

WIRELESS MULTIMEDIA SENSOR NETWORK USING MICAz MOTES



By

Hamna Anwar

Hamza Tariq Khan

**Submitted to the Faculty of Electrical Engineering, Military College of Signals,
National University of Science and Technology, Rawalpindi in partial
fulfillment for the requirements of a B.E. Degree in Telecommunication
Engineering
JUNE 2012**

**It is hereby certified that the project is duly completed by a group of EE
department under the supervision of Lecturer ObaidUllah Khalid.**

Name: Lecturer ObaidUllah Khalid

Date: _____

ABSTRACT

WIRELESS MULTIMEDIA SENSOR NETWORK USING MICAz MOTES

Wireless Multimedia Sensor Network (WMSN) employs multimedia gathering devices connected to a low power, low bandwidth sensor network. These networks are a step ahead of the WSNs, which only cater for scalar data.

A camera is integrated to a MICAz mote through the interfacing circuitry. The microcontroller buffers the image and sequentially passes it to the attached mote. This mote transmits the information to the base station mote via multi hopping through a relay mote. The base station aggregates the data from the sensor network and sends it on to a PC or computing device through the MIB600CA gateway. The application accepts data from the TCP socket and processes it into the required form. WMSNs can support a lot of diverse applications but for the project a prototype is designed to support traffic monitoring and control. The methodology to dynamically change the duration of the traffic signals is the intended solution for regulating and managing traffic.

DEDICATION

Allah, beginning with the Name of – The Most Gracious, The Most Merciful

To our parents

ACKNOWLEDGEMENTS

Praise be to Allah, who has showered more blessings than can be realized. The group members are grateful to their parents whose support got them going in the bleakest of hours and to their friends for their words of encouragement.

Due extension of gratitude to the project supervisor Lecturer Obaid-Ullah Khalid for his continuous technical guidance and moral support throughout the project. His calm nature ensured smooth work and his composed manner helped in working around obstacles with perseverance.

Special thanks to Sir Fauzan Marwat who helped jump start the work in tinyOS. Maj. Dr. Adnan Rashdi indebted the group members by his help and support and his encouraging smile. The group members also thank Maj. Dr. Muhammad Faisal for his proper guidance towards help. The Department of EE, especially Lt. Col. Umar Khalid facilitated with flexible lab timings and the group owes him words of gratitude. It would be cold to not appreciate the Lab Attendants of Broadband and Networking Lab; Naveed-ur-Rahman and Shakir Mehmood for their patience for working off the hours.

Special thanks to Professor Bruce Land and Under-Graduate Brian Harding for addressing the queries on the image grabber over emails.

DECLARATAION

No portion of the work presented in this dissertation has been submitted in support of another award or qualification either at this institution or elsewhere.

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TALBES	xii
ABBREVIATIONS USED	xiii
Chapter 1	1
INTRODUCTION	1
1.1 Background.....	1
1.2 Project Overview	2
1.3 Objectives.....	3
1.4 Outline Of Tasks.....	3
1.5 Platforms Used	4
1.6 Organization Of The Document.....	4
CHAPTER 2	6
LITERATURE REVIEW	6
2.1 Introduction.....	6
2.2 Communication Protocols.....	6
2.2.1 IEEE 802.15.4.....	6
2.2.2 Serial Communication	10
2.2.2.1 UART	11
2.2.2.2 SPI	12
2.2.2.3 I2C.....	14
2.2.3 TCP/IP Communication.....	15
2.3 Camera Specifications.....	17
2.3.1 Progressive Scan Read Out Mode.....	17
2.3.2 CIF/QCIF Resolution.....	17
2.4 Memory.....	18
2.4.1 EEPROM.....	18
2.4.2 SRAM	18
2.4.3 Flash Memory	19
2.5 Composition Of Image	19
2.5.1 Gray Scale Image	20
2.6 Review Of The Studied Wireless Sensor Networks.....	20
2.6.1 Cyclops.....	20

2.6.2 Wireless Smart Camera (WiCa).....	21
2.6.3 MeshEye.....	21
2.6.4 Meerkats	21
2.6.5 Explorebots.....	22
2.6.6 SensEye	22
2.6.7 WSN At BWN Laboratory, Georgia Tech.....	22
Chapter 3	24
NETWORK TOPOLOGY and DESIGN	24
3.1 Overview	24
3.2 Modules	25
3.2.1 CMOS Camera	25
3.2.2 Image Grabber.....	26
3.2.3 MDA100CB Data Acquisition Board.....	27
3.2.4 MPR2400CB IEEE802.15.4/ ZigBee Compliant MICAz Motes (2.4GHz).....	28
3.2.4.1 Visual Sensor Node	29
3.2.4.2 Intermediate Mote	29
3.2.4.3 Base Station Mote	30
3.2.5 MIB600CA Ethernet Gateway	30
3.2.6 PC/ Other Computational Device	31
3.2.7 Traffic Controller Circuitry	32
3.3 System Design Block Diagram.....	33
3.4 Project Overview	34
3.5 Conclusion	35
Chapter 4.....	36
WIRELESS NETWORK OF MICAz MOTES.....	36
4.1 Overview	36
4.2 Operating System For Programming MICAz Motes.....	36
4.3 Installation Of TinyOS And nesC Compiler	37
4.4 Connection Specifications In Lantronix	39
4.5 MICAz Motes For Data Transmission.....	39
4.5.1 Visual Sensor Module	40
4.5.2 Intermediate Module.....	40
4.5.3 Base Station Module.....	40
4.6 Listen Tool.....	41
4.7 Creating A TCP/IP Server	42
4.8 Packet Structure.....	43
4.9 Importing Data To The Application	44

4.10 Data Rate Optimization	44
4.11 Conclusion.....	47
CHAPTER 5.....	48
IMAGE ACQUISITION	48
5.1 Introduction.....	48
5.2 Programming The Microcontroller.....	48
5.2.1 Interfacing With Camera C3088	48
5.2.2 Interfacing With Flash.....	49
5.2.3 Interfacing With The MICAz Mote.....	50
5.2.4 Schematic Diagram.....	51
5.3 Other Features	52
5.4 Conclusion	53
CHAPTER 6.....	54
PROCESSING IN MATLAB	54
6.1 Introduction.....	54
6.2 Receiving Data's Packet Structure.....	54
6.3 Application Development.....	55
6.3.1 Background Subtraction Algorithm	57
6.3.2 Template Matching Algorithm.....	58
6.3.3 Difference Between The Algorithms	58
6.3.4 Results	59
6.4 GUI	60
6.5 Conclusion.....	61
CHAPTER 7.....	62
FUTURE WORK AND CONCLUSION	62
7.1 Introduction.....	62
7.2 Project Limitations.....	62
7.3 WMSN Advantages And Applications.....	62
7.3.1 Surveillance	63
7.3.2 Environmental Monitoring	63
7.3.3 Advanced Healthcare Delivery	64
7.3.4 Industrial Process Control.....	64
7.4 Testing And Validation.....	64
7.5 Conclusion	65
APPENDIX A- nesC CODE FOR MICAz MOTE IN VISUAL SENSOR NODE	66
APPENDIX B- nesC CODE FOR RELAY MICAz MOTE.....	69
APPENDIX C- nesC CODE FOR BASE STATION MICAz MOTE.....	72

APPENDIX D- C CODE FOR MICROCONTROLLER ATmega644 in AVR Studio 5.0.....	77
APPENDIX E- MESSAGES FROM BASE STATION MOTE TO VISUAL SENSOR NODE	84
APPENDIX F- MATLAB GUI CODE	85
BIBLIOGRAPHY	102

LIST OF FIGURES

Figure No.	Page No.
2.1 DSSS	9
2.2 UART Bit Sequence	12
2.3 SPI Connections.....	13
2.4 SPI Communication	13
2.5 I2C Communication	15
3.1 CMOS C3088 Camera	25
3.2 ATmega644.....	26
3.3 Flash Memory	26
3.4 MDA100CB.....	27
3.5 Pinout Table for MDA100CB.....	27
3.6 MICAz Mote.....	28
3.7 Connections For Visual Sensor Node	29
3.8 MIB600	31
3.9 Application.....	32
3.10 Communication Protocols Used	33
3.11 System Block Diagram.....	34
4.1 Mote And MIB600 Connection.....	38
4.2 Lantronix Setup.....	38
4.3 Lantronix UART Settings.....	38
4.4 Ping To IP Of The MIB600.....	39
4.5 Packets Received	42
4.6 SerialFowarder Window	43
4.7 Increased Packet Length In MATLAB.....	45
4.8 Dropped Packets, Counter Value Changes From EB To F1	46
5.1 VSYNC And HREF Clock Signals.....	49
5.2 How The Microcontroller Writes To The Flash	50
5.3 Schematic Diagram	51
5.4 Control Message Structure From Base Station To Visual Sensor Node	52
6.1 Packet Structure.....	55
6.2 Array Of One Complete Image.....	55
6.3 Background Subtraction Results.....	59
6.4 Template Matching Results.....	60
6.5 GUI.....	61

LIST OF TABLES

Table No.	Page No.
4.1 Packet Structure	43
E.1 Control Messages.....	84

ABBREVIATIONS USED

DSSS	Direct Sequence Spread Spectrum
GTS	Guaranteed Time Slot
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial Scientific Medical
PAN	Personal Area Network
PC	Personal Computer
POE	Power Over Ethernet
QoS	Quality of Service
RF	Radio Frequency
RS232	Recommended Standard 232
TCP/IP	Transmission Control Protocol/ Internet Protocol
UART	Universally Asynchronous Receiver/ Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
WSN	Wireless Sensor Network
WMSN	Wireless Multimedia Sensor Network

INTRODUCTION

1.1 Background

Wireless Sensor Networks are an evolving technology in the field of distributed computing and sensor based networks. The network can consist of many nodes, which can be equipped with any type of sensor to gather data and distribute control commands. Traditionally, these networks had been restricted to gathering and working with scalar data, such as temperature and humidity levels. This definition is changing by using the sensor networks to gather multimedia information. Multimedia information comprises of video, audio and still images.

The distinguishing characteristics of a wireless scalar sensor network which challenged wireless multimedia sensor network are the low cost of hardware, installation and setup of the network. It is a low power consuming, low bandwidth system where high latency is acceptable with regards to scalar data, application specific QoS (Quality of Service) is not required to be implemented and there is no requirement of source coding techniques for scalar data transmission. Also, an unreliable link can be used to transmit scalar data.

These and other such features present a challenge for enabling wireless multimedia sensor network. A few of the challenges/requirements are that it requires application specific QoS, multimedia information requires high bandwidth, multimedia source encoding techniques are required, multimedia in-processing system is required for reducing transmission of redundant information, power

consumption must be kept at a minimum when transmitting multimedia content and that the network architecture must be flexible to support heterogeneous applications and hardware.

These concepts were kept in mind while developing a wireless multimedia sensor network. For the project, still images were selected as the type of multimedia information.

1.2 Project Overview

The project demonstrates a Wireless Multimedia Sensor Networks using MICAz nodes. Still images were selected as the type of multimedia information. The sensor network consists of the nodes, MICAz node. Every edge node is interfaced with a CMOS (Complementary Metal-Oxide Semiconductor) camera through the data acquisition board thus making a visual node. The camera captures images and the visual sensor node transmits it to a base station. Nodes can be made to relay the data between the visual node and the base station. The multimedia data, after traversing the network reaches the base station. The sink gets the data from a base station and further process/saves it. The sink is a computer or any other computational device.

The sink acts as the central computational and monitoring platform. The application running on the sink receives data from the TCP/IP (Transmission Control Protocol/Internet Protocol) socket, processes and displays it accordingly. The application enables viewing of the situation at the visual sensor node by displaying the image. The images are also stored in the sink for future reference. Finally, the central/base station node can not only receive data from the sensor network but also send

commands from the user at the sink to the edge visual sensor node. Thus, the network can be centrally controlled and configured.

To demonstrate application of this project, the architecture is used to monitor road traffic at traffic signals. The visual sensor nodes are stationed at the traffic signal. Images will be taken at the time of red traffic light, and sent to the PC (Personal Computer). These images are processed to determine the duration of green traffic signal. Higher the volume of traffic, longer will be the duration of the green traffic signal and vice versa.

1.3 Objectives

Firstly, configure the wireless sensor nodes, MICAz motes, in their own operating system, TinyOS, and programming language, nesC. Develop a sensor network using MICAz motes, interface imaging device with the MICAz motes through the data acquisition board, MDA100CB. Then, develop an application software to receive, process and display the images and store them for a record. Develop a user friendly GUI (Graphical User Interface), optimize the system in a way that it leads to as low latency as possible. Finally, show the working of the whole architecture for a specific application.

