improved LUT algorithm with \( dx \) comparison produces a 0.0056 less error output compared with \( f dx \) comparison. Since the errors between the two improved LUT algorithms are quite similar, \( f dx \) or \( f dy \) comparison can be used to replace \( dx \) or \( dy \) comparison for reducing operational complexity. As a result, the improved lookup table algorithm with \( f dx \) and \( f dy \) comparison and ratio in power of 2 can provide a low requirement of operational complexity with an acceptable storage cost and a small increased RMS error compared with the most accurate vector-based algorithm for limited embedded systems or neighborhood-level parallel processors.

This MATLAB simulation compares and analyses different calculation algorithms to determine a most theoretically suitable algorithm for implementation in neighborhood processors or other embedded systems. For different hardware or platform, these algorithms are used to satisfy different requirements such as limitation of run-time or storage. The actual implementation of a dual camera system using neighborhood processors will be discussed in the future work section.
Figure 216: RMS error analyzing in different directions for each sample by using regular lookup table, vector-based, improved LUT with $dx$ comparison, improved LUT with $fdx$ comparison algorithm.
Chapter 3

Fast Scale, Rotation and Shift Invariant Recognition Application

3.1 Overview

Compared with other image processing tasks, object recognition in embedded systems is much more computationally-intensive. However, the implementation of recognition algorithms with neighborhood-level parallel processors is challenging due to the limited calculation and storage resources. Therefore, complex object categorization or recognition in low level embedded systems or neighborhood parallel processors is difficult to achieve in real-time processing even when a less complicated recognition algorithm is implemented.

In order to maximize the algorithm efficiency and recognition performance, target objects for recognition in neighborhood processing smart camera project are set to basic geometric shapes such as polygons or specific letters. Although the recognition is limited to some simple geometric shapes, it can be applied to distinguish traffic sign, helipad or other symbols. Also, it is possible to construct a scale, rotation and
shift invariant recognition with more rapid execution.

3.2 Recognition Algorithm Analysis

Based on different hardware or targets, recognition algorithms typically belong to either imaging processing or artificial intelligence. Most algorithms contain multiple procedures include pre-processing, quantization and comparison. With more procedures, more accurate results can be generated with more resource consumption. Overall, two different concepts are commonly used to implement recognition algorithms into embedded systems or intelligent cameras with a better performance and efficiency. One method is to use pre-processing or other quantization to simplify the image in the first few steps. After that, the image information can be processed or compared using signal processing approaches such as cross-correlation or root mean squared error. Another way is to simplify an artificial intelligence algorithm such as convolutional neural network (CNN) or a combination of neural network algorithms with other relatively simple embedded algorithms using performance limited hardware.

3.2.1 Convolutional Neural Network

Convolutional neural networks (CNN) as a class of deep, feed-forward artificial neural networks are commonly used to analyse or recognize image since they produces a much higher recognition performance compared with other algorithms. They are constructed with multiple pooling layers and convolutional layers to construct a large feed-forward neural network. Pooling layers are used to combine the output neuron clusters into a single neural in the next layer [9]. Pooling layers are easy to achieve in embedded systems such as max pooling which uses the maximum value from each
of a cluster of neurons at the prior layer [10]. A convolutional layer is applied to calculate convolutional results and neural data pass through filters. Without a GPU (e.g., image processing structure or neural network hardware module) convolutional operations and filtering are difficult to achieve in limited embedded systems such as parallel processors. Based on its own characteristics, CNN requires extra procedures to handle scale, rotation and shift invariant recognition in most cases. Overall, CNN has has good prospects for recognition development but may difficult to implement in the current version of the intelligent camera. This recognition algorithm may still be applied to some high performance vision chip or an upgraded version of neighborhood-level parallel processing camera in the future.

