A TOOL FOR TRACKING OBJECTS THROUGH V1KU, A NEURAL NETWORK SYSTEM

Hitesh Wadhwani
B.E, Gujarat University, India, 2007

PROJECT

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010
A TOOL FOR TRACKING OBJECTS THROUGH V1KU, A NEURAL NETWORK SYSTEM

A Project

by

Hitesh Wadhwani

Approved by:

______________________________, Committee Chair
Dr. V. Scott Gordon

______________________________, Second Reader
Dr. Du Zhang

______________________________
Date
Student: Hitesh Wadhwani

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the Project.

__________________________, Graduate Coordinator       ________________
Dr. Nikrouz Faroughi                          Date

Department of Computer Science
Abstract

of

A TOOL FOR TRACKING OBJECTS THROUGH V1KU, A NEURAL NETWORK SYSTEM

by

Hitesh Wadhwani

The intent of this project is to explore the tracking capabilities of V1KU a neural network system. V1KU is a product by General Vision Company that comprises of CogniMem neural network chip for real-time image learning and CogniSight image recognition engine. The board also consists of Micron/Aptina monochrome CMOS sensor for visual input. The board has powerful capability to learn and recognize objects simultaneously within a fraction of a second. Due to this ability an application is developed which uses board’s capabilities to track a learned object in real-time.

The development of this application has gone through various phases of experiments as during initial development stages the board was quite new and very little support was available. After applying the methodology of trial and error I was able to achieve a real-time tracking working with this board. The people at General Vision also gave their inputs on how to optimize the code so that tracking works efficiently. The board has the capabilities to track multiple objects simultaneously, but at this present time the goal is to effectively track a single object. The new version of the board with casing came out recently which has some mounting space that can be
utilized in future to mount servo motors to automate the tracking process. The output of this
application forms a basis for stereoscopic tracking of various objects in real-time.

_______________________, Committee Chair
Dr. V. Scott Gordon

_______________________
Date
DEDICATION

To my father and mother who let me pursue my dreams.
ACKNOWLEDGEMENTS

Before going into the details of this project, I would like to add few warm words for the people who gave me support, directly and indirectly to complete this project. It was pleasure that Dr. V Scott Gordon allowed me to work on this project with him. Dr. V. Scott Gordon was instrumental to my success in this project and I must thank him for providing all the necessary resources and help to finish this project. I would like to thank people of General Vision Company for providing all the support and knowledge about the V1KU board. I would also like to thank Dr. Du Zhang for being the second reader in this project. My wholeheartedly thanks go to all the faculty members of Computer Science and Software Engineering Department for helping me finish my graduation at California State University, Sacramento. Lastly special thanks to friends who have always cheered me during my tough times and have provided their valuable support and advice, especially Bhumi Patel and Hemal Mehta.
# TABLE OF CONTENTS

Dedication ....................................................................................................................................... vi

Acknowledgements ........................................................................................................................ vii

List of Tables ................................................................................................................................... x

List of Figures ................................................................................................................................... xi

Chapter

1 INTRODUCTION AND MOTIVATION .................................................................................... 1

1.1 Related Work ....................................................................................................................... 2

2 BACKGROUND .......................................................................................................................... 4

2.1 Architecture of CogniMem Neural Network Chip .............................................................. 8

2.2 Learning an Object .............................................................................................................. 9

2.3 Recognizing an Object ...................................................................................................... 10

3 INTERFACING WITH V1KU ................................................................................................... 12

3.1 V1KU Registers ................................................................................................................ 14

4 SOFTWARE DESIGN ............................................................................................................... 19

4.1 Development Environment ................................................................................................ 19

4.2 High Level Application Structure ..................................................................................... 20

4.3 Internal Application Architecture ...................................................................................... 21

4.3.1 Main Application Class ............................................................................................ 21
LIST OF TABLES

1. Table 1: List of CogniMem Registers [6] ................................................................................. 16
2. Table 2: List of CogniSight Registers [7] .................................................................................. 18

x
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Figure 1: Hardware/Software Overview of the Project</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Figure 2: V1KU within Metal Casing</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Figure 3: V1KU’s Metal Casing with Mount Space</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Figure 4: V1KU without Metal Casing</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Figure 5: V1KU Components [5]</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>Figure 6: CogniMem Inside [3]</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>Figure 7: Learning Example</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>Figure 8: Recognition Example</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>Figure 9: Class Diagram of C# DLL</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>Figure 10: High Level Flow Diagram of Application</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Figure 11: Class Diagram of Added Classes</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>Figure 12: Learning an Object</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Figure 13: Tracking the Learned Object (Locate box is ticked)</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>Figure 14: Tracking the Learned Object after Moving the Object within Frame</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>Figure 15: V1KU Board with Stepper/Servo Motor</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>Figure 16: Triangulation using two V1KU Boards</td>
<td>29</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION AND MOTIVATION