1.4 Outline Of Tasks

The project started with literature review, where IEEE (Institute of Electrical and Electronics Engineers) research papers were studied to understand the architecture, hardware, topologies and deployment strategies of Wireless Multimedia Sensor Networks. Particular features of many sensor networks were

studied and the architecture of visual nodes used were analyzed. This enabled selecting suitable network architecture and the hardware to meet the project requirement. Further study was conducted on the selected hardware. Then the hardware was appropriately coded and configured using hardware specific operating systems and programming languages.

An application was developed in MATLAB to process the incoming multimedia data from the sensor network. Finally, all the modules and hardware elements were individually optimized and then integrated. Every stage of integration was tested and thoroughly debugged. A user friendly GUI was developed for ease of operation and to make the system more interactive. Finally, the sensor network system was deployed for a model traffic signal, tested and optimized as a whole.

1.5 Platforms Used

Platforms used are AVR Studio 5.0 for programming the image-grabbing microcontroller. TinyOS, the operating system for the MICAz motes and nesC, the programming language for MICAz motes are used for the motes. MATLAB is used for processing the data gathered from the sensor node and Proteus for simulating the microcontroller and the flash memory.

1.6 Organization Of The Document

Chapter 1 contains the introduction to the project. Chapter 2 highlights the literature review while Chapter 3 entails the design of the project. Chapter 4 and 5 explain the wireless sensor network and the image acquisition process respectively. Chapter 6 comprises of the image processing techniques used for the application.

Chapter 7 explains the future work, testing and validation and then concludes the report.

LITERATURE REVIEW

2.1 Introduction

The literature review was carried out to further understand the technologies and protocols being used by individual components of the project. This helps in understanding how the components are to be controlled and applied. A literature review is pivotal in understanding how different components of will be interfaced and integrated together. Chapter 2 highlights various standards and protocols used in this project.

2.2 Communication Protocols

The communication protocol used by the MICAz mote on their radio interface is the IEEE 802.15.4. Therefore, it is essential that the principals and specifications of this protocol be studied in detail. This helps understanding how the MICAz motes transmit and receive messages and control information on their radio interface. The protocol is described in the following section.

2.2.1 IEEE 802.15.4

This is the protocol used by the MICAz motes on the air interface. It is created by the IEEE. It deals with the physical and medium access control layers of LR-PANs (low rate personal area networks). IEEE 802.15.4 does not define the working of the upper layers of the OSI model. It is used with the objective of low cost, low speed

ever present communication between nearby devices, which requires minimum basic infrastructure. This helps in further reducing power consumption.

The features which distinguish the 802.15.4 protocol are low manufacturing cost, low operation cost, it is technologically simple and flexible protocol. Power consumption can be further reduced by decreasing the data rates and is used in applications with low or no QoS requirement. It can supports large network nodes (less than or equal to 65,000 nodes).

The implemented security levels can be selected form three options, Privacy (encryption), Sender Authentication and Message Integrity. Handshaking is carried out to ensure reliability of transfer, beaconless operation is also available. It employs channel access technique, CSMA-CA (Carrier Sense Multiple Access-Collision Avoidance). It guarantees time slots in star topology for low latency devices and supports Star or Peer-to-Peer network topologies. It is capable of extremely low duty-cycle (less than 10 ppm). IEEE 802.15.4 also implements Channels Energy Scan (PLME-ED request). Also, sensor nodes can communicate directly to each other and can be supported on batteries when operating on IEEE 802.15.4.

The frequencies for 802.15.4 are present in three bands which comprises of 27 different channels. Firstly, 16 channels in the 2.4 GHz ISM (Industrial Scientific Medical) band (2.4 to 2.48GHz) Worldwide. It supports data rates of 250 kbps. Secondly, 10 channels in the 915 MHz ISM band (902.0-928.0MHz) EEUU. It supports data rates of 40/250 kbps. Lastly, 1 channel in the 868 MHz band (868.0 - 868.6MHz) Europe. It supports data rates of 20/100/250 kbps.

The basic structure considers a 10-meter range with a data rate of 250 kbps. At the start, lower transfer rates of 20 and 40 kbps were defined. Presently, 100 kbps data rate is also present. Important features are real-time suitability through GTS (Guaranteed Time Slot), channel access using CSMA/CA and support for secure communications.

The characteristics of the physical layer are that it provides the service of data transmission; it provides an interface for management of physical layer which provides access to management function of every layer and keeps information database on the PAN (Personal Area Network). Also, it manages the physical RF (Radio Frequency) transceiver, selects channels, and carries out energy and signal management functions.

The characteristics of the medium access control layer are that it provides data service, provides an interface for management and provides access to the physical channel and a guarantee of time slots. Furthermore, it takes care of node associations frame validation.

As mentioned earlier, 802.15.4 implements the Channel Energy Scan (PLME-ED request). Through this, the energy available in one or several channels is found before using the channel. The energy of the channel depicts the activity, interference or noise. This enables saving energy while choosing free channels when setting the network. There are three different behaviors when facing the energy detection problem. Firstly, Energy Mode scans the channels and then report the energy which is present on the channel. Secondly, CCA (Carrier Sense) Mode, scans the medium instead and informs of the 802.15.4 transmissions. Lastly, CCA and Energy Mode,

scans the medium and informs if there are 802.15.4 transmissions above the set energy threshold.

The 802.15.4 protocol can combat noise because it uses DSSS (Direct Sequence Spread Spectrum) technique to modulate the information. Signal before and after DSSS modulation is shown in Figure 1. Each information bit is modulated into four different signals. This results in a greater bandwidth and lower power spectral density per signal. Therefore there is lower interference in the frequency bands, enabling easier detection as well as decoding of signal at the receiver thus improving SNR (Signal to Noise Ratio).

Many DSSS techniques can be used, depending on the physical limitations of the circuit and the number of symbols being processed at a time. Examples of DSSS techniques are BPSK (Binary Phase Shift Keying), O-QPSK (Offset Quadrature Phase Shift Keying) and PSSS (Parallel Sequence Spread Sequence).

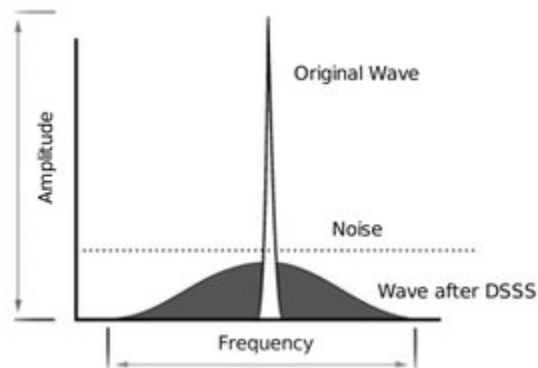


Figure 2.1 DSSS

Along with that, 802.15.4 employs CSMA-CA or GTS technique, preventing all the devices to start simultaneous transmission.

CSMA-CA is the most common technique. Every node listens to the medium before transmitting. If the energy found on the transmission medium is higher than a

specific/threshold level then the node waits for some random amount of time before trying again.

The GTS technique employs a central node, a PAN coordinator. This coordinator hands out time slots out of 16 possible time slots to every node, so they know when they have to transmit. In the first step, a node sends a GTS request message to the PAN coordinator. In response, the coordinator sends a beacon message. This beacon contains the information on allotted slot and the number of assigned slots [1].

The 802.15.4 protocol works with low duty cycles. The transceiver does not need to be on and powered up all the time, as for most of the time the transceiver is idle with no packets to transfer. So with low duty cycles, the transceiver can be sleeping when it is idle, reducing power consumption. The percentage depends on the communication model used.

2.2.2 Serial Communication

There are various serial protocols being used at different segments of the project. The protocols used are UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit). A thorough understanding of the protocols is required to understand the working of the modules. It makes the implementation of the individual segments easier as well as their integration. UART is being used between the microcontroller and the MDA100CB Data Acquisition Board, and also between the MDA100CB Data Acquisition Board and the MICAz mote. UART communication enabled transmission and reception of packets to and from the camera and the MICAz mote. SPI is used by

the microcontroller and the Flash memory. I2C is used between the microcontroller and the camera.

2.2.2.1 UART

UART sends the data in a sequential bit by bit manner, the receiver USART re-assembles the data back to complete bytes. Data can be transmitted without a clock signal. Both, the transmitter and the receiver, decide the parameters before the transmission and bits such as the start and stop bits are added to every set of bits. These are used to synchronize the transmitter and the receiver.

The Start Bit is added to the start of every to be transmitted word. It alerts the receiver that a word is going to be sent. It replaces the clocking signal because it forces the clock of the receiver to be in sync with the clock of the transmitter. The clocks must be precise and not have the frequency drift by more than 10% during the transmission of the remaining bits in the word [2].

Then, individual data bits are transmitted. The LSB (Least Significant Bit) of the word is sent first. The time period of each bit is the same. For each bit, the receiver hears at the transmission wire for half of the time period of one bit to determine if it is a 1 or a 0. The transmitter is unaware of this action of the receiver and begins transmitting the next set of data when the clock says.

After the MSB (Most Significant Bit) is transmitted, the transmitter may add a Parity Bit. Parity Bit enables receiver to compute simple error checking. Parity Bit may follow one of the two methods, Even Parity or Odd Parity. This bit is optional and its usage has to be decided by both, the transmitter and the receiver, before the start of transmission.

The last bit sent is the Stop Bit, which tells the receiver that one transmission has ended. Upon checking by the receiver, if the Stop Bit is not present where it should have been then the receiver deems the whole transmission to be erroneous and will report a Framing Error to the host processor. What most commonly leads to a Framing Error is the non synchronized clocks of the receiver and the transmitter, or interruption of the signal.

The UART removes the Start, Stop and the Parity bits. If UART at the transmitter and receiver is set up correctly these bits are not passed as user data. As the asynchronous data synchronizes itself, the transmission line remains idle if there is no data to transmit. Figure 2 shows the UART bit sequence.



Figure 2.2 UART Bit Sequence

2.2.2.2 SPI

Serial Peripheral Interface is also called the 'four' wire serial bus. It is a synchronous serial protocol. Multiple slave devices can be selected with individual slave select (chip select) lines. The synchronization signal is provided by the master device. The synchronization (clock) signal controls when the data can change and when it is valid for reading. This do's away with the need to accurately time the data for transmission. A change in the clock signal does not disturb the data. The clock signal signals the start and end of transmission of valid data. SPI connection between master and slave is shown in Figure 3.

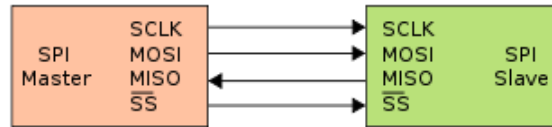


Figure 2.3 SPI Connections

SPI is a Master-Slave protocol. Master device yields the clock line, SCLK (Serial Clock). The change in state on the clock line enables the data to be sent. All the slaves are controlled by the master device. The SPI signal flow between master and slave is shown in Figure 4.

There are four logical signals present for SPI communication, namely, SCLK or serial clock (output from master), MOSI or master output, slave input (output from master), MISO or master input, slave output (output from slave) and SS or slave select (active low, output from master).

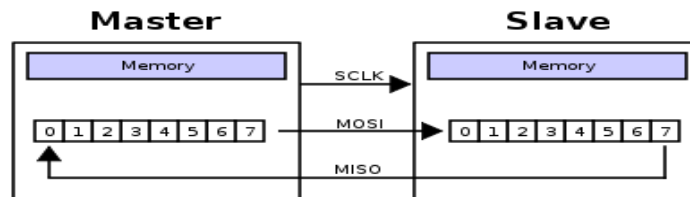


Figure 2.4 SPI Communication

SPI is a Data Exchange protocol. A device has to be both, transmitter and receiver, and need to have two data lines, one for input and the other for output. With the clocking out of data, new data clocks in. The data must be read after a transfer has taken place [3].

Slave select signal tells when the master needs to communicate with a device and access it. This signal is used for when multiple slaves exists in a system, but is optional for a system with only a single slave. It is better to use the slave select

signal. A low Slave select signal indicates the SPI is active and the slave listens for SPI clock and data signals, while a high will signal inactivity. It retunes the SPI slave for receiving the next byte.

In SPI, data changes during the rising or falling edge of SCK, synchronizing the data and the clock signal. Logically, the point of reading and changing of the data is opposite.

CKP selects the SPI clock polarity, it selects if the clock will be idle high or low. When CKP is 1, SCK will idle high. Conversely, if CKP is 0, SCK will idle low.

2.2.2.3 I2C

I2C (Inter-Integrated Circuit) is also known as the two wire interface. It is a serial, 8 bits oriented, bi-directional data transfer protocol. Only two bus lines are required: a SDA (Serial Data Line) and a SCL (Serial Clock Line). It was originally designed to interact with small number of devices.

Master-slave relationship exists at all times. The master device is one which controls the SCL, data transfer and device addressing. Masters can either operate as master-transmitters or as master-receivers. Master-transmitter transmits data to slave-receiver. Master-receiver receives data from the slave-transmitter. I2C can also handle multi-masters, with the feature of collision detection and arbitration, if multiple masters start simultaneous transmission. It has three modes, Standard mode (100kbps), Fast mode (400kbps) and High speed mode(3.4Mbps).