3.2.2 Cross-correlation-based Algorithm

For simple shape recognition, the CNN structure is too complicated and unnecessary, so low complexity algorithms are more suitable for implementation. For signal processing, cross-correlation is used to measure the similarity of two signals. It has been applied in image processing by computing the cross-correlation from small blocks of the input image and the target image. Since the operation of cross-correlation is similar to the block convolution in CNN, the operation complexity is still quite high for limited embedded systems. However, cross-correlation does not require multiple layers which reduces computational complexity compared with CNNs. Based on operational behavior, regular cross-correlation recognition algorithms are shift invariant but their performance suffers with scale or rotation. Thus, cross-correlation based algorithms do not match the requirement of recognition in neighborhood-level parallel processor and a more efficient algorithm is necessary.
3.2.3 Scale and Rotation Invariant Analysis

In the majority of cases, a CNN is not scale or rotation invariant based on the feature in itself. Similarly, cross-correlation-based recognition algorithms can not deal with image scale or rotation either. To develop scale or rotation invariant algorithms, additional processing procedures are required. Pyramiding in image processing is a multi-scale signal representation which can be used to construct scale invariant image comparison. To generate rotationally invariant recognition, multiple rotated templates are constructed for comparison in most situations. Overall, as additional features, both scale or rotationally invariant recognition algorithms have reduced algorithm efficiency and portability and are not recommended for direct application with limited embedded systems.

3.3 Key Points Algorithm

For a simple closed geometric shape, the length of each edge and the position of each vertex contain a significant amount of object information and geometry. To detect and locate these vertices, the shape of recognized object is determined and it can be compared with reference object. By using efficient vertices detection methods, a key points algorithm is developed and it can provide fast recognition with limited performance devices.

3.3.1 Basic Algorithm Introduction

The vertices detection method for the key points algorithm are derived from the surrounded regions problem in the field of data structures. Instead of computing convolution or cross-correlation at the image level, the processing of the key points
algorithm occurs at the pixel level which provides an $O(\text{mn})$ time complexity where $m$ and $n$ are the column and row resolutions. For a closed acute triangle object, key points detection procedures and sample graphs are shown in Figure 31 and Figure 32. In both figures, the same projected triangular objects are located within the simulated frames. In the beginning, a tracer scans each pixel in left to right and up to down order using a pixel value threshold. After the tracer detects the first object pixel, it starts to search next edge pixels. The tracer searches the next edge pixel by left-up-right-down order relative to its original direction as shown in Figure 31, and the overall tracer trajectory is shown in Figure 32. In most cases, there are no more than two tracer directions on a side so that two registers are used to store tracer directions for each side. When a direction to the next edge pixel is different to either of two stored directions, the tracer located pixel will be recorded as a key point. These detected key points are shown in Figure 32 as dark red pixels.

3.3.2 Additional Processing, Multi-object Detection

In exceptional cases, sides of triangles or other shapes are entirely straight lines in frames as shown in Figure 32 which causes detection error. There is only one tracer direction in a side and a new key point will be detected until the tracer third changes its direction. Although detection errors are ignored for high resolution frames or large projected objects, exceptional key points can be detected by locating the pixel where the tracer reverses its direction as shown by the yellow pixel in Figure 32.

In the previous section, the key points algorithm is limited to acute triangle detections. Thus, additional processing is applied to this algorithm to detect obtuse angles or complex shapes as shown in Figure 33. In the figure, an obtuse triangle has two sides with same tracer directions which requires extra processing to count and record
Figure 31: A projected acute triangular object in a simulated frame. Colored arrows and numbers present next-edge-pixel searching priorities based on left-up-right-down to its original direction. Dark red pixels: detected key points, Dark gray pixels: tracked edges

the number of pixels in the two directions. When tracer first change its direction, the number of pixels in previous direction is stored. The number of pixels in second direction is also stored when tracer changes its direction a second time. The ratio or the difference of these two stored counts are used to compare with the other sides. Finally, when a ratio or difference in two sides changes significantly, a key point is detected as shown in Figure 33.

The scan tracer terminates when it detects the first object pixel or has scanned all pixels in the frame. Thus, to detect multiple objects, additional processing is applied to the algorithm. After the edge tracer detects an edge pixel, it marks the current pixel before moving to another edge pixel by writing a special pixel value. The scan tracer will directly move to right when the pixel is marked as an edge detected pixel. When a scan tracer moves to a marked pixel and its right adjacent pixel has an
Figure 32: A projected acute triangular object in a simulated frame. Black arrows: trajectory of edge detecting tracer, Dark red pixels: detected key points, Yellow pixel: special key point

object value, a triggered register is activated to make scan tracer ignore all object pixels and keep moving until it moves to another marked pixel as shown in Figure 33. This process can avoid duplicate or infinite detection and it still maintains an overall $O(mn)$ time complexity.