The V1KU is a board made by General Vision Company, comprised primarily of a neural network chip, a camera, a FPGA and a USB port all on the same board [1]. By combining the power of artificial intelligence and efficient image recognition system, the V1KU board is very good and accurate at learning an object quickly and also at same time recognizing the learned object promptly. The ability of the image recognition relies on the parallel neural network which is capable of learning by example and generating models automatically. The neural network can recognize patterns that are identical or similar to the models stored in the neurons. These powerful capabilities of quick image recognition make it a great device for tracking the objects. Several practical systems have been developed such as the fish inspection [2] utilizing V1KU’s powerful capabilities. However a system in which one can track objects utilizing V1KU hasn’t yet been developed. The figure below gives an overall idea of how tracking is implemented utilizing V1KU board.

Figure 1: Hardware/Software Overview of the Project
The final deliverable of the project is a .NET C# application (called “BirdView”) that communicates with the V1KU and tracks the object in real time to show in which region in the camera’s field of view the object resides. The end user who is utilizing the application can make V1KU learn multiple objects by defining different categories (i.e. assigning different names to different type of objects) but the immediate goal is to effectively track a single learned object. Once learned the user can move the object from its learned position to any position within the camera region. The application is then able to locate that learned object despite the displacement from the original position and also will be able track the object’s movement. This project is of practical interest because such a system could later be expanded to perform stereoscopic tracking.

1.1 Related Work

Various works are currently being done in the field of stereoscopic tracking using neural networks. Out of the three closest references that I found with my work, one of them talks about the use of Recursive Shortest Spanning Tree (RSST) approach the second one focuses on Back-propagation Neural Network (BPNN) and the last one uses the Radial Basis Function (RBF) approach used for stereoscopic tracking.

The first reference, Unsupervised Tracking of Stereoscopic Video Objects Employing Neural Networks Retraining [8] deals with the RSST approach. This is a three step approach in which retraining algorithm for adapting the network weights to current conditions is used as first step. The second step involves semantically meaningful object extraction. The final step comprises of a decision mechanism for determining when network retraining should be used or activated. Object extraction is accomplished by utilizing depth information, provided by
stereoscopic video and incorporating a multi-resolution implementation of RSST. This system is still in the development stage and is not yet deployed.

The second reference, Design and Implementation of Neural Network for Real-Time Object Tracking [9] uses the BPNN approach. The approach uses BPNN with one (or more) sigmoid-type hidden layer(s) and a linear output layer can approximate any arbitrary (linear or non-linear) function. The objective of this application was to locate an airplane in the frames grabbed from a movie clip playing at a speed of 25 frames/seconds. This approach has been proposed but not yet implemented.

The third and the last reference, Auto-associative Gaze Tracking System based on Artificial Intelligence [10] uses the RBF approach. This system is currently being used in certain webcams which comes with eye-tracking mechanism. The approach consists of three stages: calibration, training and application. In the first stage the system is calibrated by the user which comprises of collection of gray scale images captured. In the training stage the captured gray scale images are used to train the radial basis function network. Finally the application stage will use the trained neural network for locating eye movements. This system is deployed and is being used effectively in commercial webcams, but is limited to tracking eye movements.
Chapter 2

BACKGROUND

The V1KU board is an evaluation module for the CogniMem [3] and CogniSight [4] technology applied to video and image recognition. The board features a CogniMem chip with 1024 neurons, a high quality aptina monochrome video sensor, a reconfigurable Actel Field Programmable Gate Array (FPGA), 16MB of SDRAM, 4 MB of Flash memory, one high-speed USB2 port, two RS485 ports, 2 opto-relays, and one opto-isolated input line. The image given below shows the V1KU board with metal casing.

The V1KU board is being used...
in harsh environmental conditions such as fish inspection where it’s exposed to water inputs which would damage typical computer systems. General Vision has embedded the board within a metal casing which provides additional ruggedness and safety against odd climates, and adds sturdiness to the V1KU board. The metal casing also comes with a mount screw hole, which can be used to mount the board to a robot or other fixture. Our ultimate plan is to use the mount to attach the board to a servo motor which can rotate the V1KU board based on the position of a located (tracked) object. The application of using a servo motor is beyond the scope of this project but can be used in the future for performing stereoscopic tracking. The image below shows the mount space provided in the metal casing.

Figure 3: V1KU’s Metal Casing with Mount Space
The image below shows the actual teardown of the V1KU revealing the board itself.

![V1KU without Metal Casing](image)

**Figure 4: V1KU without Metal Casing**

The V1KU includes a CogniMem CM1K chip which is a neural network chip and is a high speed non-linear classifier. The CogniMem CM1K chip can learn and recognize pixel data coming directly from the Micron sensor to produce a feature vector. CM1K has many benefits such as high-speed pattern recognition, expandable neural network, low power consumption and trainability. CogniMem learns by examples. It can save and restore the details of what it has learned, and can learn additional images and append those new cases to its stored knowledge.
The V1KU also includes a CogniSight image recognition engine, which resides in the board’s FPGA. The CogniSight engine can be used to identify an object based on variety of learned features or a single feature. The level of accuracy required for recognition is also able to be specified.

