

國立交通大學

電機與控制工程學系

碩 士 論 文

用在移動式機械臂之嵌入式影像處理平台

An Embedded Image Processing Platform for a  
Mobile Manipulator



研 究 生：沈 栢 瀚

指 導 教 授：宋 開 泰 博 士

中華民國九十三年七月

用在移動式機械臂之嵌入式影像處理平台

An Embedded Image Processing Platform for a Mobile  
Manipulator

研究生：沈柏瀚

Student: Andrian, Henry

指導教授：宋開泰 博士

Advisor: Dr. Kai-Tai Song

國立交通大學

電機與控制工程學系

碩士論文



A Thesis

Submitted to Department of Electrical and Control Engineering  
College of Electrical Engineering and Computer Science  
National Chiao-Tung University  
in Partial Fulfillment of the Requirements  
for the Degree of Master  
in  
Electrical and Control Engineering  
July 2004  
Hsinchu, Taiwan, Republic of China

中華民國 2004 年 7 月

# 用在移動式機械臂之嵌入式影像處理平台

研究生：沈 栢 瀚

指導教授：宋 開 泰 博士

國立交通大學電機與控制工程學系

## 摘 要

本論文的目的在於發展硬體及軟體以實現移動式機械臂的控制系統。此控制系統採用的是影像資訊之迴授。在硬體實現方面包括一 CMOS 影像擷取系統以及使用德州儀器 C6416 主處理器之數位信號處理發展板。使用所建構之嵌入式平台之好處在於可以相對低的價格來提供高效能之影像處理。在軟體實現方面本文提出了透過基於行為模式的方式去設計一視覺伺服系統，以解決移動式機械臂的控制問題。

# An Embedded Image Processing Platform for a Mobile Manipulator

Student: Andrian, Henry      Advisor: Dr. Kai-Tai Song

Department of Electrical and Control Engineering  
National Chiao-Tung University

## Abstract

This thesis aims to construct a hardware and software implementation that provides a control scheme for a mobile manipulator. The control scheme uses visual information as feedback. The hardware implementation consists of a CMOS image board and a TI DSP DSK board which uses C6416 as the main processor. The merits of using these two boards are the construction of a low cost embedded platform with high performance. Meanwhile the software implementation proposed a visual servoing using a behavior-based approach to solving the control scheme of the mobile manipulator problem.

# Acknowledgments

First and certainly foremost, I wish to acknowledge my advisor, Dr. Kai-Tai Song, for his continuous encouragement, insight, patient guidance and invaluable contribution throughout this study. It was his hard work that has ensured that I was always supported both financially and with the equipment my research required.

I would like to thank Dr. Kuu-Young Yang, Dr. Yu-Lun Huang and Dr. Kun-Wei Lin, for their comments and suggestions for the editing of my thesis.

I would like to thank my uncle Ir. Muis Moxsin with his family for giving and supporting financial aid for me to study at National Chiao Tung University (NCTU).

I would like to thank my family, for providing me with the opportunity to undertake my studies at National Chiao Tung University (NCTU). They have provided me with endless support and opportunity to reach new heights.

Lastly, I thank my colleagues in ISCI lab, Chia-How Lin, Chih-Chieh, Chih-How and Yao-Qing for sharing experiences and knowledge during the time of study. I thank my senior Ph.D student Chi-Yi for his helping and guidance during the time of study.

# CONTENTS

|  |            |
|--|------------|
| <b>ABSTRACT(CHINESE)</b> .....   | <b>I</b>   |
| <b>ABSTRACT(ENGLISH)</b> .....   | <b>II</b>  |
| <b>ACKNOWLEDGE</b> .....   | <b>III</b> |
| <b>CONTENTS</b> .....  | <b>IV</b>  |
| <b>LIST OF TABLES</b> .....  | <b>VI</b>  |
| <b>LIST OF FIGURES</b> .....   | <b>VII</b> |
| <b>1 . INTRODUCTION</b> .....  | <b>1</b>   |
| 1.1. Motivation .....  | 1          |
| 1.2. Background and Related Work .....   | 2          |
| 1.2.1. Behavior-based controller.....  | 3          |
| 1.2.2. Visual servoing controller.....   | 4          |
| 1.2.2.1. Position-based visual servo (PBVS) control .....                      | 5          |
| 1.2.2.2. Image-based visual servo (IBVS) control .....                         | 6          |
| 1.2.2.3. Camera configuration .....  | 7          |
| 1.3. Problem Statements .....  | 9          |
| 1.4. Organization of the thesis .....  | 10         |
| <b>2 . EMBEDDED IMAGE PROCESSING PLATFORM.....</b>                             | <b>11</b>  |
| 2.1. System Overview.....  | 11         |
| 2.1.1. Stand-alone .....   | 12         |
| 2.1.2. Communicate directly with other module .....                            | 12         |
| 2.2. CMOS sensor.....  | 15         |
| 2.3. DSK6416 daughter board .....  | 20         |
| 2.3.1. Frame buffer (AL422B).....  | 21         |
| 2.3.2. 7 ports GPIO of DSK6416 for emulating 4 COM ports RS232 interface ..... | 25         |
| 2.4. FPGA (Altera UP2) .....   | 31         |
| 2.4.1. I2C module .....  | 32         |
| 2.4.2. Clock divider module .....  | 34         |
| 2.4.3. Buffer_controller module .....  | 34         |
| 2.4.4. Implementation of the Buffer_controller.....                            | 36         |
| 2.4.4.1.Acquired image for 15 fps .....  | 36         |
| 2.4.4.2.Acquired image for 30 fps .....  | 39         |
| 2.5. DSK6416 Board .....   | 42         |
| 2.5.1. EMIF.....   | 43         |
| 2.5.2. EDMA (Enhanced DMA) controller .....                                    | 45         |
| 2.6. Experimental result Image processing .....                                | 47         |
| <b>3 . VISUAL SERVOING</b> .....   | <b>50</b>  |
| 3.1. System architecture .....   | 50         |
| 3.2. Switching mode .....  | 53         |
| 3.3. Barcode-like Object .....   | 54         |

|            |  |            |
|------------|--|------------|
| 3.3.1.     | Definition.....  | 54         |
| 3.3.2.     | Remarks.....   | 57         |
| 3.4.       | Barcode-like identification .....                          | 58         |
| 3.5.       | Foot motion control .....                                  | 61         |
| <b>4 .</b> | <b>OBSTACLE AVOIDANCE USING OPTICAL FLOW .....</b>         | <b>66</b>  |
| 4.1.       | Introduction to the algorithm.....                         | 66         |
| 4.2.       | Implementation of mixed optical flow algorithm .....       | 67         |
| 4.2.1.     | Acquired two subsequent images .....                       | 69         |
| 4.2.2.     | Down sampling.....   | 69         |
| 4.2.3.     | Smoothness constraing (Pre-processing).....                | 70         |
| 4.2.4.     | Calculate the Ex, Ey and Et.....                           | 72         |
| 4.2.5.     | Search the (u,v) over brightness constraints.....          | 72         |
| 4.2.6.     | Correlation constraint .....                               | 75         |
| 4.2.8.     | Create Depth Histogram or TTC .....                        | 77         |
| 4.2.9.     | Safety distribution histogram .....                        | 79         |
| 4.3.       | Experiment results with mixed optical flow algorithm ..... | 81         |
| <b>5 .</b> | <b>EXPERIMENTAL RESULTS .....</b>                          | <b>83</b>  |
| 5.1.       | The experimental mobile robots.....                        | 83         |
| 5.2.       | Experiment of grasping .....                               | 84         |
| 5.3.       | Experiment of grasping of an object from a person .....    | 89         |
| 5.4.       | Experiment of obstacle avoidance.....                      | 94         |
| <b>6 .</b> | <b>CONCLUSIONS AND FUTURE WORK.....</b>                    | <b>99</b>  |
| 5.1.       | Conclusions .....  | 99         |
| 5.2.       | Future work .....  | 100        |
|            | <b>REFERENCES .....</b>                                    | <b>101</b> |

# LIST OF TABLES

|   |    |
|---|----|
| Table 2.1 List of CMOS sensor built in function module that has been change. .... | 19 |
| Table 2.2 Configuration of McBSP pins as GPIO.....                                | 27 |
| Table 2.3 Baud rate value detect from HR_SoftUartSpeedDetect function. ....       | 29 |
| Table 2.4 Configuration of the CMOS sensor. ....                                  | 34 |
| Table 2.5 DSK6416 board specification.....  | 42 |
| Table 2.6 List of setup time, strobe time and hold time. ....                     | 45 |
| Table 3.1.Barcode-like feature to distance transform.....                         | 60 |
| Table 3.2.The scale factor value for foot motion control. ....                    | 64 |





# LIST OF FIGURES

|   |    |
|---|----|
| Figure 1.1 Subsumption architecture which introduced by Brookds.....                  | 3  |
| Figure 1.2 Position-based visual servo (PBVS) structure.....                          | 5  |
| Figure 1.3 Image-based visual servo (IBVS) structure.....                             | 6  |
| Figure 1.4 Static camera mounted for robotic manipulator .....                        | 7  |
| Figure 1.5 Eye-in-hand camera configuration .....                                     | 8  |
| Figure 2.1 Hardware architecture of embedded image processing.....                    | 13 |
| Figure 2.2 Software architecture of embedded image processing .....                   | 14 |
| Figure 2.3 EVT202 and its application .....   | 16 |
| Figure 2.4. Frame buffer (AL422B).....  | 22 |
| Figure 2.5.AL422B Functional block diagram .....                                      | 23 |
| Figure 2.6 Daughter board for 4 COM ports RS232 interfaces .....                      | 25 |
| Figure 2.7.Block diagram of McBSP pins configuration for 4 COM ports.....             | 27 |
| Figure 2.8.UART Auto Baud rate detection.....   | 28 |
| Figure 2.9 HR_SoftUartInChar UART data fetch in .....                                 | 29 |
| Figure 2.10 HR_SoftUartOutChar UART data fetch out.....                               | 30 |
| Figure 2.11 Altera UP2 board and its specification .....                              | 31 |
| Figure 2.12 The complete control signal of FPGA, DSK6416 and CMOS sensor.....         | 33 |
| Figure 2.13 I2C module .....  | 32 |
| Figure 2.14 Clock divider .....   | 34 |
| Figure 2.15 Buffer controller module .....  | 35 |
| Figure 2.16 Full timing diagram for 15 fps configuration .....                        | 37 |
| Figure 2.17 Timing diagram of Frame Buffer in writing stage.....                      | 37 |
| Figure 2.18 Timing diagram of Frame Buffer in reading stage .....                     | 38 |
| Figure 2.19 Full timing diagram of writing and reading stage .....                    | 40 |
| Figure 2.20 Full timing diagram of 30 fps configuration.....                          | 40 |
| Figure 2.21 Timing diagram for reading and writing stage in 30 fps configuration..... | 41 |
| Figure 2.22 DSK6416 board with its main list components .....                         | 42 |
| Figure 2.23 Basic EMIF asynchronous control signal .....                              | 43 |
| Figure 2.24 EMIF control signal.....  | 44 |
| Figure 2.25 The configuration of the EDMA controller .....                            | 45 |
| Figure 2.26 The flow chart of the implementation of the EDMA transfer .....           | 46 |
| Figure 2.27 Original image of the CMOS sensor .....                                   | 48 |
| Figure 2.28 Applying sobel mask operation in the original image .....                 | 48 |
| Figure 2.29 Applying binary operation in the original image.....                      | 49 |
| Figure 3.1. Hand-in-eye camera configuration .....                                    | 51 |
| Figure 3.2. Architecture of the visual servoing based on behavior-based.....          | 52 |
| Figure 3.3. Communication line between each behavior module.....                      | 53 |
| Figure 3.4. Flow chart of grasping mode .....   | 55 |
| Figure 3.5. Barcode-like feature as the main feature of our object.....               | 56 |

|  |    |
|--|----|
| Figure 3.6. The flow chart of barcode-like identification.....   | 59 |
| Figure 3.7. The model of non-holonomic mobile robot system.....  | 61 |
| Figure 3.8. Foot motion control .....  | 65 |
| Figure 4.1. Flowchart of mixed optical flow algorithm.....   | 67 |
| Figure 4.2. Flowchart of mixed optical flow algorithm implement in embedded image<br>processing platform ..... | 68 |
| Figure 4.3. EDMA transfer for down sampling method .....   | 70 |
| Figure 4.4. Smoothness constraint of optical flow.....   | 71 |
| Figure 4.5. Sobel mask to calculate Ex, Ey and Et .....  | 72 |
| Figure 4.6. The imaging geometry .....   | 73 |
| Figure 4.7. The motion constraint line .....   | 74 |
| Figure 4.8. Brightness constraint for possibly flow motion.....  | 74 |
| Figure 4.9. Correlation constraint algorithm.....  | 76 |
| Figure 4.10.a. The camera moves toward the object.....   | 78 |
| Figure 4.10.b. The object moves toward the camera .....  | 78 |
| Figure 4.11. The representation of 3D space to 2D ZX-plane .....   | 80 |
| Figure 4.12.a Optical flow field .....   | 81 |
| Figure 4.12.b Histogram of TTC (7 region).....   | 81 |
| Figure 4.13.a Optical flow field .....   | 82 |
| Figure 4.13.b Histogram of TTC (7 region).....   | 82 |
| Figure 5.1. Mobile manipulator (H2) platform .....   | 84 |
| Figure 5.2. The robot is faced to the object that put in top of chair .....                                    | 84 |
| Figure 5.3.a The mobile robot is turned left about 30° .....   | 85 |
| Figure 5.3.b The mobile robot is turned right about 30° .....  | 86 |
| Figure 5.4. The robot change its orientation and speed after it locates the object in front of<br>it .....     | 87 |
| Figure 5.5.a The mobile manipulaotr is approaching the object.....   | 87 |
| Figure 5.5.b The mobile manipulaotr is approaching the object (cont'd).....                                    | 88 |
| Figure 5.6.a The object is get into the area of the gripper .....  | 88 |
| Figure 5.6.b The mobile manipulator grasp the object .....   | 89 |
| Figure 5.7. The mobile manipulator robot is faced to the human that hold the object .....                      | 90 |
| Figure 5.8.a The robot is turned left about 30° .....  | 91 |
| Figure 5.8.b The robot has located the object that holds by human.....   | 91 |
| Figure 5.9.a The robot is approaching the object.....  | 92 |
| Figure 5.9. b The robot is approaching the object (cont'd).....  | 92 |
| Figure 5.10. a The object is in the gripper area.....  | 92 |
| Figure 5.10. b The gripper grasped the object .....  | 92 |
| Figure 5.11. The trajectory of the mobile manipulator .....  | 92 |
| Figure 5.12. The robot orientation .....   | 94 |
| Figure 5.13. The trajectory approach of the avoid obstacle.....  | 96 |
| Figure 5.14.a. ....  | 96 |
| Figure 5.14.b. ....  | 96 |


Figure 5.14.c ..... 97  
Figure 5.14.d..... 97  
Figure 5.14.e. .... 97  
Figure 5.14.f..... 97  
Figure 5.15.the trajectory approach of the mobile navigation experiment ..... 98



# 1. Introduction

## 1.1. Motivation

There are many existing mobile robots with a manipulator such as XR4000 platform [1][2], Stanford mobile platform [3], COSMOS system [4], YAMABICO Type-Ten[5] and our previous self-constructed guide-robot mobile manipulator[6] and etc. These existing mobile manipulators somehow use a personal computer (PC) as the main processor for computation and control of the mobile manipulator. There are fewer researchers who focus on embedded platform for control of the mobile manipulator.



So it motivates us in this study to develop an embedded platform for controlling mobile manipulation that are less expensive using low cost components, and can operate independently like a module. This module has its main processor, operate together and also communicate directly with other module such as motion control board or PC. This embedded platform will have built-in vision board, so there is no need for an additional vision system.

Later, the functional of this embedded platform can be used to control the mobile manipulator to locate certain object and grasp it; meanwhile the robot can also move autonomously to avoid obstacles. Therefore by using this embedded platform, the computational load of the main processor (PC) of the manipulator robot can be reduce and the robot can do any other task such as face recognition, localization and etc. Also these tasks can also interact or combine with the manipulation task to complete a more complex task.

## 1.2. Background and related work

The advances of robotics research in the last two decades have led to new generations of robotics system and new scenarios for applications. The successful introduction of robots into human environments will depend on the development of competent and practical systems that are dependable, safe and easy to use. To work, cooperate, assist, and interact with human, this new generation of robots need mechanism that accommodates interaction while fitting into our unstructured, sizeable and unpredictable environments.

Many methodologies and techniques have been proposed by many researchers to address this challenge. Many robots have also been developed into the human environment to help human for doing tasks such as cleaning (*Roomba* which is produced by iRobot Corporation), elderly-care (*Wakamaru*, which is produced by Mitsubishi Heavy Industries) and the extreme one is humanoid robots (*Asimo* [7], which is produced by Honda) which is able to accurately understand and respond to a range of human motions. Another kind of a new application of robots is personal-assistant robots. This robotic capability to aid human is in the accomplishment of a variety of physical operations and presents various control strategies developed for vehicle-arm coordination.

Like a common personal assistant robot, our lab also introduced a new kind application of a personal-assistant which is so-called Easybot (Easy is pronunciation of ISCI Lab, while bot is Robot). Easybot is a personal assistant robot which is equipped with manipulator that can interact with people and environment autonomously. The term interact here means that the

robot can recognize people, locate certain object and grasp it and move autonomously in dynamic environment with its manipulator attached on it.

This new emerging application involved a depth study of mobile manipulation research area. Because it is not just a mobile robot but there is a manipulator attached onto the mobile robot the system introduces a new control scheme problem in the scope of the mobile manipulation robotics area.

### 1.2.1. Behavior-based controller

In the early stage of the mobile manipulator research, J. H. Connell introduced a behavior-based arm controller [8]. The goal of the system is to locate and retrieve empty soda cans in an unstructured environment using a variety of local sensors. The arm controller consists of 15 independent behaviors which each of these behaviors contains some grain of expertise concerning the collection task and cooperates with the other to accomplish its goal. The structure of the arm controller is based on a *subsumption architecture* [9], which is proposed by Brooks (see figure 1.1).

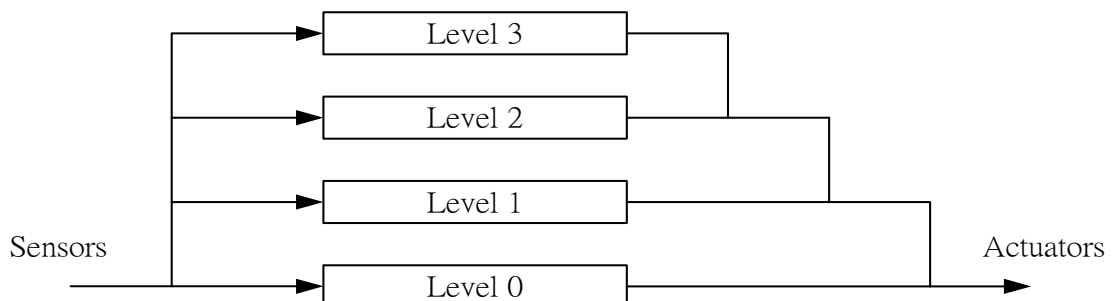


Figure 1.1 subsumption architecture which introduced by Brooks [9]

Each of the 15 independent behaviors run concurrently, in real time, on a set of eight loosely coupled on-board 8-bit microprocessors. In the experimental result, the arm controller work well in practice and can be used in many different environment. The advantage of this control scheme is that the control problem of mobile manipulator can be solved by behavior-based controller that can relax the kinematic of manipulator and the motion control problem.

Later work that also used behavior-based but combined with fuzzy logic [10] and a hierarchical behavior based [11] as the control scheme is introduced by Wasik and Saffioti. However but their implementation is limited to the control of a robot manipulator of 5 DOF (degree of freedom) to do a pick and place task. In the behavior-based design, they introduced a hierarchical behavior-based design that consists of a simple behavior that implement a *simple* control strategy and complex behavior by *combining* these simple behaviors. Meanwhile the fuzzy logic is used to fuse behaviors in the control scheme.



## 1.2.2. Visual servoing controller

Recent advances in vision sensor technology and image processing allow the effective use of vision data in the control of a robot. Based on this fact, many researchers are focusing to use vision sensor data as the feedback control for their control scheme. This kind of controller is called visual servoing controller. In the tutorial on visual servo control [12], the problem of the visual servoing control scheme and configuration of the vision sensor on the robot is described.

### 1.2.2.1. Position-based visual servo (PBVS) control

In the position-based visual servoing control (see figure 1.2), features ( $f$ ) are extracted from the image and used to estimate the pose of the target with respect to the camera. Using these values, an error between the current ( $cx_a$ ) and desired pose ( $cx_d$ ) of the robot is defined in task space (Cartesian coordinate). These errors are then given to the control law to correct the current position to the desired position and still in the task space (Cartesian coordinate). Finally the result of control law in Cartesian coordinate is transformed to the joint controller coordinate by computing the inverse kinematics for each joint of the manipulator.

The advantage of this configuration is that the Cartesian control law is easy to design. Respectively, the disadvantage over this configuration is that the camera need calibration related to the camera and the manipulator position that introduce some camera calibration error, and somehow the pose estimation can waste the time because of the computing delay.

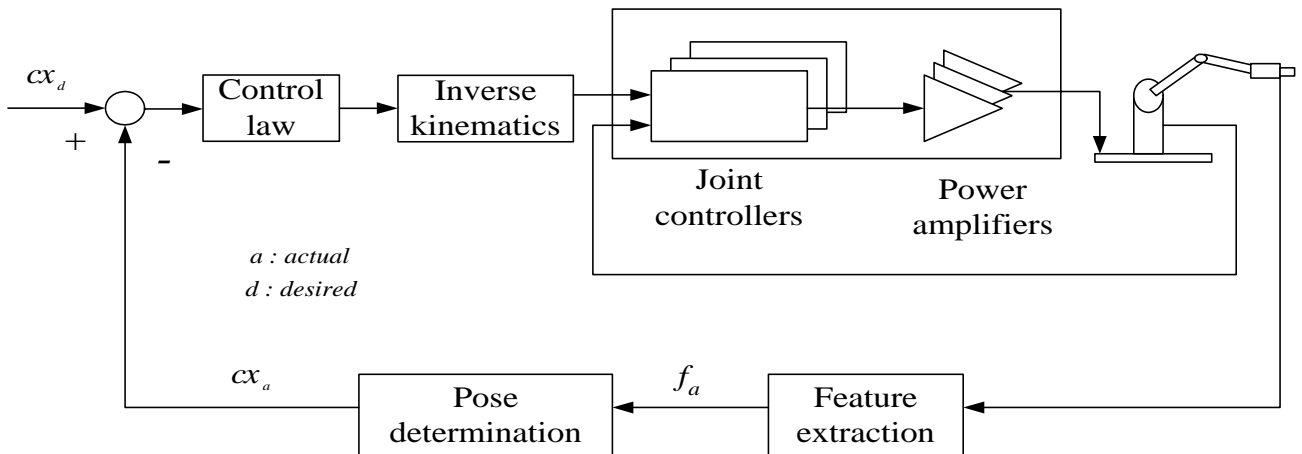


Figure 1.2 Position-based visual servo (PBVS) structures [12]



### 1.2.2.2. Image-based visual servo (IBVS) control

In image-based visual servo control (see figure 1.3), feature ( $f$ ) of desired object is extracted from the image, and will be compared with the desired feature in the image plane ( $f_d$ ) by subtracting the desired feature in the image with the current feature in the image plane. Usually the desired feature is the center ( $x,y$ ) of image, for example: an image with 160 x 120 res. the center of the image is at (80, 60).

Later the error between the current features with the desired feature will be given to the feature space control law to transform the feature into each joint controller of the manipulator, after that the position of the manipulator will change to the new desired position.

The advantage of this configuration is the computational delay may be reduced because there is no need to estimation the pose of the object. Also it can eliminate errors due to camera calibration because in this scheme there is no need to do a camera calibration. The challenge of this configuration is to design the feature space control law.

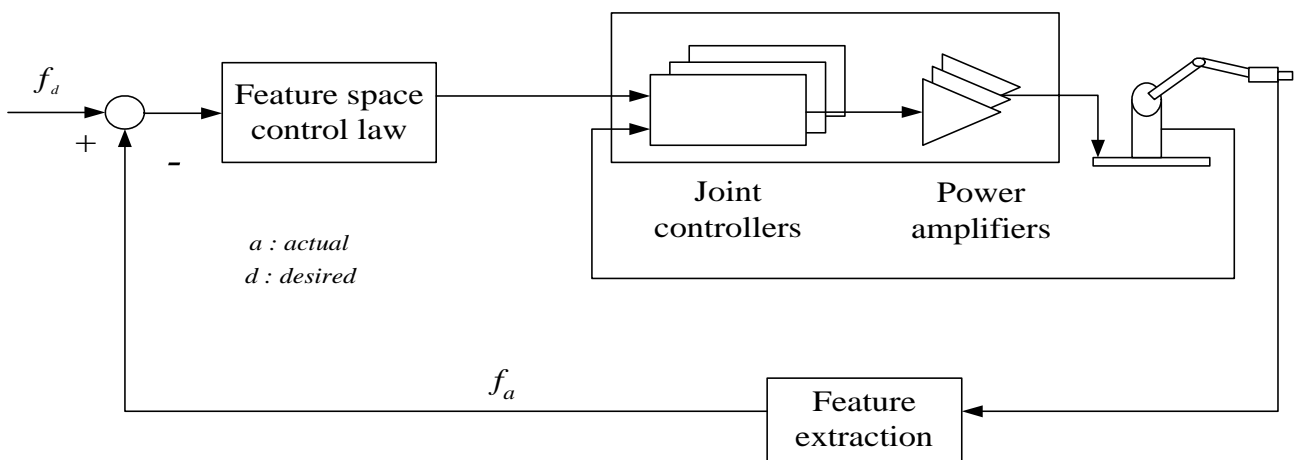


Figure 1.3 Image-based visual servo (IBVS) structures [12]

### 1.2.2.3. Camera Configuration

The choice of the camera mounting position can cause drastic changes in the basic system design. Camera poisoning in visual servoing technique involves a choice between statically mounted camera (see figure 1.4) and robot mounted camera (eye-in-hand configuration) (see figure 1.5).

In a robotic manipulator with fixed bases, the statically mounted camera means the camera position is not in the body of the robot so the camera will act like an observer because the working space of the space already defined.

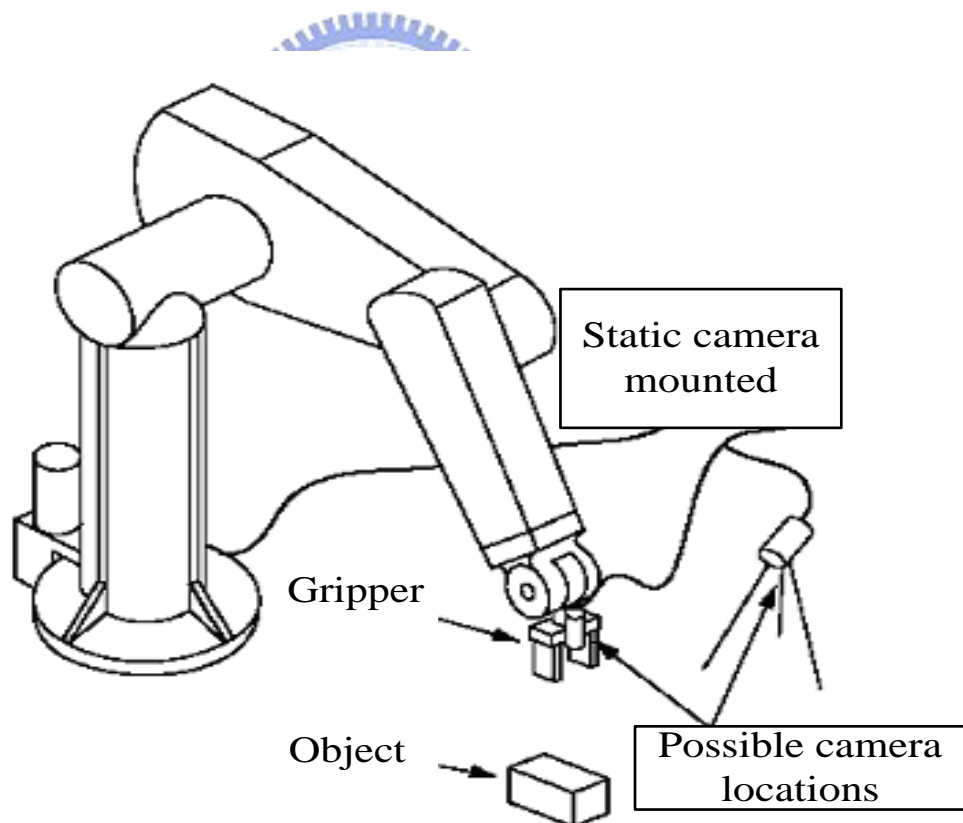


Figure 1.4 static camera mounted for robotic manipulator [13]

In this case the static camera must observe both the manipulator and the object that want to be grasped. Since the camera position is static and it is assumed that any motion energy in the view of the camera is the object to be grasped.

A static camera simplifies the control algorithms, but raises a calibration issue for precise camera-to-manipulator coordinate system transformation. The second configuration of camera positioning is the camera mounted in the manipulator (eye-in hand configuration). In this configuration, the camera does not have to observe the manipulator; it just observes the desired object to be grasped. The choice of an eye-in-hand configuration reduces the need for a calibrated camera-to-manipulator coordinate transformation.