Each device has a unique software address. There are two types of addressing schemes supported by I2C, 7 bit addressing and 10 bit addressing. However, the

maximum number of devices to be supported is restricted by the maximum bus capacitive loading, 400pF.

The start condition (S) occurs when SDA transitions from 1 to 0 when SCL is 1. The stop condition (P) occurs when SDA transitions from 0 to 1 when SCL is 0. The bus is free between a consecutive stop and start condition. The bus is busy after start condition and before the next stop condition. Figure 5 shows how multiple slaves are attached to a master through SPI.

I2C communication takes place when SDA and SCL are both high. The bus is free. A start bit is placed to indicate the usage of the bus. Slave devices listen to the data bus to see whether they are being addressed. Clock signal (SCL) is provided. It is used to give the reference time at which each data bit on the SDA will be correct and useable. The data must be valid when the SCL is valid. Then, address of the slave device to start communication with it is put on the line.

Then a bit is put on the SDA telling whether the master wants to read or write data. At this point the slave must send a one bit Acknowledgement. After the Acknowledgement is OK, data can be transferred. Acknowledgement after every 8-bit data is required by the sender. After end of transmission, the bus is freed up after transmission of a Stop bit [4].

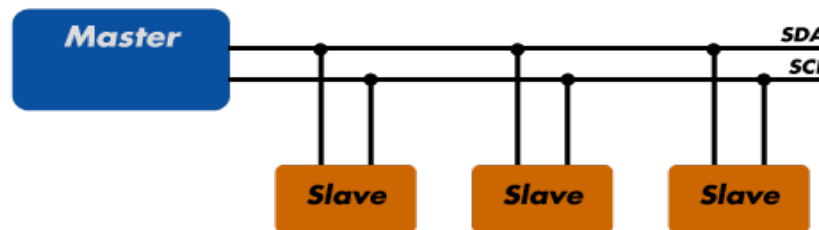


Figure 2.5 I2C Communication

2.2.3 TCP/IP Communication

TCP/IP is abbreviated for Transmission Control Protocol/Internet Protocol. It is set of rules, for computers to communicate through the Internet. TCP/IP is TCP and IP working together on two layers. The upper layer, TCP, assembles of a message into smaller packets for transmission over the Internet. The TCP layer of the receiver reassembles the packets to make the original message.

The lower layer, IP, is responsible for addressing of the packets for reaching the correct destination. Despite the route, the packets will be reassembled correctly at the receiving computer. For communication, TCP/IP employs client server model. It is also point to point communication.

TCP/IP is stateless, as each client request is taken as a new one, a connection is only maintained for the transmission of all the packets of one message, after which the connection is freed for continuous use elsewhere in the network.

Application layer protocols which use TCP/IP are HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) and Telnet. Computers with Internet connection through the analog phone modem use SLIP (Serial Line Internet Protocol) or PPP (Point-to-Point Protocol) to access the Internet.

IP packets are encapsulated through these protocols, so that they can be transmitted using a dial up phone. Protocols related to TCP/IP are UDP (User Datagram Protocol) [5]. UDP is a standard of TCP/IP and defines unreliable communication between TCP/IP hosts.

2.3 Camera Specifications

An understanding of the camera specifications was necessary to understand how the camera functioned. Then a study of the camera protocol is necessary to understand what protocol and format is to be used to communicate with it and to understand the format and protocol of the output of the camera. This will help in understanding how the camera can be operated and controlled. This will also help in integrating the camera in the circuitry. A thorough review and understanding of the data sheet of the camera is an absolute must to feed an appropriate input into the camera and obtain an appropriate output.

2.3.1 Progressive Scan Read Out Mode

It is also called as noninterlaced scanning. It displays and transmits moving images in which all the lines of each frame (or rows) are drawn in sequence from top to bottom, and not in an alternate manner as had been done in interlaced scanning. It results in a smoother, more detailed image, with all the fine details easily visible. It yields good and precise results, especially with text, as the screen does not flicker.

2.3.2 CIF/QCIF Resolution

CIF stands for Common Intermediate Format. This is the format produced by low resolution digital cameras for the horizontal and vertical resolution of pixels of YCbCr video signals. It has a resolution of 352 pixels by 288 pixels.

QCIF, Quarter Common Intermediate Format, has one-fourth of the area of a CIF frame. It has the resolution of 176 pixels by 144 pixels. It is smaller than CIF, QVGA (Quarter Video Graphics Array), and VGA (Video Graphics Array) [6].

2.4 Memory

The memory required by the interfacing circuitry was first decided by first studying the types of memory available, the characteristics of each of these types and the requirements of the interfacing circuitry. Also, compatibility of the microcontroller and the memory was first established and then the vendor was chosen so that no issues of compatibility will arise.

2.4.1 EEPROM

It stands for Electrically Erasable Programmable Read-Only Memory. It is a nonvolatile memory used in devices which holds variables and programs which need to survive power off. It can only hold small amounts of data, for larger amount of data Flash memory is used. EEPROM can be erased and reprogrammed by users through application of higher than normal electrical voltage. The voltage can be generated internally or externally. EEPROM can be programmed and erased in circuit. Initially, they supported byte by byte operation, but then it moved onto multi-byte operation. It can be reprogrammed by a certain number of times [7].

2.4.2 SRAM

Static Random Access Memory (SRAM) is a volatile memory that uses bistablelatching circuitry to store each value of bit. Static RAM, unlike DRAM (Dynamic RAM), does not need periodical refreshes. It is more expensive, faster and has lower power consumption than DRAM. SRAM is less dense as compared to DRAM, and is not used for low-cost, high-capacity workload. For this reason, it is not used in personal computers.

SRAM can be in either one of the three states: standby, reading and writing. It is used where low power consumption, bandwidth or both are prime factors for consideration. It has two types, non-volatile SRAM and Asynchronous SRAM [8].

2.4.3 FLASH MEMORY

It is a nonvolatile memory which can be electrically erased and reprogrammed. It has fast read access times, though not of the type presented by static RAM or ROM. Flash memory needs to be erased in a specific manner before it can be rewritten.

Extra features include mechanical shock resistance, high durability, and ability to endure high pressure and high temperature. Devices using Flash memory include medical electronics, personal computers, mobile phones, industrial robotics, digital audio players, video games, digital cameras, scientific instrumentation etcetera [9].

2.5 Composition Of Image

In order to receive, store, break down, packetize, transmit and rearrange an image the composition of the image must be understood thoroughly. An image is an array of data in a matrix form, consisting of values within a range defined by the type of image. For a binary image, the value can either be 1 or a 0. For 8 bit representation, 256 shades of gray are present, therefore the values in the image matrix varies from 0 to 255. As the data in the project is an image, the composition and properties of the image must be studied. Then, depending on the camera and the memory limitations, the type of image for transmission is selected. This further affects the delay in the transmission, buffering and reception of the image. The hardware of the project is also affected.

2.5.1 Gray Scale Image

It is an image in which the pixel value contains only the information about the luminescence of the received light signal. Unlike monochrome images, or binary images, gray scale images consist of shades of gray, ranging from black where there is least intensity to white where there is the most intensity. A pixel's intensity is given from 0 to 1, where 1 represents white. Such images are also referred to as 8-bit gray scale. The binary representation denotes 0 as black and 255 as white or as dependent on the output of a camera.

Nowadays, gray scale images represent 8 bits per pixel value, generating 256 different intensities, the shades of gray. Images required for technical use, such as medical imaging require more number of bits to represent one pixel for accuracy and counter the quantization error. Normally, 16 bits per pixel are used [10].

2.6 Review Of The Studied Wireless Sensor Networks

There are numerous wireless sensor network testbeds established in various universities around the world. These testbeds are used to set up WSNs, research is carried out on them and the results of various experiments and setups are then published. These testbeds help in understanding the basic architecture of a WSN so that a WMSN can be created appropriately.

2.6.1 Cyclops

The platform, Cyclops, is a collaboration between Agilent Technology Inc. and the University of California. It consists of low-resolution imaging device which can be integrated with any WSN technology, such as MICA and MICAz mote. Power

consumption of Cyclops is minimal, this means that it can be used for large scale deployment and has an extended lifetime. This also results in harsh constraints in its computational power and imaging size. Therefore, Cyclops can be used for many types of applications.

2.6.2 Wireless Smart Camera (WiCa)

This sensor node consists of two cameras and is based on the ZigBee protocol and the 8051 microcontroller. It is compatible with the IEEE 802.15.4 communication protocol. It has the capability to store two images of 256 by 256 pixels. This can enable distributed applications to be supported. It can be attached with external memory to further support numerous applications.

2.6.3 MeshEye

The MeshEye architecture has two layers, based on the ARM7. It is used for real time object detection. It has a lower resolution system which determines the position, range and size of the moving objects. The higher resolution color camera is used for image processing [11]. It offers reduced complexity, response time, and power consumption over conventional solutions.

2.6.4 Meerkats

The Department of Computer Engineering at University of California, Santa Cruz has developed 'Meerkats', a testbed to research monitoring and surveillance of wide areas. It consists of eight visual sensor nodes, each of which consists of a Stargate, battery, power supply and IEEE 802.11.b wireless card with a laptop acting as

information sink [11]. The nodes in Meerkats are capable of sufficient computational and storage facility.

2.6.5 Explorebots

The University of North Carolina-Charlotte has developed an experimental test bed 'Explorebots'. It has mobile, wireless-controlled robots capable of moving around and transmitting multimedia content at 320x240 at 15 fps. This enables a mobile WMSN with multi-hop facility. Reliability on human mobility is eliminated in this technology.

2.6.6 SenseEye

SenseEye by the University of Massachusetts is a powerful testbed consisting of four different cameras, the Agilent Cyclops, the CMU cam Vision sensor, a Logitech Quickcam Pro Webcam and a Sony PTZ (Pan Tilt Zoom) camera and three different platforms; Crossbow Motes, Intel Stargates and mini-ITX embedded PC. These different platforms are used at different tiers and test bed is organized in a hierarchy. It is capable of object detection, recognition and tracking [12] [13].

2.6.7 WSN At BWN Laboratory, Georgia Tech

Broadband and Wireless Networking Laboratory at Georgia Tech has developed an experimental test bed for multimedia wireless sensor network and integrated with scalar sensor network test bed. iMotes and MICAz are used for scalar sensing and MICAz from MEMSIC and Stargate platforms are used for low and high-end imaging respectively. The high-end PTZ cameras are installed on a mobile robot. The testbed

also has storage and computational hubs, for performing computationally intensive multimedia processing algorithms.

NETWORK TOPOLOGY and DESIGN

3.1 Overview

A wireless sensor network has a few basic parts, wireless sensor nodes, base station node(s) and a central, computational device to manage the incoming data from the sensor network and send commands into the network. A database is often required on the computational device to store the incoming sensor data for use. The communication links between these separate entities is an important part of the network as well. An individual sensor node comprises of a sensor, a radio transceiver, a small computing device such as a microcontroller and a power supply. The type of sensor depends on the application of use the sensor network is catering to. A few scalar sensors, such as humidity, light and temperature measurement devices are already present on the add-on data acquisition board of the node. Other sensors have to be separately interfaced with the node to enable it to gather the required kind of data. Due to their small size, long life and ability to communicate wirelessly with the other nodes they are the popular choice for establishing setups for gathering scalar and multimedia data.

A collection of such nodes make up the sensor network. Among these nodes, some or all can be equipped with a sensor to gather data; some can be intermediate only used for multi-hopping/relaying purposes. A central base station node is usually required to aggregate the data from all the nodes and interface the sensor network with the computational device. Various communication links have been used to

communicate data between the nodes, such as 802.15.4, Bluetooth, ZigBee and ZigBee Beta. In this chapter the modules and design of the project will be explained.

3.2 Modules

The project is divided into seven modules. The modules in a sequential manner are the camera, the image grabber, which consists of a microcontroller and a Flash memory. Then the next modules are the MDA100CB Data Acquisition Board and MICAz mote. They are followed by the MIB600 Ethernet Gateway and a computational device such as a PC. The last module is the traffic controller circuitry.

The salient features of the modules and their functions are listed below:

3.2.1 CMOS Camera

The camera comprises of the OmniVision imager OV6620. The images are read in a progressive scan read out mode, enabling it to be used for quality imaging. The camera has a SCCB (Serial Camera Controller Bus) interface, which is a subcategory of I2C interface. It supports CIF/QCIF Resolution and low power operation enables it to be used with the sensor network. The camera produces up to 60 frames per second. CMOS cameras consume less energy than their CCD (Charged-Coupled Device) counterparts and hence are a popular choice in wireless sensor networks [14]. The camera is shown in Figure 6.