On V1KU’s CogniMem chip there are two network models implemented which can be selected: Radial Basis Function (RBF) and K-Nearest Neighbor (KNN). The RBF model is highly adaptive and is ideal for non-linear applications. It is focused on learning and recognizing objects quickly rather than generalizing. KNN is more suitable for exact pattern matching and accurate recognition. The image below shows all the components of V1KU board.

Figure 5: V1KU Components [5]
2.1 Architecture of CogniMem Neural Network Chip

CogniMem stands for Cognitive Memory and is breakthrough technology for massive parallel pattern identification and classification as well as anomaly and novelty detection. The concept of CogniMem was invented in 1993 by Guy Paillet, founder of General-Vision. CogniMem is a neural processor for high-speed pattern learning, recognition and decision making. The CogniMem is a silicon chip with identical neuron cells connected in parallel and operating simultaneously. The operation works at low clock frequency and consequently low-power consumption. Recognition and learning cycles are independent from the number of neurons in use. The chip can be cascaded with no decrease in speed performance and no change to inter-neural associations. The contents of the chip can be saved and loaded on the fly. CogniMem chip uses both Radial Basis Function and K-Nearest Neighbor classifiers. The image is close-up photograph of the CogniMem chip.

![CogniMem Inside](image)

*Figure 6: CogniMem Inside [3]*
2.2 Learning an Object

Before going into the details of how to learn an object using V1KU there are certain terms that the user must be familiar with. The first is Region of Interest (ROI) which is a small rectangular area including the object. So whenever the user executes the application, a small window comes up on the screen which is ROI. The user then manually positions the ROI on the object which he/she wants to learn. The second term is Region of Search (ROS) which is again a small rectangular area where the recognition engine CogniSight will search for any learned object. So, when the user has made the V1KU learn a particular object and then wants the V1KU to recognize the learned object, at that time the V1KU will search for any learned object within the ROS. The last term is Vector Count (VO) which is an array that maintains counts for each neuron that successfully recognizes an object. It also includes the X, Y coordinates and RecoDist (distance variable) from the learned object. The figure below gives an overview on how an object is learned by V1KU.

![Learning Example](image)

**Figure 7: Learning Example**

Learning an object using V1KU board usually consists of 3 simple steps. The first step is to specify the ROI such that it contains the particular object which you want to learn. The next step is to name or categorize the object. Categorization is important when learning multiple
objects. The last step is to specify that the V1KU “learn” what is contained in the ROI, which will make V1KU learn that particular object. Despite the apparent simplicity, in the CogniMem chip, particular neurons stores particular features including the user-defined category of that particular object along with pixel information. At the same time the incoming data from the image is stored in the VO array.

2.3 Recognizing an Object

After the user has made V1KU learn a particular object, it is then capable of identifying that learned object. Recognizing the learned object consists of 2 steps. The first step is to move the ROS window to the object which the board has learned. If the user had learned several objects of different categories the user may select a particular category of object which he/she wants the board to recognize. The final step is to specify that the V1KU “recognize” which will trigger V1KU to recognize any learned object in the particular ROS window. It is common to set the ROS slightly larger than the ROI, so that if the object has moved slightly, it can still be identified.

![Recognition Diagram](image)

**Figure 8: Recognition Example**

Here similar to learning an object, the front scenario looks very straightforward but at the back the CogniSight engine present in the FPGA scans the whole ROS window in a snake pattern to identify any successful hits. If any of the patterns present in the ROS window matches one or
more neurons, then the VO array will contain the nearest X, Y coordinates of the image which matches the selected category. The CogniSight engine which is high-speed image recognition engine tries to identify the signatures of the ROS window with the data stored in neurons. The engine creates an array of distance between the ROS and the information in the VO array. The application can then use this distance information to find the coordinates with the closest match.
Chapter 3

INTERFACING WITH V1KU

General Vision provides a C#.NET DLL for communicating with the V1KU. The DLL was quite new at the start of this project, as the engineers at General Vision had just developed it, so there was very little documentation available on what functions were available in the DLL and what they do. Instead of the documentation they provided Easy Trainer software [11] with source code. The Easy Trainer software handled basic learning and recognizing of objects but did not do tracking.