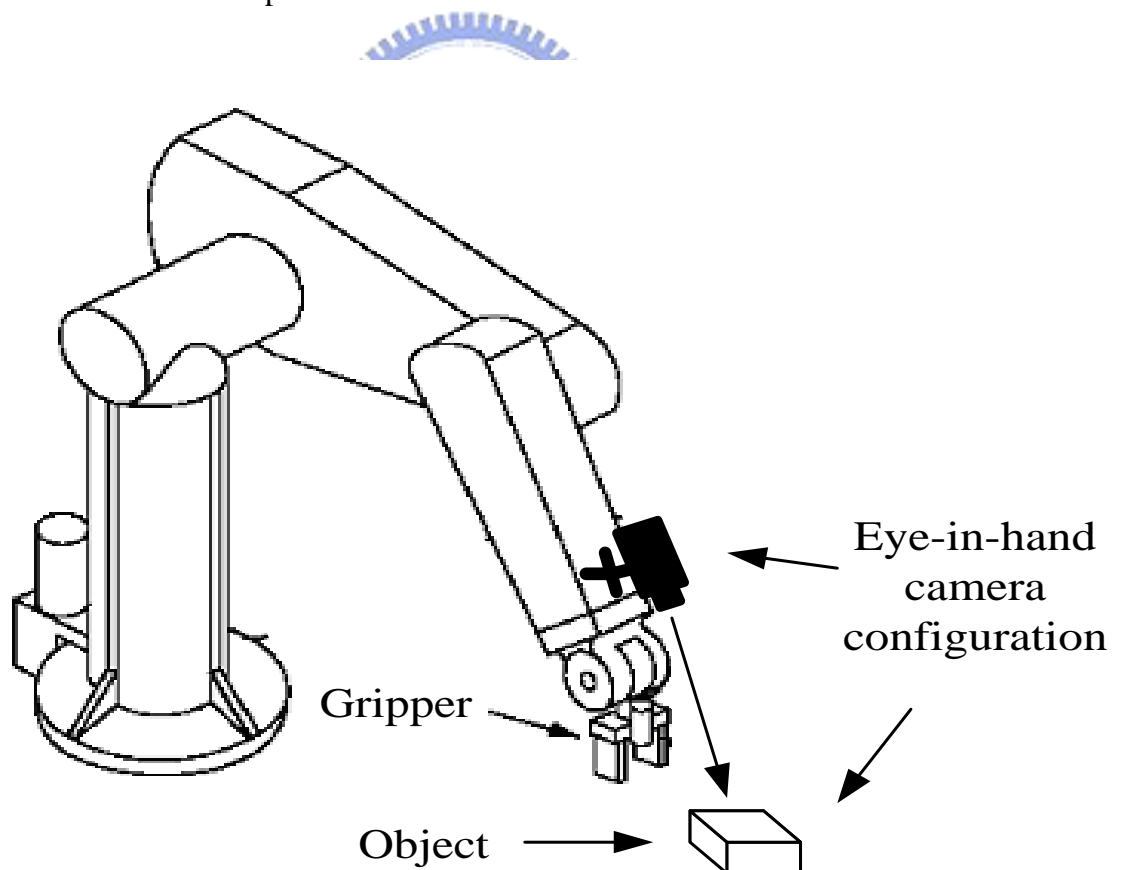


Figure 1.5 eye-in-hand camera configurations

It also closed the control loop by providing visual feedback (robot motion causes camera motion) without requiring the vision system to track the end-effector, as would have been the case with the static camera. The movements of the manipulator were predefined and no details of the control algorithm required implementing the motion nor were the vision algorithms required to track the given object's position.

## 1.3. Problem statements

This study aims to construct hardware and software implementation that solves a mobile manipulator problem.

The strategy of hardware implementation is to develop an embedded platform with a less expensive vision system built-in. The design allows to use low cost components and construct a stand-alone module because it has its main processor and accomplish real-time image processing including acquiring image from vision sensor, and communication directly with other modules such as motion control DSP-board and PC (main processor of the mobile manipulator) to receive and sending commands.

The second strategy on software implementation is to propose an arm controller that can do a specific task such as to locate a specific object and grasp it. Also the robot needs to autonomously to avoid obstacles.

Therefore, by combining the hardware and software implementation, we develop a stand-alone arm controller for a mobile manipulator that can do a basic task of manipulator and navigation by combining this stand-alone arm controller with other controller that already

provided for the mobile manipulator such as localization and face recognition.

## 1.4. Organizations of the thesis

In the following chapters, we present the details of the selected problem, our proposed algorithm, and the experimental results that demonstrate the validity of our approach. We begin by presenting hardware implementation of our embedded platform including the choosing selected low cost components and hardware design of our embedded platform in the chapter 2. After presenting the hardware implementation, next we will present the software implementation that run on our embedded platform. The software implementation is divided into two chapters (chapter 3 and chapter 4). In chapter 3 we begin explanation our proposed behavior-based algorithm that combined with visual servoing. In the same chapter, we present the use of this algorithm for grasping a certain object. In the following chapter (chapter 4), we present the use of this algorithm for mobile navigation. Experimental results of grasping and vision-based navigation are included in the chapter 5. Finally we summarize the thesis and detail contributions of the thesis in chapter 6. We also discuss what we believe should be the next future work in the research beyond the content of the thesis.

# 2. Embedded Image Processing Platform

## 2.1. System Overview

In this chapter, we introduced the proposed embedded image processing platform. The components of the embedded image processing platform include a CMOS sensor board from IC Media as the sensor board and the DSK6416 board from Texas Instrument as the main processor unit. The selection of the CMOS sensor as the sensor is based on that the image quality of the CMOS sensor is good enough and the price of CMOS sensor less expensive compared with CCD sensors. The term good enough means here is the image produce by the CMOS sensor still can be used for further processing for machine vision applications. Meanwhile the selection of the DSK6416 board as the main image processing board which is based on that C6416 has highest-performance fixed-point Digital Signal Processors (DSPs) with 600 MHz clock rate, can do instruction for 4800 MIPS with 1.67-ns instruction cycle time.

In the interface design, the C6416 offers an interface called EMIF (External memory interface) which supports glueless memory architecture interface such as FIFO or SDRAM. In the memory controller, C6416 offers EDMA (Enhanced DMA) controller that can be used to move a block of memory from current address memory into desire address memory.

### **2.1.1. Stand-alone**

The first criterion of our developed embedded image processing platform is stand-alone. The term stand-alone means that the embedded image processing can run independently without any devices such as PC, so even there is no PC in the robot, the embedded image processing platform still can control the mobile manipulator robot.

### **2.1.2. Communication directly with other modules**

The second criterion of our developed embedded image processing platform is that it can communicate directly with other modules. The selection of the communication link is important. The available communication link on module is RS232 communication line. Unfortunately, C6416 does not support UART (RS232 communication standard) standards natively. To solve this problem, we emulate the UART signal by using the available GPIO (General Purpose Input Output) in the C6416. The implementation of the UART emulation is using software implementation. The description how to create the UART emulation is described in the section 2.3.2 in this chapter.

The hardware and software architecture of our embedded imaging processing platform is illustrated in figure 2.1 and 2.2. There are three boards in the architecture of embedded platform. The first board is the main board, which is the DSK6416. The second board is the vision sensor board, which has CMOS sensor that using ICM205B as the main vision sensor. The third board is the DSK6416 daughter board which is a supporting board for DSK6416 board. The DSK6416 daughter board consists of two sub-boards.

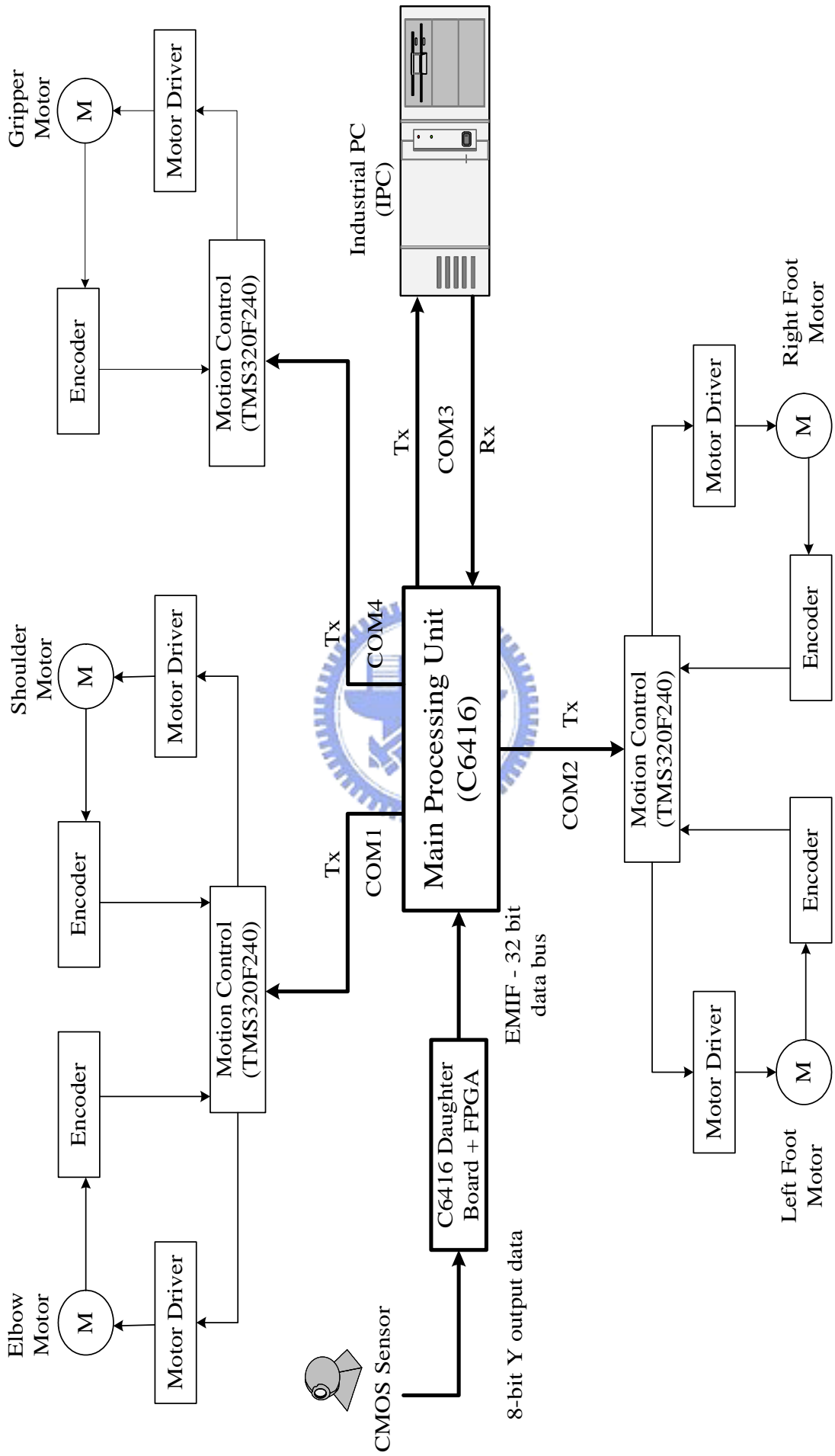


Figure 2.1 hardware architecture of embedded image processing platform



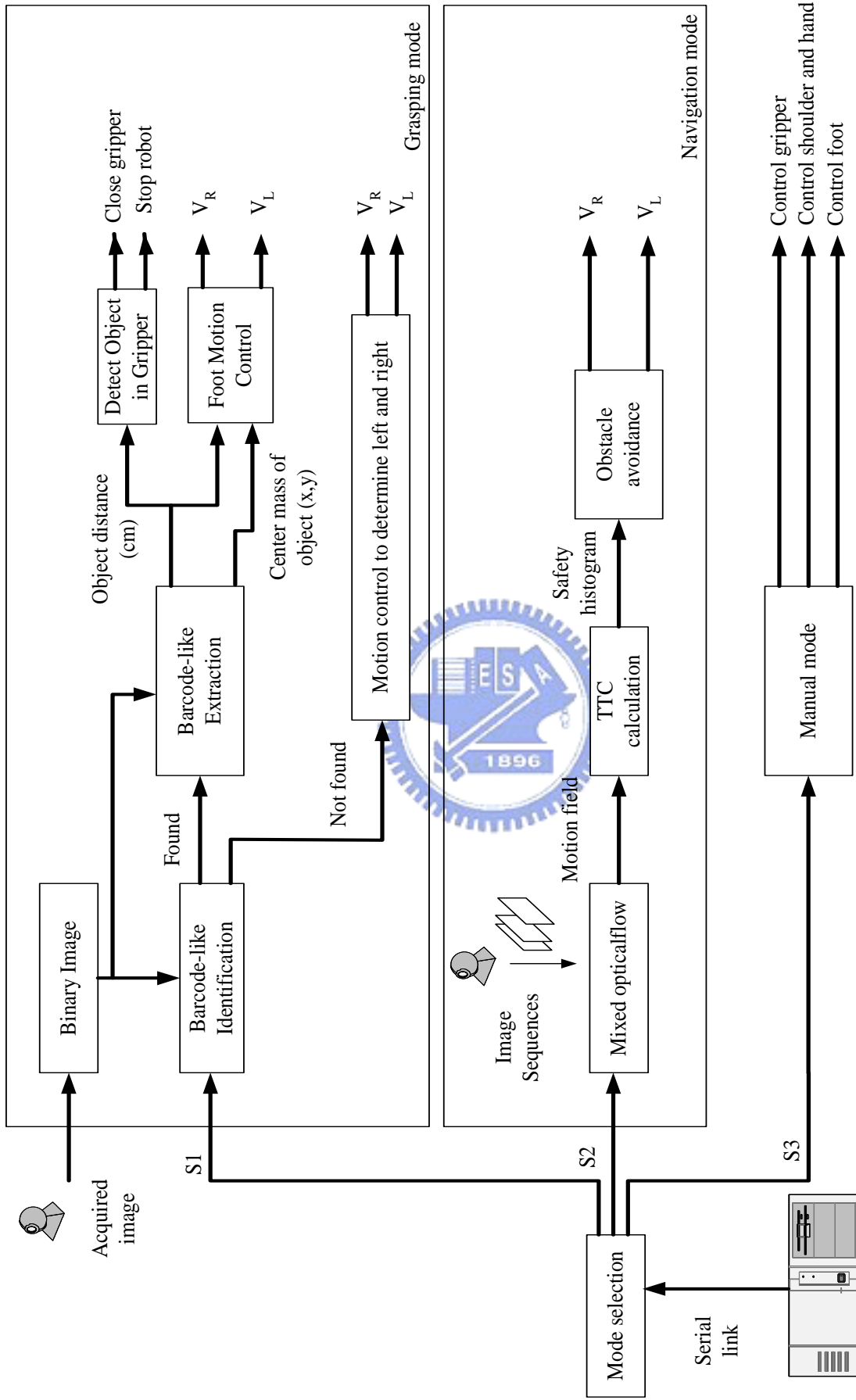


Figure 2.2 software architecture of embedded image processing platform

The first sub-board is the temporary storage board which consists of frame buffer (AL422B) and FPGA (Field Programmable Gate Array) which use Flex10K70RC240 from Altera. The second DSK6416 daughter board is the RS232 communication lines which have 4 COM ports to communicate with other modules or PC.

Next, the embedded imaging processing platform has one input, which comes from the first DSK6416 daughter board and have four outputs to the 4 COM ports to communicate with three DSP motion control card, which is controlling ELBOW and SHOULDER motor (COM1), FOOT motor (COM2), GRIPPER motor (COM4) and one to communicate with PC for receiving or sending command (COM3).

The image acquired from CMOS sensor is first stored in the frame buffer for certain time, after that it is sent to the DSK6416 for storing in the SDRAM. When the image frame is already in the SDRAM, the C6416 begins to process the image. After processing the image, the obtained information is used to control the motor by sending a command to the COM port. In the following section, we will discuss each component of the embedded image processing platform.

## 2.2. CMOS sensor board

The used CMOS sensor board is EVT202. EVT202 specification:

- Number of Active Pixels: 640 x 480
- Number of Physical Pixels: 650 x 490
- Main Clock Frequency: 3 to 24 MHZ
- Frame rate: from 1 to 30 fps

- Output data format: 8/16-bit YCbCr, 24-bit RGB, 16-bit RGB and 8-bit raw data
- Input / output interface: SIF

EVT202 is a CMOS sensor boards with an ICM205B vision chip from IC media Corp [18]. Figure 2.3 Illustrates the EVT202 CMOS sensor board. The ICM205B sensor chip is a single-chip digital color imaging device. It incorporates a 640 x 480 sensor array operating at 1 - 30 frames per second in progressive manner. Each pixel is covered by a color filter which formed a so-called Bayer pattern.

There are several module built in the sensor that can be used for enhanced the image quality, such as the brightness of the scene by adjusting the digital gain for all pixels, color correctness by using automatic white balance circuit, image sharpening by correcting the value of the sharpening function and gamma correction to boost darker signal by selecting the appropriate gamma value. The modules that available inside the chip are described as below:

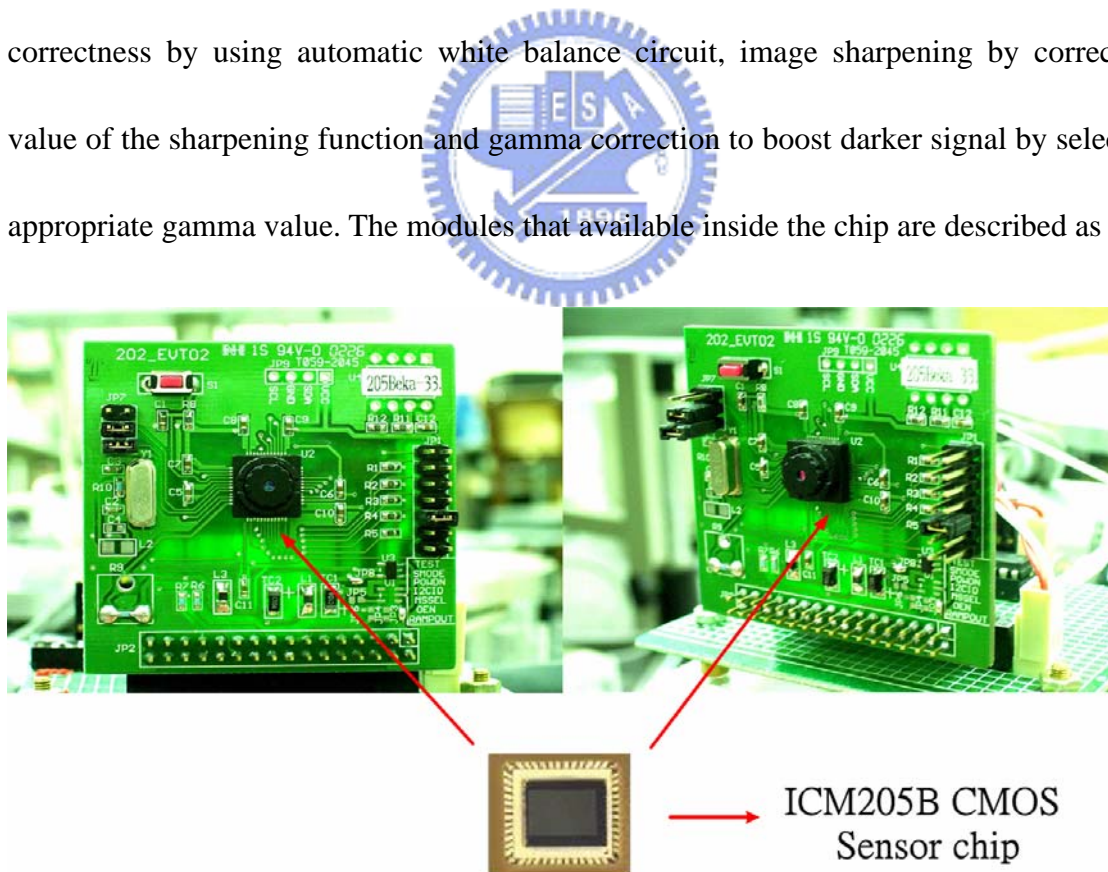


Figure 2.3 EVT202 and its specification

- Real time color interpolation

The functionality of this module is to reconstruct each pixel that covered by a color filter that form so-called Bayer pattern into complete set of RGB values.

- Programmable / Automatic exposure control

The Automatic Exposure (Auto Exposure) module is to control the Exposure time, Digital gain and Anti-flickering function automatically.

- Programmable / Automatic white balancing and color correction

The Automatic white balance is to perform white balancing when the lighting is suddenly changed in the environment by changing the R (Red), G (Green) and B (Blue) to produce a new value of white color. When this function is disabled, the value of the red gain, green gain and blue gain is set manually.

- Programmable sharpening control

The functionality of this module is to sharpening the edge of the image. The edge will sharpen according to the weighting value that given to the CMOS sensor.

- Programmable gamma correction

The gamma correction module is to boost the image from darker. The gamma correction function of the CMOS sensor is  $V_o = V_i^{\frac{1}{\gamma}}$ , where  $V_i$  is normalized (ranged from 0 to 1) R, G, or B signal coming from the white balancing module and  $V_o$  is normalized final output. So by adjusting the  $\gamma$  value we can adjust gamma correction.

- Programmable brightness correction

The brightness correction is to increase / decrease the brightness level of the CMOS sensor.

- Programmable anti-flickering (50 Hz, 60 Hz or off, like outdoor)

The functionality of this module is to prevent flicker that can occur from light source by making the flicking stable at 50 Hz / 60 Hz.

All of these modules can be changed or activated via serial bus control which known as I2C by changing the value of the register in each modules. Although these feature are available inside the chip, but not of these entire feature are activated or changed. It is because some of available function can affect the image that want to be process, such as sharpening function, auto exposure and auto white balance function. The sharpening function can sharpen the edge of the image by changing its value but because the inconsistency of the sharpening edge between image at time  $t$  and time  $t+1$ , so it can make false motion detection for our optical flow algorithm so we disabled the sharpening module.

Another function that we disabled is the auto exposure time function, by disabling this function the frame rate of the image will be fix although there is any changes of the environment such as lighting changes that coming from the light source so by fixing the frame rate, we can acquire the image from CMOS sensor with a fix frame rate for all the time.

The last function that we disabled is auto white balance function. Activated the function of auto white balance function can also make false motion detection for our optical flow algorithm. It is because when there are any suddenly changing in environment from dark to

white the auto white balance function will try to adapt by changing the R, G and B value to make a new white color for several image frames. In this time, there will be false motion detection even there is no any motion in the image. Generally, the need of motion detection algorithm is that the image must be consistently same for all the time without any suddenly changes so it will be clearly detect motion occur in the image instead of the changing of the environment. Instead of the above module, we also change the gamma correction from the gamma=1 to gamma=1.5 for boosting the darker signal so the image will be brighter. Meanwhile the brightness correction will not be change; it is no necessary to change the value of brightness correction after changing the gamma correction value. The value of the module that has been changed or disabled can be seen in the table 2.1.

Another configuration of the CMOS sensor that should be considered is data output format and data output mode. The ICM205B CMOS sensor offer a various data output format such as: 8/16 bit YCbCr, 24-bit RGB, 16-bit RGB and 8-bit raw data and also output modes such as VGA and its sub-sampling QVGA/QQVGA.

Table 2.1 list of CMOS sensor built in function module that has been changed

| No | Module name           | Default value        | New value          |
|----|-----------------------|----------------------|--------------------|
| 1  | Auto exposure         | Automatic            | Disable            |
| 2  | Auto white balance    | Automatic            | Disable            |
| 3  | Sharpening            | Sharpen weight = 2   | Disable sharpening |
| 4  | Gamma correction      | Gamma = 1 (no gamma) | Gamma = 1.5        |
| 5  | Brightness correction | 0                    | No changes         |

In our embedded image processing platform, we configure the output data as 16-bit YCbCr with the 8-bit low output data is Y and the next 8-bit high output data is sequential output of Cb/Cr which we are just using the 8-bit low output data (Y), so we doesn't use the 8-bit high output data (Cb/Cr). And for the output data mode, we are using VGA data output mode. The YCbCr data format itself is a similar with YUV color space format which is used in European TVs.

## 2.3. DSK6416 daughter board

The DSK6416 daughter board consists of two kind interfaces for embedded image processing platform. The first daughter board interface uses for temporary 8-bit image data storage of CMOS sensor data before the image data send to the SDRAM in DSK6416 board. DSK6416 daughter board specification:



- Frame buffer (AL422B): 384 Kbytes
- 7 ports GPIO of DSK6416 for 4 port RS232 emulation

This daughter board uses AL422B from Averlogic Corp [21]. There are several reasons why we choose this frame buffer (AL422B). First, the frame buffer has enough capacity (384 Kbytes) that can store image data up to one image frame (640x480). Second, the speed of the frame buffer is fast enough to operate at 50 MHZ, so it can match with our embedded image processing platform speed that run at 3 MHZ. Third, the control signal of the frame buffer is simple comparable with other similar frame buffer which provide by other company. So the AL422B match with our criteria for temporary image data storage for CMOS sensor data.

The second daughter board consists of 4 COM ports of RS232 interfaces. The need of these 4 COM port of RS232 interfaces are to communicate with other module such as motion card to control motor and IPC. So by having these interfaces, the embedded image processing can run independently to control motor or to communicate with any other module directly. These 4 COM ports of RS232 interfaces emulate from 7 ports GPIO (General Input Output) of the C6416 chip [20]. The emulation use software emulation that writing in the C6416 chip. The software emulation can emulate two ports of GPIO to generate signal RX and TX for one COM port RS232 interfaces signal.

In order to emulate 4 COM ports we need at least 8 ports GPIO, because there are only 7 ports GPIO available so one of the 4 COM ports will have no RX signal which means that one of the 4 COM ports will only have transmit signal (TX) while the other 3 COM ports will have both TX and RX signal. The software emulation can also emulate several baud rates for the COM port RS232 interfaces. The baud rates speed that can emulate by the software emulations are 2400, 4800, 9600, 14400, 19200 and 57600 bps. So within this various baud rate configuration, software emulation can emulate COM port RS232 for several COM port configurations.

### **2.3.1. Frame buffer (AL422B)**

The frame buffer (AL422B) is a video frame buffer consists of DRAM that work like a FIFO which long enough to hold up to 393,216 bytes (384 Kbytes) of picture information and fast enough to operate at 50 MHz [21]. AL422B specification (see figure 2.4):

- 384K (393,216) x 8 bits FIFO organization



- Independent read/write operations (different I/O data rates acceptable)
- Read/write cycle time: 20ns
- Output enable control (data skipping)
- 5 or 3.3. V power supply

From the functional block diagram in figure 2.5, we can see that the frame buffer consists of two kind of separating part that are writing part and reading part. The writing part and reading part have three control signals to be able to write and read data in and out.

These control signals are described as below:

### 1. Writing part

In the writing part, there are three control signals that control the data from outside that connected to the DI0 ... DI7 to be written in to the memory cell array in the frame buffer. These control signal are WCK, /WRST and /WE. The functional of each control signal are:

- WCK (Write Clock)

This control signal is to give a periodic clock to the frame buffer, because the frame buffer memory cell uses DRAM as the storage memory so it required a periodic clock to refresh its data for certain time so the data will still available in

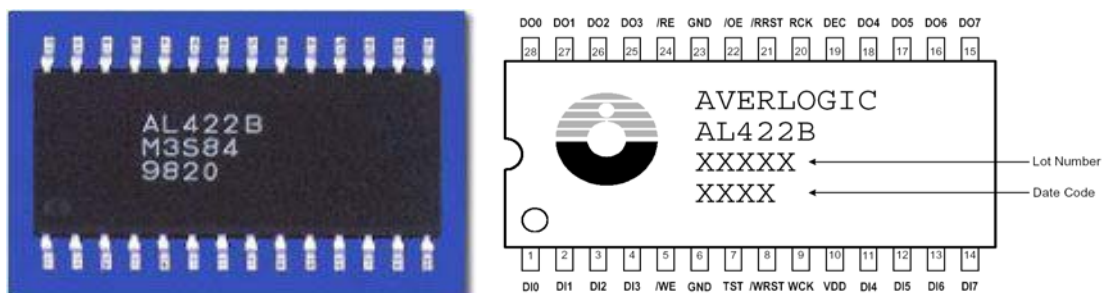


Figure 2.4 frame buffer (AL422B) [21]

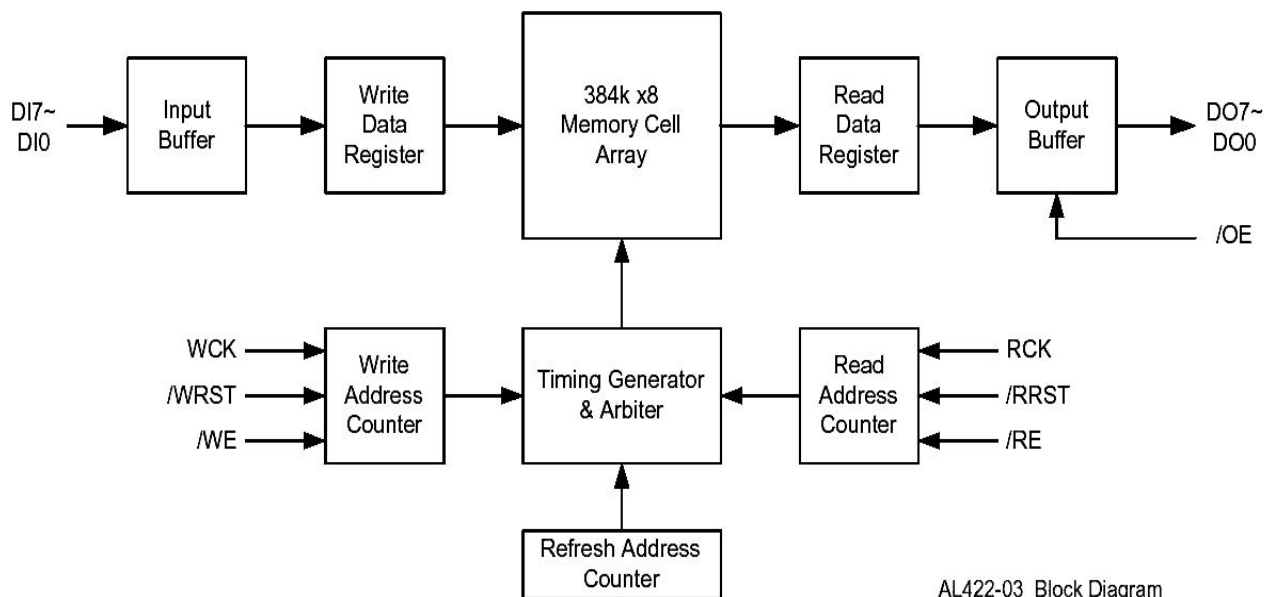
the memory cell array. Instead of refreshing the data in the memory cell, this control signal is used to synchronizing incoming data from DI0 ... DI7 to be writing into the memory cell array and will increase the internal counter address memory of the frame buffer.

- /WRST (Write Reset)

This control signal is used to reset the internal counter from current address memory into beginning address memory and active low. Its mean that if we want to write the memory cell array from beginning address, we must to use this control signal to reset the current memory address into the beginning memory address.

- /WE (Write Enable)

This control signal is used to enable the writing operation and active low, so that the data coming from DI0 ... DI7 can write into memory array cell.



AL422-03 Block Diagram

Figure 2.5 AL422B Functional block diagram [21]

## 2. Reading part

In the reading part, there are four control signals that use to read the data out from memory array cell into DO0 ... DO7. For the first three control signals, the functional of these control signals will be same as the control signal in the writing part. These three control signals are RCK (Read clock) similar with WCK (Write clock), /RRST (Read Reset) similar with /WRST and /RE similar with /WE. The other control signal that is different with the writing part is /OE (Output Enable).