Figure 3.1 CMOS C3088 Camera

3.2.2 Image Grabber

It comprises of microcontroller ATmega644 and External Memory 45DB321D-SU. ATmega644 is an 8 bit microcontroller, it consists of 64 Kbytes of in-system programmable Flash program memory. It has 2 Kbytes of EEPROM with 4Kbytes worth of internal SRAM. It supports USART, SPI and I2C interfaces. It is in-system programmable through JTAG (Joint Test Action Group). It requires 5 volts for operation [15]. The microcontroller is shown in Figure 7.



Figure 3.2 ATmega644

The External Memory, 45DB321D-SU, requires 2.5 to 3.6 volts or 2.7 to 3.6 volts for operation, supports SPI interface and provides 32 Mbit memory. It provides two SRAM buffers, their sizes are 512 and 528 bytes. By using buffers, data can be received even when the main memory is being reprogrammed. Also, data can be written in a continuous manner. Its programming and erase cycles are self-timed [16].The external memory is shown in Figure 8.



Figure 3.3 Flash Memory

The functionalities of the image grabber are that it enables the MICAz mote to capture images and saves motes battery life by doing all the computations itself,

thereby shifting the storage and packetization workload from the mote’s memory and processor to itself.

3.2.3 MDA100CB Data Acquisition Board

The MDA series sensor boards have a precision thermistor, a light sensor/photocell, and a general prototyping area. It provides connections to USART, SPI, ADC (0-7) amongst many others. The prototyping area on the MDA100 is used to interface the MICAz mote and the image grabber on a UART link..The MDA100CB board is shown in Figure 9.



Figure 3.4 MDA100CB

The diagram and table of the prototyping area layout is shown in Figure 10. The prototyping area has 45 unconnected solder holes for breadboard connectivity, for connecting external devices or sensors to the mote [17]. UART pins on the MICAz mote are accessed through the MDA100CB board, the pins on the MDA100CB board provide the connectivity as shown in Figure 10.

	A	B	C	D	E	F
1	GND	GND	GND	VCC	VCC	VCC
2	OPEN	OPEN	USART1_CK	INT3	ADC2	PW0
3	OPEN	OPEN	UART0_RX	INT2 [†]	ADC1 [†]	PW1 [†]
4	OPEN	OPEN	UART0_TX	INT1	ADC0 [†]	PW2
5	OPEN	OPEN	SPI_SCK	INT0	THERM_PWR	PW3
6	OPEN	OPEN	USART1_RX	BAT_MON	THRU1	PW4
7	OPEN	OPEN	USART1_TX	LED3	THRU2	PW5
8	OPEN	OPEN	I2C_CLK	LED2	THRU3	PW6
9	OPEN	OPEN	I2C_DATA	LED1	RSTN	ADC7
10	OPEN	OPEN	PWM0	RD	PWM1B	ADC6
11	OPEN	OPEN	PWM1A	WR	OPEN	ADC5
12	OPEN	OPEN	AC+	ALE	OPEN	ADC4
13	OPEN	OPEN	AC-	PW7	OPEN	ADC3
14	GND	GND	GND	VCC	VCC	VCC
15	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
16	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
17	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

Figure 3.5 Pinout Table for MDA100CB

3.2.4 MPR2400CB IEEE802.15.4/ ZigBee Compliant MICAz

Motes (2.4GHz)

MICAz is a 2.4GHz wireless transceiver which empowers low power WSNs. It operates on IEEE 802.15.4 compliant RF transceiver, in the ISM band (2.4 to 2.48 GHz). It employs DSSS radio and RF power of -24dBm to 0dBm. MICAz has a data rate of up to 250kbps [18]. Figure 11 displays a MICAz mote.



Figure 3.6 MICAz Mote

MICAz mote is a low power and low bandwidth wireless transceiver widely used for enabling many research oriented sensor networks. The MICAz mote consists of Atmel ATmega 128L processor, an 8 bit microcontroller, operating at 4 megahertz. It has 128 kilobytes of flash memory for storing the mote's program. It requires low voltage for operation leading to consumption of mere 8 milliamps while it is running, as compared to 15 micro amps in sleep mode.

This low power consumption allows a MICAz mote to run for more than a year with two AA batteries. However, the programmers will typically write their code so that the CPU is asleep majority of the time, allowing it to extend battery life considerably [19].

MICAz motes have a 10-bit A/D converter to digitize sensor data. When receiving data, it consumes 10 milliamps. When transmitting, it consumes 25 milliamps. Conserving radio power is the key to long battery life [20].

The MICAz mote has been used in three ways in the project: as a node interfaced with the camera to become a visual sensor node, as an intermediate mote and as a base station mote. The functionality of each is explained below:

3.2.4.1 Visual Sensor Node

These motes are interfaced with the camera as shown in Figure 12.

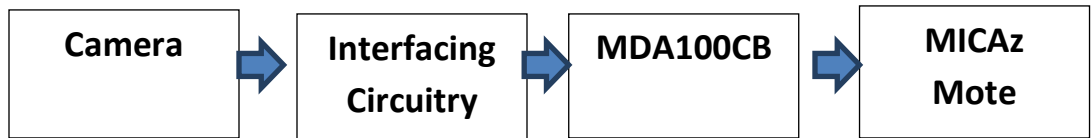


Figure 3.7 Connections For Visual Sensor Node

This enables the mote to become a wireless, visual sensor capable of transmitting multimedia information. These motes are placed at locations where visual information of the surroundings is required. These motes transmit the multimedia information to the nearest mote for multi hopping towards the base station mote.

3.2.4.2 Intermediate Mote

This mote is not equipped with a camera; rather it only relays the radio messages it receives from the visual sensor node to the base station mote and vice versa. This enables the extension of the network and allows working around the constraints of LOS (Line Of Sight) and MICAz's radio range. The addition of Intermediate mote affects the delay in reception of an image. More number of intermediate motes means a higher delay.

3.2.4.3 Base Station Mote

MICAz mote can be programmed as a base station. When connected to an Ethernet gateway, MIB600, it aggregates data coming from multiple sensor-mote hybrid for transmission onto the PC. It also transmits the user commands from the PC to the sensor network.

For convenience and ease of understanding, the sensor networks are organized hierarchically, with the base station serving as the central node to collect the data from the sensor network. The base station mote, along with the MIB600 gateway, collects and forwards data to and from the PC or any computational device. The base station is typically resource-rich in terms of its computational ability, storage capacity, and energy lifetime compared to individual sensor nodes [21].

The base station mote maintains wired and wireless connectivity. For the project, the base station is situated near the laptop, but it can be placed on top of a command van or may have limited mobility enough to be guided to a favorable location in the sensor network topology.

3.2.5 MIB600CA Ethernet Gateway

The MIB provides Ethernet connectivity for communication and in-system programming. Sensor network can be accessed by the MIB600 through the TCP/IP socket. The MIB600 bridges wired and wireless segments of a network. The MIB600 offers two different ports. One is dedicated to in-system mote programming and a second for routine data communication over the LAN (Local Area Network). The built-in POE (Power Over Ethernet) feature removes the need for an external power source, Figure 13 shows MIB600 Gateway.



Figure 3.8 MIB600

The gateway interfaces between the base station node of the wireless sensor network and the PC where the data is aggregated and processed. An application running on the PC appropriately processes the data. The gateway used is MIB600CA which is an Ethernet based gateway. The functions of the MIB600 are to transfer sensor data to the PC and transmit user commands and code updates to one or more of the network's nodes. The MIB600CA serial server connects directly to a 10 Base-T LAN like any other network device [22]. The MIB600 is only connected to the base station node through the 51 pin connector, using UART connectivity.

3.2.6 PC/ Other Computational Device

A PC is used to acquire and parse packets from the MIB600 as required. An application is developed to acquire the data from the TCP/IP socket of the MIB600, extract the data from the headers and process the data. The only incoming data in the network are images from the visual node. The image is stored on the PC for future reference. In this application, the visual sensor node is envisioned to be placed on the traffic signal and will be interfaced with the traffic signal. This is depicted in Figure 14. An image of the cars parked at the red traffic signal will be sent to the PC using the aforementioned architecture. Image received at the PC will

be processed using two different algorithms: background subtraction and template matching.

If more cars are present at the traffic signal the decision to turn the green traffic signal on for a longer period of time is relayed back to the visual sensor node. The visual node is interfaced with the traffic signal and will turn the green signal on for the duration of time set by the PC. For lesser number of cars, the green signal will be turned on for a shorter duration.

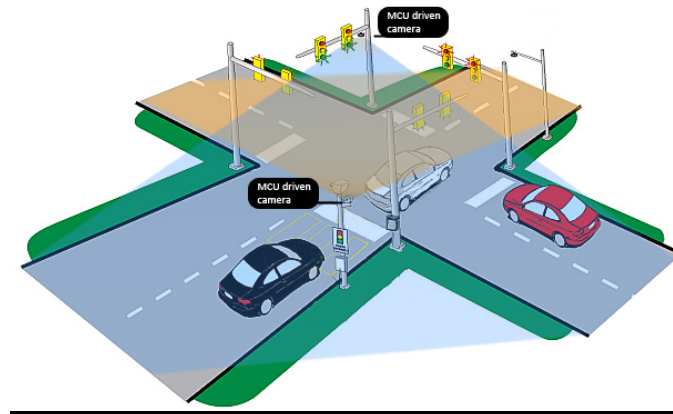


Figure 3.9 Application

For ease of access and user interactivity a GUI is created. It carries out the following functionalities of opening and closing TCP/IP connections, receiving the raw data, extract data from the packet and discard headers, displaying the image and network parameters such as network efficiency, separating the coding from the functionalities of the system, transmitting the calculated duration of the green signal back to the visual node and allowing user to control camera parameters.

3.2.7 Traffic Controller Circuitry

This will be interfaced with the visual sensor node. It is primarily controlled by the microcontroller, ATmega644, and consists of red and green LEDs (Light Emitting

Diode). Turning on of the red LEDs depict number of cars lesser than the set threshold. Turning on of the green LEDs depict number of cars more than the set threshold.

3.3 System Design Block Diagram

The communication protocols which enable one component to communicate and integrate with another are highlighted in Figure 15. The communication protocols have already been explained. The I2C bus is a 8 pin bus, meaning 8 bit will represent one sample. Therefore a gray scale image will be received from the camera.

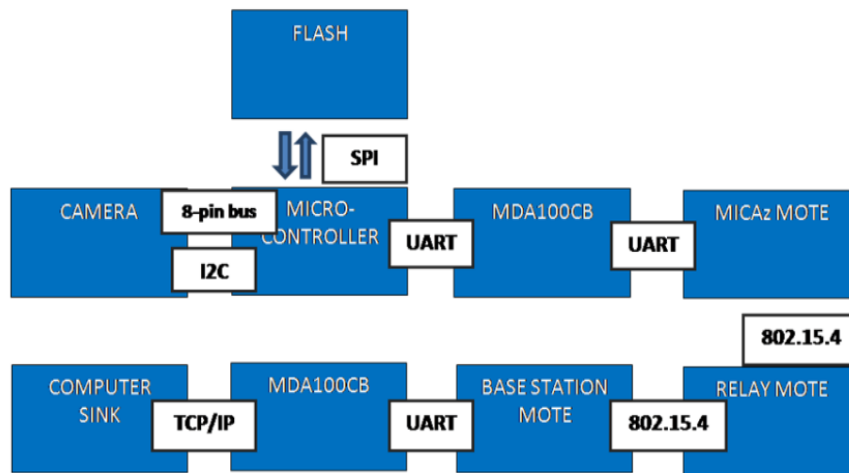


Figure 3.10 Communication Protocols Used

The system design is represented in a block diagram in Figure 16. The block diagram shows the generic model for the WMSN. The end processing and the end circuit which receives the result information depend on the application for which the WMSN is being used for. The decision is relayed back to the visual sensor node, which can be interfaced with the appropriate circuitry and can act according to the received results.