Easy Trainer was limited to learning and recognizing of objects with the ROI and ROS in fixed position (settable by the user), but did not move the ROS automatically and thus could not track learned objects. So the DLL was the only source which had to be explored by experimenting with different functions and variables. After some weeks of rigorous experiments with the DLL and using trial and error methodology, rudimental tracking was achieved. After these successful results, a meeting at General Vision with the results of our experimentation enabled us to document some of the useful DLL calls. The class diagram below lists all the DLL calls used by V1KU.
Figure 9: Class Diagram of C# DLL
BirdView only uses a fraction of the DLL function calls available. The following functions and variables were useful:

- **V1KU_CogniSight**: V1KU CogniSight’s class object which will contain method calls defined in the CogniSight class.
- **V1KU_Sensor**: V1KU Sensor’s class object which will contain method calls defined in the Sensor’s class.
- **CS_CSR**: Control Status Register variable is used for setting status register which triggers V1KU into different modes such as learn ROI, recognize ROI and scan the ROS.
- **CS_CATL**: Category variable for learning the category entered by the user.
- **CS_ROILEFT**: Variable which stores the left position of the ROI window.
- **CS_ROITOP**: Variable which stores the top position of the ROI window.
- **VO_COUNT**: Variable which maintains the stored vector count of neurons.
- **VO_X**: Vector object’s X co-ordinate stored value.
- **VO_Y**: Vector object’s Y co-ordinate stored value.
- **ROIDIST**: Variable which maintains the distance in form of vector of the learned object in neuron.
- **CS_ROSLEFT**: Region of Search window’s left position value.
- **CS_ROSTOP**: Region of Search window’s top position value.
- **CS_ROSWIDTH**: Region of Search window’s width value.
- **CS_ROSHEIGHT**: Region of Search window’s height value.

3.1 V1KU Registers

The CogniMem CM1K neural chip contains various network registers and neurons which are mentioned in the table below. The last column showing an asterisk (*) shows the usage of that register in BirdView application.
<table>
<thead>
<tr>
<th>Register</th>
<th>Hex Code</th>
<th>Description</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM_NCR</td>
<td>0x00</td>
<td>Bit[6:0], neuron context Bit[7], norm Bit[15:8], neuron identifier[23:16]</td>
<td>0</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_COMP</td>
<td>0x01</td>
<td>Component</td>
<td>0</td>
<td>W, R/W</td>
</tr>
<tr>
<td>CM_LCOMP</td>
<td>0x02</td>
<td>Last Component</td>
<td>0</td>
<td>W, R/W</td>
</tr>
<tr>
<td>CM_DIST</td>
<td>0x03</td>
<td>Distance (Range 1 to 65535). A distance 0 means that vector matches exactly the model stored in top neuron A distance of 32767 (0xFFFF) means that no neuron has fired and vector is not recognized</td>
<td>0xFFFF</td>
<td>R *</td>
</tr>
<tr>
<td>CM_CAT</td>
<td>0x04</td>
<td>Bit[14:0], category value ranging between 1 and 32767 Bit[15], Degenerated flag which indicates that vector is recognized, but close to zone of uncertainty If category is equal to 65535, the vector is unknown</td>
<td>0xFFFF</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_AIF</td>
<td>0x05</td>
<td>Active Influence Field</td>
<td>0x4000</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_MINIF</td>
<td>0x06</td>
<td>Minimum Influence Field</td>
<td>2</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_MAXIF</td>
<td>0x07</td>
<td>Maximum Influence Field</td>
<td>0x4000</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_NID</td>
<td>0x0A</td>
<td>Neuron identifier[15:0]</td>
<td>0</td>
<td>R</td>
</tr>
<tr>
<td>CM_GCR</td>
<td>0x0B</td>
<td>Global Norm and Context</td>
<td>1</td>
<td>W</td>
</tr>
<tr>
<td>CM_RESET</td>
<td>0x0C</td>
<td>Point to the 1\textsuperscript{st} neuron</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>CHAIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM_NSIR</td>
<td>0x0D</td>
<td>Network Status Register Bit[2], Uncertain status Bit[3], Identified status Bit[4], SR mode Bit[5], KNN Classifier</td>
<td>0</td>
<td>R/W</td>
</tr>
<tr>
<td>CM_FORGET</td>
<td>0x0F</td>
<td>Clear neuron registers, but not their memory. Also reset Minif, Maxif and GCR global register.</td>
<td></td>
<td>W</td>
</tr>
<tr>
<td>CM_NCOUNT</td>
<td>0x0F</td>
<td>Return the number of committed neurons</td>
<td></td>
<td>R *</td>
</tr>
<tr>
<td>CM_RSR</td>
<td>0x1C</td>
<td>Recognition Status Register Bit[0], Real-time recognition login ON Bit[1], Output bus enable Bit[2], Uncertain recognition status Bit[3], Identified recognition status</td>
<td>0</td>
<td>R/W</td>
</tr>
</tbody>
</table>
Bit[4], Frame valid if VI_EN=1
Bit[5], Recognition in progress

CM_RTDIST 0x1D Real-time distance or the distance of the neuron with the best match 0xFFFF R *

CM_RTCAT 0x1E Real-time category or the category of the neuron with the best match 0xFFFF R *