The functional of the reading part control signal are described as below:

- RCK

This control signal is to give a periodic clock to the frame buffer, so the data memory still remains for certain time in the memory cell array, also used to synchronize the data read out from memory cell array into DO0 ... DO7 and to increased internal counter address. So when there is an incoming clock into RCK, the data will readout from memory cell array into DO0 ... DO7 and after that will increase the internal memory address.

- /RRST

This control signal is used to reset the current internal counter address into beginning address memory and active low. Its mean that if we want to read out the data from beginning address of the memory cell array we must to use this control signal to reset the current memory address back to the starting memory address.

- /RE

This control signal is used to enable the data to read out from the memory cell array into DO0 ... DO7 and active low.

- /OE

This control signal is used to enable reading the data that already available in the DO0 ... DO7 and active low. If this control signal is disabling, the output data of the DO0 ... DO7 will be in high impedance state.

According to above description of the frame buffer control signal, it is important to give a correct timing for the control signal to write data into frame buffer or to read data out from frame buffer. For this part, we already design the correct timing and it will describe in the following section while describing the FPGA part.

### 2.3.2. Seven ports GPIO of DSK6416 for emulating four COM ports RS232 interfaces.

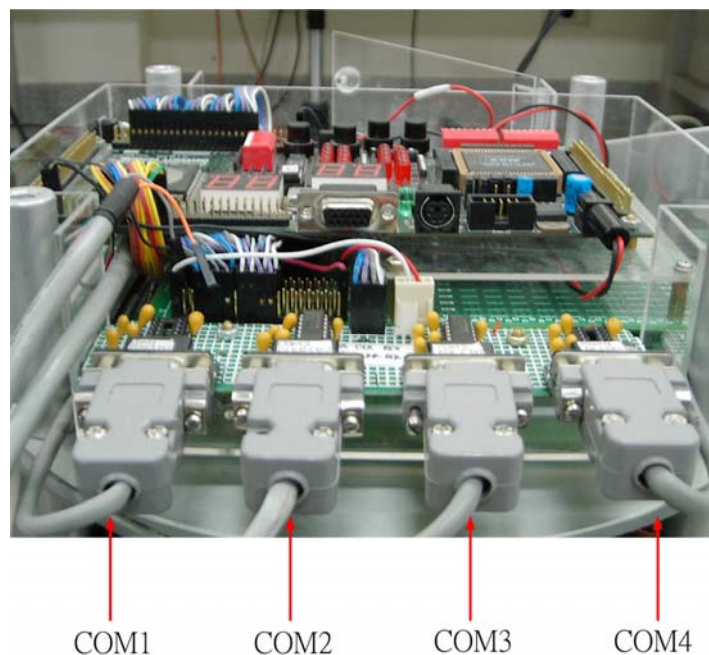


Figure 2.6 daughter boards for 4 COM ports RS232 interfaces

The COM ports RS232 interface in our embedded image processing platform is very important for communication with other modules such as motion control module and PC directly. The C6416 itself does not provide COM port or RS232 interface that support UART standard, but the C6416 provides so-called multichannel buffered serial port (McBSP) that have seven ports that can be configured for General Input Output (GPIO) function [20]. Figure 2.6 shows daughter board for four COM ports RS232 interfaces. So we need to emulate these seven ports GPIO to work as COM port RS232 that supporting UART standard. One way to emulate these GPIO port are using software to emulate it. The software need to emulate two ports of GPIO to act like RX and TX as part of the COM port RS232 interfaces signal which RX for receive signal and TX for transmit signal. To configure the McBSP pins as GPIO, we must configure two registers of the McBSP. These registers are SPCR (Serial Port Control Register) and PCR (Pin Control Register).

The configuration of the McBSP pins as GPIO can be seen at table 2.2 and the block diagram of the McBSP configuration as GPIO can be seen in the figure 2.7. After configuring the seven ports of the GPIO into for four COM ports RS232 interfaces, next we will explain about the software implementation to emulate the TX and RX signal. The software implementation itself has four functions to emulate the COM port RS232 interface. Below is the description of these three functions:

1.) `void HR_Initiate_GPIO(void);`

This function sets the McBSP in GPIO mode by setting the SPCR and PCR registers according to the table 2.2.

Table 2.2 configuration of McBSP pins as GPIO [20].

| Pins | GPIO enabled When ...<br>(SPCR register) | Selected as Output<br>When ...<br>(PCR register) | Selected as Input<br>When ...<br>(PCR register) |
|------|--|--|---|
| CLKX | /XRST = 0,<br>XIOEN = 1                  | CLKXM = 1  | ---   |
| FSX  |  | FSXM = 1   | ---   |
| DX   |  | Always   | Never   |
| CLKR | /RRST = 0,<br>RIOEN=1                    | CLKRM = 1  | ---   |
| FSR  |  | ---  | FSRM = 1  |
| DR   |  | Never  | Always  |
| CLKS |  | Never  | Always  |

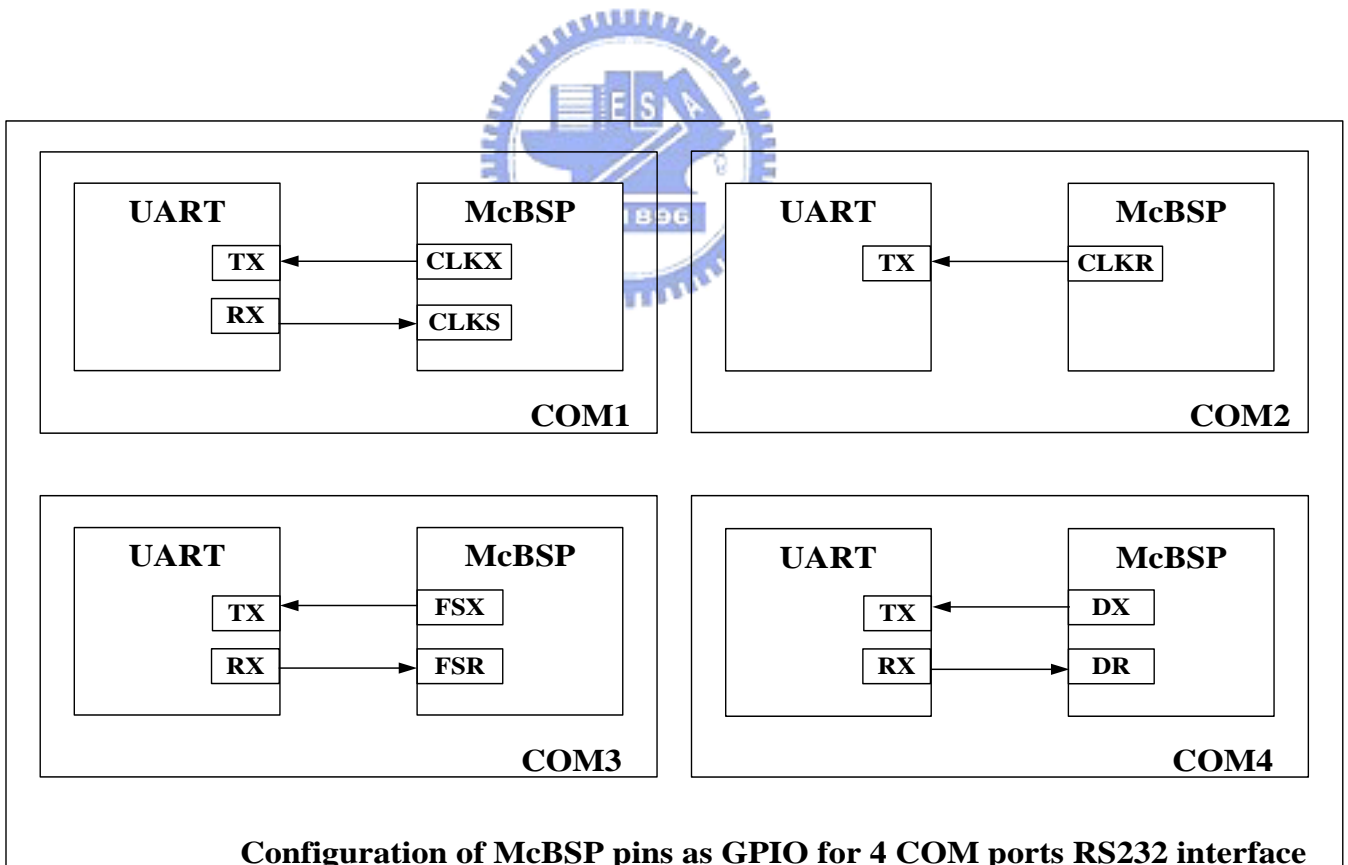


Figure 2.7 block diagram of McBSP pins configuration for 4 COM ports

2.) unsigned int HR\_SoftUartSpeedDetect(void);

This function of HR\_SoftUartSpeedDetect performs Auto-Baud rate detection by measuring the length of the Start bit, plus the length of the first data bit (logic high) by using character <cr> (carriage return = 0x0d). So while this function tries to detect the baud rate, users need to ensure (in software) that the first character sent is <cr> or any other character that the first data bit has to be a logical one. This is shown in figure 2.8. The time T is determined by this function that using a software counter incremented by one until the second transition from high to low is detected (D0) by reading the serial data input (RX). T represent twice the time of the time of a bit length. This measurement is required because the RX signal from UART is not always very clean. Simply measuring the length of the Start bit to determine the baud rate is not accurate enough so we need to use the first data bit which must be at logic high to fix the accuracy. The time reference value  $UART\ speed = T/2$  is returned from HR\_SoftUartSpeedDetect function which this value will use as reference for the Baud rate speed.

The return value of this baud rate will depend on the clock speed of the main processor. In our case, we are using C6416 with 600 MHz, and the return value of this function according to its baud rate speed can be seen in the table 2.3.

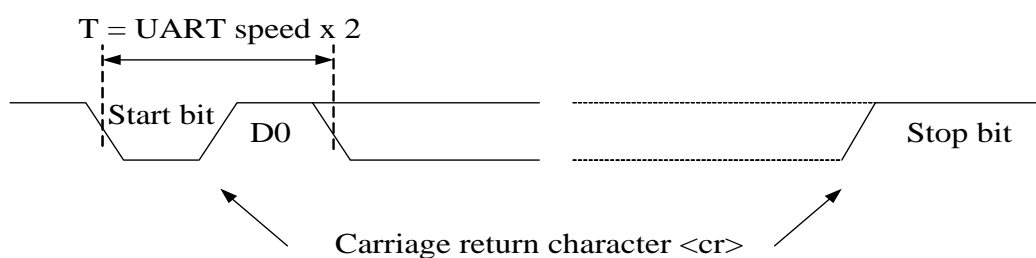


Figure 2.8 UART Auto Baud rate detection [20]

Table 2.3 baud rate value detect from HR\_SoftUartSpeedDetect function

| No | Baud rate speed (bps) | Return value from HR_SoftUartSpeedDetect function |
|----|-----------------------|---|
| 1  | 2400                  | 0x144E (5198)                                     |
| 2  | 4800                  | 0xA27 (2599)                                      |
| 3  | 9600                  | 0x513 (1299)                                      |
| 4  | 14400                 | 0x362 (866)                                       |
| 5  | 19200                 | 0x289 (649)                                       |
| 6  | 57600                 | 0xD8 (216)  |

This result is obtaining by transmitting character <cr> from PC to each of the COM ports.

3.) char HR\_SoftUartInChar(int BaudRate, int COM\_Port);

The HR\_SoftUartInChar takes the value of the Baud-rate speeds detected from HR\_SoftUartSpeedDetect and select which COM Port that use for receiving the serial data input (RX). Every single 8-bit character read from the serial data input returned from this function. This function parses bit-by-bit the UART data on the serial data input and detects the Start bit by poling the first transition from inactive (logic 1) to active (logic 0) state.

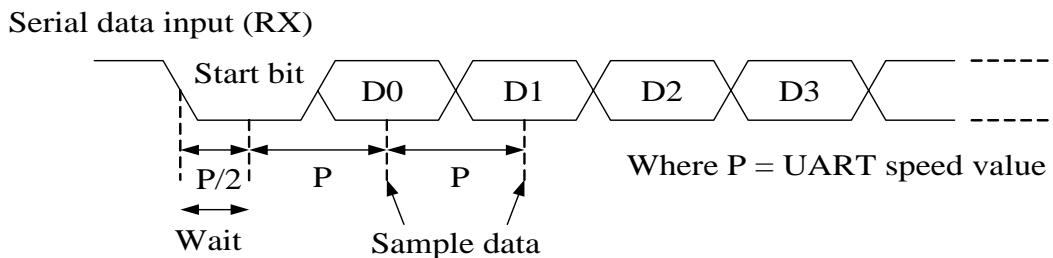


Figure 2.9 HR\_SoftUartInChar UART data fetch in [20]



The 8 data bits are transmitted by the UART device immediately after the Start bit. The best time to fetch the right value of each data bit is in the middle of the data bit waveform. Figure 2.9 shows how HR\_SoftUartInChar function read out the serial data input. Firstly, it waits for half of the UART speed ( $P/2$ ) which detected from HR\_SoftUartSpeedDetect function during the Start bit. Then for each of the eight valid data bits, it samples the serial data input in the middle of the data bit waveform. Finally this function shifts each binary bit result into a single 8-bit character

4.) void HR\_SoftUartOutChar(int BaudRate,char Buffer, int COM\_Port);

The HR\_SoftUartOutChar function is based on the same mechanism as the HR\_SoftUartInChar function. It has had three input arguments, which are Baud-rate speed that detected from HR\_SoftUartSpeedDetect function, the single 8-bit character to be sent via serial data output (TX) and last argument is to specify the COM port that one wants to use. At the beginning of a transfer, HR\_SoftUartOutChar function writes a logic low ('0') to the serial data output as Start bit. Subsequently, it transmits each data bit until 8-bit already send to serial data output. The transmit character is first placed into the least significant 8 bits (LSB) in a register padded with three stop bits (0x00000700).

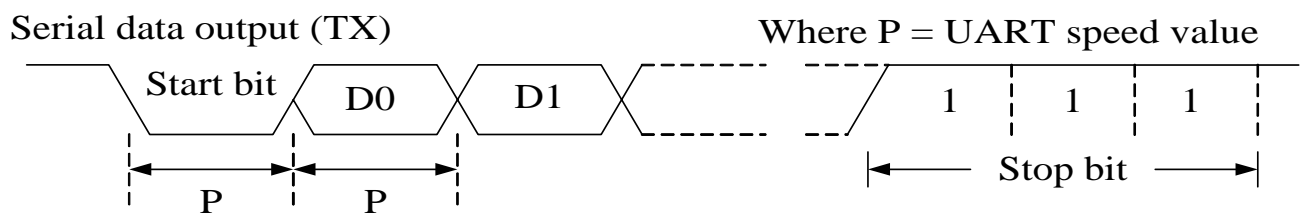


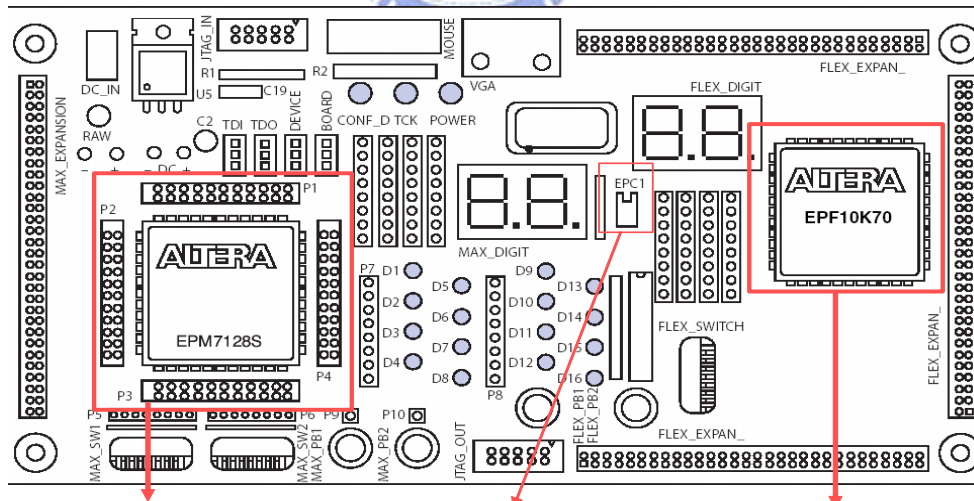
Figure 2.10 HR\_SoftUartOutChar UART data fetch out

For example, the character 'A' with ASCII code is 0x41 will be placed in the padded register to become 0x00000741. Figure 2.10 shows how HR\_SoftUartOutChar function sends out the serial data out.

So by emulating the GPIO into COM port RS232 interfaces, there will have more additional COM port RS232 interfaces which will depend on the availability of the GPIO itself. In our case, we can emulate up to four COM ports. But by emulating the GPIO into COM port RS232 interface, the way to receive serial data from other module is not supporting serial port interrupt, so we can just use polling method to receive serial data from other module.

## 2.4. FPGA (Altera UP2)

### Altera UP2



EPM27128S (MAX7000S)  
 -Gate count : 2500  
 -Max I/O : 104  
 -EEPROM-based programmable

Altera configuration device (EPC1)

EPF10K70RC240 (Flex10K)  
 -Gate count : 70,000  
 -Max I/O : 358  
 -SRAM-based programmable

Figure 2.11 Altera UP2 board and its specification [22]

The Altera UP2 board (see figure 2.11) consists of two FPGA chips that have different gate counts, and number of I/O (Input and output) [22]. The first FPGA chip (EPM27128S) has 2,500 gate count, max 104 I/O and EEPROM-based programmable. Meanwhile the second chip (EPF10K70RC240) has 70,000 gate counts, max 358 I/O and SRAM-based programmable.

We choose the second chip (EPF10K70RC240) because of the larger gate count and have more available I/O. But this chip (EPF10K70RC240) needs additional boot ROM (EPC1) because of the architecture is SRAM-based

The use of FPGA (Field Programmable Gate Array), Altera UP2 board (see figure 2.11) in our embedded image processing platform is to generate control signals for CMOS sensor, Frame buffer and DSK6416. The design of control signal in FPGA consists of three modules (see figure 2.12). These three modules are I2C, Buffer controller and 8bit\_to\_32bit\_clk.

### 2.4.1. I2C module

The I2C module is used to generate SDA and SCL signal (I2C signal) in bidirectional input port *SDA* and *SCL* to initiate the CMOS sensor (see figure 2.13).

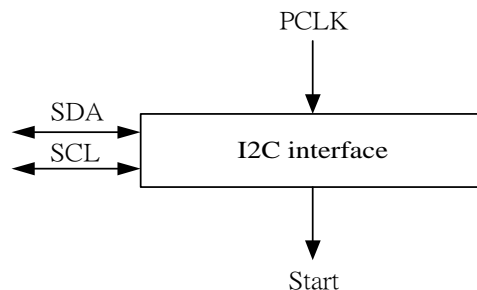


Figure 2.13 I2C module

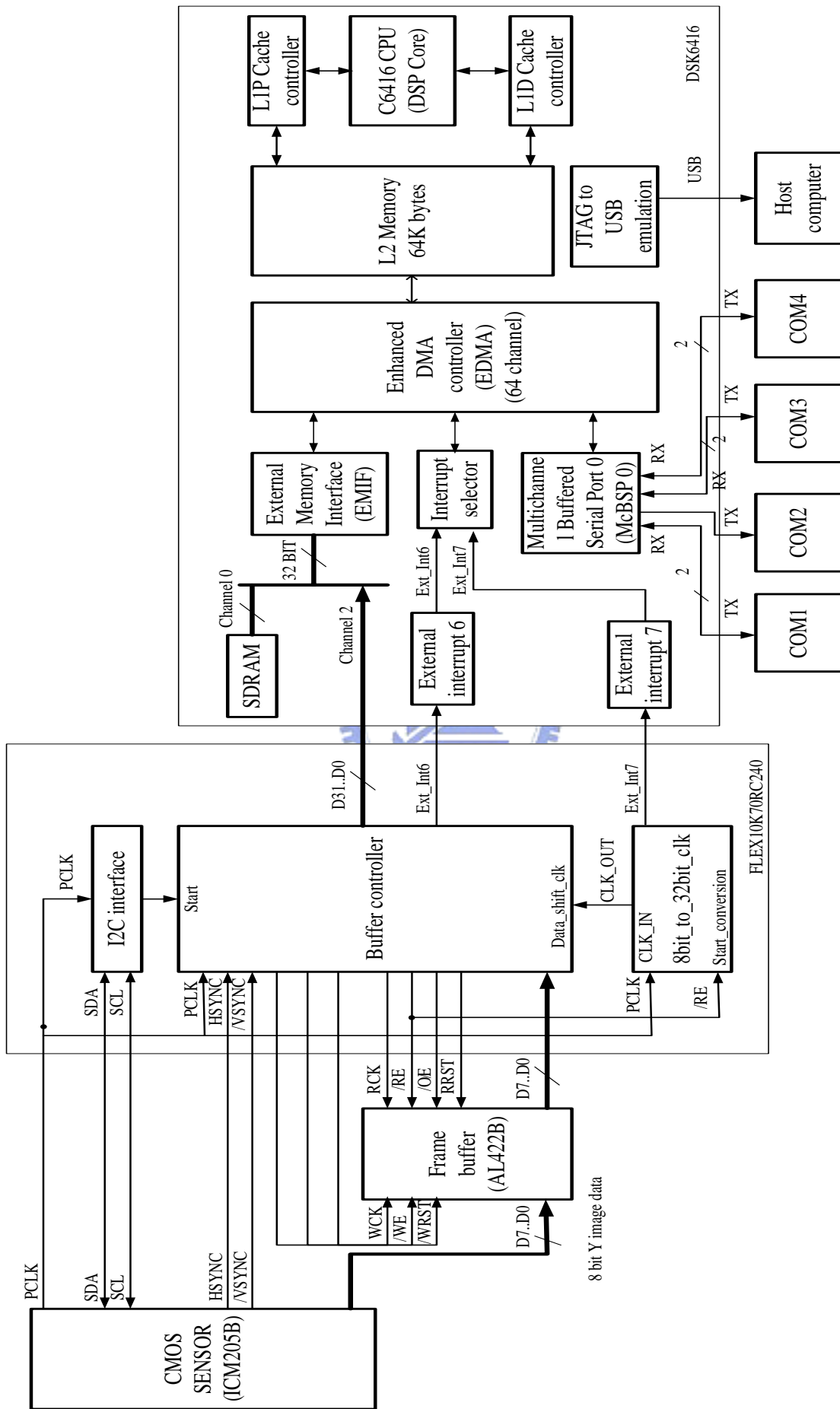


Figure 2.12 the complete control signal of FPGA, DSK6416 and CMOS sensor

Table 2.4 configuration of the CMOS sensor

| No | Feature name       | Status             |
|----|--------------------|--------------------|
| 1  | Auto white balance | Disabled           |
| 2  | Image sharpening   | Disabled           |
| 3  | Auto exposure      | Disabled           |
| 4  | Gamma correction   | Gamma = 1.5        |
| 5  | Data output format | 16-bit 4:2:2 YCbCr |
| 6  | Data frame rate    | 30 fps             |

As we already mentioned in the previous section (section.2.2) the feature of the CMOS sensor can be disabled or enabled by using this I2C module. After completion initiation the CMOS sensor, this module generates a “high” signal in the output port *Start* to the next module (buffer controller module) to starting to work. The complete configuration of the CMOS sensor that initiates this module is shown in table 2.4.

### 2.4.2. Clock divider module

The clock divider module (see figure 2.14) that divides the input port *CLK\_IN* by four and the output can be taken from *CLK\_OUT* and *Ext\_Int7* output port. Later, “high” signals on *Start\_conversion* will active this module.

### 2.4.3. Buffer\_controller module

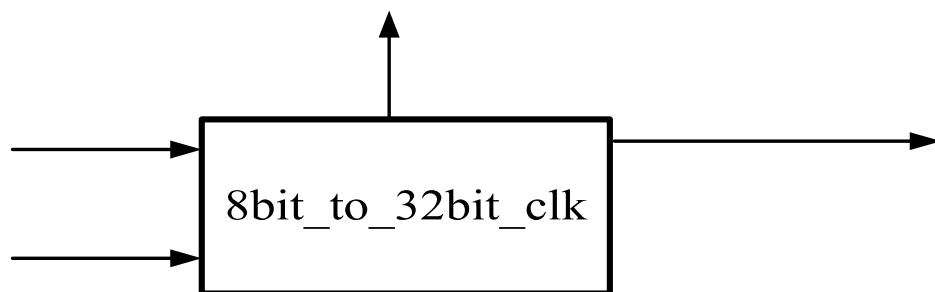


Figure 2.14 clock divider

The Buffer\_controller module (see figure 2.15) is the main module of the three other modules in the FPGA design. The functional of this module (see figure 2.12) is to read out the correct 8-bit image data from CMOS sensor according to the *PCLK*, *HSYNC* and *VSYNC* control signal. The correct 8-bit image data is then first stored in the frame buffer by generating a correct write control signal (*/WE*, */WRST*, *WCK*).

After the correct 8-bit image data are stored for a certain time in the frame buffer, the Buffer\_controller generates a correct read control signal (*/RE*, */OE*, */RRST*, *RCK*) to read out the 8-bit image data from the Frame Buffer and stored the data in the FPGA. When the 8-bit image data already stored in the FPGA, it is then converted to the 32-bit image data by shifting the 8-bit image data four times.

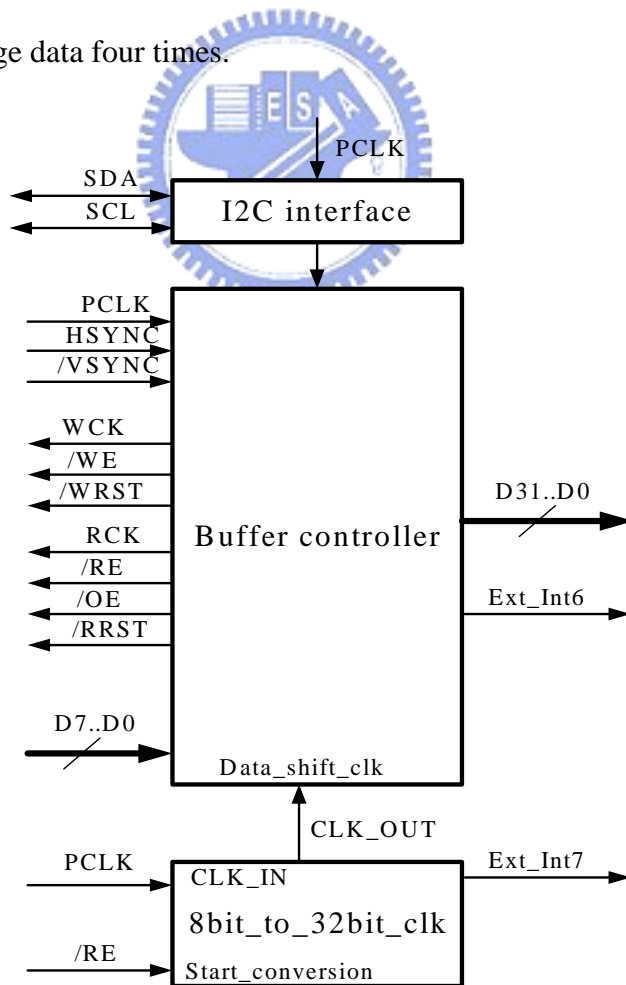


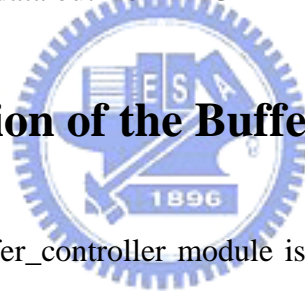
Figure 2.15 buffer controller modules

After completing shifting the 8-bit image data for four times, the *CLK\_OUT* control signal, that comes from Clock divider module, (see section 2.4.2) tells the Buffer\_controller to shift the 32-bit image data [D31 ... D0] out from the FPGA.

At the same time, the Clock divider module generates an interrupt signal to C6416 using *Ext\_Int7* control signal (see figure 2.14) to tell C6416 to fetch 32-bit image data out from FPGA. The last control signal of Buffer\_controller is *Ext\_int6* control signal.

The *Ext\_Int6* control signal is used to interrupt C6416 that there is an incoming one frame image data. So after receiving this interrupt the C6416 open an EDMA channel to starting fetch the 32-bit image data out from FPGA

#### **2.4.4. Implementation of the Buffer\_controller**



The implementation of Buffer\_controller module is successful that we can acquire image from CMOS sensor for 15 fps and 30 fps. The acquiring methods for 15 fps and 30 fps are different especially in controlling the timing diagram of frame buffer. Next we will discuss these two methods for acquiring images.

##### **2.4.4.1. Acquired image for 15 fps**

In the earlier implementation of the Buffer\_controller, the way of controlling the frame buffer is divided into two stages; namely, writing stage and reading stage (see figure 2.16). So in the first incoming */VSYNC*, the writing stage is active while the reading stage is not active and the next incoming */VSYNC* it switches to the reading stage while the writing stage is not

active (see figure 2.16).