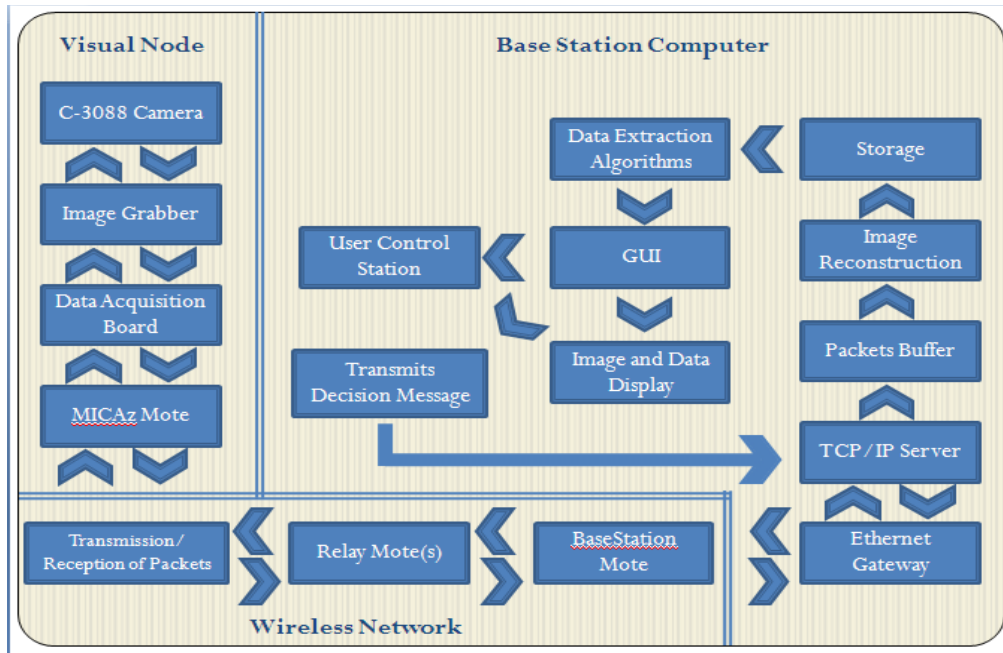


Figure 3.11 System Block Diagram

3.4 Project Overview

The C3088 camera will snap an image and pass it onto the interfacing circuitry in a progressive scan read mode. The microcontroller in the interfacing circuitry receives the packets and transfers them to the external memory, 45DB321D-SU. The flash first stores the entire image and sequentially passes it to the MICAz mote.

The MDA100CB acts as the data acquisition board for the MICAz motes. The MDA100CB is attached with the MICAz mote to facilitate the access of the pins of the mote's microcontroller, ATmega128L. The pins accessed on the mote are the UART transmit, UART receive and Ground. The interfacing circuitry is interfaced to the mote through the MDA100CB board.

MICAz mote transmits data at 2.4 GHz in the ISM band. The mote transmits the information to a relay mote. This intermediate mote transmits the information to the base station mote. The base station mote aggregates the data from the sensor

network and sends it on to a PC or computing device through the MIB600CA gateway.

The MIB600CA gateway collects data from the base station mote and delivers it to the TCP socket of the PC. The application running on the PC accepts data from the TCP socket and processes it into the required form.

3.5 Conclusion

The individual modules were studied, their datasheets were parsed. The modules were coded, tested and developed individually. They were then individually optimized before integrating them with the other modules to develop the project's model.

WIRELESS NETWORK OF MICAz MOTES

4.1 Overview

This chapter describes the programming of the MICAz motes for their three roles in the network. To recap, the MICAz motes are the only wireless nodes in the network and are used for three purposes, as the wireless transceiver in the visual node, as the wireless transceiver at the base station and as relays in the intermediate network between the visual nodes and the base station.

The motes are programmed in TinyOS, the operating system for enabling low power, low bandwidth wireless sensor networks. They require to be programmed in the nesC language. The code installed on the MICAz mote is written in nesC, the compiler converts it into the binary format prior to installation [23].

In this chapter the steps for installation of the TinyOS and compiling codes on the MICAz motes are explained in detail. They include the code for transmission of data from the visual sensor mote to the intermediate mote, from there to the base station mote, to the Ethernet gateway and creating TCP server on the PC.

4.2 Operating System For Programming MICAz Motes

TinyOS is a compact, simple and lightweight operating system explicitly designed for low-power wireless sensors. TinyOS focuses on extremely low power operation, ideal for power consuming devices such as the MICAz motes. TinyOS is designed for the small, low-power microcontrollers' motes have. With TinyOS, applications for

motes can be made more easily. It provides a set of important services and constructs, such as sensing, communication, storage, and timers.

It upholds a concurrent execution model, so programmers can construct applications out of reusable interfaces and components without worrying about unanticipated interactions. TinyOS runs over multiple platforms, most of which are open to addition of new sensors and hardware. Furthermore, TinyOS's structure makes it easy to port to new platforms. nesC is a dialect of C with features to reduce RAM and size of the code, to optimize and help prevent low-level bugs like conditions [23].

4.3 Installation Of TinyOS And nesC Compiler

There are multiple options for installing TinyOS and nesC but for the project, TinyOS has been installed in VMWare Workstation. TinyOS 2.1.0 was selected. The procedure is to download XubunTOS 2.1 virtual machine from

<http://sing.stanford.edu/TinyOS/dists/xubuntos-2.1-vm.tar.gz>

Then run the XubunTOS iso file, open the VMware Player, open XubunTOS and debug the terminal using the command

```
CLASSPATH=.:$CLASSPATH:/opt/TinyOS-2.1.0/support/sdk/java/TinyOS.jar
```

Then add serial port and Ethernet adapter in the virtual machine and update the Java package called Java Development Kit or "JDK". These packages are needed for the java GUI for several applications and install "sun-java6-jdk" from synaptic manager. Now, the system is ready to program the MIB600CA board. Firstly, power off the MICAz mote and connect the MIB600CA programming board through the Ethernet cable as shown in Figure 17.

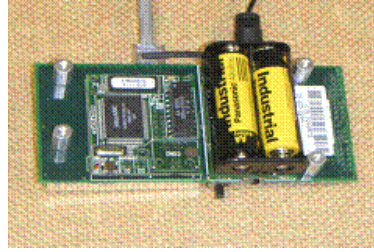


Figure 4.1 Mote And MIB600 Connection

Then assign IP address to the MIB600CA programming board by using Lantronix Device Installer software and assign parameters to the programming board through the Internet Explorer on the host PC as shown in Figure 18 and 19.

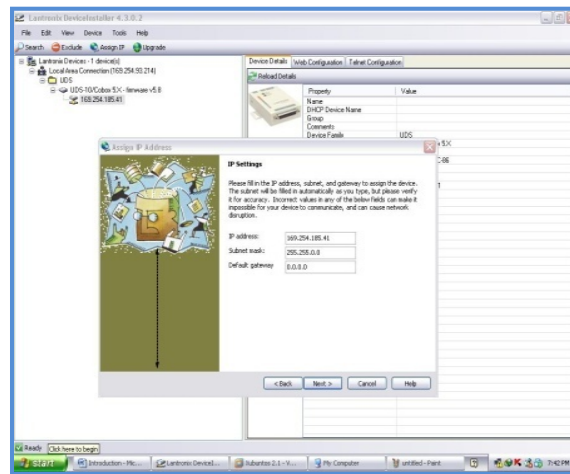


Figure 4.2 Lantronix Setup

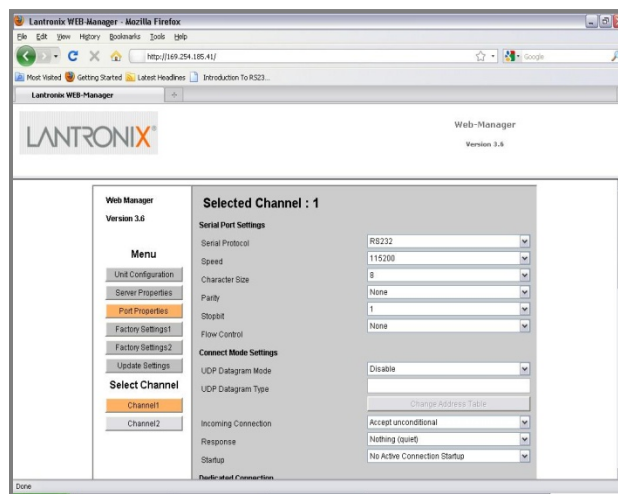
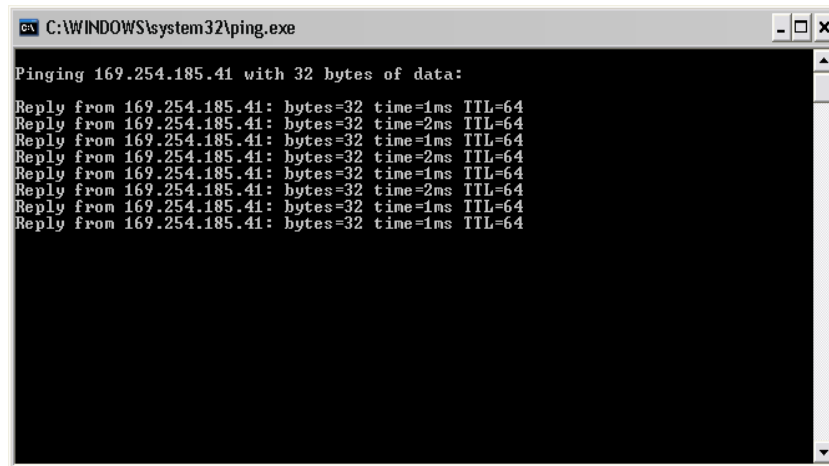


Figure 4.3 Lantronix UART Settings

The connection between XubunTOS and the MIB600CA can be checked by accessing the device in Mozilla Firefox. Enter the IP of the programming board in the address bar, if page similar to the above opens up with the same information, it means that the programming board is successfully connected [23]. Also, by sending a ping request at the MIB 600's IP address was verified that a successful PC to Gateway connection was established. A successful ping is shown in Figure 20.



```
C:\WINDOWS\system32\ping.exe
Pinging 169.254.185.41 with 32 bytes of data:
Reply from 169.254.185.41: bytes=32 time=1ms TTL=64
Reply from 169.254.185.41: bytes=32 time=2ms TTL=64
Reply from 169.254.185.41: bytes=32 time=1ms TTL=64
Reply from 169.254.185.41: bytes=32 time=2ms TTL=64
Reply from 169.254.185.41: bytes=32 time=1ms TTL=64
Reply from 169.254.185.41: bytes=32 time=2ms TTL=64
Reply from 169.254.185.41: bytes=32 time=1ms TTL=64
Reply from 169.254.185.41: bytes=32 time=1ms TTL=64
```

Figure 4.4 Ping To IP Of The MIB600

4.4 Connection Specifications In Lantronix

The parameters assigned in Lantronix are the port number, the default baudrate, the implemented baudrate, the character size and the number of stop bits. Port assigned was USART0 of the ATmega128. The default baud rate was set as 115200 bps, the implemented baud rate was set as 115200 bps. The character size is kept as 8 bits and Stop bit was chosen as 1.

4.5 MICAz Notes For Data Transmission

TinyOS is an open source software, many programs have already been developed which can be freely used. Basic applications, interfaces and configurations were

used and developed for creating the desired applications for the project. The motes were coded on the basis of their usage, edge visual sensor node, intermediate node and the base station mote. These TinyOS codes are attached in the appendix A to C.

4.5.1 Visual Sensor Module

A MICAz mote part of the visual sensor node receives control messages from the central station and passes them on to the camera controller/image grabber and receives image data from the image grabber and transmits them towards the base station. Since the link between the image grabber and MICAz mote is a reliable serial link, the mote is hard programmed to receive the exact number of bytes per image (25920 bytes).

$$25920 \text{ bytes} = (88 \text{ payload bytes} + 2 \text{ synchronization bytes})/\text{packet} * 288 \text{ packets}$$

---- Eq No (1)

4.5.2 Intermediate Module

Motes are used in between the visual sensor node and the base station to increase the distance and work around LOS constraints. These motes are programmed to receive packets, change their destination address to the next mote in their destination and forward them.

4.5.3 Base Station Module

In the stage, the packets have to be transmitted to the computer. For this, the program of BaseStation.nc was used which acts as a simple bridge between the serial and radio links. It includes queues in both the serial and radio links, with a guarantee that once a message enters a queue and it will eventually leave on the

other interface. It only acknowledges a message arriving over the serial link if that message was successfully queued for delivery to the radio link. The program was edited to include transmission of received UART bytes from the computer to enable two-way communication throughout the network.

4.6 Listen Tool

TinyOS needs information about Network IP to be accessed and the Port number for acquiring data from the sensor node. The base station mote is then connected to the MIB 600. Listen Tool is a program which prints the raw data of each packet received from the serial port. To receive data on the PC the following steps were taken, the prerequisite being installation of Java and the javax.comm package are to firstly, connect the basestation mote with the MIB600. Then cd to the following directory

/opt/TinyOS-2.1.0/support/sdk/java directory

There, type

make export MOTECOM=network@<IP address>:<port number>

The environment variable MOTECOM tells the java Listen tool which packets it should listen to. Here “network@<IP address>:<port number>” says to listen to a mote connected to a network with the mentioned socket address. The command used was:

export MOTECOM=network@169.254.185.41:10002

Then set MOTECOM appropriately, then run

javanet.TinyOS.tools.Listen

Packets received will be displayed on the monitor, as shown in Figure 21.