CM_LEFT 0x11 Left position of the ROI 200 R/W *

CM_TOP 0x12 Top position of the ROI 120 R/W *

CM_NWIDTH 0x13 Width of the ROI 340 R/W *

CM_NHEIGHT 0x14 Height of the ROI 220 R/W *

CM_BWIDTH 0x15 Width of the inner block 20 R/W

CM_BHEIGHT 0x16 Height of the inner block 20 R/W

CM_ROIINT 0x1F Reset ROI to default W

<table>
<thead>
<tr>
<th>Register</th>
<th>Hex Code</th>
<th>Description</th>
<th>Default</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_CSR</td>
<td>0x60</td>
<td>Control Status Register (AUTO_RESET register) Bit [0], grab Bit [1], recognize ROI Bit [2], learn ROI Bit[3], scan the ROS and append the position and category of all recognized ROIs in a hit list</td>
<td>0</td>
<td>R/W</td>
</tr>
<tr>
<td>CS_LEFT</td>
<td>0x61</td>
<td>Left position of ROI</td>
<td>200</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Table 1: List of CogniMem Registers [6]

The V1KU also comes with reconfigurable image recognition CogniSight as described in the previous chapter. CogniSight also contains a list of registers which are used heavily in the BirdView application. The table below shows the list of registers which CogniSight uses for image recognition. The last column showing an asterisk (*) shows the usage of that register in BirdView application.
<table>
<thead>
<tr>
<th>Field</th>
<th>Address</th>
<th>Description</th>
<th>Value</th>
<th>Mode</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS_TOP</td>
<td>0x62</td>
<td>Top position of ROI</td>
<td>120</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_RECODIST</td>
<td>0x63</td>
<td>Distance of the last processed ROI</td>
<td>0xFFFF</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_RECOCAT</td>
<td>0x64</td>
<td>Category of last processed ROI</td>
<td>0</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_CATL</td>
<td>0x65</td>
<td>Category to Learn</td>
<td>1</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_ALEFT</td>
<td>0x66</td>
<td>Left position of the ROS</td>
<td>0</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_ATOP</td>
<td>0x67</td>
<td>Top position of the ROS</td>
<td>0</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_AWIDTH</td>
<td>0x68</td>
<td>Width of the ROS</td>
<td>752</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_AHEIGHT</td>
<td>0x69</td>
<td>Height of the ROS</td>
<td>480</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td>CS_HITCOUNT</td>
<td>0x6A</td>
<td>Number of identified ROIs in ROS</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_HITLEFT</td>
<td>0x6B</td>
<td>Left position of the next ROI in the hit list.</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_HITTOP</td>
<td>0x6C</td>
<td>Top position of the next identified ROI</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_HITDIST</td>
<td>0x6D</td>
<td>Distance of the next identified ROI</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_HITCAT</td>
<td>0x6E</td>
<td>Category of the next identified ROI</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_INIT</td>
<td>0x6F</td>
<td>Reset all the above to their default values</td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CS_PAGENUM</td>
<td>0x70</td>
<td>Page number in the memory map to store a video frame.</td>
<td>0</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>CS_FWIDTH</td>
<td>0x71</td>
<td>Width of the image in memory</td>
<td>752</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>CS_FHEIGHT</td>
<td>0x72</td>
<td>Height of the image in memory</td>
<td>480</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>CS_STEPX</td>
<td>0x73</td>
<td>Horizontal scanning step</td>
<td>16</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>CS_STEPY</td>
<td>0x74</td>
<td>Vertical scanning step</td>
<td>16</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td>Address</td>
<td>Description</td>
<td>RW</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>CS_RSR</td>
<td>0x75</td>
<td>Recognition Status Register&lt;br&gt;Bit[2:0] describe what to report as a Hit Point found in the ROS:&lt;br&gt;000, all recognized objects&lt;br&gt;001, all unknown objects&lt;br&gt;010, objects recognized with certainty&lt;br&gt;100, objects recognized with uncertainty&lt;br&gt;Bit[3] = reserved&lt;br&gt;Bit[4] = 1, launch continuous grab&lt;br&gt;Bit[5] = 1, automatic ROI recognition after grab&lt;br&gt;Bit[6] = 1, automatic ROS scanning after grab&lt;br&gt;Bit[8] = 1, do not use CogniSight reco-logic but use the CogniMem reco-logic</td>
<td>RW</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>CS_MINHIT</td>
<td>0x76</td>
<td>Minimum number of identified ROI at the end of a scanning</td>
<td>1</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>CS_FCOUNT</td>
<td>0x77</td>
<td>Number of consecutive frames to append automatically in memory</td>
<td>1</td>
<td>RW</td>
<td>*</td>
</tr>
<tr>
<td>CS_FCOUNTED</td>
<td>0x78</td>
<td>Number of consecutive frames appended in memory since the last Write CS_FCOUNT</td>
<td></td>
<td>R</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: List of CogniSight Registers [7]
Chapter 4
SOFTWARE DESIGN