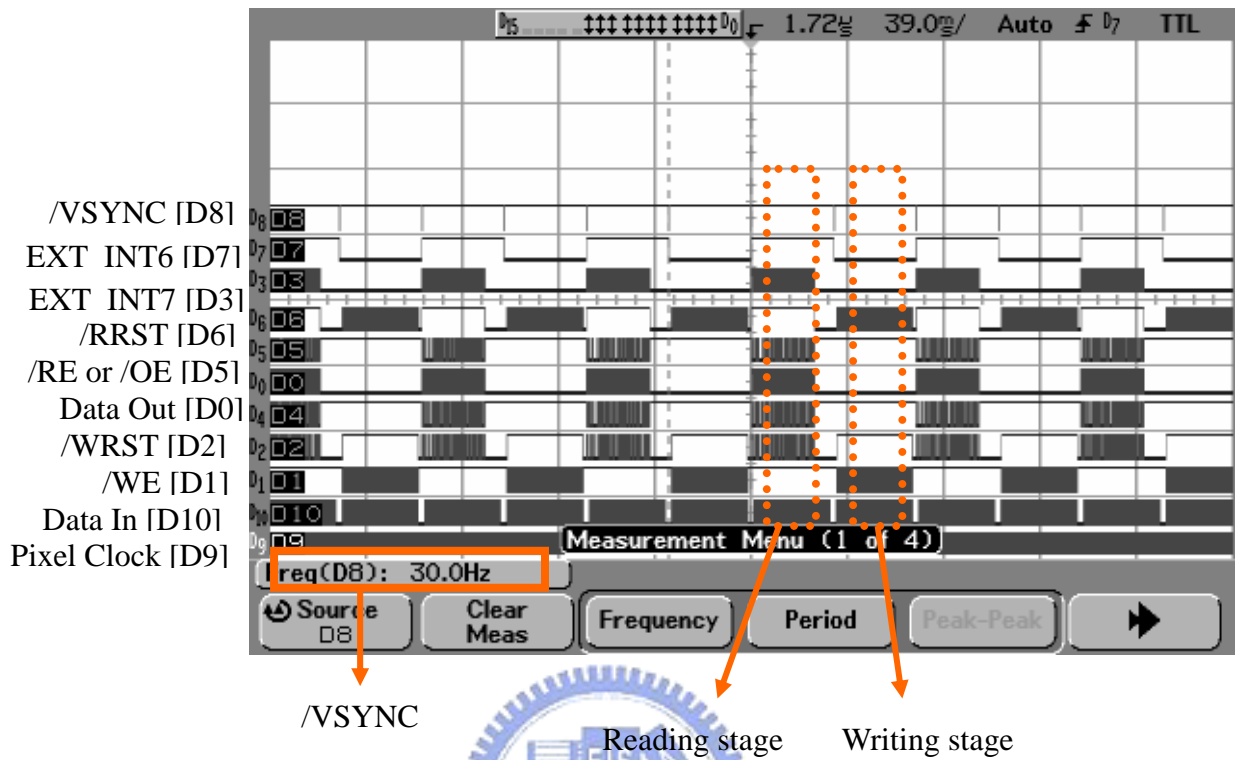


Figure 2.16 full timing diagrams for 15 fps configuration

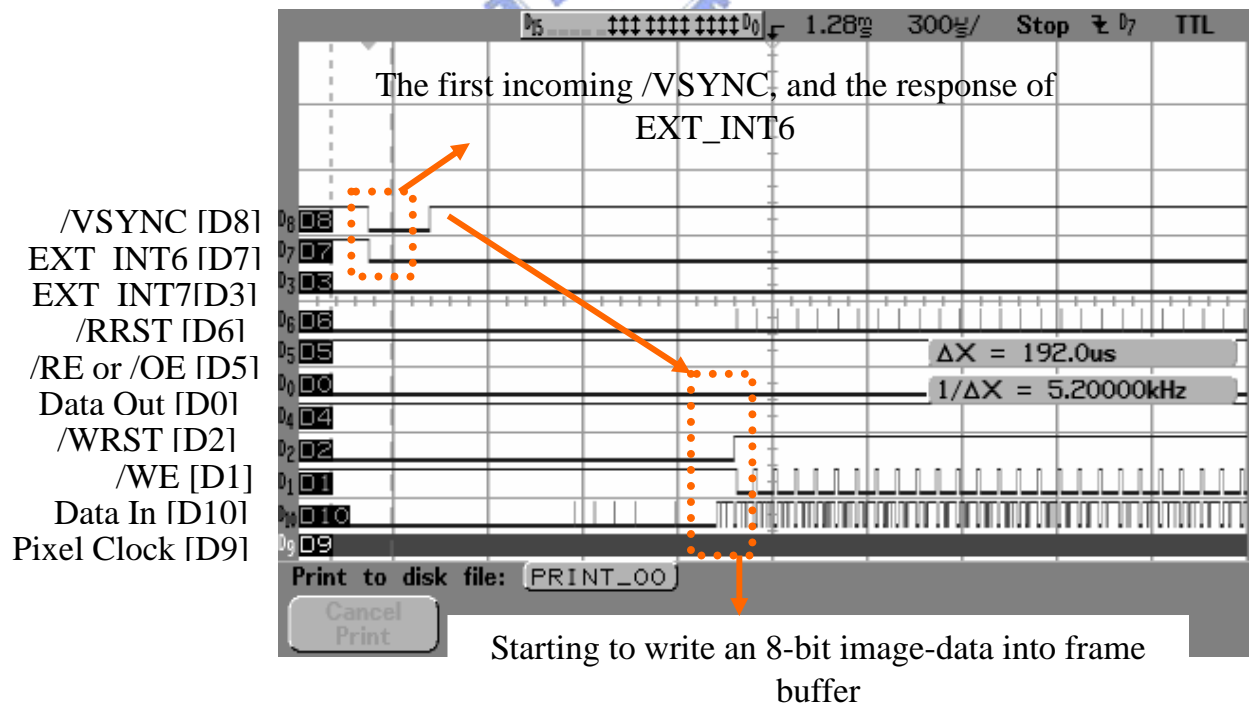


Figure 2.17 timing diagram of Frame Buffer in the writing stage



The writing stage here means that the Buffer\_controller enables the writing part control signal of frame buffer to stored 8-bit image data from CMOS sensor for 640x480 pixels (see figure 2.17) by asserting the control signal  $\overline{WE}$  ( $\overline{WE}=LOW$ ), and disasserting  $\overline{WRST}$  ( $\overline{WRST}=HIGH$ ) which are active low. Meanwhile the reading part of the frame buffer is disabled by disasserting the control signal  $\overline{RE}$  ( $\overline{RE}=HIGH$ ),  $\overline{OE}$  ( $\overline{OE}=HIGH$ ) and asserting  $\overline{RRST}$  ( $\overline{RRST}=LOW$ ).

On the other hand, the reading stage here means that in the second incoming  $\overline{VSYNC}$ , the buffer controller enabled the reading part control signal of the frame buffer to read out 8-bit image data that already stored in frame buffer for 640x480 pixels (see figure 2.18) by asserting the  $\overline{RE}$  ( $\overline{RE} = LOW$ ),  $\overline{OE}$  ( $\overline{OE} = LOW$ ) and disasserting the  $\overline{RRST}$  ( $\overline{RRST} = HIGH$ ) control signal. Meanwhile the writing part of the frame buffer is disabled by disasserting the control signal of  $\overline{WE}$  ( $\overline{WE} = HIGH$ ) and asserting  $\overline{RRST}$  ( $\overline{RRST} = LOW$ ).

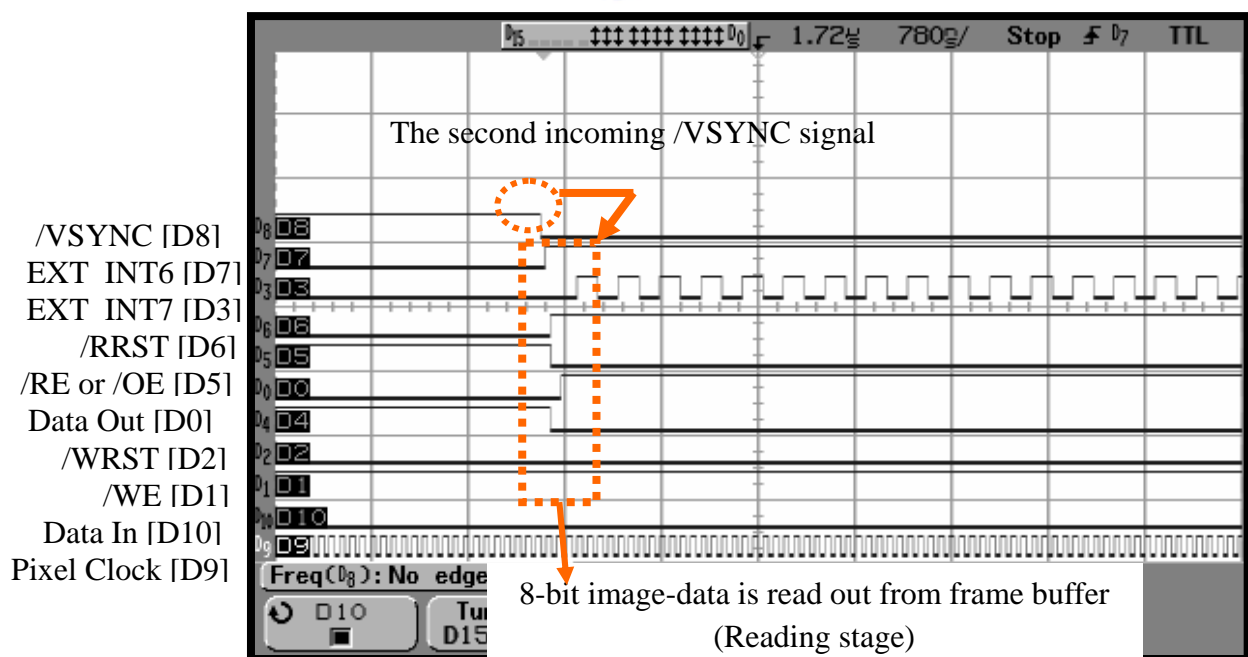


Figure 2.18 timing diagram for Frame Buffer in the reading stage

It is because the Buffer\_controller needs two  $\overline{VSYNC}$  signal to write the 8-bit image data into frame buffer, read it out from frame buffer and move it to the FPGA before it is moved to DSK6416, Therefore the acquire image speed from CMOS sensor until the DSK6416 is

$$Frame\ rate = \frac{\overline{VSYNC}}{2} = \frac{30}{2} = 15\ fps$$

From figure 2.18, we also can see that in the reading stage, the incoming one image frame ( $\overline{VSYNC} = \text{LOW}$ ), will active  $EXTINT6$  ( $EXTINT6 = \text{HIGH}$ ) signal. Since The  $EXTINT6$  signal is connected to the interrupt of the C6416 (see figure 2.12) so the active  $EXTINT6$  signal will interrupt the C6416 to tell for an incoming one image frame. Therefore the C6416 enable EDMA channel and open its channel to begin to move 32-bit image data from FPGA to the SDRAM until  $640 \times 480$  by counting  $EXTINT7$  signal.

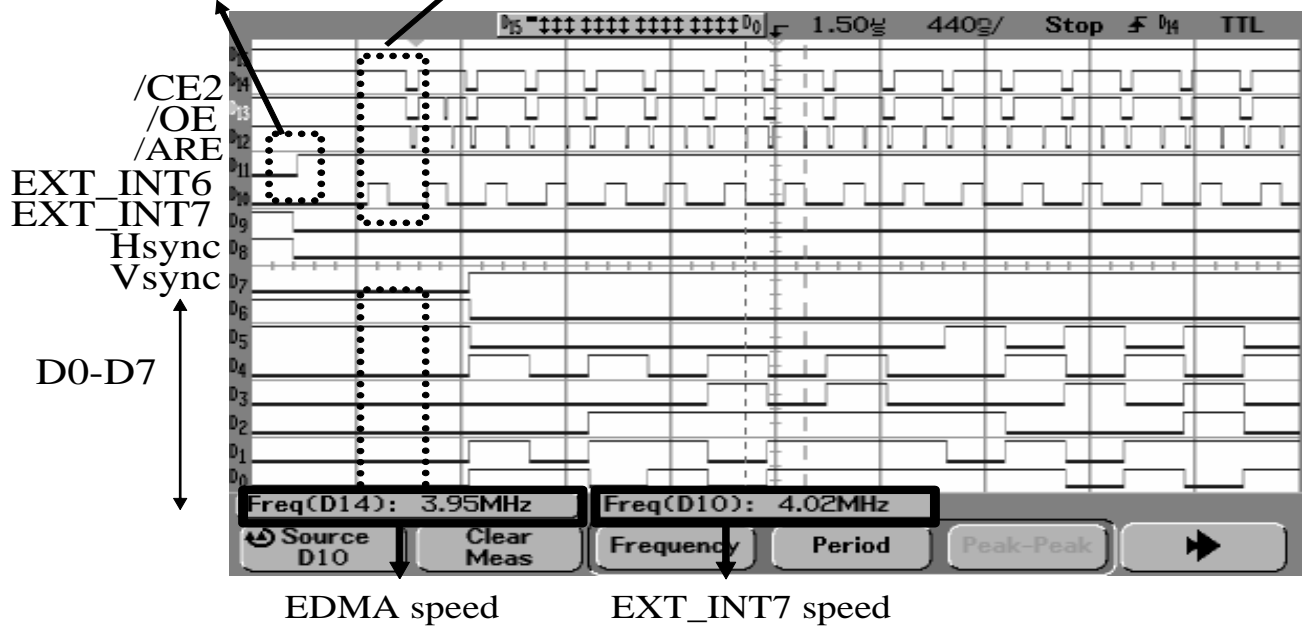
#### 2.4.4.2. Acquired image for 30 fps

In the second implementation of the Buffer\_controller time delay that caused by the switching mode between writing stage and reading stage is reduced that two  $\overline{VSYNC}$  signals is reduced to one  $\overline{VSYNC}$  signal. It means that in the same  $\overline{VSYNC}$  signal, the switching mode between writing stage and reading stage occurs (see figure 2.19).

The time difference between writing stage occurs and the following reading stage in one  $\overline{VSYNC}$  signal (see figure 2.20) is 192 ns or 5.2 KHz. So at the first stage, the Buffer\_controller will first store 8-bit image data from CMOS sensor (writing stage) up to 3 x 640 image data into frame buffer by asserting the  $\overline{WE}$  and disserting  $\overline{WRST}$  control signal,

An incoming EXT\_INT6 signal, to tell C6416 to read out 32-bit from FPGA

C6416 response the EXT\_INT6 signal by enabled EDMA channel, then 32-bit is read out using EMIF channel 2



EDMA speed

EXT\_INT7 speed

Figure 2.19 full timing diagrams of writing and reading stage

/VSYNC [D8]  
 EXT INT6 [D7]  
 EXT INT7 [D3]  
 /RRST [D6]  
 /RE or /OE [D5]  
 Data Out [D0]  
 /WRST [D2]  
 /WE [D1]  
 Data In [D10]  
 Pixel Clock [D9]

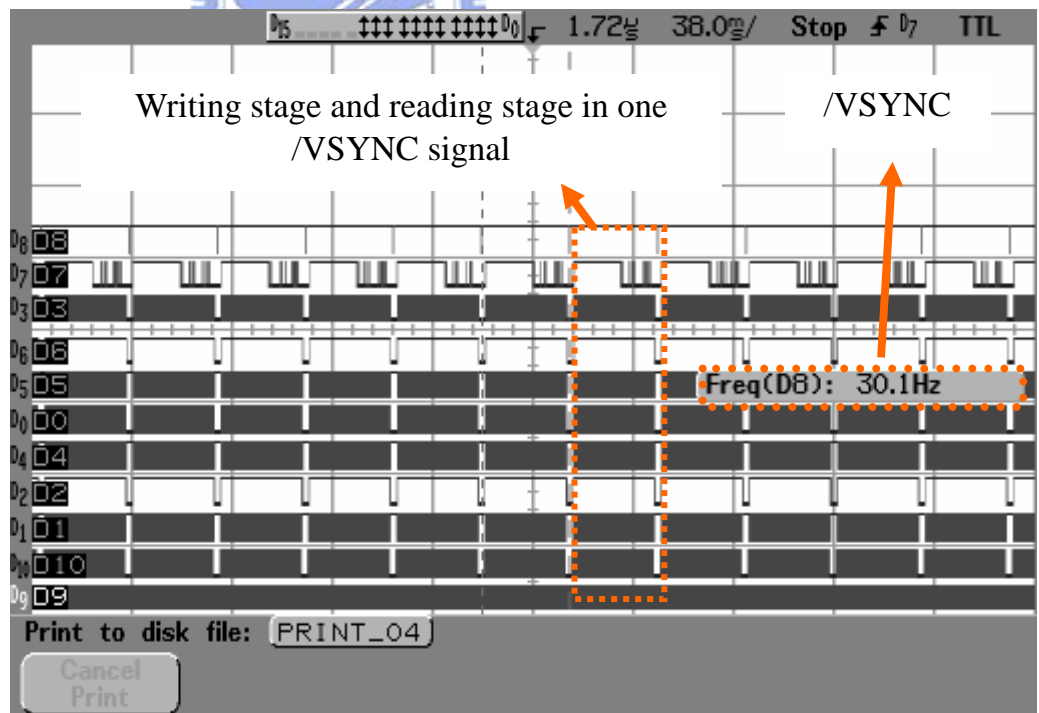
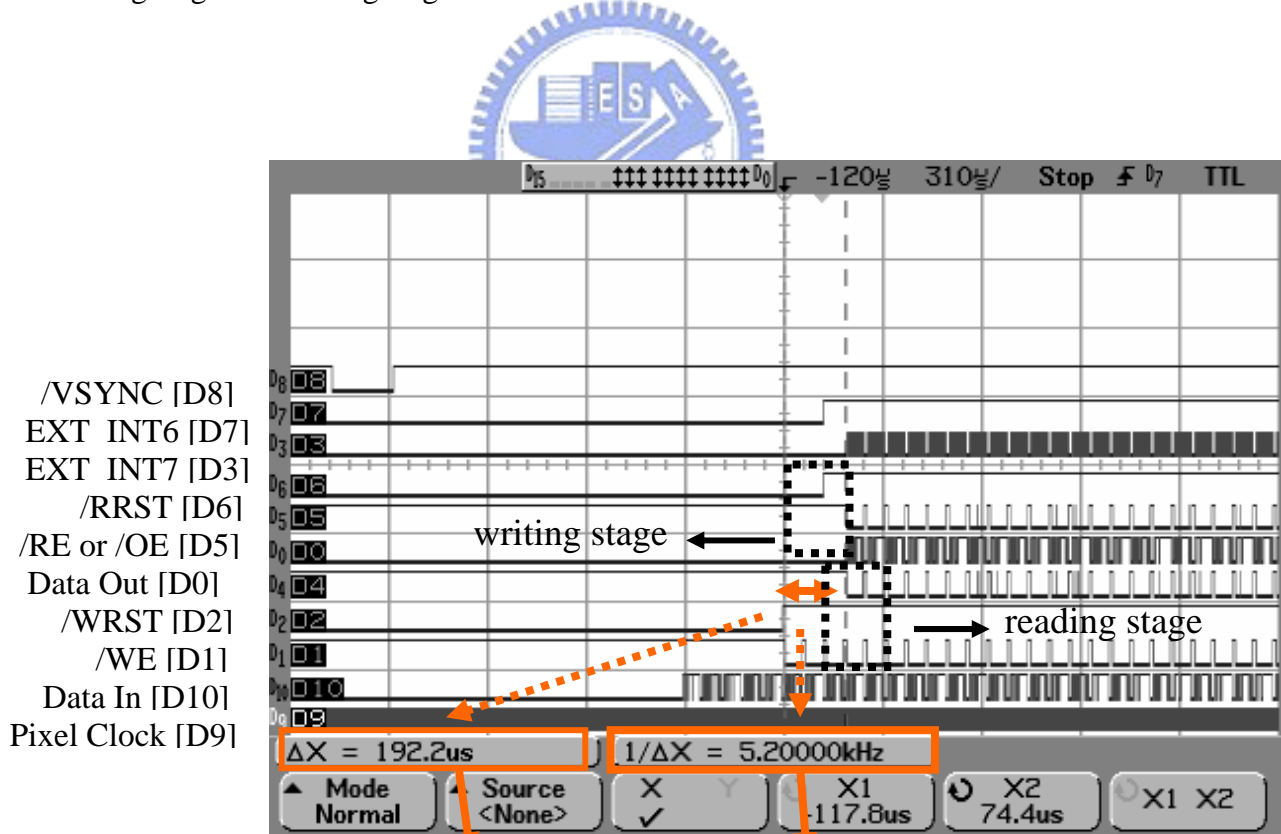


Figure 2.20 full timing diagram of 30 fps configuration

after that the Buffer\_controller activates the reading stage by asserting the  $/RE$ ,  $/OE$  control signal and disasserting  $/RRST$  control signal to read the 8-bit image data out from the frame buffer to move into FPGA, (before it moves to DSK6416). So at the instant of 192 ns (see figure 2.21), the writing stage and reading stage will run concurrently to write and read the 8-bit data image in the Frame Buffer. According to this new configuration method, the frame rate speed of the acquiring image from CMOS sensor is

$$Frame\ rate = \frac{1}{\Delta X} = \frac{1}{192.2\ \mu s} = 5.20000\ kHz = 30\ fps .$$

The 30 fps frame speed can be reach because there is no additional delay between the writing stage and reading stage in the Frame Buffer.



The different time between writing stage and reading stage

Figure 2.21 timing diagram for reading and writing stage in 30 fps configuration

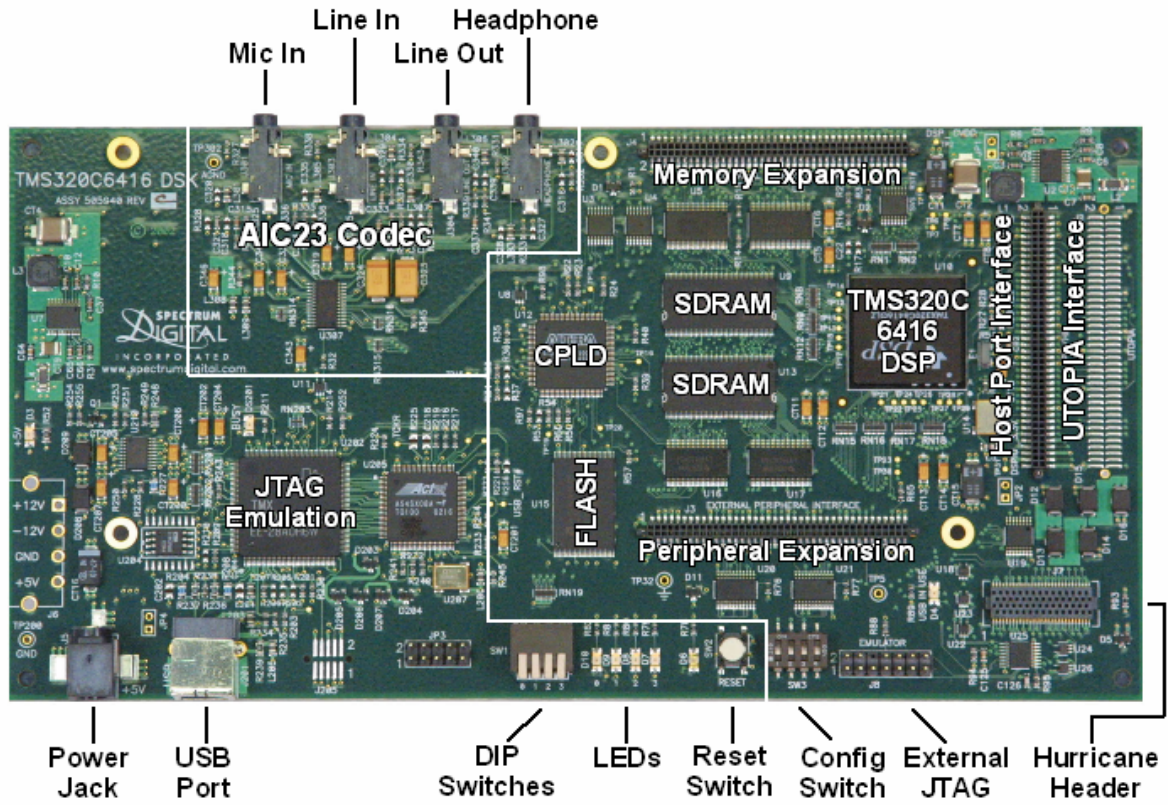


Figure 2.22 DSK6416 board with its main list components [19]

## 2.5. DSK6416 board

Table 2.5 DSK6416 board specification

| No. | Feature name            | Details   |
|-----|-------------------------|---|
| 1   | TMS320C6416 DSPs        | 600 MHz, fixed point, 1 Mbytes internal RAM                               |
| 2   | External RAM            | 16 Mbytes, 64-bit interface   |
| 3   | External Flash          | 512 Kbytes, 8-bit interface   |
| 4   | Daughter card Expansion | Allows users to enhance functionality with add-on daughter cards          |
| 5   | HPI Expansion Interface | Allows high speed communication with another DSPs                         |
| 6   | Embedded JTAG Emulator  | Provides high speed JTAG debug through widely accepted USB host interface |

The DSK6416 is a standalone platform which is equipped with a high-performance DSP chip (C6416), external RAM for a bigger storage solution, external Flash for booting process, daughter card expansion which enabled the DSK6416 to communicate with daughter card / daughter board, with HPI expansion interface to communicate with other DSPs and last Embedded JTAG emulator which enabled the DSK6416 to communicate to Code Composer studio (CCS) while in the development process (see table 2.5) [19]. In our proposed embedded image processing platform, DSK6416 is the main processing platform including acquiring image from CMOS sensor. Next, we will discuss about software implementation acquiring image from CMOS sensor into DSK6416 which need a basic understanding about EMIF for interfacing to FPGA that holds 32-bit image data and EDMA controller for selecting an appropriate transfer mode according to the control signal from FPGA.

### 2.5.1. EMIF (External Memory Interface)

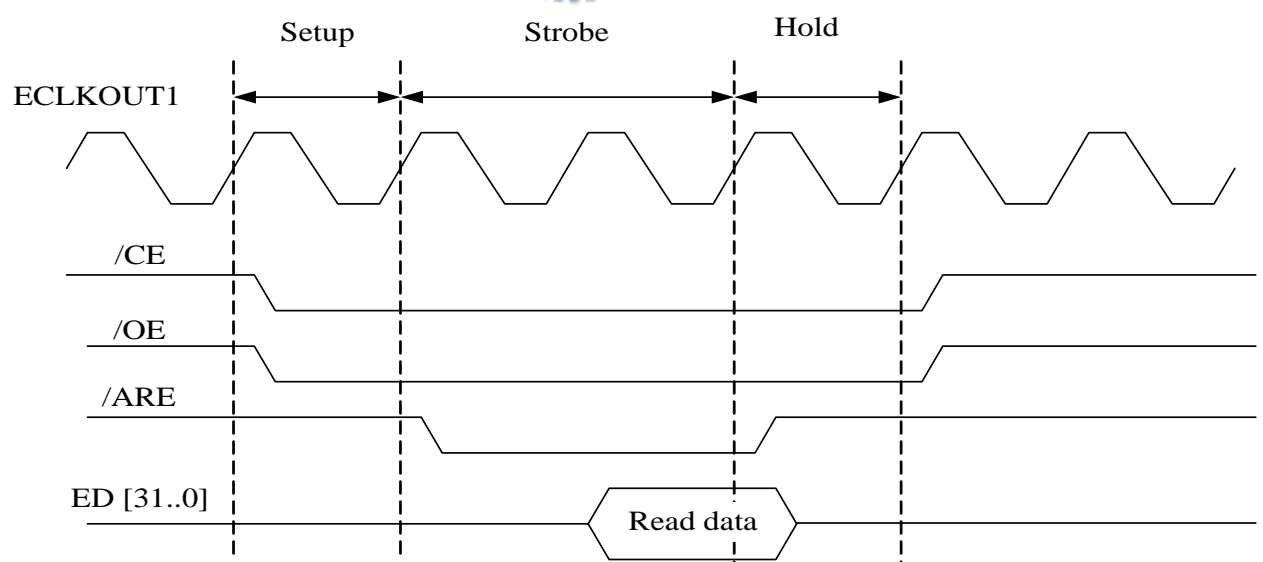


Figure 2.23 basic EMIF asynchronous control signal [23]

The EMIF is designed for a flexible interface to a wide variety of external memory devices [23]. We use the EMIF as interface to FPGA to read 32-bit image data from FPGA. We treat the FPGA as an asynchronous external memory device which is equivalent to FIFO memory architecture. Figure 2.23 showed the basic asynchronous control signal of EMIF.

In order to read 32-bit data through asynchronous interface from FPGA (see figure 2.12), the timing of the EMIF asynchronous control signal must be chosen carefully. The design of required timing can be done by setting the correct setup time, strobe time and hold time for the EMIF timing diagram. In our implementation we choose the correct setup time, strobe time and hold time for the EMIF asynchronous control signal shown in table 2.6.

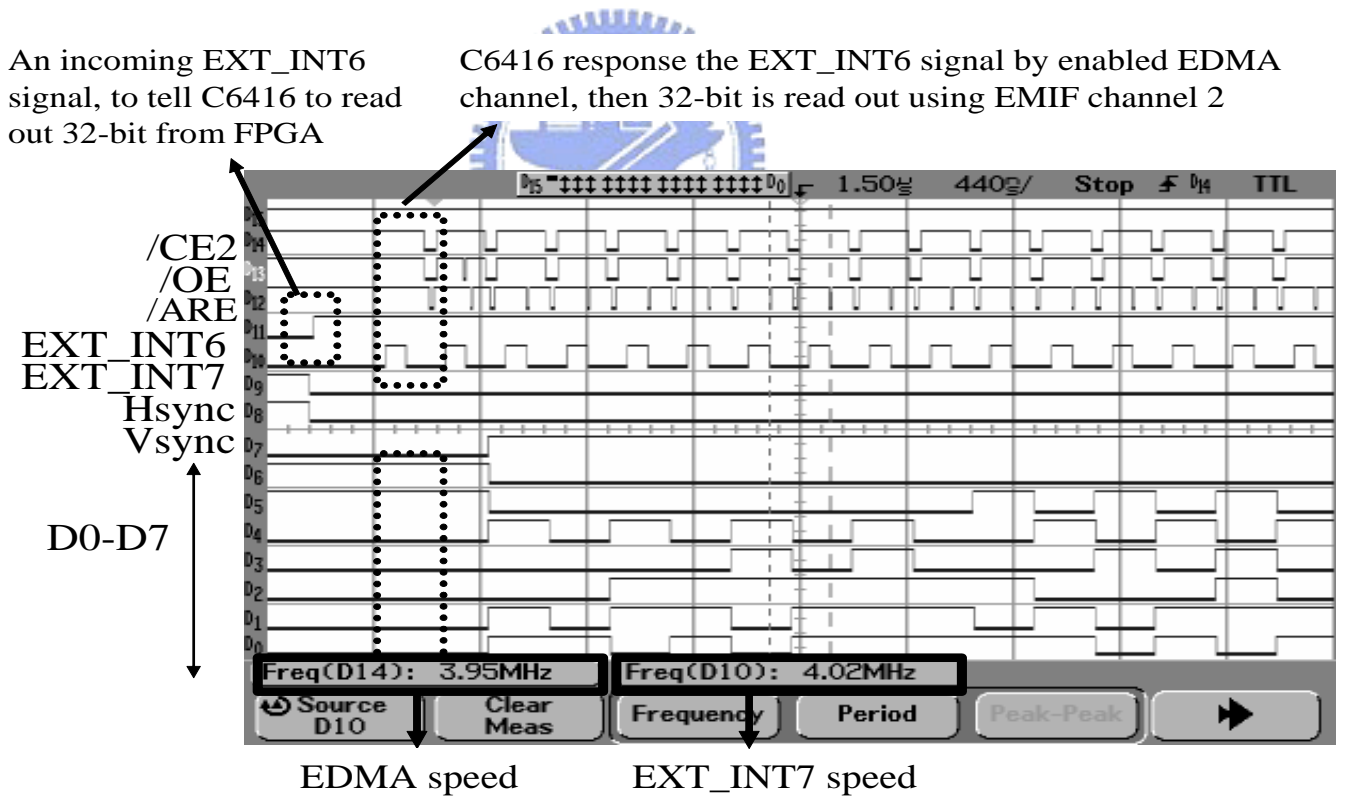


Figure 2.24 EMIF control signal

Table 2.6 lists of setup time, strobe time and hold time

| No | Timing diagram | Value (ns) |
|----|----------------|------------|
| 1  | Setup time     | 60         |
| 2  | Strobe time    | 60         |
| 3  | Hold time      | 20         |

In the figure 2.24, the *EXT\_INT7* control signal shift out the 32 bit image data from FPGA. We can see that every EMIF control signal (*/CE*, */OE* and */ARE*) matches with the *EXT\_INT7* control signal. So the correct 32-bit image data are successfully read out from FPGA and will be stored in the SDRAM.