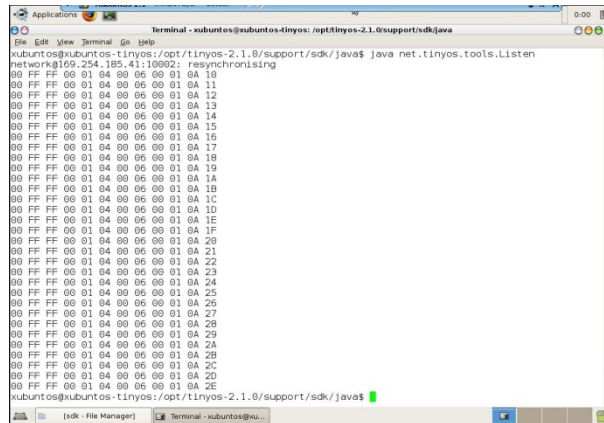


Figure 4.5 Packets Received

4.7 Creating A TCP/IP Server

Serial forwarder is used to create a TCP server. The purpose of this server is to access the data coming from MIB 600 board in raw form, and forward the data on specified port of server. In this way, multiple clients can access that server and get the data in any application.

The SerialForwarder program is used to read packet data from a serial port and forward it over an Internet connection, so that other programs can be written to communicate with the sensor network over the Internet.

SerialForwarder does not display the packet data itself, but rather updates the packet counters in the lower-right hand corner of the window. Once running, the serial forwarder listens for network client connections on a given TCP port, and simply forwards TinyOS messages from the serial port to the network client connection, and vice versa.[24]

The steps to establish a TCP server are to

`cd to /opt/TinyOS-2.1.0/support/sdk/java`

then type the following command

javanet.TinyOS.sf.SerialForwarder -port 9001 -comm network@169.254.185.41:10002

This opened up the GUI window shown in Figure 22.

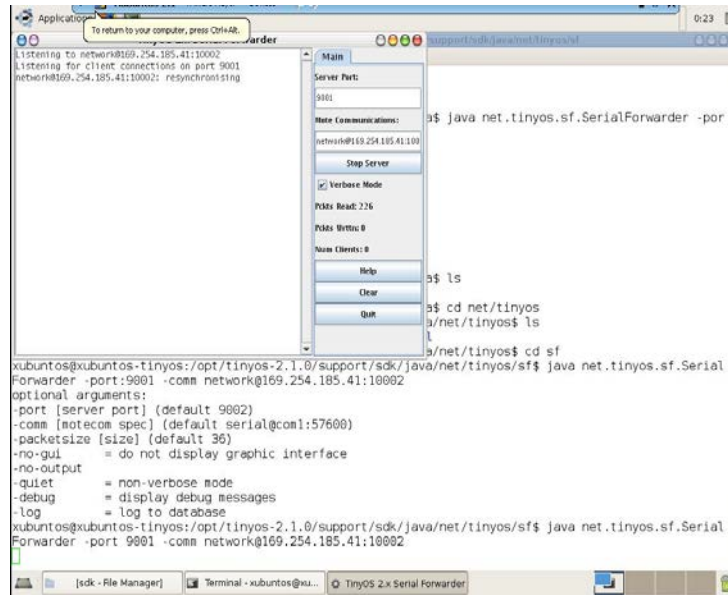


Figure 4.6 SerialFowarder Window

The `-comm` argument specifies where the packets SerialForwarder should forward come from. Unlike most other programs, the `-port` command specifies the incoming TCP port. The rest of the command is similar to the command used to define the MOTECOM in Listen tool [24].

4.8 Packet Structure

Each data packet that comes out of the mote contains several fields of data. The description of a BlinkToRadio packet is given in Table 1:

Table 4.1 Packet Structure (Cont'd)

Packet Structure (Hex)	Description
00	Indicates that it is an AM packet

FFFF	The destination address follows the00 identifier, in this case it is the broadcast address, (FFFF)hex
00 01	Source Address
04	Packet length (bytes)
00	Group ID
06	Defines the type of AM packet
0001	Node ID

4.9 Importing Data To The Application

The base station mote transmits the packets to MIB600 using its USART connection. The MIB600 transmits the data packets to PC through Ethernet. On the PC, MATLAB is used to receive the data packets.

A program was written in MATLAB, to carry out the tasks of creating a TCP client which will access the MIB600 on the assigned port number and IP, acquiring the packets received by the base station, removing escape/delimiter bytes in the packets, extracting the pixel values and their position in the image from each packet, detecting which (if any) of the packets are lost and which portion of the image it corresponds to and finally arranging and showing/saving the image

4.10 Data Rate Optimization

The packet length specified in the dummy program originally used to test the network was set at 17 bytes. The time taken for transmission of one packet was specified as 250msec.

To increase the data rate, the packet length was increased. In the file, Opt/TinyOS-2.1.0/tos/types/message.h, the packet length is defined by the variable TOS_DATA_LENGTH. It was increased from 28 bytes to 110 bytes.

For TinyOS, 37 bytes is the optimum packet size, 9 bytes are reserved for header and footer fields, 28 bytes are available for the message/payload. To increase the packet length, the default parameters need to be changed.

The new packet length becomes 119 bytes, the time period is kept as 250msec. These changes had to be implemented in both the programs, so the two motes were burned again. The buffer length in the MATLAB code was increased to 119 bytes.

Maximum packet length the CC2420 radio on the MICAz mote can handle is 128 bytes. The new received packet in MATLAB is shown in Figure 23.

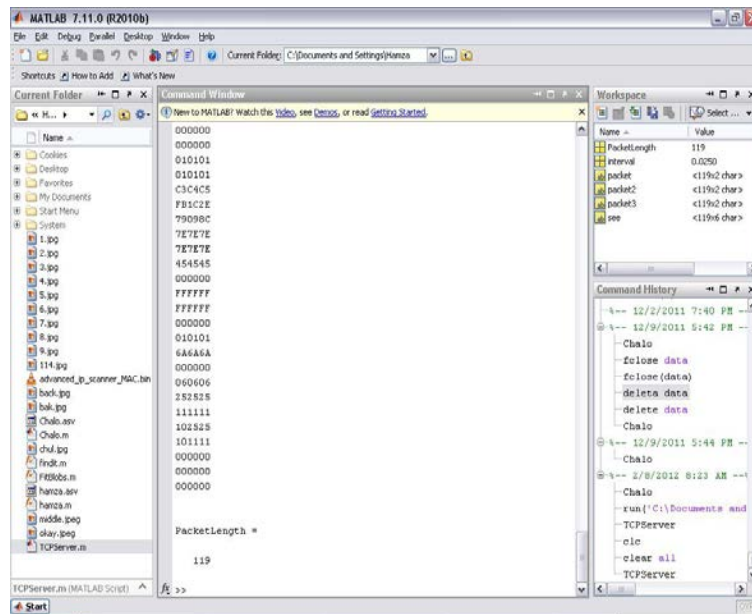


Figure 4.7 Increased Packet Length In MATLAB

To further enhance the data rate, the time taken to transmit a packet is decreased. 119 bytes are being transmitted in 250msec, the maximum data rate of the CC2420

radio is 250Kbps, so to reach the maximum data rate, the following calculation was used:

$$\text{New time interval} = 119 \text{ bytes} / (250,000/8) \text{ bytes per second} = 3.808 \text{ msec}$$

--- Eq No 2

The new time interval is set in `opt/TinyOS-2.1.0/tutorials/BlinkToRadio.h`, as 4msec. `TIMER_PERIOD_MILLI=4`. The BlinkToRadio mote was burned with the altered code. After using the new program, the Green LED on the base station mote starts to toggle, but immediately the Yellow LED starts to toggle as well. This means packets are being dropped. Packet dropping is shown in Figure 24.

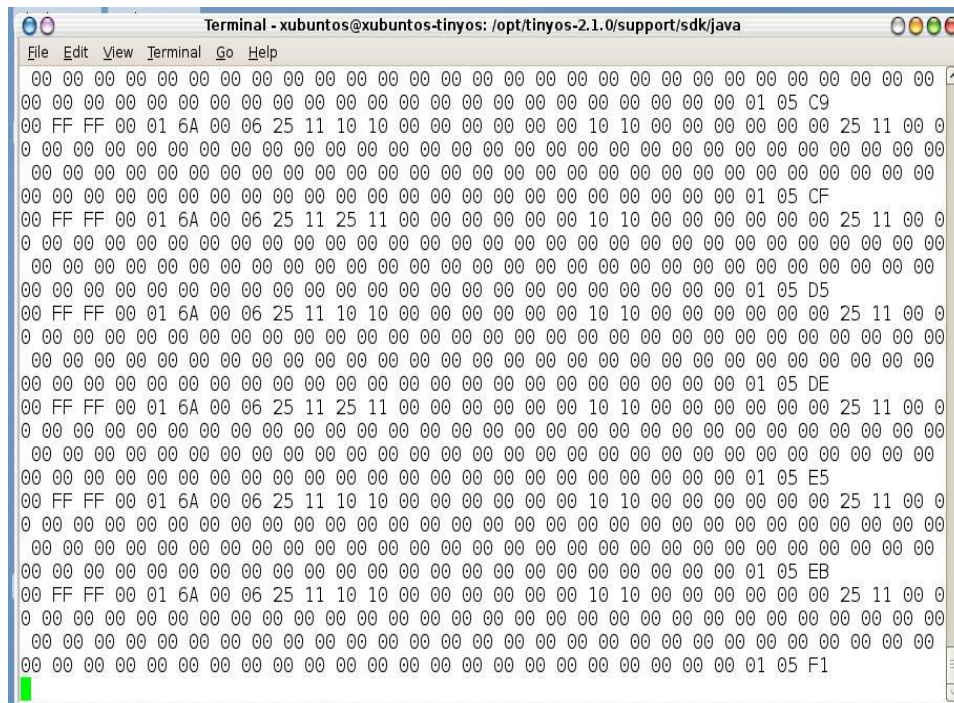


Figure 4.8 Dropped Packets, Counter Value Changes From EB To F1

In the file, `opt/TinyOS-2.1.0/apps/BaseStationP.nc`

```
UART_QUEUE_LENGTH=12;
```

```
UART_QUEUE_LEN=12;
```

UART_QUEUE_LENGTH was increased to 24 to decrease packet dropping, but the total number of packets of this length a MICAz can buffer is 24. So the UART_QUEUE_LEN was changed to 0. This delays the packet dropping on the UART interface.

In the file, opt/TinyOS-2.1.0/tos/platforms/micaz/hardware.h

PLATFORM_BAUDRATE was increased to 57600bps.

In the file, opt/TinyOS-2.1.0/tos/chips/Atm128UartP.c the following line was added:

```
else if (PLATFORM_BAUDRATE == 115200UL)
    m_byte_time = 34;
```

Maximum baudrate MIB600 can support is 115200 bits per second. Hence the MICAz baudrate was set to this value. This makes the MIB600 the bottleneck of the network. Baud rate of USART1 in Lantronix is also changed to 115200bps. The TIMER_PERIOD_MILLI in the BlinkToRadio header file was changed to 17msec. The data rate being received was around 8kBps.

4.11 Conclusion

By first deploying a sensor network and dummy edge sensor nodes which sent dummy data, the network was ready to acquire images from an edge sensor node and carry them to the sink.

IMAGE ACQUISITION

5.1 Introduction

The C3088 camera works above 8MHz. The MICAz works at 8MHz. Therefore the MICAz cannot be directly used to acquire images from the camera. ATmega644 was used at 16MHz to grab images from the camera. An image with a 176 x 144 resolution is a QCIF image. A black and white QCIF image is $176 \times 144 \times 1 = 25344$ bytes. The ATmega644 doesn't have enough on-board memory to store even a quarter of this image. Therefore an external memory is needed to buffer an image during its reception.

5.2 Programming The Microcontroller

C language is used to program the microcontroller and AVR Studio 5.0 is used as the programming tool. The program grabs the image from the camera, through I2C, and transfers this data to Flash memory, through SPI. It then receives this data from the Flash in a sequential manner, appends extra bytes to it and transfers it to the MICAz, through UART, for transmission. The code is attached in appendix D.

5.2.1 Interfacing With Camera C3088

The registers on the C3088 camera are programmable through the SCCB interface. This is a variant of the I2C serial protocol and can be implemented by the I2C hardware on the ATmega644. Three signals from the camera signify the beginning of

an frame, a line and a pixel and are used to keep the camera and image grabber in synchronization. The exact timing of the three signals is shown in Figure 25.

A high to low conversion of the VSYNC (Vertical Synchronization) signals the beginning of a new frame. A low to high conversion of the HREF (Horizontal Reference) signals the beginning of a new line of pixels. The PCLK or the Pixel Clock goes high each time a valid pixel is sent by the camera. After the pixel is transmitted, the PCLK goes low until a new pixel value is to be output.