4.1 Development Environment

The application ("BirdView") was developed using Microsoft Visual Studio 2008 which utilizes the .NET 3.5 framework. The application is created from the base application provided by General Vision. As mentioned in the previous chapter General Vision provided the Easy Trainer application which handled basic learning and recognizing of objects. Easy Trainer software is intended for use with the V1KU module when configured with the default CogniSight engine for object recognition. The application lets users adjust the video acquisition settings, define a region of interest in the video images, and teach the neurons what users want to recognize. However there are certain limitations of the Easy Trainer software which includes objects cannot be tracked in real-time environment; they can only be recognized, limited adaptivity to change of scale and orientation and limited to 7 categories of object of the same size. Easy Trainer software is easy to use, but does not show the full potential of the CogniMem technology available on the camera.

The application ("BirdView") extends and explores the capabilities of V1KU which was previously used for only learning and recognizing objects but now can be used to track objects in real-time. The neural network chip on the V1KU utilizes a radial-basis architecture that lends itself to specific object recognition rather than generalization so the application concentrates on identifying and tracking specific trained objects rather than finding similar objects. The software also displays the camera image for visual validation.
4.2 High Level Application Structure

The above figure shows how the application works. The application starts, it asks the user if the object is already learned or not. If the object is not learned the user then focuses ROI box on the particular object and enters the category for the same. If the object is already learned, the user can select the category of the object which it wants to recognize and track. Upon selection of recognition the object is recognized with a box on it in the image and is also tracked simultaneously. If the user does not want to either learn or recognize the object, the user can also quit the application which is the final state shown in the diagram.

Figure 10: High Level Flow Diagram of Application
4.3 Internal Application Architecture

4.3.1 Main Application Class

The main application class initializes various variables which are required by all the other classes and functions. The class also creates instance of V1KU_CogniSight and V1KU_Sensor class. These class variables are already defined in the C# DLL provided by General Vision. Initial video frame canvas, positions of ROI and CogniSight variables are set in this class.

Figure 11: Class Diagram of Added Classes

The diagram shows the class structure and relationships of the added classes. It includes:
- Main Application
  - V1KU_CogniSight
  - V1KU_Sensor
- Learn Object
  - V1KU_CogniSight.CATL
- Recognize Object
  - V1KU_CogniSight.CSR
- Track Object
  - V1KU_CogniSight.CS

These components are integral to the application's functionality, providing a clear and structured approach to handling various aspects of image processing and analysis.
4.3.2 Learn Object

Learning an object involves several steps which could be better explained using pseudocode:

1. Set CogniSight.CATL category which is entered by the user.
2. Set CogniSight.CSR = 4 which triggers V1KU to learn the object present in ROI.
3. Learn CogniSight.ROILEFT, CogniSight.ROITOP for North West, North East, South West and South East.
   3.1 While learning ROILEFT, ROITOP make V1KU unlearn surrounding objects by setting CATL=0
4. Set the ROILEFT and ROITOP to the original coordinates from where it started.

4.3.3 Recognize Object

Recognize an object also involves several steps which could be better explained using pseudocode:

1. Set CogniSight.CSR = 2 which triggers V1KU to recognize ROI.
2. Get the ROIDIST from CogniSight.ROIDIST.
3. Calculate the percentage hit ratio of fetched ROIDIST.
4. Get the Category name of the found object from CogniSight.ROICAT.
5. Display the text which is the identified object category and also matching percentage.

4.3.4 Track Object

Tracking an object is more a complicated task than learning or recognizing an object. Tracking involves a process in which the ROS window expands itself until it is able to locate the object within the frame. Once it finds the object in the window, it moves the ROS window to
center around that object. The following pseudocode describes the complete steps on how tracking works:

1. First test if there are any learned objects or not. This is done by checking CogniMem.NCOUNT value. If returned value is zero then exit.

2. Set the ROS window slightly larger than ROI window.

3. If there are any learned objects learned by neurons then set CogniSight.CSR = 8 which will trigger VIKU to scan ROS and append the position and category of all recognized ROIs in the hit list.

4. Now determine the number of hits during search. This is done by checking CogniSight.VO_COUNT value.
   
   4.1 If value of VO_COUNT is zero then we have to expand the size of ROS window by calling MoveWindow function (step 5.) by passing calculated ROS.X value, calculated ROS.Y value, calculated ROS.WIDTH value and calculated ROS.HEIGHT value. This expands the ROS window 1.5 times than the original size. This can happen repeatedly in which it may call the function until the window is expanded to the actual frame size and it hasn’t found any object.

   4.2 If the value of VO_COUNT is not zero, then we grab the values stored in Vector Object. We scan through the VO array and grab the best value which is determined by the nearest ROIDIST present in the array.