## 2.5.2. EDMA (Enhanced DMA) controller

The EDMA provides two types of data transfer, 1-dimensional (1D) and 2-dimensional (2D) transfer. The number of dimensions a transfer has determines the makeup of a frame of data. In a 1-D transfer, frames are made up of a number individual element.

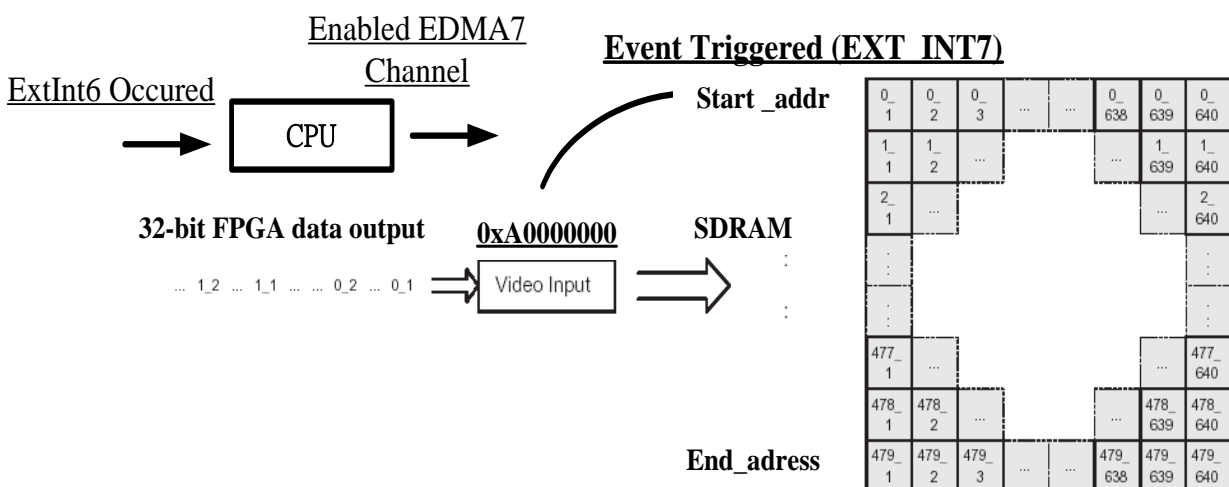


Figure 2.25 the configuration of the EDMA controller [24]



In a 2-D transfer, blocks are made up of a number of arrays, each of which is made up a number of elements. According to our previous implementation of Buffer\_controller, the control signal that provided from Buffer\_controller is similar to 1-dimensional (1D) transfer. It is because that the output of the Buffer\_controller are made up of a number individual which is a individual 32-bit image data with following two control signal that are *EXT\_INT6* and *EXT\_INT7*. In the figure 2.25, we can see that an incoming *EXT\_INT6* will tell the CPU that there is one incoming image frame. The CPU then will open EDMA channel 7 that is already directly mapping with the *EXT\_INT7* signal.

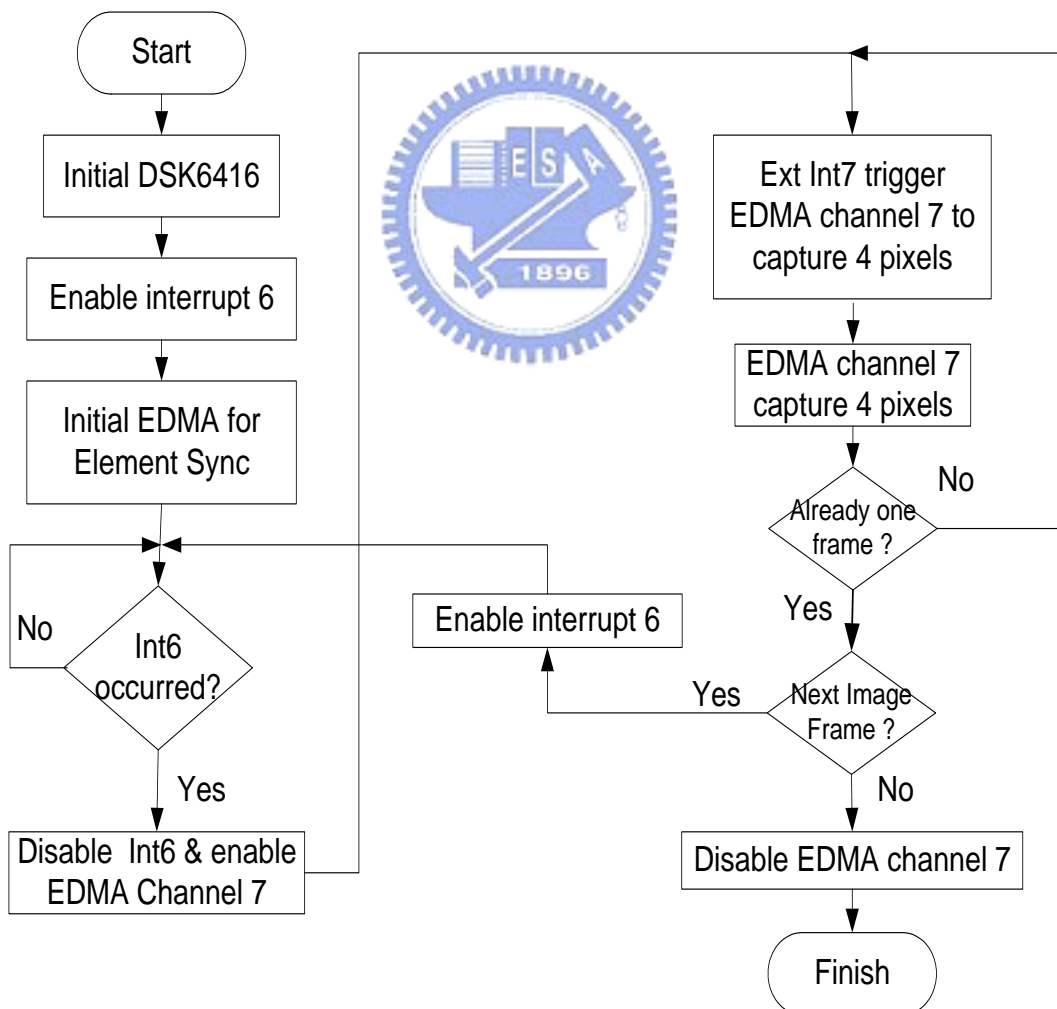


Figure 2.26 the flow chart of the implementation of the EDMA transfer

So for every incoming *EXT\_INT7* interrupt will trigger the EDMA to read out an individual 32-bit image data from the FPGA. And this is repeated until 640 x 480 image data been read out from FPGA. The flowchart of the implementation of EDMA transfer is shown in the figure 2.26.

## 2.6. Experimental result Image processing

In this section, we show experiment results of acquiring image from the embedded image processing platforms. After acquiring image from the embedded image processing platform, we do a simple processing with the acquired image by using simple algorithm. In the first processing of the image, we use sobel mask to find an edge detection of the image. Later we use image threshold to make a binary image. The embedded image processing platform uses the second configuration with a frame rate of 30 fps. The image resolution is 640 x 480 pixels.

The first experiment is acquired one image frame with 640 x 480 pixels, from the embedded image processing platform. Figure 2.27 show the original image acquired from CMOS sensor. Figure 2.28 shows the original image while applying the sobel mask. Figure 2.29 shows the binary image from thresholding.



Figure 2.27 original image of the CMOS sensor

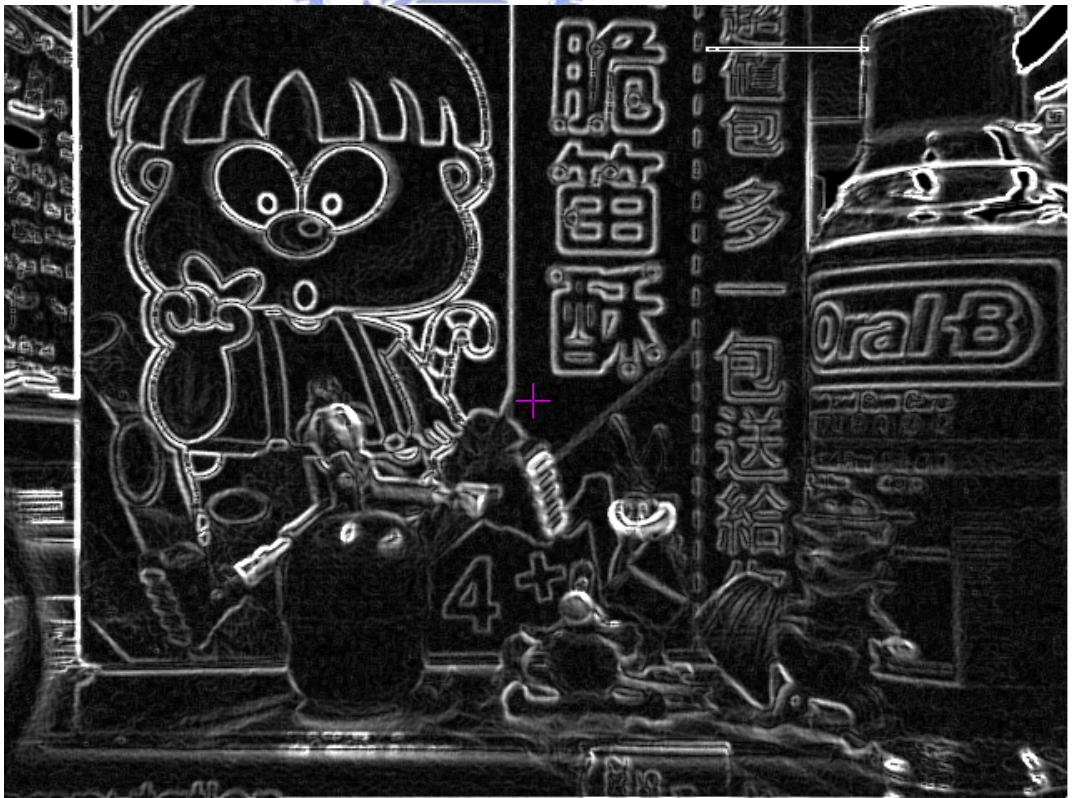


Figure 2.28 applying sobel mask operation in the original image



Figure 2.29 applying binary operation in the original image



# 3. Visual Servoing

## 3.1. System architecture

In the early stage of the mobile manipulator research, Connel introduced a behavior-based arm controller in his paper [8]. In his research, Connel combined a multi sensor feedback with the eight loosely-coupled on-board 8-bit microprocessors using behavior-based arm controller. Similar with the work, we propose a motion control for mobile manipulator by combining a behavior-based controller that does not use multiple sensor feedback but only use one visual sensory feedback. The proposed behavior-based algorithm is based on *subsumption architecture* which is introduced by Brooks [9] (see figure 1.1). The *subsumption architecture*, control is layered with higher layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.

In the design of the visual servoing, we choose hand-in-eyes camera configuration to reduced the need for calibrated camera-to-manipulator coordinate transform, and it closed the control loop by providing visual feedback (robot motion caused camera motion) without requiring the vision system to track the end-effector's (see figure 3.1).

Our proposed algorithm is implemented in the embedded image processing platform for controlling the mobile manipulator robot. The architecture of visual servoing design can be seen in the figure 3.2.

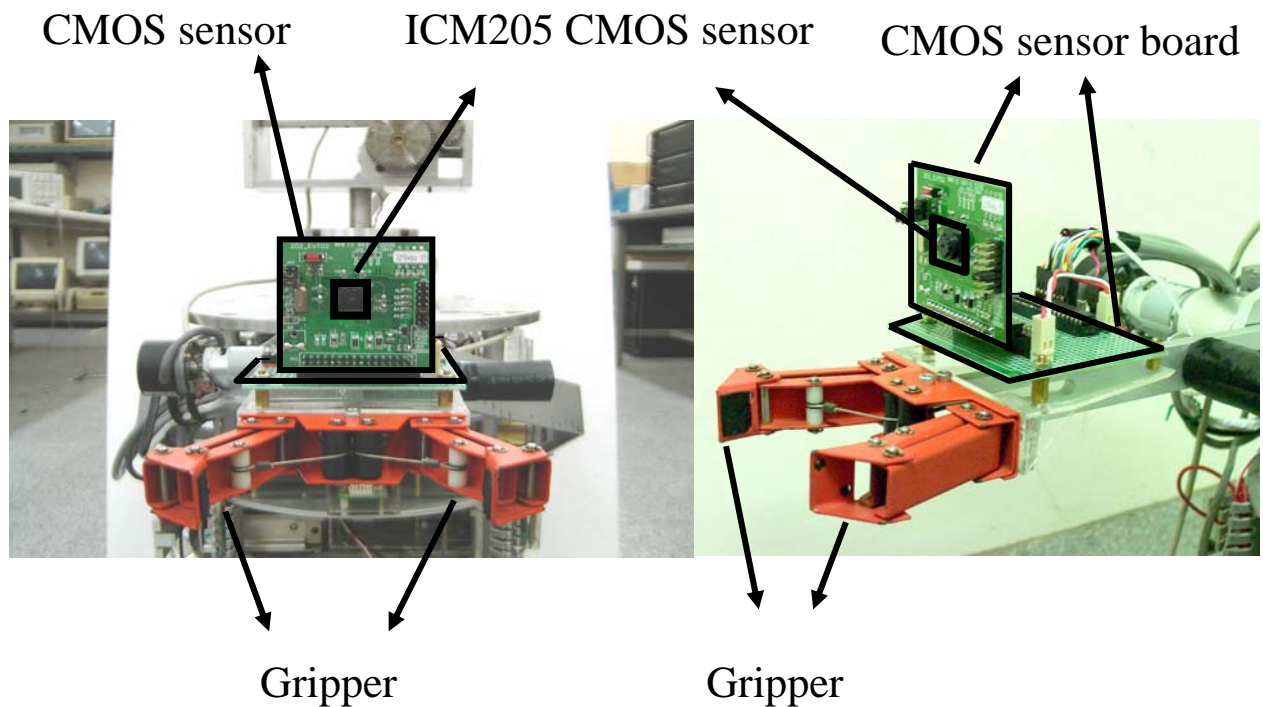


Figure 3.1 hand-in-eye camera configurations

The visual servoing based on behavior-based architecture is divided into three layers. These three layers are navigation layer, vision guide layer, and grasping layer. The navigation layer uses mixed optical flow algorithm [14] to navigate the robot to avoid obstacles in dynamic environments. The second layer is vision-guided layer; the functional of this layer is to guide the manipulator approaching certain object with barcode-like print on the body of the object. The third layer is grasping layer that has functional to grasp the object while the object is already detected in the gripper of the mobile manipulator. Based on the *subsumption architecture*, the higher level of the visual servoing based on behavior-based (Grasping layer) will subsuming the lower layer (Vision-guided layer and navigation layer) example: if there are already detected the object in the gripper so the gripper will closed and the robot will stop.

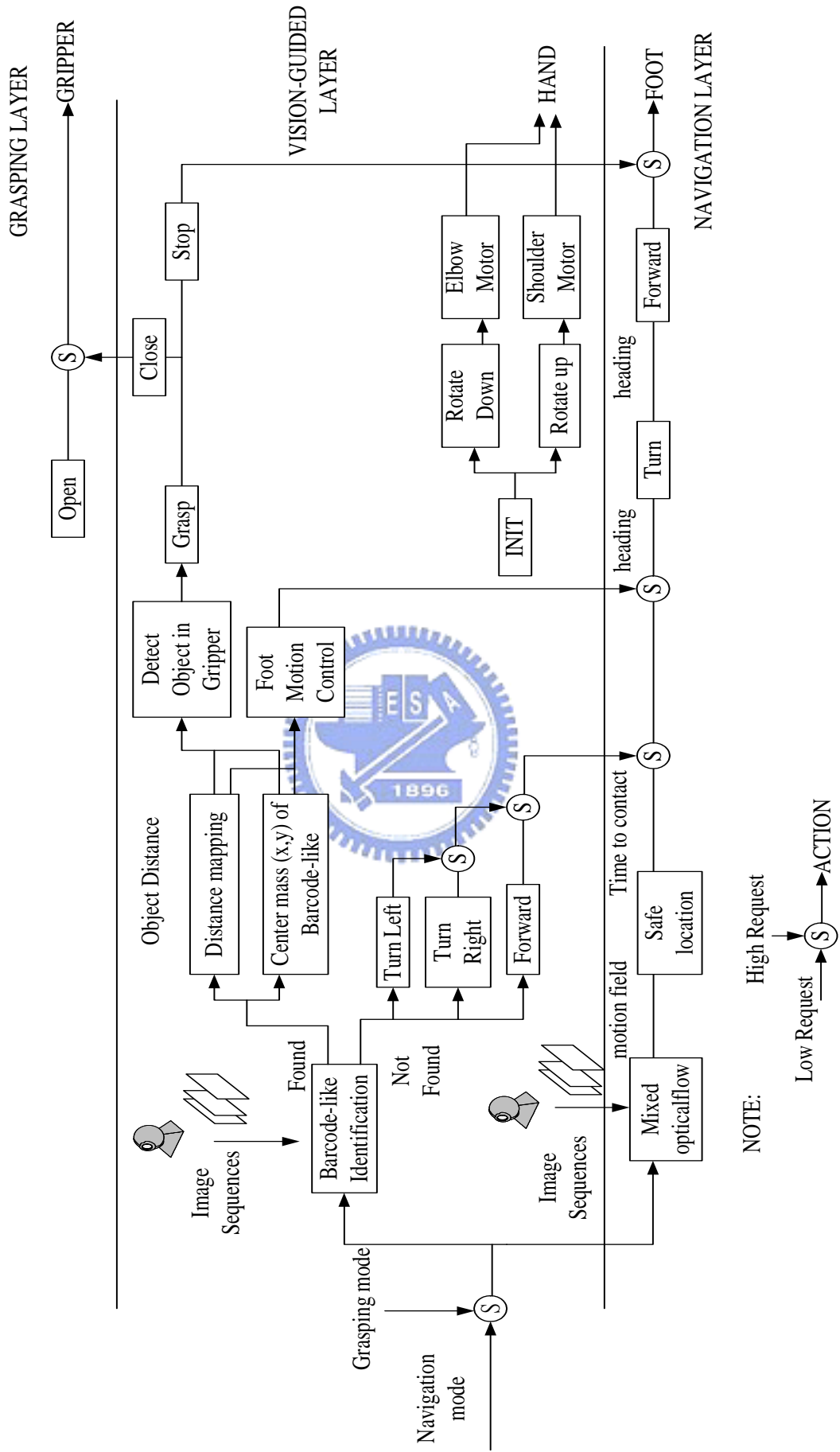
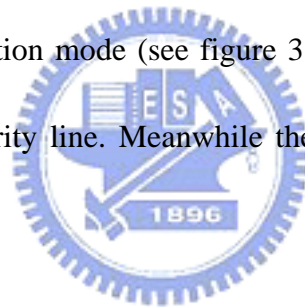


Figure 3.2 Architecture of the visual servoing based on behavior-based

Later, the communication line between each behavior uses priority request. Behavior module with high priority request will suppress the low behavior priority request (see figure 3.3). The lower priority request behavior module can only pass the communication line if there is no any request from higher priority behavior module. Using this communication line, it will guarantee the high level behavior can always pass the communication line.

## 3.2. Switching mode

In order to choose which layer that will active at certain time, the grasping mode (see figure 3.4) and the navigation are allocated in the same communication line but the grasping mode can suppress the navigation mode (see figure 3.3). It is because the grasping mode is put in the higher request priority line. Meanwhile the navigation mode is put in the lower request priority line.



If the navigation mode is activated and the grasping mode is not activated, the mobile manipulators will activate the navigation layer. If the grasping mode is activated, the mobile manipulators will activate the vision-guide layer and grasping layer to grasp a barcode-like object.

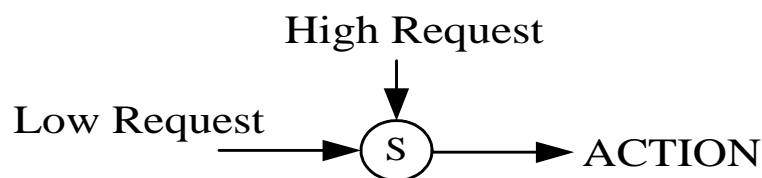


Figure 3.3 communication lines between each behavior module



This selection mode behavior is sent by main processor of robot (IPC) to the embedded image processor platform whether to switch which one of the modes to be activated although the navigation mode is still active because of the priority of the grasping mode is higher than the navigation layer.

A simple example scenario of using our proposed algorithm: when the mobile manipulator does not find the object or the object location is far away from mobile manipulator, the mobile manipulator robot still can move autonomously in the environment by activating the navigation layer. Meanwhile, when mobile manipulator robot located near with the barcode-like object, then by activating the grasping mode (see figure 3.4) so the mobile manipulator robot can try to locate the barcode-like object by scanning its environment, and when the mobile robot locate the barcode-like object it will approach to the object by changing its speed and orientation until the barcode-like object get in to in the gripper and then it grasp the object.

## **3.3. Barcode-like Object**

### **3.3.1. Definition**

It is important to define an object that the robot wants to grasp first. By knowing the object, the mobile manipulator robot then can locate the object and grasp it. To knowing or identifying an object using visual information, it is required some good feature on the object over another objects or environment so the object can be identified.

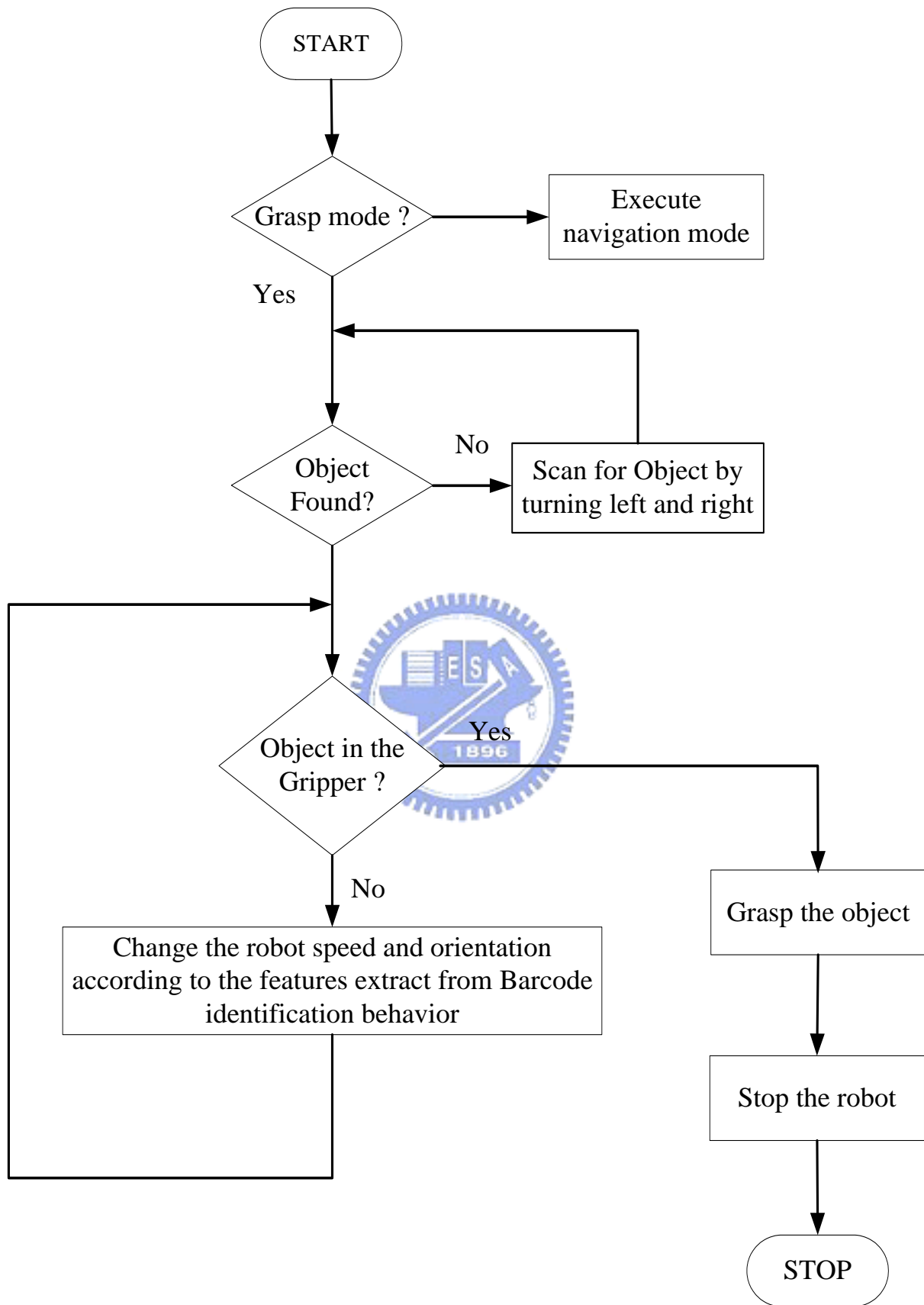


Figure 3.4 flow chart of grasping mode

In our proposed embedded vision system, the image data only provide 8-bit gray level information without any color information so we can not use color information as the feature of the object. The feature of the object can only extract gray-level information. This however is not a limitation of our platform, because by using gray-level information the feature of the object still can be extracted. There are many research has done related on this problem. Yoon[15] that using 3D geometric model to identify object, Kaneko[16] that using pyramidal-like objects as the feature to identify the object and Richards[17] using “figure/ground” approach to identify the object.

Related to this problem, we create a unique feature for our object to be easily identified. For the unique feature, we choose a barcode-like feature as object feature (see figure 3.5).

The barcode-like feature is divided into two kinds of bars which are vertical bar that located in the middle of the barcode-like and the horizontal bar that located in the top and bottom of the vertical bar.

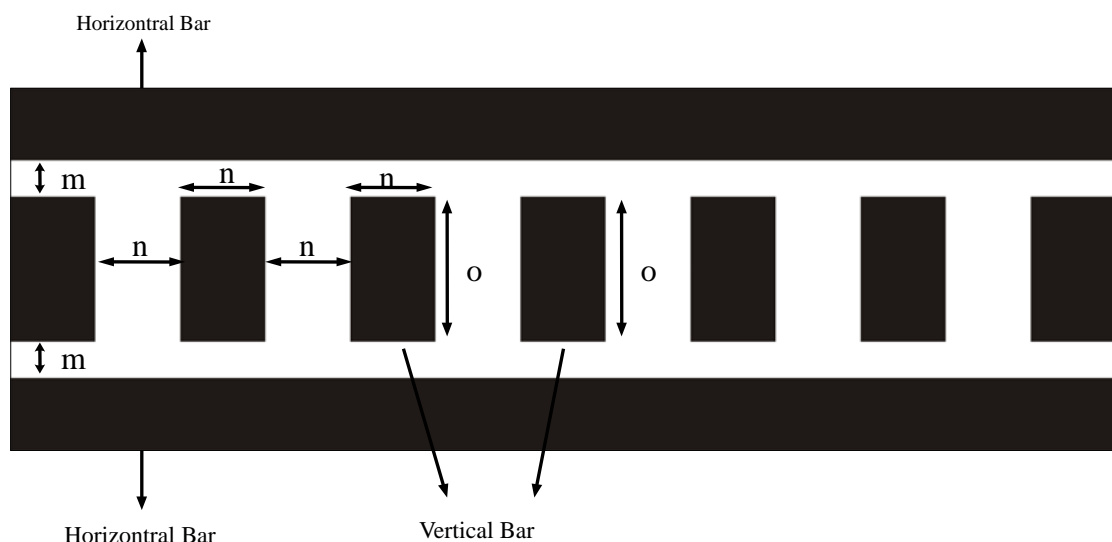
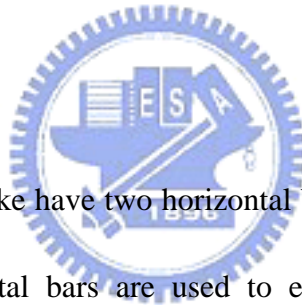


Figure 3.5 barcode-like features as the main feature of our object

The height ( $o$ ) and width ( $n$ ) of each vertical bar is same with other vertical bar. The length of the horizontal bar somehow is design to cover the vertical bar. The spacing ( $n$ ) between first vertical bars with other vertical is same as the width. The spacing ( $m$ ) between top horizontal bars and vertical bars is same as the spacing ( $m$ ) bottom of horizontal bars and vertical bars.

To use the barcode-like feature on the object, the barcode-like must be labeled in the object. So any object that is labeled with the barcode-like can be identified by our mobile manipulator robot. The idea of choosing the barcode-like feature is that it is simple to search. It does not take a long time to identify the barcode-like, unique structure so it can differentiate the barcode-like features with other object in the environment.

### 3.3.2. Remarks



The design of the barcode-like have two horizontal bars that cover the middle vertical bars (see figure 3.5). The horizontal bars are used to extract the width of the barcode-like. Meanwhile the vertical bars are used to extract the height of the barcode-like.

The structure of our barcode-like is different from ordinary barcode structure that uses to label manufacture product. The ordinary barcode structure is thinner than ours. So when applying this kind of barcode, it has a weak feature to identify it.

The size of the barcode-like is already well defined ( $m$ ,  $n$  and  $o$ ). This well defined size is already matched with the lookup table (see table 3.1), so it can be used to transform to estimate the real distance of the object. Although the size is already well defined, it still can be re-defined. When the size of the barcode-like redefined, the lookup table (see table 3.1) also

has to change according to the new size of the barcode-like.

### 3.4. Barcode-like identification

If an object in the image frame is labeled with barcode-like, the object will have features that can be used to identify. In this section, we will discuss our proposed algorithm to identify the barcode-like features. The algorithm begins by down sampling the acquiring image from CMOS sensor into 160 x 120 pixels.

After that the image is converted into binary image. The converted binary image is then filtered by 1D horizontal and vertical filter. This filter is to filter a noise such as a small bar in the horizontal and vertical direction. After that it begins to find the barcode-like feature in vertical direction.