3.3.3 Object Comparison

For a fast scale, rotation and shift invariant recognition algorithm, conversion of the coordinates of these key points to lengths of each side is highly efficient for comparing two triangles. In theory, the length of sides comparison generates exception errors for quadrilaterals or any other shapes with more than four sides. However, this method still can be applied for most embedded or low resolution recognitions with relatively simple objects. The Pythagorean theorem $l = \sqrt{x^2 + y^2}$ for conversion contains square and square root operations which do not work for the current project. Thus,
Figure 33: A projected obtuse triangular object in a simulated frame. Black arrows: trajectory of edge detecting tracer, Dark red pixels: detected key points

an approximate pythagorean equation \( l = y + \text{round}(x/3) \) is implemented for replacing the exact equations which highly improves computational efficiency.

After calculating lengths of each side, these data can be used for comparison with reference data. The side lengths of a reference object can be constructed using the same algorithm. Three calculated reference lengths are regrouped in order but with the minimum length first. Also, the three lengths of detected object are regrouped in the same way and can be used to compare with reference lengths directly. One of the series of data for comparison will need to be scaled up or down to make the first length in both series equal. Finally, the difference between the two scaled series can be used to determine the goodness of fit for the comparison. A match is selected based on a set threshold.
Figure 34: Two projected triangular objects are in a simulated frame. Black arrows: object detecting tracer, Dark red arrows: edge detecting tracer, Dark gray pixels: marked detected edge

### 3.3.4 Simulation and Results

The feasibility analysis of the key points algorithm is presented in a MATLAB simulation program. The simulation program randomly generates a triangle and projects it to a simulated $64 \times 80$ frame. The algorithm detects and highlights sides and key points as shown in Figure 35. The key points algorithm provides accurate recognition when recognized object has complete and clean stepped edges in the frame. However, this algorithm may detect dummy points if the edge has an unusual shape such as a single raised pixel. Erroneous recognitions can be eliminated or reduced by setting more suitable thresholds when detecting edge pixels.

Overall, the key points algorithm has no complicated frame level image operation such as convolution or cross-correlation and most of the operations are required comparisons. For other frame level recognition algorithms, computational complexities increase exponentially when resolution increases. However, pixel level recognition
Figure 35: Simulated recognitions of triangular objects with key points algorithm. Light gray pixels: detected object edges, White pixels: detected key points

algorithms provide linear increases in similar conditions. Although the key points algorithm has a lower deviation correction ability and a low recognition accuracy for complicated shape objects, it provides a good balance between storage and computational performance for simple shaped objects in performance limited vision processors or embedded systems.

3.4 Recognition Algorithm Application

Since key points algorithm work better when detecting simple shapes rather than complex objects, applications with this algorithm are more related to distance mea-
surement, tracking, coordinate orientation, etc. An indoor orientation application of mini unmanned aerial vehicles (UAV) with the key points detection algorithm is discussed in this section.

3.4.1 Application Introduction

For unmanned aerial vehicles (UAV), commonly known as drones, GPS provides precise orientation and navigation when drones can receive GPS signals. However, GPS typically cannot be used indoor where signal strength becomes weak. In most cases, indoor drones have difficulty maintaining altitudes and positions without GPS support. For some of the latest drones with bottom view cameras, drifts can be detected or prevented by comparing images from these cameras. To achieve indoor orientation, distance sensors or local orientation signal generators are commonly applied. However, most distance sensors have maximum and minimum range limits and local orientation requires additional transmitter hardware.

Figure 36: Drone camera models for bottom view camera and front pitch-adjustable camera.
An application with key points algorithm can provide accurate indoor orientation with a simple graphic reference. To detect the size of an imaged reference in the camera frame, the vertical distance between camera plane and referenced object plane can be calculated. In Figure 36, a bottom view camera generates a scaled image with a limited recognition range but front adjustable camera generates a two-dimensional deformation image with a much wider recognition range. The relative coordinate position of a drone can be calculated by computing the relative shapes between original reference and image objects.

A MATLAB model is applied for simulating and analyzing deformations of right triangles with different camera positions and directions. The camera model is based on pinhole imaging and the image in the frame is projected by the straight line
through pinhole and the original object. In the model, the camera lens points to the centroid of the object in the scene. The original object scene is shown in Figure 37 and every pixel in the original object scene are projected to the simulated camera frame through a simulated pinhole. Simulated frames with different camera positions and directions are shown in Figure 38 and Figure 39. By comparison, not only the orientations of these projected triangles but also the lengths of these triangle sides are different from each other. As a result, these images with varying positions of vertices or lengths of sides can be used to determine the position coordinates of the camera.