   4.2.1 If the best value which is grabbed from VO array is not the present ROS window coordinates then we call the MoveWindow function (step 5.) by passing the best value grabbed from VO array which contains X and Y coordinates. Thus the new ROS is centered on the learned object which we were able to locate within the frame.
5. MoveWindow function has four input parameter which accepts all integer co-ordinate values. It performs calculation on the input values and moves the ROS window according to passed and then calculated values.
Chapter 5

RESULTS

The results were astonishing when one observed the speed of object learning, recognizing and tracking all occurring in fraction of seconds. Here the application developed has the basic layout in which it continuously captures the incoming video stream on the canvas and displays it to the user. Initially the user sees a ROI (Region of Interest) window (the square marked “REC”) which is used for focusing on object which is to be learned.

![Image](image.png)

**Figure 12: Learning an Object**

After the object is learned, the object is recognized simultaneously. The user now has an option to locate (track) the object in the frame of the camera.
If the object is moved within the frame, then the ROS (Region of Search) window (the now slightly enlarged square) will expand gradually (if necessary) to search for the learned object within the frame. Upon successful hit, the ROS window will locate the object and center on it. If no learned object is found in the frame region the ROS window will expand itself until it finds one.

![Figure 13: Tracking the Learned Object (Locate box is ticked)](image)

In the above image the object is being tracked, so now if the user moves the object manually to any place within the region of the frame, V1KU will be able to track the object’s new location. Figure 14 shows the result upon movement of the object.
Figure 14: Tracking the Learned Object after Moving the Object within Frame
Chapter 6

FUTURE WORK

V1KU has incredible capabilities which can be explored more and can be used for stereoscopic tracking in 3D space. This project serves as the base foundation for more to be done in this field. One future enhancement which I see is attaching a servo motor to the V1KU board. Presently in the application if the learned object moves out of the frame, the board searches for the object and the user has to move it manually if he wishes to center the object on the screen. By attaching a stepper servo motor, the application can get the coordinates of the object and can control the movement of the servo motor which would move the board according to object’s movement, keeping the ROS centered on the camera window. The figure below gives a future enhancement overview.

Figure 15: V1KU Board with Stepper/Servo Motor
By adding the servo motor we would be able to add autonomous control to the board. However another enhancement would be having two V1KU boards which are connected to servo motors on a common base. This type of arrangement would lead to real stereoscopic tracking of object in 3D space. The servo motors would facilitate triangulating the coordinates passed by both the V1KU boards (after centering the object in each camera) and would move both of them in accordance to the movement of the object. The below figure gives an overview of the proposed concept of two V1KU boards connected to two servo motors.

Figure 16: Triangulation using two V1KU Boards
Another enhancement would be offloading the application to the board itself. Presently the application runs on the host machine, so sometimes the user may witness delays in tracking the objects due to the transfer of the images across USB. By offloading the application on the board’s flash memory, the processing would be faster which would result in faster recognition and tracking of objects. The camera images are not necessary for tracking, and are there only for visual confirmation that the system is working.
APPENDIX A

Code

//main partial class
int ObjSizeW = 64;   // Object learning ROI size width
int ObjSizeH = 64;   // and height
int Offset = 5;      // Distances for NW, SW, SE, NE
int FPS = 0;
int RecoDist = 0;
int RecoCat = 0;
bool Learning = false;
V1KU_CogniSight myCS = new V1KU_CogniSight();
V1KU_Sensor mySensor = new V1KU_Sensor();
Rectangle ROI = new Rectangle();
Rectangle ROS = new Rectangle();
Pen ROIPen = new Pen(Color.Green, 2);
SolidBrush ROIBrush = new SolidBrush(Color.Green);
Pen ROSPen = new Pen(Color.Red, 2);
SolidBrush ROSBrush = new SolidBrush(Color.Red);
string dString;
Font dFont = new Font("Ariel", 12);
String myVersion;

public Form1()
{
    InitializeComponent();
    if (myCS.Connected==true)
    {
        mySensor.Comm = myCS.Comm;
        myVersion = myCS.Version;
        mySensor.SetBinning(2);     // 2=Halfsize Window 1=Fullsize
        myCS.CogniMem.FORGET = 0;
        ROI.X=156;
        ROI.Y=88;
        ROI.Width=ObjSizeW;
        ROI.Height=ObjSizeH;
        myCS.ROILEFT = ROI.X;
        myCS.ROITOP = ROI.Y;
        myCS.ROIWIDTH = ROI.Width;
        myCS.ROIHEIGHT = ROI.Height;
        myCS.BWIDTH = 4;
        myCS.BHEIGHT = 4;
        myCS.ROSSTEPX = 4;
        myCS.ROSSTEPY = 4;
    }
}
else Application.Exit();
}