We choose vertical direction because we want to find the horizontal bar first before finding the vertical bar in the barcode-like feature. It is to prevent a false detection of barcode-like feature because the horizontal bar is covering the vertical bar. This 1D vertical search is to find the corner of the barcode-like feature. Meanwhile, the distance of the barcode-like object can be extracted from barcode-like distance transform. The barcode-like distance transform is constructed using a look up table (see table 3.1) to extract the distance. We choose look up table because this method is suitable for hardware implementation and faster than other method. Figure 3.6 illustrates the flow chart of the proposed algorithm.

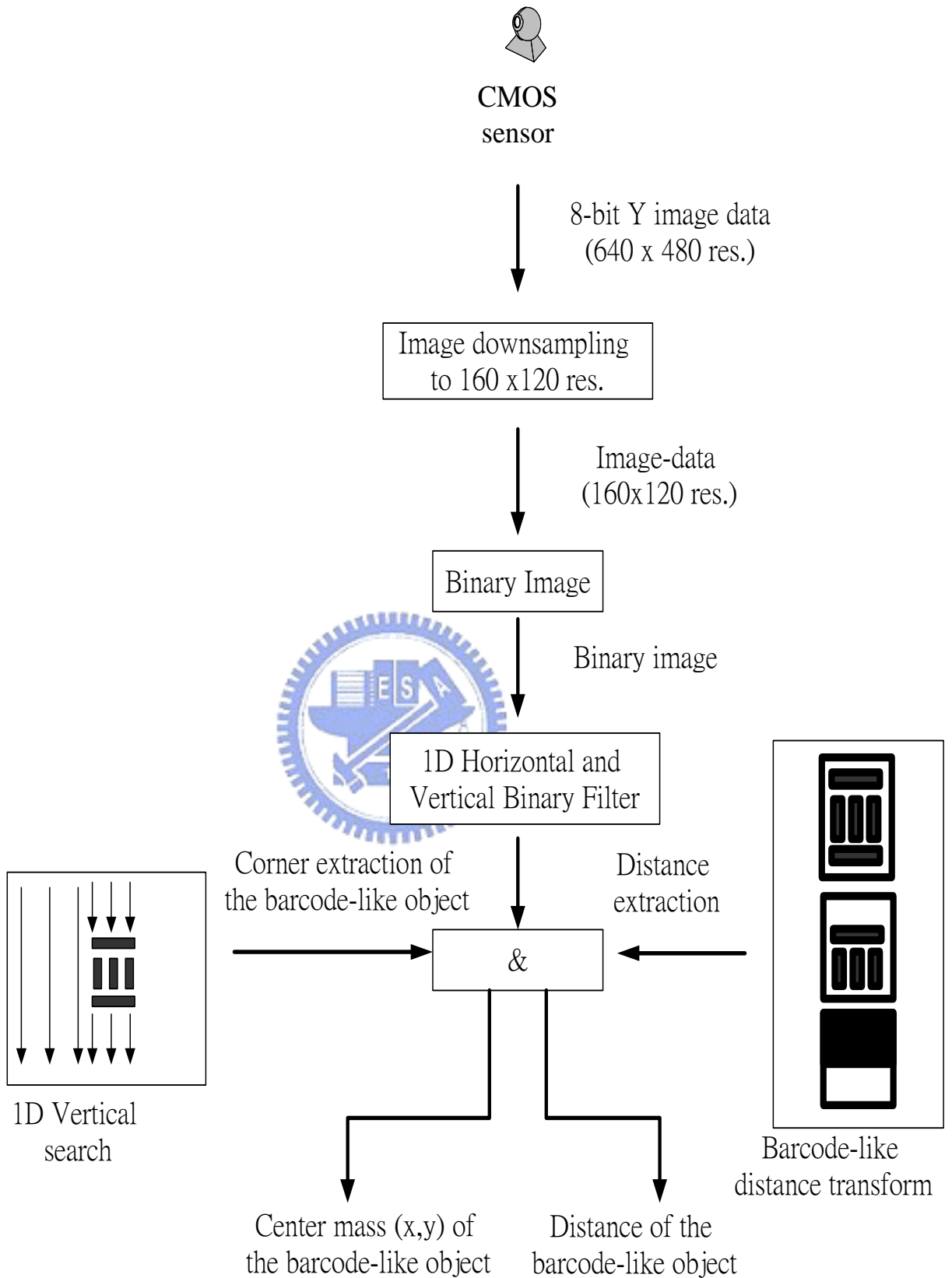
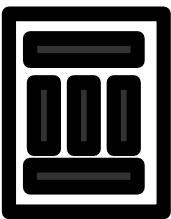




Figure 3.6 the flow chart of barcode-like identification

The barcode like distance transform uses three kinds of possibilities of the barcode-like features. The first barcode-like feature is to detect whether the barcode-like still consist of two horizontal bars, if it so then the possible distance are 80 cm, 60 cm or 40 cm. The second barcode-like features is to detect whether the bottom horizontal is missing if it is so then the possible distance are 30 cm or 20 cm. The last barcode-like features is detected when the object already in the gripper, with the distance is 5 cm.

This transformation distance is then compared with the current barcode-like feature that was found by 1D vertical search to extract the correct distance of the barcode and the center mass of the barcode-like object (see table 3.1)

Table 3.1 barcode-like feature to distance transform

| No | Barcode-like features   | Pixel to distance mapping                      |                        |                    |
|----|---|--|------------------------|--------------------|
|    |   | Height of Top Horizontal Barcode-like (pixels) | Estimate distance (cm) | Real distance (cm) |
| 1  |  | 1 - 3  | 80                     | 70 – 80            |
|    |   | 4 - 6  | 60                     | 50 – 60            |
|    |   | 7 - 10   | 40                     | 30 – 40            |
| 2  |  | 11 - 14  | 30                     | 20 – 30            |
|    |   | 15 - 20  | 20                     | 10 – 20            |
| 3  |  | 21 - 90  | 10                     | 5 – 10             |

The transformation of object distance begins by comparing the top horizontal barcode feature found by the 1D vertical search. For example: If the current top horizontal barcode feature found by the 1D vertical search has height about 1 – 3 pixels, according to the look-up table, the estimate distance of the object is about 80 cm. The difference between estimate distance and real distance is the estimate distance will be used in the computational for calculate the speed of the mobile robot while approaching the object while the real distance is the real distance of the object but in the certain range.

### 3.5. Foot motion control

The foot motion control behavior is used to control the orientation and velocity of the mobile robot while approaching the barcode-like object. Figure 3.7 show the model of the mobile robot system. According to the model, we can derivate:

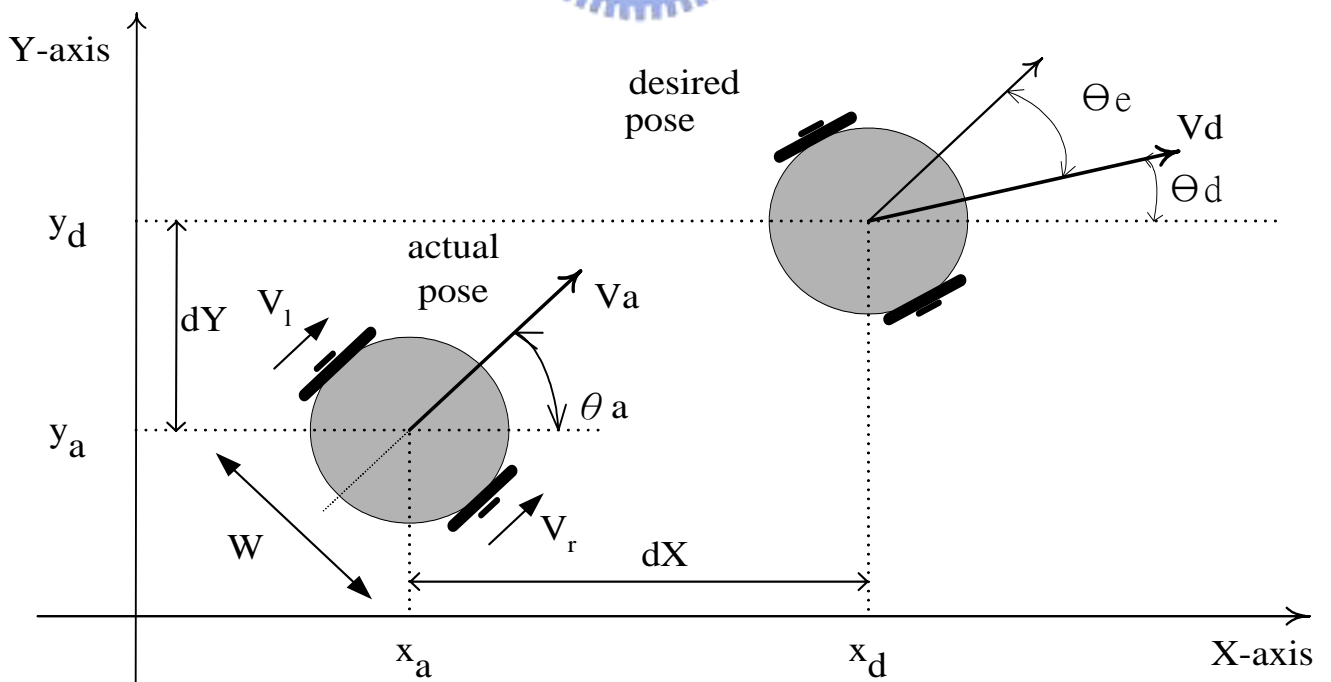


Figure 3.7 the model of non-holonomic mobile robot system



$$v_d = \frac{v_l + v_r}{2} \quad \dots\dots\dots (3.1)$$

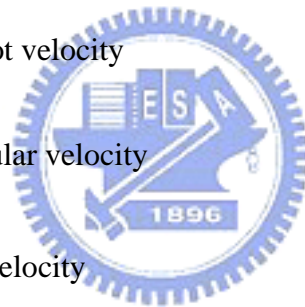
$$\theta_d = \omega_d = \frac{v_r - v_l}{D} \quad \dots\dots\dots (3.2)$$

$$dX = x_d - x_a \quad \dots\dots\dots (3.3)$$

$$dY = y_d - y_a \quad \dots\dots\dots (3.4)$$

Where:

- $V_a$  is the actual robot velocity cm/s
- $\theta_a$  is the actual angular velocity rad/s
- $V_d$  is the desired robot velocity cm/s
- $\theta_d$  is the desired angular velocity rad/s
- $V_L$  is the left motor velocity cm/s
- $V_R$  is the right motor velocity cm/s
- $D$  is the diameter of the robot 30 cm
- $X_a$  is the actual location of the robot in the x-axis cm
- $Y_a$  is the actual location of the robot in the y-axis cm
- $X_d$  is the desired location of the robot in the x-axis cm
- $Y_d$  is the desired location of the robot in the y-axis cm
- $dX$  is different location between estimate with actual in x-axis cm



- $dY$  is different location between estimate with actual in y-axis                      cm
- $\theta_e$  is different between the actual angular speed with estimate angular velocity  
rad/s

From equation 3.1 and 3.2, we can derivate the left motor speed and right motor speed:

$$V_L = \frac{2V_d - D\omega_d}{2} \quad \dots\dots\dots \quad (3.5)$$

$$V_R = \frac{2V_d + D\omega_d}{2} \quad \dots\dots\dots \quad (3.6)$$

The left motor speed (3.5) and the right motor speed (3.6) can be controlled directly by sending a speed command to the DSP motion card that controlling the left motor and the right motor. Figure.3.8 shows the foot motion control system. The foot motion control is designed using the concept of image-based visual servo (IBVS) control. In this control scheme the closed loop control scheme uses sensor information is used as the visual feedback which is the features of the barcode-like.

The features of the barcode consist of two kind's information which is the barcode-like distance (cm) and the center-mass of the barcode like in the x-axis (pixels). The difference between the center mass of the barcode-like in x-axis and center mass of image plane in x-axis (error in pixels) is then transformed into  $\omega_d$  (desired angular velocity – rad/s) by scaling down the error (pixels) using  $1/k$  scaling factor to desire of the orientation of the robot. Similarly, extracted barcode-like distance (cm) is also directly transformed into  $V_d$  (desired robot velocity – cm/s) by scaling down the distance (cm) using  $1/m$  scaling factor to desire of

the velocity of the robot. Tables 3.2 show the value of the k and m that are used in the foot motion control.

The value k in the table 3.2 consists of two values. These two values will depend on the distance of the barcode-like. The idea behind this is to make the estimate orientation of the robot can be adaptive. While the object distance is greater than 30 cm the robot orientation tried to locate in bigger orientation. Meanwhile, the object distance is smaller than 30 which it means that the robot is approaching the object; the orientation of the robot is made to be smaller.

On the other hand, the value of m is to control the velocity of the robot according to the object distance of the robot. We define the value of m is constant to the object distance of the robot, it is because we want to make the velocity of the robot is constant so the robot can move smoothly while approaching the object.

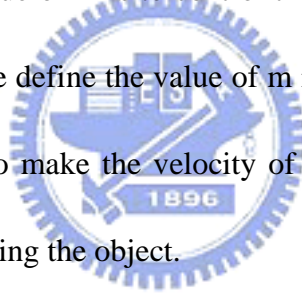


Table 3.2 the scale factor value for foot motion control

| No | Scale factor | Value | Status                           |
|----|--------------|-------|----------------------------------|
| 1  | K            | 16    | < 30 cm (barcode-like distance)  |
|    |              | 8     | >= 30 cm (barcode-like distance) |
| 2  | M            | 4     | ---                              |

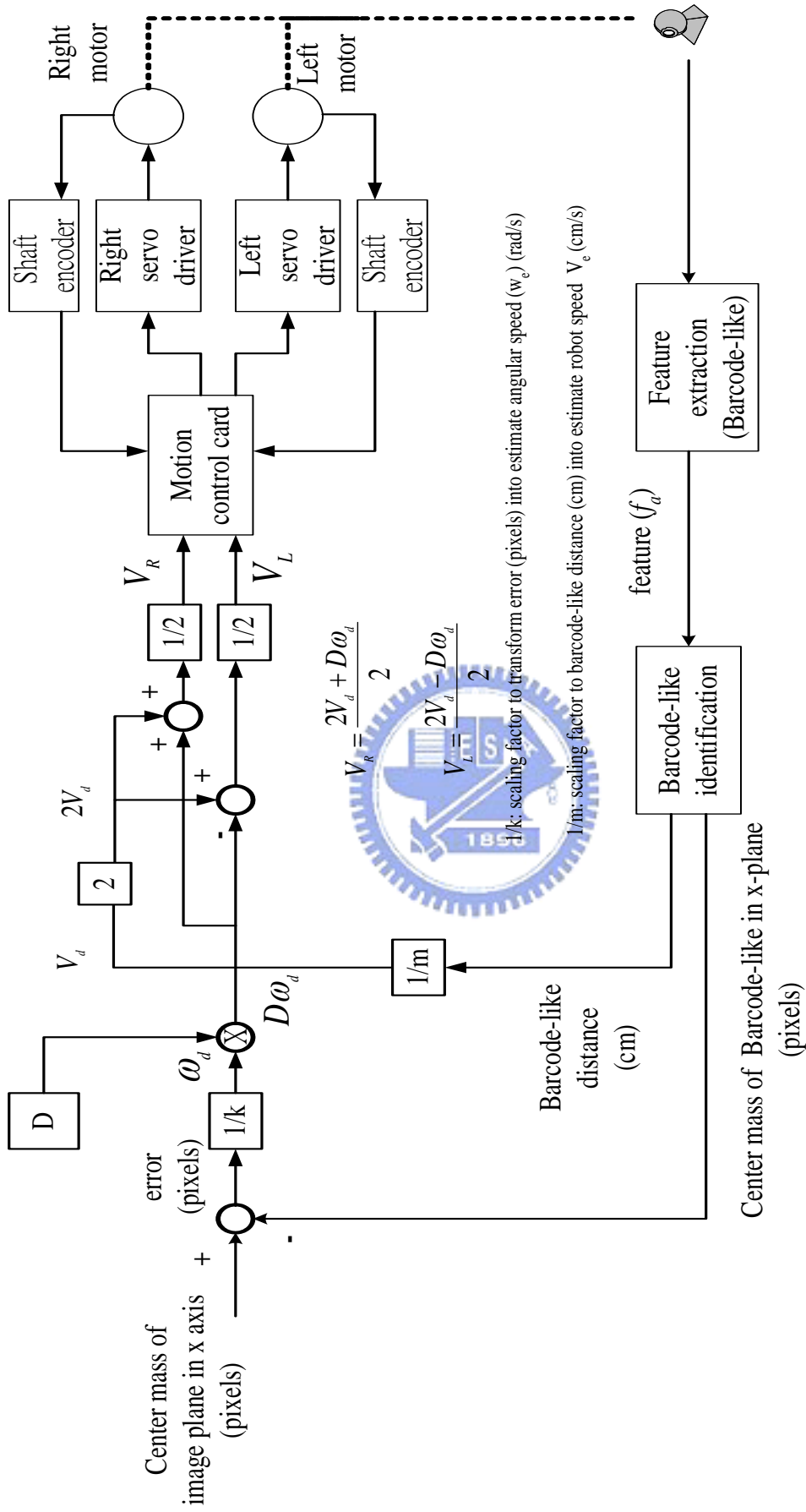
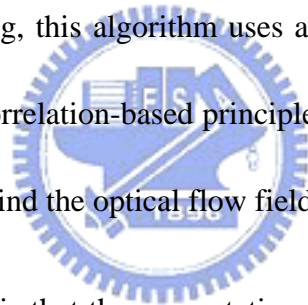


Figure 3.8 Foot motion control

# 4. Obstacle Avoidance Using Optical Flow

## 4.1. Introduction to the algorithm

Mixed optical flow algorithm was proposed by an earlier graduate student in our lab [14] this algorithm features a combination of the conventional correlation-based principle and the differential-based method for optical flow estimation. By employing image intensity gradients as features for pattern matching, this algorithm uses a brightness constraint to configure the search area. After that using correlation-based principle to search the best match between two successive images and finally find the optical flow field.



The merit of this scheme is that the computation load can be greatly reduced and in the mean time the possibility of estimation error is decreased. After obtaining the estimated optical flow field, we can calculate the scene depth and time-to-collision (TTC).

This depth information of object is used as input to an obstacle avoidance algorithm. This algorithm was implemented using Pentium 233 MMX and took 0.9 second to perform one calculation of obstacle avoidance. The flowchart of the proposed algorithm can be seen in figure 4.1[14]

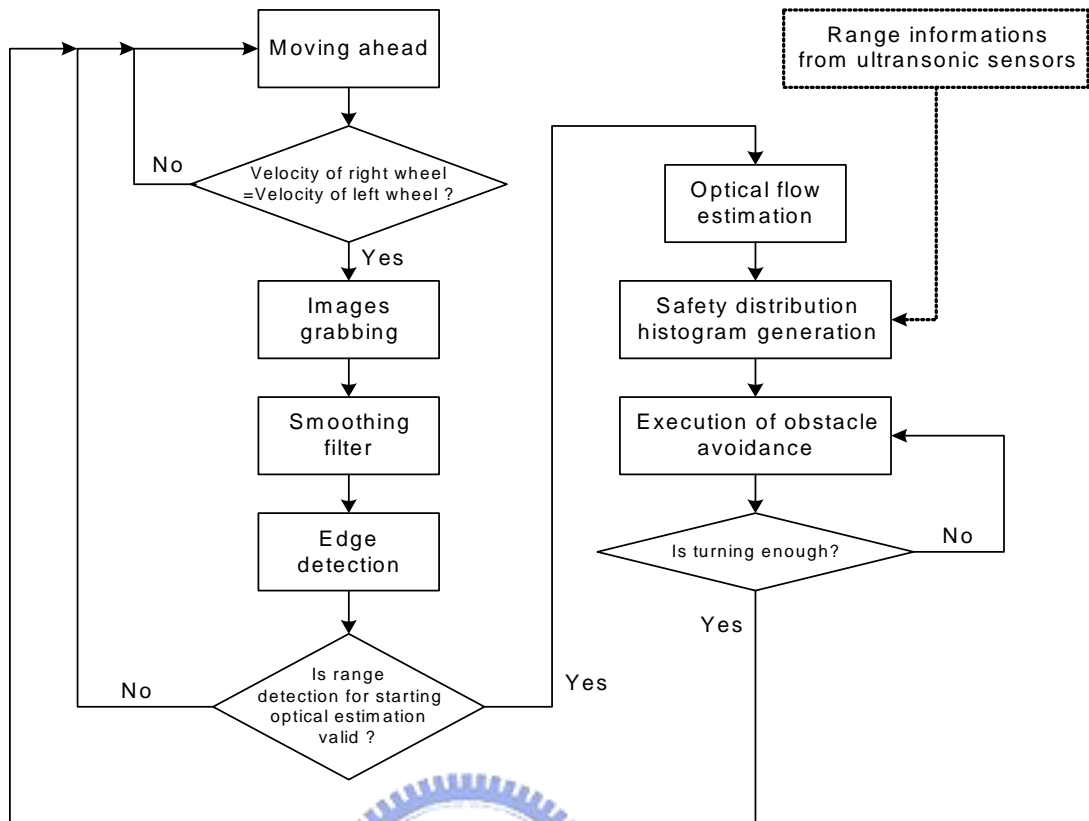


Figure 4.1 flowchart of mixed optical flow algorithm [14]

## 4.2. Implementation of mixed optical flow algorithm

In this thesis, the mixed optical flow algorithm is implemented in the embedded image processing platform. We implement the mixed optical flow algorithm as part of our behavior-based architecture in the navigation layer for navigating our mobile manipulation robot that already describe in previous section. Figure 4.2 showed the flow chart of implementation mixed optical flow algorithm in our embedded image processing platform.

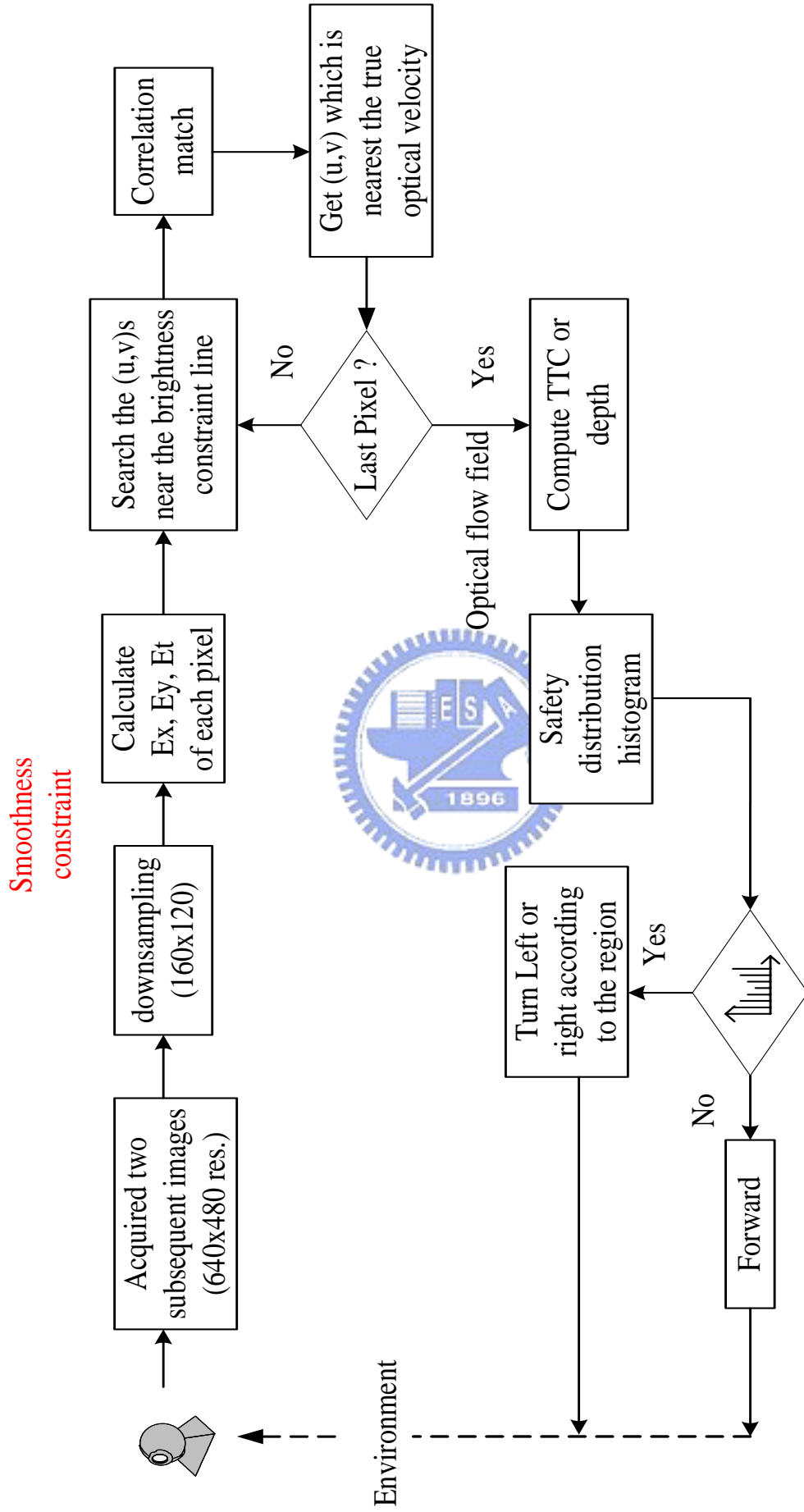


Figure 4.2 Flowchart of mixed optical flow algorithm implement in our embedded image processing platform

There are several differences between the original mixed optical flow algorithms with the implementation in our embedded image processing platform. The next following section will describe the implementation of mixed optical algorithm in our platform.

### **4.2.1. Acquired two subsequent images**

The first step of the mixed optical flow acquires two subsequent images. These two images is acquired from CMOS sensor, these two subsequent images resolution are 640 x 480 and just acquired 8-bit Y image data. Although the embedded image processing platform can acquire image for 30 fps, but we choose to use 15 fps. It is because our robot speed is not fast enough (15 cm/s) so if we use 30 fps for acquired image the mixed optical flow algorithm can hardly detect motion field because these two subsequent images nearly same so it is not detect any motion field. We do not increased the speed of our robot instead of reducing the frame rate, it is because the robot speed is already ideal according to robot heavy and after there is a manipulator attach on it. So we consider about the safety for the movement of the robot.

### **4.2.2. Down sampling**

The method to do the down sampling from 640 x 480 to 160 x 120 is to fetch original image data (640x480) out for every 3 pixel (1, 4, 7, 10, ..., 640 x 480 ) and put in the new address location. After the data already fetch out, the new image data resolution is reduced to 160 x 120. We choose this method because it fit for hardware implementation which can be done by using EDMA transfer that already available in the C6416 processor. The simplicity



and need not any computational to do the down sampling that can make this method faster than other method such as *nearest neighbor interpolation* or *bilinear interpolation* [25]. The result of this method also is good enough and without reduces the feature of the image such as edge or corner so we use this method instead of other down sampling method. The implementation of the down sampling method is using two channel of EDMA to move two images (640 x 480) that located in SDRAM into new location address that locate in L2 Cache for 160 x 120 res. After the EDMA transfer already done to two subsequent images into L2 Cache, the image 160 x 120 is ready for next processing step. Figure 4.3 showed the EDMA transfer for this down sampling method.

### 4.2.3. Smoothness constraint (Pre-processing)

Commonly, the moving objects viewed in the world are opaque and undergo rigid motion or deformation. In this case, the neighboring points on the objects have similar velocities. It is same in the optical flow case; we can assume that the optic flow varies smoothly in small neighborhoods in the visual field [26].

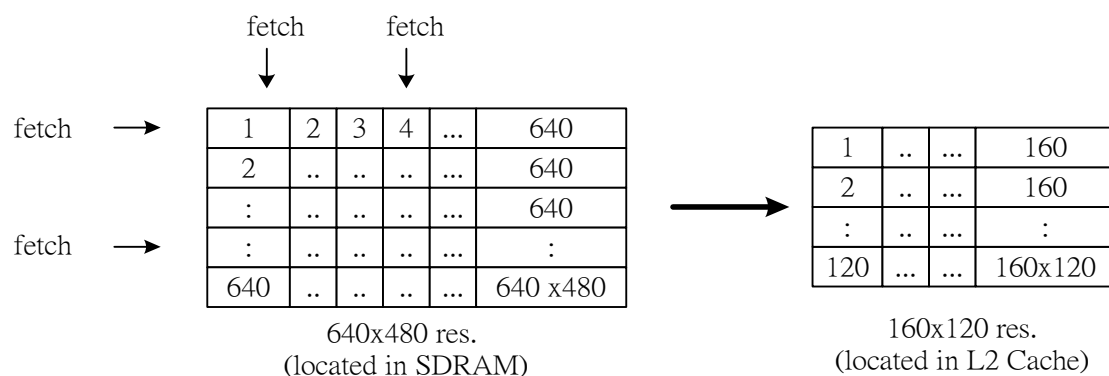


Figure 4.3 EDMA transfer for down sampling method

In the mixed optical flow algorithm, this additional constraint is combined with brightness constraint to provide an additional constraint to the underdetermined system for optic flow determination in the presence of the aperture problem.

The implementation of the smoothness constraint uses the filter averaging. The reason to choose the filter averaging is because of this filter can average each pixel of the image within its neighbor so each pixel is having equivalently gray level value within its neighborhood boundary and based on this condition, the result of the filter averaging is similar with the assumption of the smoothness constraint in optic flow. Figure 4.4 illustrate the smoothness constraint.

It is clearly that by applying the smoothness constraint the boundary of an object can be easily extract so the optical flow can detect the motion object using the boundary of the object and it can be done by applying filter averaging. The disadvantages by applying the filter averaging for the smoothness constraint are that the image after applying the averaging filter is having blurring effect. Although there have blurring effect over the image, the mixed optical flow algorithm still can detect the motion object so the blurring effect doesn't affect the mixed optical flow algorithm.