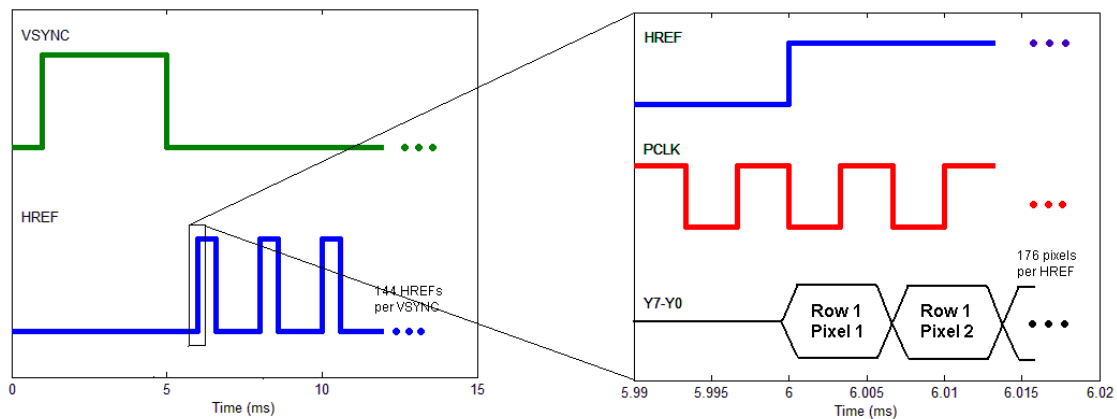


Figure 5.1 VSYNC And HREF Clock Signals [25]

The C3088 camera has two busses for data output; Y and UV. Since the luminescence data is output on the Y bus, only the Y bus is connected to the Atmega644 and this gives a black and white image.

5.2.2 Interfacing With Flash

The microcontroller has 64KBytes of Flash memory, 2KBytes of EEPROM and 4Kbytes of SRAM. The RAM is insufficient to store a 25Kbyte image. The camera outputs 144 lines of 176 pixels each. After every three lines of 176 pixels, there are 528 acquired bytes. In the time between 3rd and 4th line, the microcontroller sends

these 528 bytes to the flash buffer, this is shown in Figure 26, and this allows the microcontroller to keep up with the camera timing.

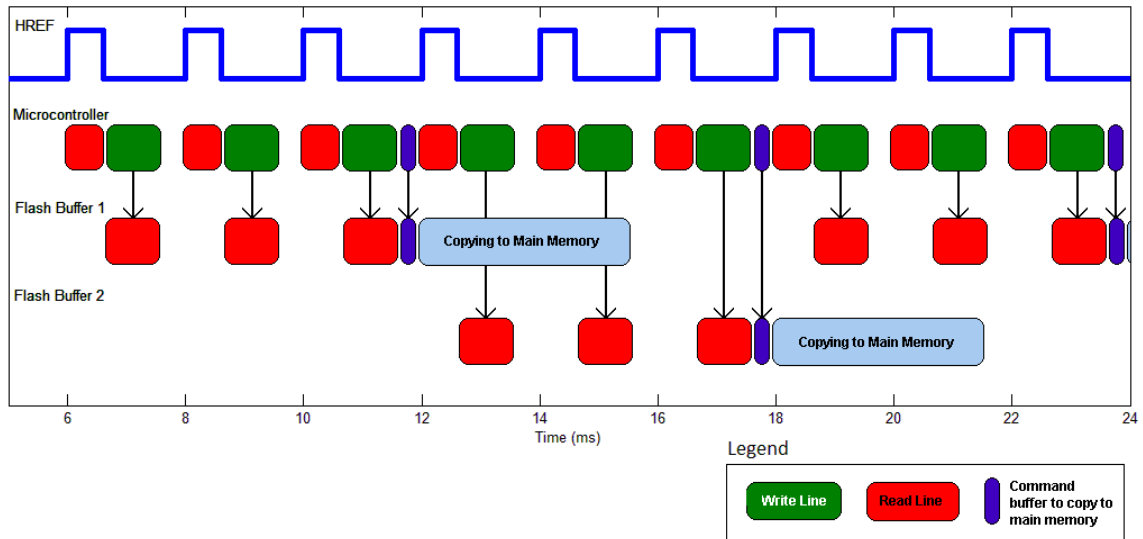


Figure 5.2 How The Microcontroller Writes To The Flash [25]

5.2.3 Interfacing With The MICAz Mote

The radio hardware on the MICAz can only transmit packets of 128 bytes length. This means the image needs to be fragmented before being fed to the MICAz mote for wireless transmission. Also, since the packet is encapsulated in header and footer fields before transmission, the payload has to be lesser than 120 bytes.

It was decided to comprise packets with 88 bytes of pixel values as the payload. Also 2 bytes were added to indicate the position of these pixel bytes in the image to allow the receiving end to correctly process and arrange the image in case of packet loss and out of order packet delivery.

5.2.4 Schematic Diagram

The schematic diagram illustrating the hardware connection of the microcontroller with the camera, flash and MICAz mote is depicted in Figure 27. The pin connections are then programmed on the microcontroller accordingly. The following circuit was implemented on a Vero board.

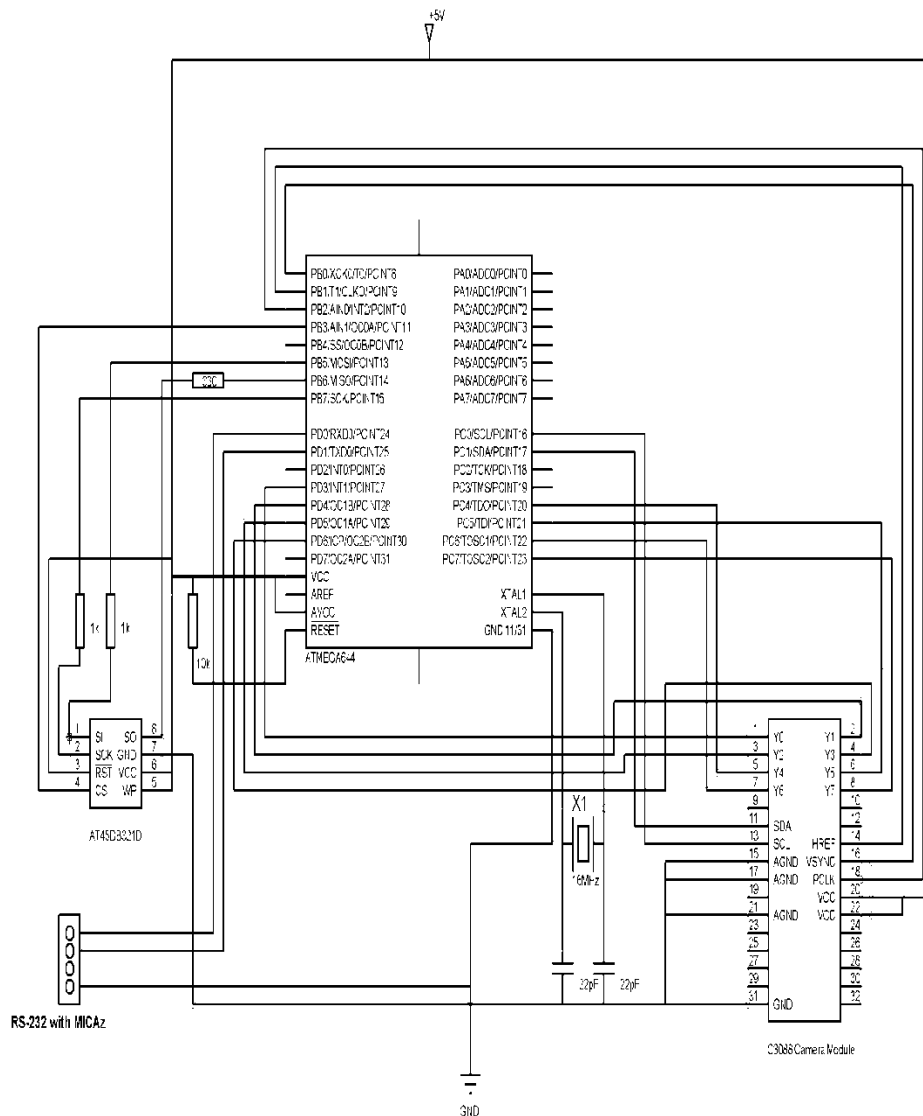


Figure 5.3 Schematic Diagram

5.3 Other Features

The ATmega644 is programmed not only to grab, save and transmit images but also to receive control packets from the base station computer and change camera control registers according to them. The features which a user can control from the base station computer are switching on/off of Automatic Gain Control and Automatic Exposure Control, setting 8 different reference values for Automatic Gain Control and Automatic Exposure Control, increase/decrease Contrast, increase/decrease Brightness, increase/decrease Gain, increase/decrease Exposure, switching on/off Automatic White Balance and resetting the camera.

The control messages between the base station and remote node are abstracted from the user through the GUI. There are 23 different types of control messages for the visual node from the Base Station. Any other format and length of message received is discarded. The format is depicted in Figure 28. The source addresses filtering is carried out at the data link layer of the MICAz mote on the visual node.

Synchronization byte is set to 0x61 and Reserved byte is set to 0x62 for no specific reason. The description field describes the response of the visual node when the message is received. The exact codes are attached in the Appendix E.



Figure 5.4 Control Message Structure From Base Station To Visual Sensor Node

5.4 Conclusion

The hardware was integrated together. Various communication protocols, UART, SPI and I2C were coded for and implemented. The microcontroller was appropriately coded. The control of the camera from the base station mote was an extra feature which enabled the user to control the camera features. Initially, dummy data was received from the microcontroller. Then after the integration of the camera, images were received.

PROCESSING IN MATLAB

6.1 Introduction

MATLAB receives the data from the network. For ease of use, a GUI is created. It carries out the functionalities such as opening and closing TCP/IP connections, receiving the raw data, extracting data from the packet and discarding headers, displaying the image, displaying network parameters such as network efficiency, separating the coding from the functionalities of the system, transmitting the calculated duration of the green signal back to the visual node and allowing user to control camera parameters.

The received data is displayed as images which can be processed according to the application for which the Wireless Multimedia Sensor Network is used.

For the presented application, traffic monitoring, the images consist of cars standing at the traffic signal. To count the number of cars in an image, the following methods are used.

6.2 Receiving Data's Packet Structure

Each QCIF image is fragmented into 288 packets of 88 pixel bytes each. The visual node appends 2 bytes corresponding to 1-288 sequence number of the block of pixel bytes. These 90 bytes form the payload of the messages sent by the MICAz mote to the base station. The packet structure is depicted in Figure 29.

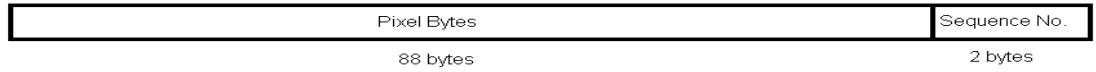


Figure 6.1 Packet Structure

A complete QCIF image is transferred in a burst of 88 such messages. The Sequence Number field is added to each of the 88 messages which make up one image by the MICAz mote prior to transmission of the packet. It allows rearranging of bytes despite out-of-order delivery and packet drops at the PC side. A complete image is depicted in Figure 30.

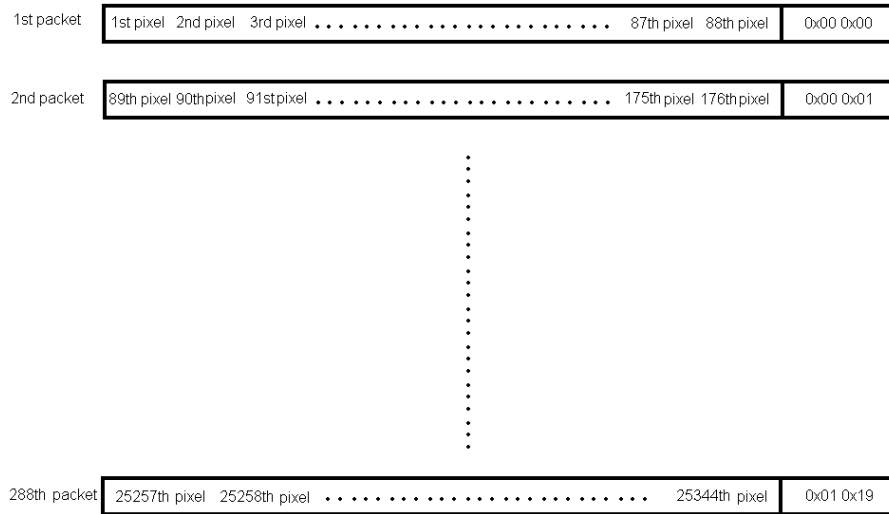


Figure 6.2 Array Of One Complete Image

6.3 Application Development

Over the last few years, there has been a tremendous increase in the road traffic density. The upshot of this is a significant loss of human resources, significantly increased inefficiencies, reduced quality of life, and a greater environmental impact at a critical time. A significant amount of time is spent waiting in traffic. A paramount amount of fuel is lost while waiting in traffic jams every day. The

monetary loss and the loss of man-hours affect individuals, families, companies and the government.