3.4.2 Neural Network Analysis

Since the relationship of images and coordinate positions of a camera can be difficult to deduce through regular methods such as non-linear regression, an artificial neural network is applied to analyze the imaging deformation. Through training by using the imaging data and the location data, the neural network with trained weights can be applied to predict the camera coordinate position. To implement a neural network on a performance limited vision chip or a low-level embedded system, the properties of the network such as learning algorithm, model type, activation function, the number of neurons and the number of hidden layers are carefully considered.

Since complex neural networks greatly reduce the efficiency of calculation for real-time recognition, a single hidden layer feed-forward network is applied to verify feasibility and performance at first. Based on the key points algorithm developed in the previous section, the locations of vertices in the frame can be obtained and converted to lengths of sides. Either lengths of sides or positions in x and y direction of vertices can be used as inputs for neural network. Six variables for position contain more information compared with three side lengths and provide better neural net-
Figure 38: Simulated images for right triangle with different camera positions and directions in $100 \times 100$ pixel frames.
Figure 39: Simulated images for right triangle with different camera positions and directions in $100 \times 100$ pixel frames.
work performance, but more inputs increase the complexity of the neural network as additional weights and neurons are required.

### 3.4.3 Side Lengths Model

For the first neural network model, three unordered side lengths obtained from key points algorithm as inputs are used for network training and 15 neurons are applied in the hidden layer as shown in Figure 310. A neural network data training model randomly generates images based on different space positions of the camera as shown in Figure 311. A data set of side lengths as inputs and a data set of camera coordinate positions as targets are applied for neural network training with MATLAB built-in backpropagation algorithm.

![Neural network structure](image)

*Figure 310: Neural network structure*

The neural network training regression of the unordered side lengths inputs model with 793.7(mm) total root mean squared error is shown in Figure 312. The regression graph represents relationship between training targets and neural network outputs and these errors are shown as black circles. Also, a trained outputs and errors comparison graph is shown in Figure 313. The neural network performance is not ideal since the overall root mean squared error is significant especially in x or y directions.
The position of camera \( p(x, y, z) \) is random where \(-500 < x < 1500, -500 < y < 1500\) and \(200 < z < 3200\) (mm).

### 3.4.4 Vertices Model

The nonlinear relationship between unordered side lengths and camera location is obtained from the neural network output. The deformation of imaging is too complicated so that it is difficult to directly implement a single hidden layer neural network with unordered side lengths inputs. Thus, RMS is hardly be reduced by simply adding more neurons. In Figure 38a and 38d, similar side lengths with same order are calculated from key points algorithm which confuse the training of neural network or generate output conflicts. This drawback produces poor performance for the neural network outputs in x or y directions. However, outputs in z direction are more accurate compared with outputs in other directions. For a neural network,
nonlinear problems can be normally solved by multi-layer models such as CNN in the previous section. Thus, RMS can be decreased by adding more hidden layers as a test. However, the increase of ANN complexity significantly affects the efficiency of a real-time implementation on neighborhood processors or other vision chips which is not an optimized solution.

Since unordered side lengths from the key points algorithm reduce neural network
performance, this problem can be solved by generating more accurate object direction data for inputs to the neural network. An efficient way to achieve this is to highlight a vertex of an object and its imagings are shown in Figure 314. The key points algorithm in this case will refrain from calculating until the edge detecting tracer detects the first highlighted vertex. The side lengths calculation process is not used in this model and the frame positions of vertices are applied as six inputs for the neural network training. The trained outputs for this vertices model are shown in Figure 315. Since the positions of vertices contain more information including scale,
rotation and deformation, the figure represents a better trained performance in both x, y and z directions. However, the vertices model neural network with six total inputs produces twice computational complexity compared with three inputs side length model neural network which is still the most efficient solution.

### 3.4.5 Spherical Model

In the side lengths model neural network, the performance in the z direction obtained low RMS compared with other directions as shown in Figure 313. This feature represents a single hidden layer feed-forward neural network performs scale or deformation processing rather than rotation. To improve three-input neural network performance, ordered side lengths data from a highlighted vertex triangle are used as inputs and position data in spherical coordinate system are used as target outputs for training. The outputs of the spherical model neural network are shown in Figure 316.