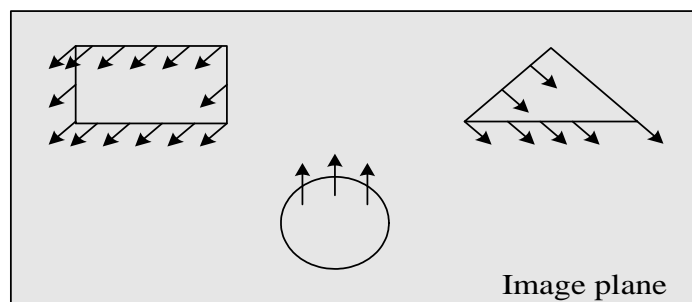


Figure 4.4 smoothness constraint of optical flow

#### 4.2.4. Calculate the $E_x$ , $E_y$ and $E_t$

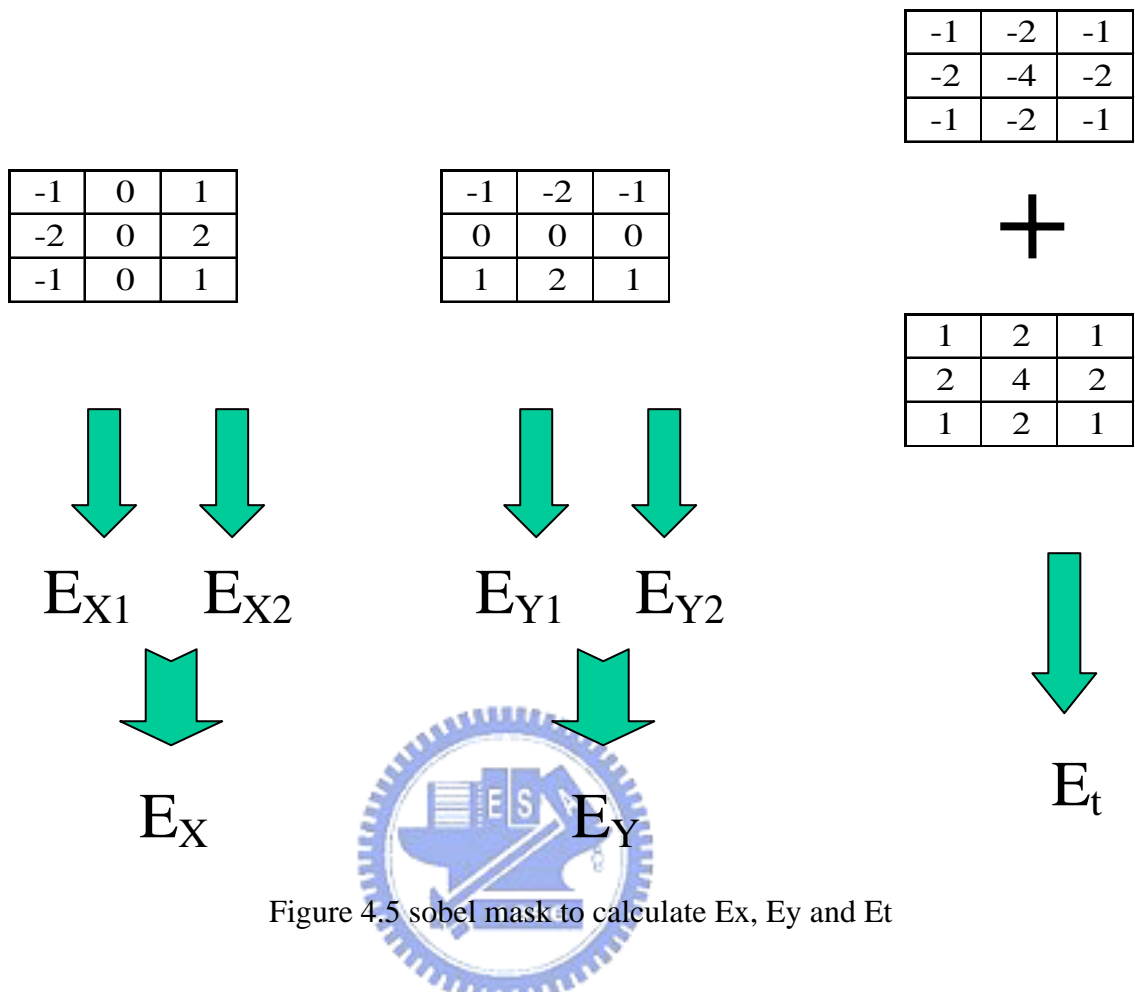


Figure 4.5 sobel mask to calculate  $E_x$ ,  $E_y$  and  $E_t$

The method to calculate  $E_x$ ,  $E_y$  and  $E_t$  is straight forward by using Sobel mask for x-direction, y-direction and t-direction. The Sobel mask for these directions can be seen in the figure 4.5.

#### 4.2.5. Search the $(u,v)$ over brightness constraints

Brightness constraints assume that for a given scene point the intensity ( $E$ ) at the corresponding image point remains constant over time. That is, if a scene point  $P$  (see figure 4.6) projects onto the image point  $(x,y)$  at time  $t$  and onto the image point  $(x + \delta x, y + \delta y)$  at time  $(t + \delta t)$ , we can write

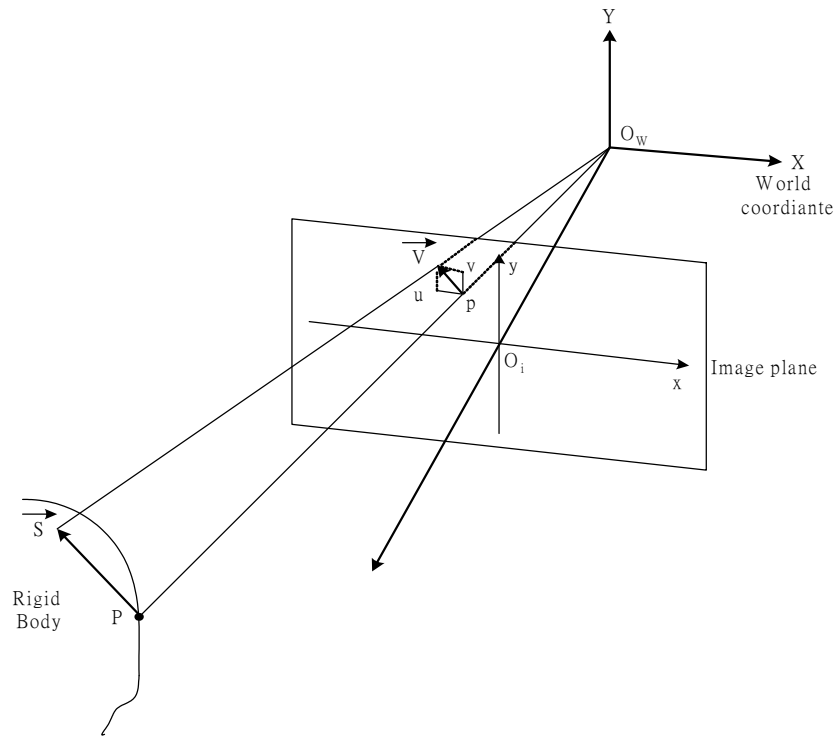


Figure 4.6 the imaging geometry

$$E(x, y, t) = E(x + \delta x, y + \delta y, t + \delta t) \quad \dots \quad (4.1)$$

We can straight forward derived the (4.1), and we get the brightness constraint formulation is

$$E_x \cdot u + E_y \cdot v + E_t = 0 \quad \dots \quad (4.2)$$

The brightness constraints based on (4.2) provide a linear equation of variables  $u$  and  $v$ . As the consequence, the velocity vector  $(u, v)$  cannot be determined locally without applying additional constraints. Equation 4.2 also referred to as the *motion constraint line*, and can be plotted in  $uv$  space, as shown in figure 4.7. In the mixed optical flow algorithm, it applies correlation constraint for the additional constraint. The next following section will describe about correlation constraint.

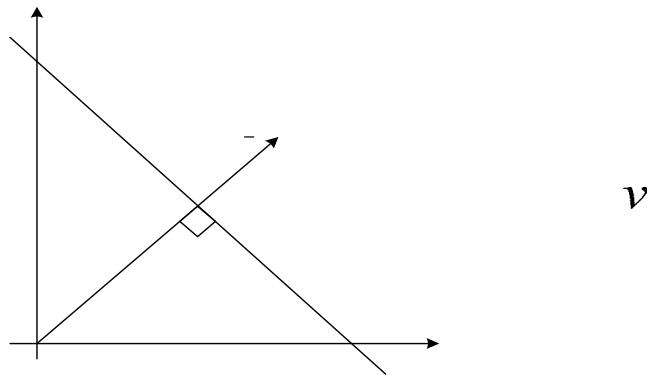
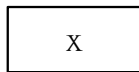


Figure 4.7 the motion constraint line

The implementation of the brightness constraint itself is based on the (4.2). After calculating the  $E_x$ ,  $E_y$  and  $E_t$  using sobel mask, it is checked whether the calculated values of  $E_x$ ,  $E_y$  and  $E_t$  satisfy for the *motion constraint line*.

If it satisfies the *motion constraint line*, it means that there is no possible flow motion on the calculated pixel. Alternatively, if it does not satisfy, it means that the calculated pixel have possibly flow motion. We apply 7x7 window to configure the search area for the flow motion.

$E_x=-29, E_y=-19, E_t=-44$



| $v \backslash u$ | -3  | -2  | -1  | 0    | 1    | 2    | 3    |
|------------------|-----|-----|-----|------|------|------|------|
| -3               | 100 | 71  | 42  | 13   | -16  | -45  | -74  |
| -2               | 81  | 52  | 23  | -6   | -35  | -64  | -93  |
| -1               | 62  | 33  | 4   | -25  | -54  | -83  | -112 |
| 0                | 43  | 14  | -15 | -44  | -73  | -102 | -131 |
| 1                | 24  | -5  | -34 | -63  | -92  | -121 | -150 |
| 2                | 5   | -24 | -53 | -82  | -111 | -140 | -169 |
| 3                | -14 | -43 | -72 | -101 | -130 | -159 | -188 |

7x7 window mask

Figure 4.8 Brightness constraints for possibly flow motion

Figure 4.8 shown an example for searching the  $(u, v)$  over the search area for possible flow motion using brightness constraint. In this example, the  $E_x$ ,  $E_y$  and  $E_t$  of a pixel is calculated and checked whether it satisfies the *motion constraint line*. The result is the values of the  $E_x$ ,  $E_y$  and  $E_t$  do not satisfy the *motion constraint line*. So a  $7 \times 7$  windows is to configure the search area and the shaded line in the  $7 \times 7$  windows mask on figure 4.8 is the possible flow motion  $(u, v)$ . The searching method to find  $(u, v)$  for the possible flow motion is looking for the transition value between + (positive) and - (negative) in the  $u$  or  $v$  direction. Below is the searching method to find the possible flow motions:

$$u, v \in \begin{cases} (u, v) \mid uE_x + vE_y + E_t \geq 0 \ \& \ (u-1)E_x + vE_y + E_t < 0 \\ OR \ uE_x + vE_y + E_t < 0 \ \& \ (u-1)E_x + vE_y + E_t \geq 0 \\ OR \ uE_x + vE_y + E_t \geq 0 \ \& \ E_x + (v-1)E_y + E_t < 0 \\ OR \ uE_x + vE_y + E_t < 0 \ \& \ (v-1)E_x + vE_y + E_t \geq 0 \end{cases} \quad \dots \quad (4.3)$$

#### 4.2.6. Correlation constraints

The correlation constraints [27] [28] of optical flow is estimated in terms of the relation of a pixel  $(x, y)$  in one frame to its next frame. It is assumed that the pixel motion in two successive frames of instant  $t$  and  $(t + \delta t)$  will retain in range of  $-N < u < N$  and  $-N < v < N$ , where  $N$  is the largest possible displacement of  $u$  and  $v$ , and  $(u, v)$  is the optical flow vector. The optical flow is determined from the correlation match of the patch represented by  $(2n+1) \times (2n+1)$  pixels centered at  $(x, y)$ , out of  $(2N+1) \times (2N+1)$  possible displacements. The correlation match equation is given by [28]:

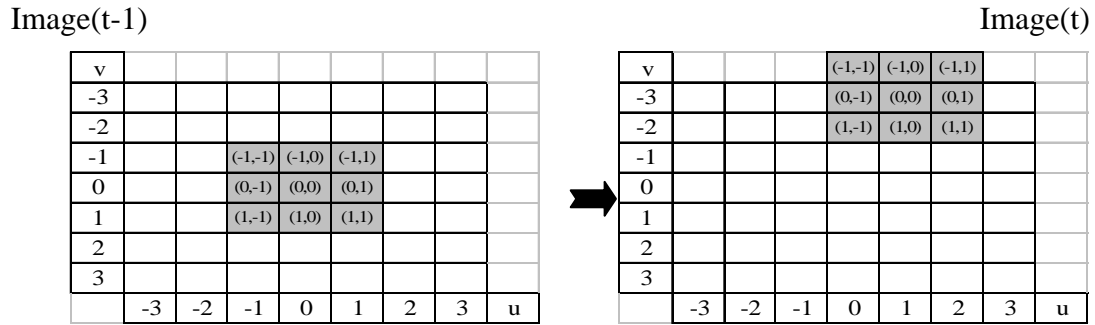


Figure 4.9 correlation constraint algorithms

$$SSD_{t,t+1}(x, y; u, v) = \sum_{j=-n}^n \sum_{i=-n}^n [I_t(x+i, y+j) - I_{t+1}(x+u+i, y+v+j)]^2 \dots (4.4)$$

where  $-N < u < N$  and  $-N < v < N$  and SSD denotes the sum of squared difference between the patch and a  $(2n+1) \times (2n+1)$  window around each pixel in  $(2N+1) \times (2N+1)$  possible displacement. Figure 4.9 shown the correlation constraint algorithm based on (4.4).

From the (4.4), we can see if the search area is increasing it will greatly increase the computation load. In order to reduce computational complexity and still hold acceptable estimation accuracy, the mixed optical algorithm proposes to include the brightness constraint [26] that already mention in previous section combine with this correlation constraint. So in the mixed optical flow algorithm, it is using the principle of correlation match of correlation-based (4.4) technique as the kernel, and adds the brightness constraint (4.2) of differential-based technique to condense the search area of correlation match.

So the implementation of the correlation constraint in mixed optical flow algorithm is straight forward. Firstly, we find the *motion constraint line* which is based on the (4.2). After we find the *motion constraint line*, we then perform SSD (sum of squared difference) based

on the (4.4), to find the displacement of current pixel in the possibly search area that lying the *motion constraint line*. After performing SSD search in all pixels that lying in the *motion constraint line*, then we look for the minimum value of all SSD value. The displacement of the pixel which is motion field vector  $(u,v)$  is the minimum value of the SSD value. After finding for one motion field vector, we repeated this step until the entire pixel in the one image frame already done. In our case, we find the motion field vector for 160 x 120.

## 4.2.7. Create Depth Histogram

Using the obtained optical flow field, we can calculate the scene depth and time-to-collision (TTC). This depth information of object is then input to an obstacle avoidance algorithm which locates in the navigation layer in our behavior-based architecture. As shown in figure 4.10, the camera focuses to the positive Z-axis direction. Let the origin of the world coordinate  $O_c$  locate at the camera center. For both the cases that the camera moves toward the object with a velocity  $(0,0, W_c)$  or the object moves toward the camera with a velocity  $(0,0, W_c)$ , the TTC or depth can be calculated using the equation below [29]:

$$\frac{d}{f} = \frac{D}{Z} \quad \dots \quad (4.5)$$

where  $d$  is distance between focus of expansion (FOE) to a point  $p$  int the image plane (see Fig. 4.10).  $f$  is focal length,  $D$  is the distance between an interest point P and the Z-axis,  $Z$  is the depth. Differentiating with respect to time and dividing by  $d$ , we can get:



$$\frac{d}{V} = \frac{Z}{W_c} = \tau, \quad \text{as the camera is still}$$

$$\frac{d}{V} = -\frac{Z}{W_c} = \tau, \quad \text{as the object is still}$$

... (4.6)

where  $V$  is the flow velocity and  $\tau$  is TTC.

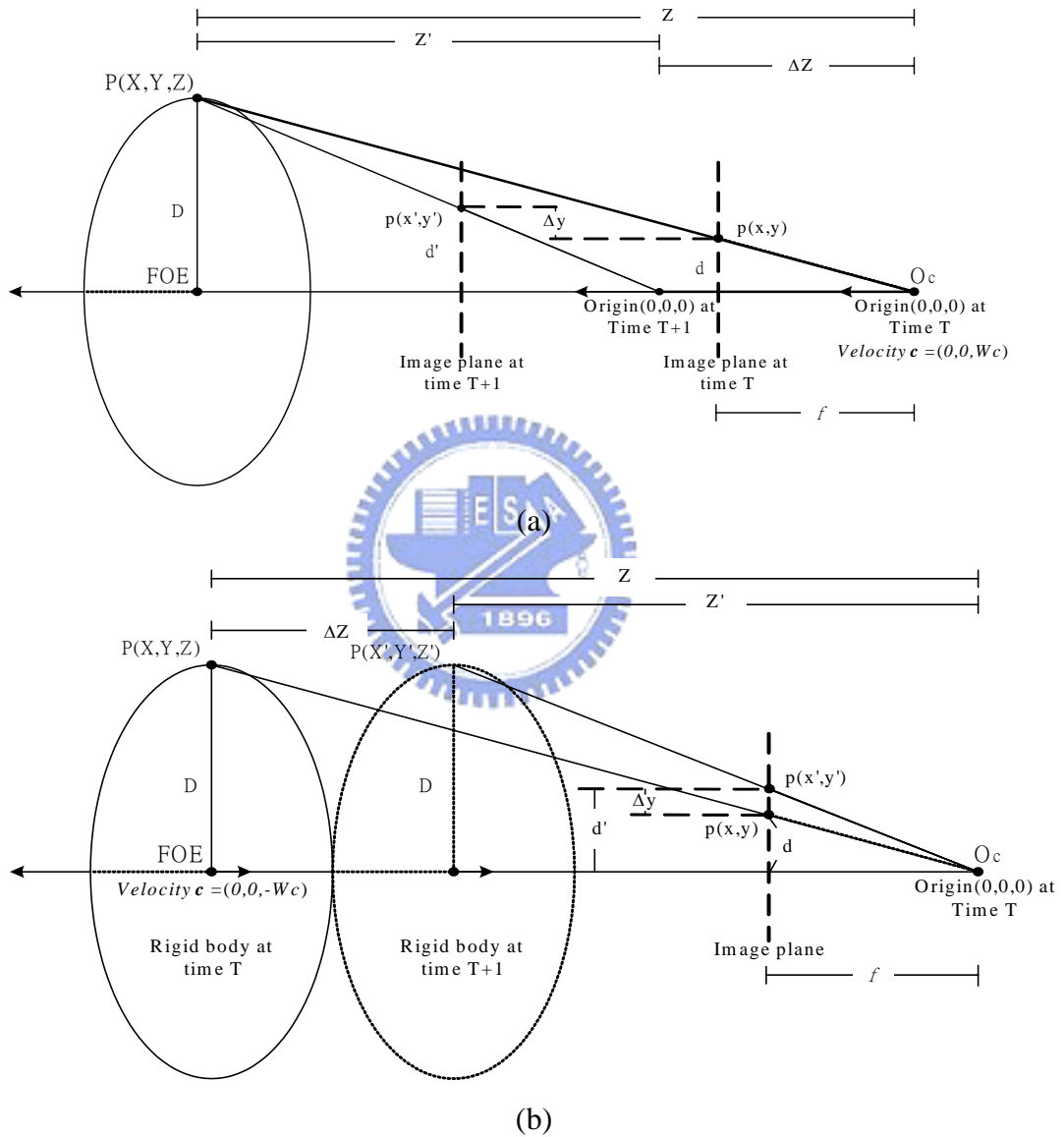


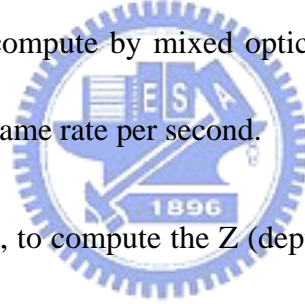
Figure 4.10 (a) The camera moves toward the object.

(b) The object moves toward the camera.

The implementation of computing TTC is also straight forward using (4.6), which is for the condition *as the object is still*. By modifying the (4.6) with our real implementation in our program we obtain the computation formula for TTC:

$$TTC(\tau) = \frac{d}{V} = \frac{abs(Est\_v\_pos - 60)}{\frac{Est\_v}{t}} \quad \dots \quad (4.7)$$

where  $Est\_v\_pos$  is the position (y-axis) for point p of the motion flow field in the image plane at T (see figure 4.10 (b)).  $Est\_v\_pos$  is subtracted with 60 (the coordinate center of y-axis with 160x120 res.) and then absolute the result value to get the d value (see figure 4.10 (b)). Meanwhile to compute V (motion field speed) we need  $\Delta y$  (see figure 4.10 (b)) or in this case  $Esv\_v$  (the motion field compute by mixed optical flow) and the sampling time of the image which equivalent with frame rate per second.



After we get the TTC ( $\tau$ ), to compute the Z (depth information) we just multiplying the TTC ( $\tau$ ) with  $W_c$  (robot speed) which is shown by Eq.4.8.

$$Z = TTC(\tau) \cdot W_c = \frac{abs(Est\_v\_pos - 60)}{\frac{Est\_v}{t}} \cdot W_c \quad \dots \quad (4.8)$$

## 4.2.8. Safety distribution histogram

In order to establish a representation of the environment configuration, we transformed the calculated depth to a safety distribution histogram. This histogram is established from a mapping from a 3D space to 2D ZX-plane using the depth information to represent the obstacle in the environment.

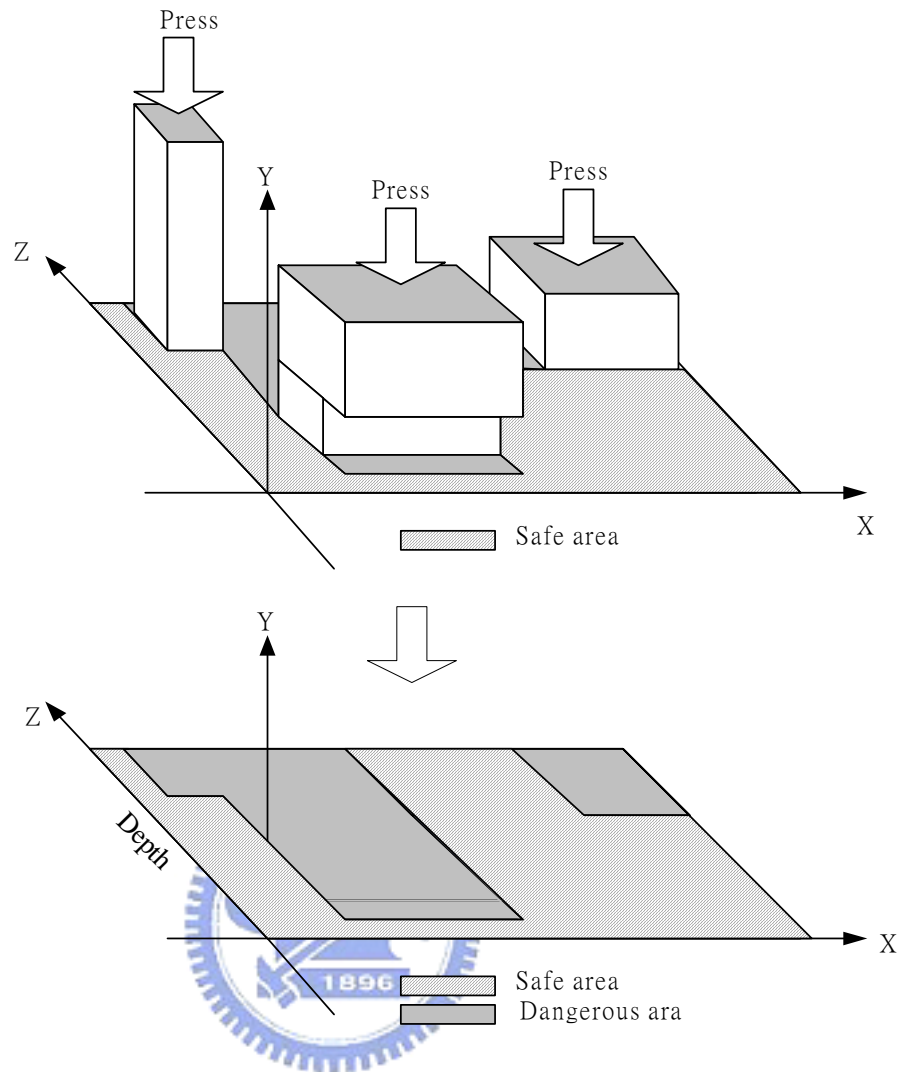


Figure 4.11 the representation of 3D space to 2D ZX-plane

Figure 4.11 illustrates the idea of safety histogram. From this figure, we see that the transformation of 3D space to 2D ZX plane can be done by pressing the y-axis value, pressing the y-axis means that the depth (Z) of the ZX histogram will depend on the y-axis value. This z-axis value is chosen the minimum value from the y-axis. So the bigger value of the z, the safer the location. On the other hand, a smaller value of z means that there are obstacles in front of the robot.

## 4.3. Experiment results with mixed optical flow algorithm

In this section, we show experiment results of implementation mixed optical flow algorithm and the computation of safety histogram distribution. In figure 4.12 (a) shows the motion field vector in uv space and figure 4.12 (b) shows the safety histogram distribution that divide in to the 7 region. The width of the safety histogram distribution is 160 pixels; with each region of the safety histogram distribution is 20 pixels.

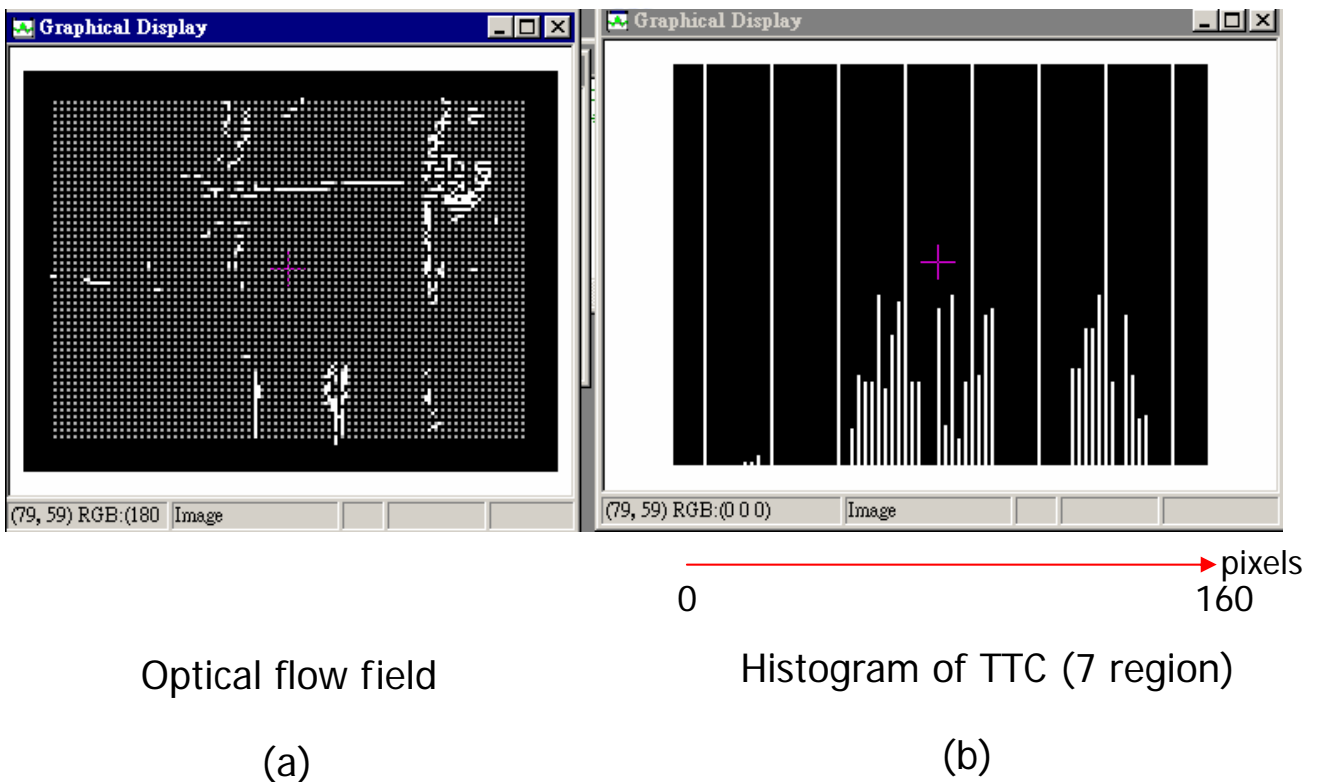


Figure 4.12 Optical flow field [a] and histogram of TTC (7 regions) [b]

Next, we divide the seven regions into three sides which are left side, front side and right side. Figure 4.13 (b) shows the three side of safety histogram distribution that derived from seven regions and figure 4.13 (a) shows its motion field vector. By using these three regions, the robot can easily and fast to determine the safety location to avoid obstacle, like in the case of figure 4.13 (b), the left side of safety histogram distribution is bigger than the right side so the robot will turn left to avoid the obstacle.

The implementation of mixed optical flow in embedded image processing platform took 0.2 second or 5 Hz to perform one calculation of obstacle avoidance which in the previous research [14] it took 0.9 second or approx. 1 Hz.