// Learn Method
private void Learn(int Category)
{
    myCS.CATL = Category;
    myCS.CSR = 4;
    // Learn NW (North West)
    myCS.ROILEFT = ROI.X - Offset;
    myCS.ROITOP = ROI.Y - Offset;
    myCS.CATL = 0; myCS.CSR = 4; myCS.CATL = Category; myCS.CSR = 4;
    // Learn NE
    myCS.ROILEFT = ROI.X + Offset;
    myCS.ROITOP = ROI.Y - Offset;
    myCS.CATL = 0; myCS.CSR = 4; myCS.CATL = Category; myCS.CSR = 4;
    // Learn SW
    myCS.ROILEFT = ROI.X - Offset;
    myCS.ROITOP = ROI.Y + Offset;
    myCS.CATL = 0; myCS.CSR = 4; myCS.CATL = Category; myCS.CSR = 4;
    // Learn SE
    myCS.ROILEFT = ROI.X + Offset;
    myCS.ROITOP = ROI.Y + Offset;
    myCS.CATL = 0; myCS.CSR = 4; myCS.CATL = Category; myCS.CSR = 4;

    myCS.ROILEFT = ROI.X;
    myCS.ROITOP = ROI.Y;
}

// Recognize Method
private void Recognize()
{
    myCS.CSR = 2;
    RecoDist = myCS.ROIDIST;
    RecoDist = RecoDist / 100;
    RecoDist = 100 - RecoDist;

    if (RecoDist < 0)
        RecoDist = 1;

    RecoCat = myCS.ROICAT;
    if (RecoCat != 0)
        txtDist.Text = Convert.ToString(RecoDist) + " %";
    else
        txtDist.Text = "0 %";
}
private void FindObject()
{
    int Dist, storeX, storeY, count, coX, coY, changed;
    if (myCS.CogniMem.NCOUNT == 0) return;
    myCS.CSR = 8; // CSR=8 causes V1KU to search
    count = myCS.VO_COUNT; // Determine number of hits during search
    if (count == 0)
    {
        // if none, expand window by 1.5x and try again
        labStatus.Text = "Searching Object";
        MoveWindow(ROS.X-ObjSizeW/4, ROS.Y-ObjSizeH/4,
                    ROS.Width+ObjSizeW/2, ROS.Height+ObjSizeH/2);
        txtDist.Text = "0 %";
    }
    else // If there was a hit, find the best hit
    {
        RecoDist = MAX_DIST; // Initialize distance to a MAX number
        changed = 0; // Flag to see if we storeXY are corner or center
        storeX = ROS.X; // Start with the old value
        storeY = ROS.Y;
        for (int i = 0; i < count; i++) // For each hit, check distance
        {
            coX = myCS.VO_X; // Always grab X first
            coY = myCS.VO_Y; // when grabbed, they are at center
            Dist = myCS.VO_DIST;
            if (Dist < RecoDist)
            {
                storeX = coX; // Remember the best hit in search
                storeY = coY;
                RecoDist = Dist;
                changed = 1;
            }
        }
        labStatus.Text = ";";
        RecoDist = RecoDist / 100;
        RecoDist = 100 - RecoDist;
        if (RecoDist < 0)
            RecoDist = 1;
        txtDist.Text = Convert.ToString(RecoDist) + " %";
        if (changed == 1)
        {
            storeX = storeX - (ObjSizeH*3)/4; // if changed, move to corner
            storeY = storeY - (ObjSizeW*3)/4;
        }
        MoveWindow(storeX, storeY, (ObjSizeW*3)/2, (ObjSizeH*3)/2);
    }
}
// MoveWindow Method
private void MoveWindow(int CtrX, int CtrY, int W, int H)
{
    ROS.X = CtrX;    // CtrX and CtrY are upper left corner of window
    ROS.Y = CtrY;
    ROS.Width = W;
    ROS.Height = H;

    if (ROS.X <= 0) ROS.X = 1; // Make sure it doesn't go out of bounds
    if (ROS.Y <= 0) ROS.Y = 1;
    if (ROS.X + ROS.Width >= 376) ROS.Width = (376 - ROS.X) - 1;
    if (ROS.Y + ROS.Height >= 240) ROS.Height = (240 - ROS.Y) - 1;

    myCS.ROSLEFT = ROS.X;   // set the region of search on the V1KU
    myCS.ROSTOP = ROS.Y;
    myCS.ROSWIDTH = ROS.Width;
    myCS.ROSHEIGHT = ROS.Height;
}

// If User moves the ROS/ROI window, then Mouse Move Method
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        if (e.X < 0)
            ROI.X = 0;
        else
            ROI.X = e.X;

        if (e.X + ROI.Width > pictureBox1.Width)
            ROI.X = pictureBox1.Width - ROI.Width - 1;

        if (e.Y < 0)
            ROI.Y = 0;
        else
            ROI.Y = e.Y;

        if (e.Y + ROI.Height > pictureBox1.Height)
            ROI.Y = pictureBox1.Height - ROI.Height - 1;

        myCS.ROILEFT = ROI.X;
        myCS.ROITOP = ROI.Y;
        Recognize();
    }
}
BIBLIOGRAPHY