This model sacrifices the performance of azimuthal angle and some performance of polar angle for a better accuracy in radial distance. Although errors of polar and
azimuthal angle are significant in the test model, output of radial distance is good enough for implementation.

3.4.6 Implementation and Results

The implementation of orientation still requires more accurate polar and azimuthal angle values besides radial distance which is from spherical model. For most UAVs, compasses are used to obtain frame directions in static or moving status. To combine offset angles from the compass and yaw of the camera, a more accurate space
azimuthal angle in spherical coordinate is determined. Also, for a drone with pitch-adjustable camera, polar angle can be determined by directly reading the pitch value in the hardware. Thus, all three variables in spherical coordinate can be obtained for camera orientation and can also be converted to a cartesian coordinate system. Although the conversion from spherical coordinate to cartesian coordinate contains four sine or cosine operations, these operations can be simplified by using sinusoidal approximations such as Bhaskara I’s sine approximation formula. The outputs and errors of the converted spherical neural network model with additional polar and az-
imuthal angle values is shown in Figure 317. In the figure, this model provides a much more accurate neural network outputs compared with other models.

![Graph](image)

Figure 317: Neural network training performance for final implementation model.
Blue: target positions, Red: neural network outputs, Yellow: errors

Table 31: Root mean squared error for different ANNs in each direction

<table>
<thead>
<tr>
<th>RMS(mm)</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th># inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Side lengths model</td>
<td>589.9</td>
<td>530.1</td>
<td>225.2</td>
<td>3</td>
</tr>
<tr>
<td>Vertices model</td>
<td>193.6</td>
<td>172.9</td>
<td>203.7</td>
<td>6</td>
</tr>
<tr>
<td>Hybrid model</td>
<td>34.2</td>
<td>24.6</td>
<td>47.4</td>
<td>3</td>
</tr>
</tbody>
</table>

The comparison of root mean squared errors and amount of inputs for different neural network models is shown in Table 31. From the table, the hybrid model con-
tains much higher accuracy compared with other models and a small number of inputs which reduces the operational complexity of the neural network. Also, the orientation system can be optimized for more operational efficiency. If spherical coordinates are used throughout the whole system including calculation, navigation and orientation then spherical-cartesian conversion can be avoided. When the number of neurons reduces to 10, the neural network increases RMS by 15%. The neural network with 10 neurons reduces 30% operational complexity compared with 15 neurons for limited hardware implementation. As a result, this UAV application which provides accurate orientation without GPS based on the imaging of a triangle reference can be applied for indoor formation flight, automatic flight, UAV carrying, indoor maintenance, etc, without altitude or distance sensors.
Chapter 4

Conclusion and Future Work

4.1 Future Work

4.1.1 Dual Camera System Implementation

The implementation of a dual camera tracking algorithm with the current version of the neighborhood-level parallel processing vision chip requires a higher communication speed between two cameras or an additional control and processing unit for coordinating each vision chip. Since the structure of current version of the processor mainly uses serial communication to achieve data transmission, this serial communication significantly reduces the overall processing speed. Thus, it is necessary to apply a parallel communication structure between the two cameras in order to maintain a high operational speed in real time. Alternatively, an additional control and processing unit for computing both data from each camera can be applied to the dual camera system with a low transmission efficiency but a high processing speed.
4.1.2 Optimization and Implementation of Neural Network

In order to obtain better neural network performance and lower complexity, an optimization for parallel processors or other embedded system is required. The current neural network is based on MATLAB simulator which is still difficult to be implemented in low level embedded systems. Moreover, implementation of the neural network requires particular optimization for satisfying the hardware specifications. Thus, the parameters of the neural network, including the number of neurons, layers, the type of activation functions, etc are determined by adapting to the corresponding hardware for better orientation performance and efficiency.

4.2 Conclusion

In this work, a smart camera application of real-time trajectory calculation and an application of scale, rotation and shift invariant recognition are described. The trajectory application contains calculation and prediction in frame level with a single intelligent camera and calculation of object coordinate position with a dual camera system. A key points algorithm is designed and applied to provide quick and accurate processing on a neighborhood-level processor or other embedded system. To construct a relationship between the camera coordinate position and the projected object in the frame with a neural network, the application of the key points recognition algorithm is available to locate the camera position via imaging.


### Bibliography