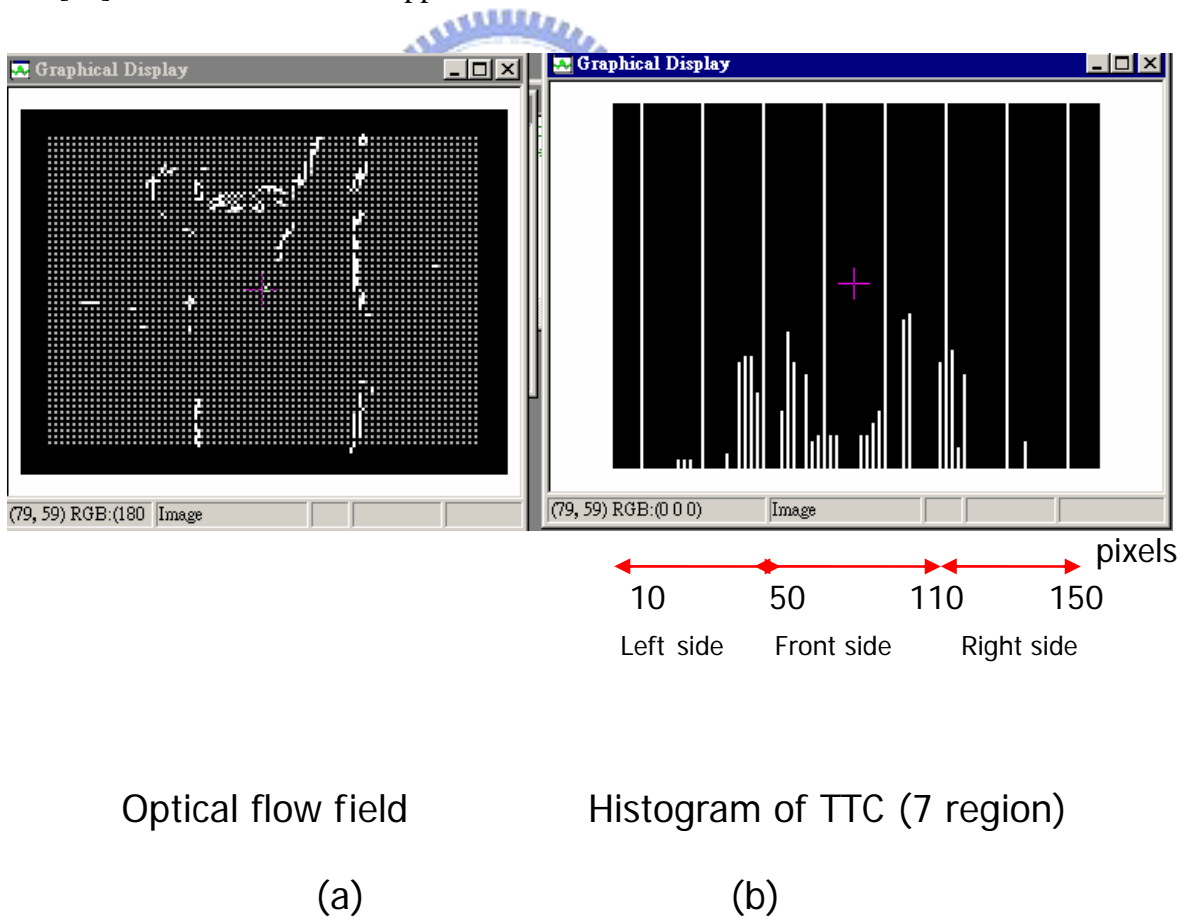


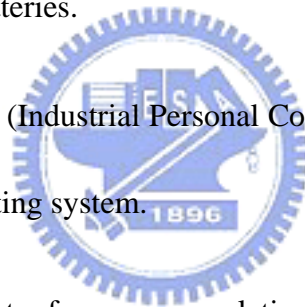
Figure 4.13 Optical flow field [a] and histogram of TTC (3 regions) [b]

# 5. Experimental Results

## 5.1. The experimental mobile robot

The experimental mobile robot is equipped with a manipulator. Below is the description of the mobile robot used in experiments:

- Two independent drive wheels (Foot motors) and two casters for mobility.
- A 3 DOF (Degree of freedom) manipulator for grasping.
- Two 12 V and two 6 V batteries.
- A Pentium III 1 GHZ IPC (Industrial Personal Computer) for main system control, using Windows XP as the operating system.
- 12 to 5 V DC – DC converter for power regulation.
- Two DSP motion boards for controlling Foot motor and Head motor, and two DSP motion boards for controlling 3DOF manipulator
- 4 COM ports RS232 interface that provide communication between the embedded image processing and DSP motion boards or IPC.
- Embedded image processing platform that using DSK6416 and CMOS sensor board for controlling the 3DOF manipulator.



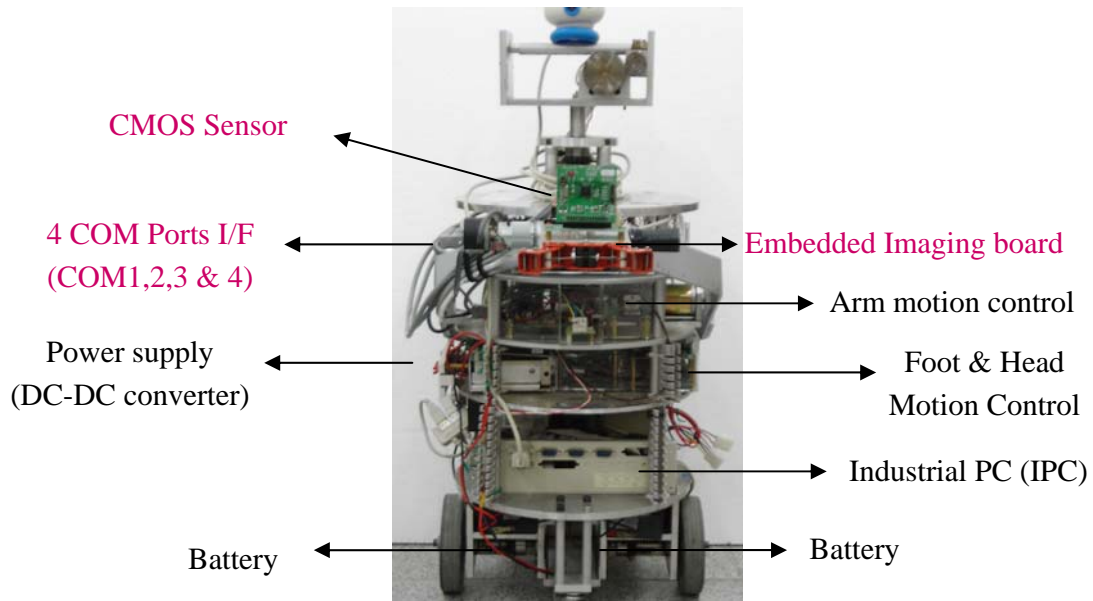


Figure 5.1 Mobile manipulator (H2) platforms

## 5.2. Experiment of grasping

The goal of this experiment is to verify the mobile manipulator can locate barcode-like object using the proposed visual servoing method.

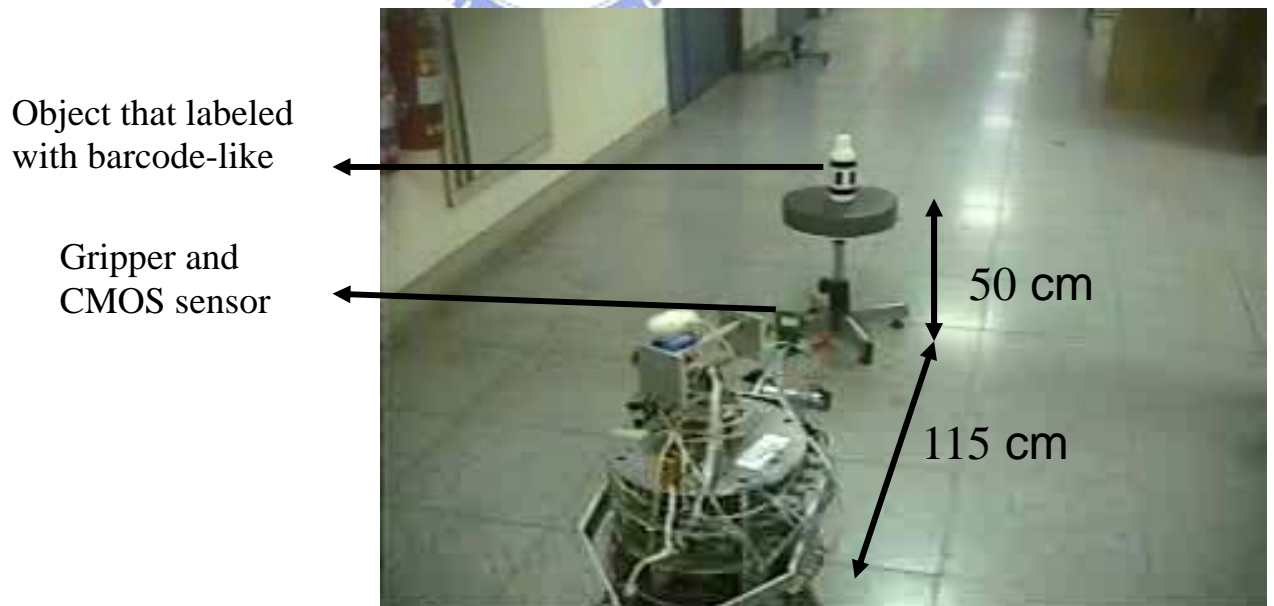


Figure 5.2 the robot is faced to the object that put in the top of chair

In the experiment the object is put in certain place, which in this experiment is put in top of a chair. Figure 5.2 show the robot faced to the object. The object which labeled with the barcode-like feature is put at a distance about 115 cm. The object itself is put in the top of chair which height to the ground is 50 cm.

In the first stage, the mobile manipulator robot scan its environment by turning left about  $30^\circ$  (see figure 5.3 (a)) and right about  $30^\circ$  (see figure 5.3 (b)) to trying to locate the barcode-like object in front of the mobile manipulator.



Figure 5.3 (a) the mobile robot is turned left about  $30^\circ$





Figure 5.3 (b) the mobile robot is turned right about 30°

The robot is tried to scan the environment for several times, and after the mobile manipulator robot located the object in front of it, it then changed its orientation and speed according to the barcode-like direction and distance. Figure 5.4 shows the mobile manipulator robot locates the object and change its orientation and speed according to the object orientation.

Next, the mobile manipulator robot is approaching the object which adaptively change the orientation and speed according to the distance and the orientation which compute from the feature of the barcode-like that get from visual sensory feedback. (See figure 5.5 (a) and figure 5.5 (b)).



Figure 5.4 the robot change its orientation and speed after it locates the object in front of it



Figure 5.5 (a) the mobile manipulator is approaching the object



Figure 5.5 (b) the mobile manipulator is approaching the object (cont'd)



Figure 5.6 (a) the object is get into the area of the gripper



Figure 5.6 (b) the mobile manipulator grasp the object

After the barcode-like object gets into the gripper area, (see figure 5.6 (a)), the mobile manipulator slowly approaches the object (see figure 5.6 (a)). Lastly, when the barcode-like distance approximately 5 cm, the mobile manipulator grasps the object and stops (see figure 5.6 (b)).

### **5.3. Experiment of grasping of an object from a person**

The goal of this experiment is to show how the mobile manipulator robot can locate barcode-like object as the object is hold by human. A human is standing while holding the object which is labeled by the barcode-like. The mobile manipulator is faced to the object

with distance about 90 cm. The object is hold by human with the height of the object to the ground about 55 cm. Figure 5.7 show the mobile manipulator faced the object that hold by the human.

Similar with the previous experiment result, the mobile manipulator is tried to scan its environment by turning left about 30° (see figure 5.8 (a)) and by the time the robot is turn back to the forward direction, its locate the object that hold by the human (see figure 5.8 (b))

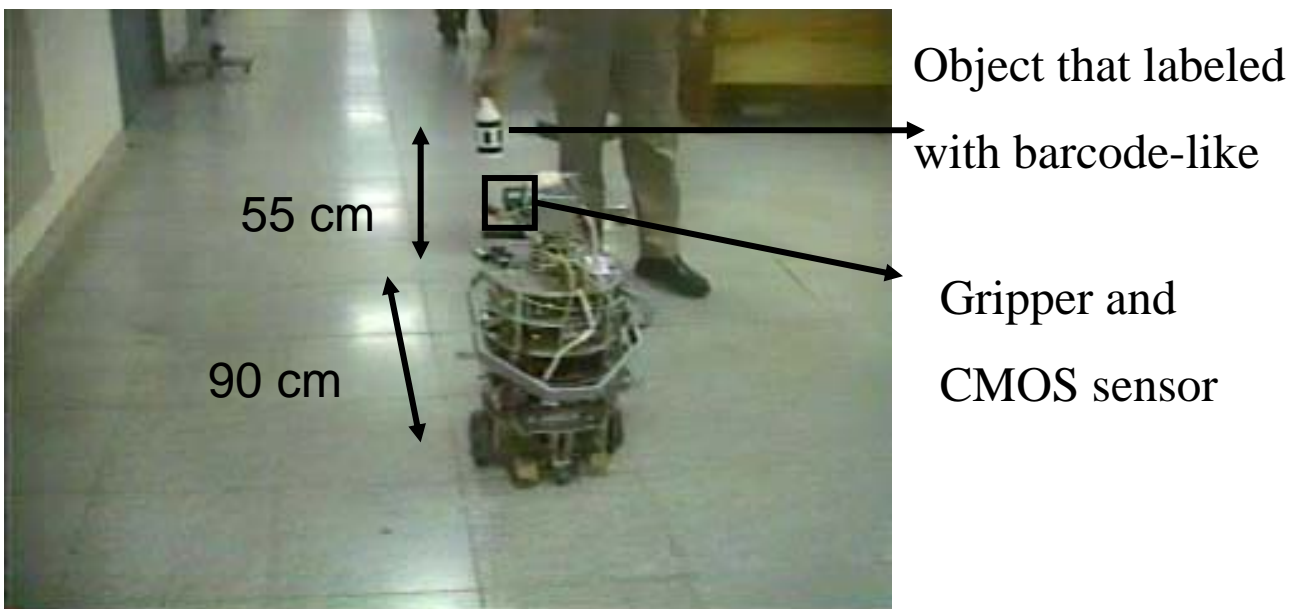


Figure 5.7 the mobile manipulator robot is faced to the human that hold the object



Figure 5.8 (a) the robot is turned left about  $30^\circ$



Figure 5.8 (b) the robot has located the object that holds by human

Next, the mobile manipulation is approaching to the object, (see figure 5.9 (a) and (b)), and when the object already gets in to the gripper, it grasps the object (see figure 5.10 (a) and (b)).



Figure 5.9 (a) the robot is approaching the object      Figure 5.9 (b) the robot is approaching [cont'd]



Figure 5.10 (a) the object is in the gripper area      Figure 5.10 (b) the gripper grasped the object

The representation of this experimental in the qualitative data is shown as below:

a. The mobile manipulator robot trajectory

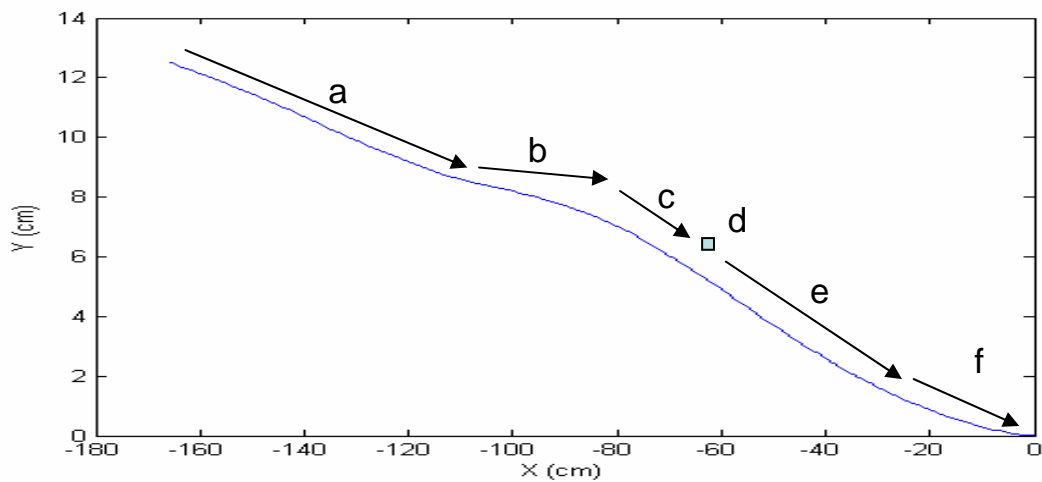


Figure 5.11 the trajectory of the mobile manipulator

Figure 5.11 has shown the recorded trajectory of the mobile manipulator. This trajectory

is drawn using encoder pulses from the motor. The movement of the robot in x-axis is recorded in X (cm), meanwhile the movement of the robot in y-axis is recorded in Y (cm). In the figure 5.11, we divided the trajectory into five parts. In the part (a) the robot is starting from its starting point and is moving forward. Because the mobile manipulator robot can not find the object, the robot is trying to scan its environment by turning left side about 30 degree (part b). After turning left about 30 degree, the robot is then back to the forward position (part c), by the time the robot is turned to the forward position. The robot locates the object. That is why in part (d), the robot change its orientation, and then facing forward to approach the object.

In the part (e), the robot is approaching the object approx 30 – 40 cm, in the last part (f) the robot is nearly the object that the distance about approx 0 - 20 cm. In this distance the velocity of the robot is reduced, and when the object get into the gripper it will grasped the object.

b. The orientation of the robot

The orientation of the robot while approaching the object can be seen in the figure 5.12. In this figure, it plots between the orientation of the robot in radian and time in second. The total time that is using to grasp the object is approx. 30 second. In the part (a), the robot is trying to scan its environment by scanning to the minus direction according to the graphic of the figure 5.12 which turning about 5 seconds. (From 10 second to 15 second).

Next, because the robot does not find the object it then turn to the respectively direction



back to the forward direction (part b). By the time, the robot want to back to the forward direction, the robot find the object, it can see in the part (c) where the robot only turn for a half of its direction. Next in the part (d), the robot already locates the object, and changing its orientation and direction. As we can see that in the part (d), there is an oscillation to make the robot stable at its orientation according to the object orientation. Later in the part (e), the robot is nearly from the object and trying to get the object into its gripper. As we can see that the trajectory in the part (e) is not changed and stable between -0.15 and -0.1.

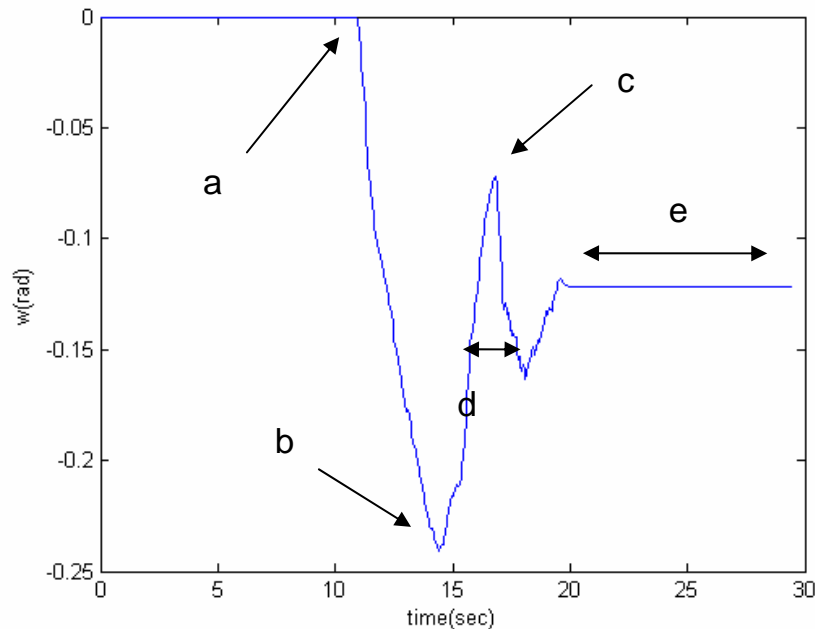


Figure 5.12 the robot orientation (rad - sec)

## 5.4. Experiment of obstacle avoidance

The goal of this experiment is to show how the mobile robot can avoid objects in front of it. If there is an obstacle in front of the robot, the robot will try to avoid the obstacle by determining the safety location between the left side and the right. After selection of the safety

location, the robot then turns about 30 degree to avoid the obstacle.

In the navigation mode the mobile manipulator robot works using the navigation layer in the behavior-based architecture. In the experiment, the mobile manipulator robot is free-running, if there is an obstacle in front of the mobile manipulator, the robot will try to find the safe location. Once the robot finds the safe location, the robot will change its direction and orientation to the safe location so it can avoid the obstacle safely.

Figure 5.13 illustrate the trajectory of this experiment. Point (a) of figure 5.13, the robot faces the person approximately about 30 cm (see also figure 5.14 (a)). The robot then tries to find the safe location, once it finds the safe location the robot starting to turn about 30 degree to the safe location (see figure 5.14 (b)). At point (a), the robot chooses to turn left about 30 degrees according to the result of safe location estimation.

Point (b) of figure 5.13, the robot faces the person and the wall beside the human (see figure 5.14 (c)). After calculating the safe location, the robot successfully chooses (see figure 5.14 (d)). Similar with points (a) and (b), at points (c) and (d), the robot finds the safe location to avoid the obstacle safely. Figure 5.15 illustrate the recorded trajectory of the robot in this experiment.

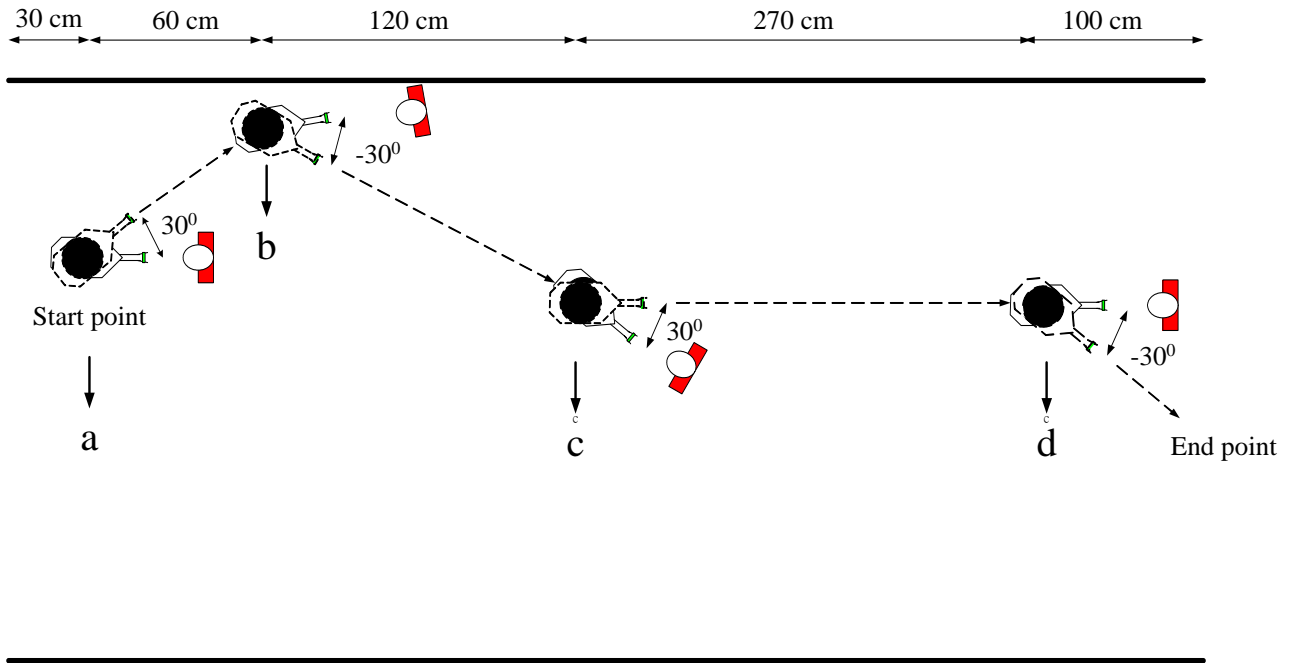


Figure 5.13 the trajectory approach of the mobile navigation experiment



Figure 5.14 (a)



Figure 5.14 (b)



Figure 5.14 (c)



Figure 5.14 (d)



Figure 5.14 (e)



Figure 5.14 (f)



Figure 5.14 (g)



Figure 5.14 (h)

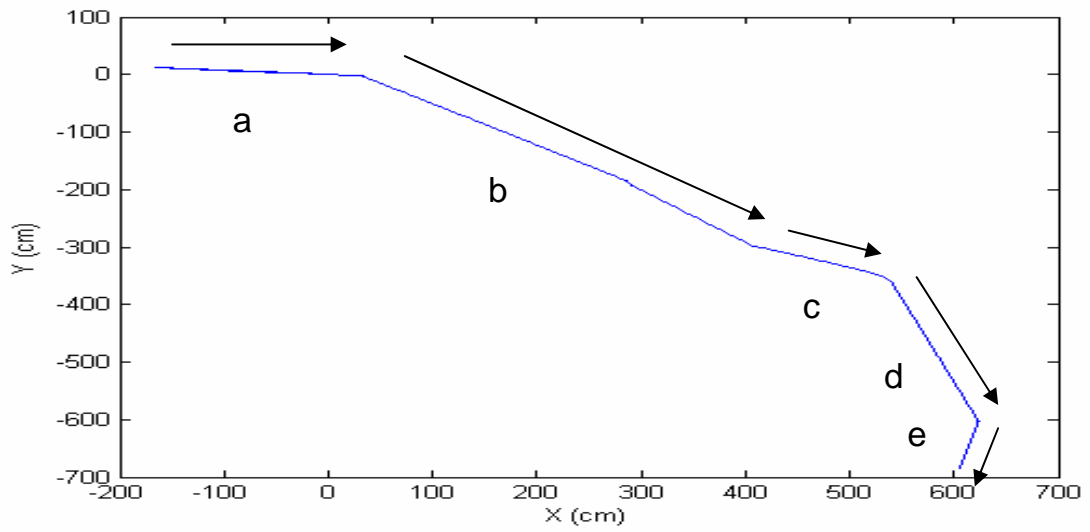


Figure 5.15 the trajectory of the mobile navigation mode



# 6. Conclusions and Future Work

## 6.1. Conclusions

This thesis presents a design and implementation of hardware and software to solve a mobile manipulator problem which is for navigation and grasping. The hardware implementation develops an embedded image processing platform which is stand-alone and can communicate directly to other module like DSP motion board and IPC. The embedded image processing consists of CMOS image sensor and DSK6416 as the main processing board. The result of this embedded image processing platform can be used for acquiring image for 15 fps and 30 fps.



The experiment results of the visual servoing algorithm have shown that the mobile manipulator can locate a barcode-like object, approaching and grasp the object. The performance of this algorithm implement in embedded image processing platform to search the barcode-like object took 0.0667 second or 15 Hz.

Meanwhile, the experiment results of the mixed optical flow algorithm can avoid a person in real time. The performance of this algorithm implement in embedded image processing platform for one calculation of obstacle avoidance took 0.2 second or 5 Hz.

## 6.2. Future work

The proposed algorithm only has shown the use of the grasping mode and navigation mode independently without combining these two modes at the same time. In the future, by combining these two modes at the same time, the mobile manipulator can autonomously navigate it while the barcode-like object hasn't detected. And if the barcode-like object already detect, the mobile manipulator can approach and to grasp it.

In this work we just control the speed and the orientation of the mobile robot. The proposed algorithm visual servoing algorithm needs to improve by controlling the manipulator as well as the robot autonomously.



# References

- [1] D. Kragic and H. I. Christensen, "Robust visual servoing," in *The International Journal of Robotics Research*, vol. 22, pp. 923-939, Oct. – Nov. 2003.
- [2] L. Petterson, D. Austin, and D. Kragic, "High-level control of a mobile manipulator for door opening," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol. 3, 31 Oct.-5 Nov. 2000, pp. 2333-2338.
- [3] O. Khatib, "Mobile manipulation: The robotic assistant," in *Robotics and Autonomous System*, vol.26, pp. 175-183, February 1999.
- [4] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *the International Journal of Robotics Research*, vol.5, pp. 90-98, 1986.
- [5] K. Nagatani and S. Yuta, "An Experiment on opening-door-behavior by an autonomous mobile robot with a manipulator," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol. 2, Aug. 1995, pp. 45-50.
- [6] K-T Song and T-Z Wu, "Visual servo control of a mobile manipulator using one-dimensional windows," in *Proc. IEEE IECON'99 Industrial Electronics Society*, vol.2, 29 Nov. – 3 Dec. 1999, pp. 686 – 691.
- [7] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of the Honda Humanoid Robot", in *Proc. of the International Conference on Robotics and Automation*,



1998, pp.1321-1326

- [8] J. H. Connel, “A behavior-based arm controller,” in *IEEE transaction on Robotics and Automation*, vol. 5, pp. 784 – 791, Dec. 1999,
- [9] R. A. Brooks, “A robust layered control system for a mobile robot,” in *IEEE journal of Robotics and Automation*, vol. RA-2, pp. 14 – 23., March 1986,
- [10] Z. Wasik and A. Saffioti, “A fuzzy behavior-based control system for manipulation,” in *IEEE/RSJ International Conference on Intelligent Robots and System*, vol.2, 30 Sept. – 5 Oct. 2002, pp. 1596 – 1601.
- [11] Z. Wasik and A. Saffioti, “A hierarchical behavior-based approach to manipulation tasks,” in *Proc. IEEE International Conference on Robotics & Automation*, vol. 2, Sept. 2003, pp. 2780 – 2785.
- [12] S. Hutchinson, G. D. Hager and P. I. Coke, “A tutorial on visual servo control,” in *IEEE transaction on Robotics and Automation*, vol. 12, pp. 651-670, Oct. 1996.
- [13] C. E. Smith, “Visually guided manipulation of static and moving objects,” Ph.D. Thesis, Univ. Minnesota, 1996
- [14] K.T. Song and J. H. Huang, “Fast optical flow estimation and its application to real-time obstacle avoidance,” in *Proc. IEEE International Conference on Robotics and Automation*, vol.3, 2001, pp. 2891 – 2896.
- [15] Y. Yoon, G. N. D. Souza and C. Kak, “Real-time tracking and pose estimation for



- industrial geometric features,” in *Proc. IEEE International Conference on Robotics and Automation*, vol.2, Sept. 2003, pp.3473 – 3478.
- [16] M. Kaneko, M. Kessler and A. Weigl, “Capturing pyramidal-like objects,” in *Proc. IEEE International Conference on Robotics and Automation*, vol. 2, May 1998, pp. 3619 – 3624.
- [17] C. A. Richards and N. P. Papanikolopoulos, “The automatic detection and visual tracking of moving objects by eye-in-hand robotics system,” in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol.3, Aug. 1995, pp. 228 - 233.
- [18] ICM205B, Datasheet, “VGA/QVGA CMOS image sensor with digital YUV output,” Oct 2002, IC-MEDIA Corp.
- [19] TMS320C6000, Reference Guide, “TMSC6000 Peripherals Reference Guide, “ Literature Number: SPRU190D, December 2002, Texas Instrument. Inc.
- [20] TMS32C6000, Reference Guide, “TMSC6000 McBSP: UART,” Literature Number: SPRA633B, May 2004, Texas Instrument Inc.
- [21] AL422B Data sheet, Data sheet, 2004, Aver Logic Inc
- [22] Altera University design laboratory package, User guide, 2001, Altera Corp.
- [23] TMS320C6000, Application Report, “EMIF to External FIFO interface,” Literature Number: SPRA543, May 1999, Texas Instrument Inc.
- [24] TMS320C6000, Application Report, “Applications using the TMS320C6000 Enhanced

DMA,” Literature Number: SPRA636A, October 2001, Texas Instrument Inc.

[25] R. C. Gonzales and R. E. Woods, Digital Image Processing, Prentice Hall, 2002.

[26] B. K. P. Horn and B. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, pp. 185-203, 1981.

[27] T. Camus, “Real-time Quantized Optical Flow,” in *Proceedings of 1995 Computer Architectures for Machine Perception*, pp. 126 – 131, 1995.

[28] A. Sigh, Optical Flow Computation: A Unified Perspective, IEEE Computer Society Press, 1991.

[29] Jorge L. C. Sanz (Ed.), Image Technology: Advances in Image Processing, Multimedia and Machine Vision, Springer-Verlag Berlin Heidelberg, 1996.