The first approach used to tackle was the much needed expansion of the major roads all over the country. However, with an economy like Pakistan's, it is impractical to expect to expand the whole road network. The second approach then employed was to set up an entire task force to manage and monitor it. It has its own drawbacks, latency and irregularity being the most complained about.

As these problems worsen, it is becoming clear that old solutions, such as increasing the number of lanes, building new roadways and enforcing traffic laws cannot solve the problem. Urban space is limited, and the resources needed to simply maintain the traffic network as roads and bridges age exceeds the monetary resources available at the local, regional and national level. A massive rebuilding of the traffic network to alleviate current congestion is an unfeasible solution. Technology usage to develop smart road traffic control however remains economical and employment of wireless networks to monitor and control traffic is the preferred choice owing to its convenient *en masse* deployment [26].

The traffic lights at road intersections are usually driven by timers. Each side of the road intersection is conventionally allowed a fixed and equal period to pass through. Such a scheme for traffic lights is inefficient whenever there are unequal number of cars on the sides of an intersection. This is worsened during rush hours when a particular road hosts much larger traffic but the traffic there is allowed to pass for only as long as the other sides with lesser traffic. Manual regulation provides a little

relief as it suffers from mismanagement, is prone to human errors and comes to a halt during harsh weather.

Also during periods of low traffic such as at night, experiences of waiting for the traffic light to go green even when there are no other cars at the intersection are common. Conventional traffic regulating systems cannot handle occasional congestion caused by unforeseen circumstances such as incidents, work zones and emergency rescue work, ambulances or weather conditions.

All this points to the need for a different scheme to operate traffic signals. This scheme needs to be dependent upon the relative traffic on each side of the road. Sensor Networks have been used only to monitor and measure road traffic. The aim is to use visual sensors and use it to not only measure traffic intensity, but also integrate it with a Wireless Sensor Network to gain a holistic view of the road conditions and thus make continuous, informed decisions on the priority of passage given to each road. To count the number of cars, two methods, of varying accuracy are used.

6.3.1 Background Subtraction Algorithm

It requires two images, one for background and the other is the test image which consists of the cars. A mask is applied onto the background image to eliminate the unwanted areas, such as the other side of a two way road, to avoid false detection of the cars or other objects in an unwanted region in the field of view of the camera.

Images are acquired, the background is masked and they are subtracted. The contrast of the resultant image is adjusted; it is binarized and thresholded to highlight the objects present in the test image. In some cases, when the cars are

standing at a very close distance to one another, they can be wrongly merged with one another in the binarized and thresholded image. To counter this, depending on the size and the distance of the merged body from the camera, the merged body was broken up into the appropriate number of cars. The other method used combats these problems in a different way.

6.3.2 Template Matching Algorithm

The windshields of the cars are chosen as the distinctive feature. Model images with the cars at a varying distance from the camera and in varying grouping were taken. From these images, the windshields of a few cars were chosen as templates. A mask is applied onto the test images to eliminate the unwanted areas, such as the other side of a two way road, to avoid false detection of the cars or other objects in an unwanted region in the field of view of the camera. As before, the image is binarized and thresholded.

The windshields of the cars were matched to identify the cars. This resulted in a greater accuracy of the application. The templates of the larger windshields were limited to matching with the cars nearer to the camera. In this way, the templates were limited to their area of applicability. This prevented false detection of labeling a merged body by single larger car.

6.3.3 Difference Between The Algorithms

The facts of the background subtraction which led to designing a new algorithm are that it requires another image, as the background. The background needs to be changed as the light intensity of the camera's surrounding changes. Background subtraction technique will detect everything present in the test image, even if it is

not a car as this technique mainly works due to the background subtraction method. It will detect and count everything present in the test image. This hinders deployment as the pedestrians and objects such as motorcycles and bicycles will be counted as cars.

This led to development of the technique, template matching, which does not require another image as the background. It only detects objects with which the template of the windshields can be matched. This removes false detection of pedestrians and other non car objects.

6.3.4 Results

The template matching method shows more accuracy. For background subtraction, the following results are obtained. The background subtraction detected seven cars out of the eight. Besides, the selection of the background and the task of keeping the light intensities of the background and the test images is difficult. The background images needs to be refreshed so very often. The results are shown in Figure 31.

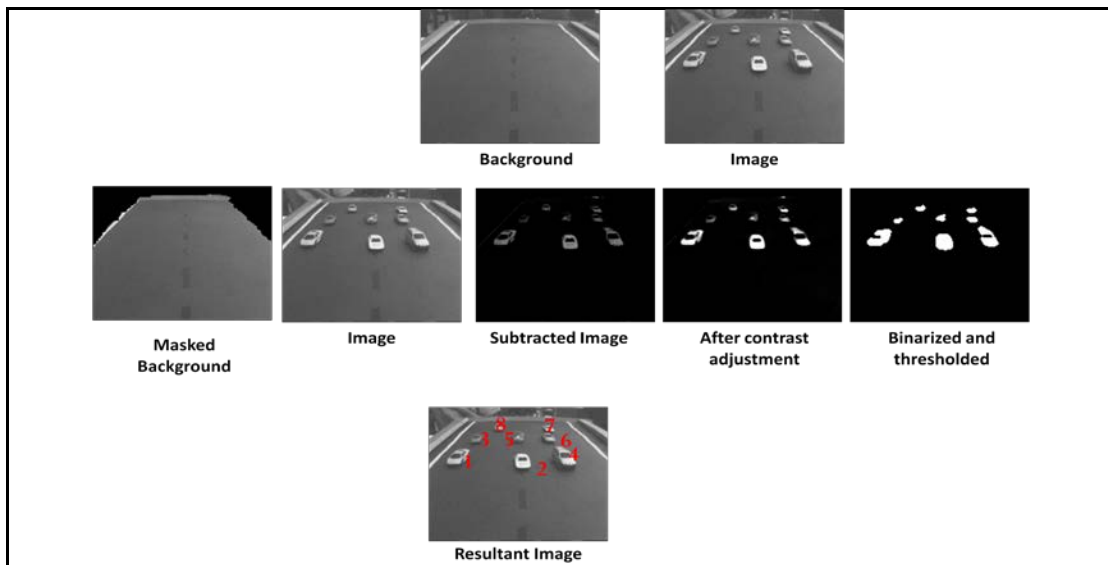


Figure 6.3 Background Subtraction Results

The template matching algorithm detected eight out of eight cars, obtaining a higher accuracy over the background subtraction method. The templates used and the results of this algorithm is shown in Figure 32. Normalized cross correlation between the test image and the templates are calculated in a sequential manner. If there is a match that area is removed from the binarized test image and is run for the same or another template until no match is found. A counter is incremented every time a match is found; this depicts the number of detected cars.

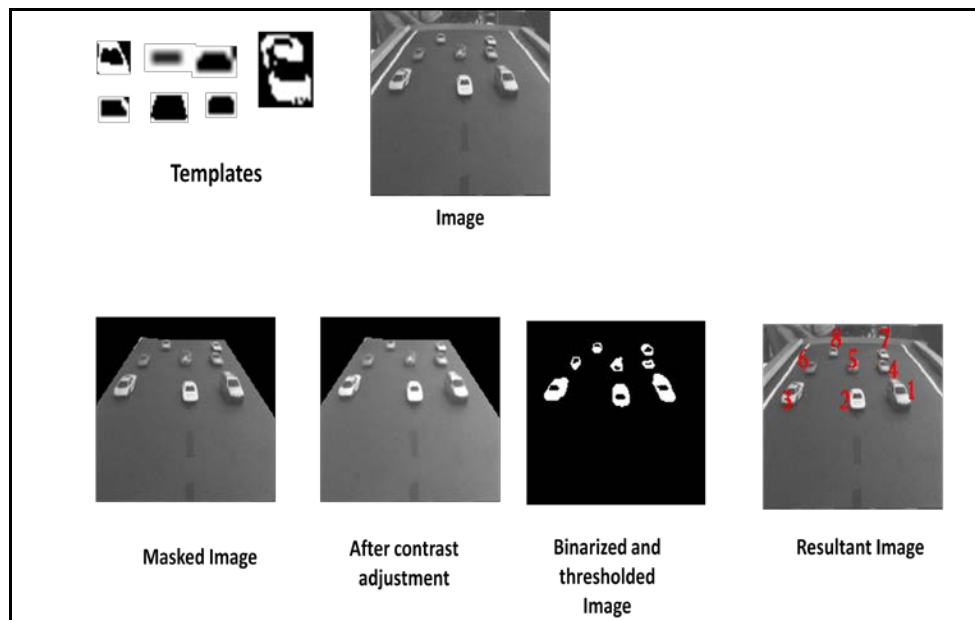


Figure 6.4 Template Matching Results

6.4 GUI

A user friendly GUI created is for ease of use. The layout of the GUI is shown in Figure 33. Separate buttons enable control of various functions for ease of understanding. The codes sent to control the camera are attached in appendix E. The code for the GUI and the processing being accomplished through GUI is attached in the appendix F.

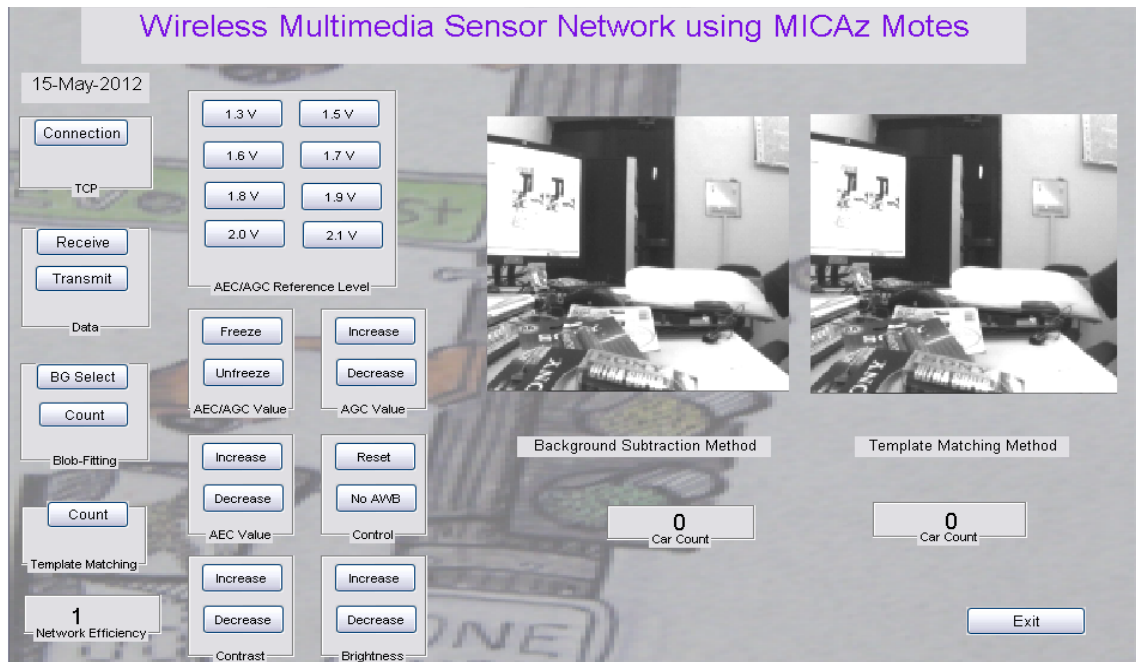


Figure 6.5 GUI

6.5 Conclusion

The data was received in MATLAB and consequently processed. The processing done to retrieve the data and for supporting the desired application was written in MATLAB. Two methods of processing the data for the application were developed, both showed different accuracies. The results have been thoroughly discussed. GUI was created for ease of use. Other applications can be developed in MATLAB depending on how the received images are intended to be used.

FUTURE WORK AND CONCLUSION

7.1 Introduction

The WMSN can be used for supporting many applications but first the project limitations are discussed to give an idea of the improvements which can be made to the architecture before deploying the system for other applications. Potential applications are briefly discussed.

7.2 Project Limitations

Communication for mote to mote communication requires Line Of Sight (LOS) but this problem can be worked around using relay motes. The camera exposure needs to be adjusted according to the brightness level of the surroundings. The whole process of image acquisition from the camera to the display of image on the PC results in a delay of 8 seconds, which can be reduced by using different techniques. The low resolution of the camera deteriorates number of successful detections per image. CoTS batteries enable around an hour and a half of battery time of the visual node.

7.3 WMSN Advantages And Applications

Advantages of WMSNs enable them to be used for such applications are mobility of the MICAz motes, inconspicuous deployment, scalable network and easy deployment. The sensor network is always-on, has low power consumption and power saving modes of operation. It has low cost, small size and there is no need of

knowing the position of the mote prior to the start of the application, allowing random deployment. Also, simple computations can be carried out in the mote's processor, instead of sending all the data to a remotely located central computing device, and the necessary information can be transmitted

Apart from the application of traffic monitoring and control, WMSN can be used for other applications as well, such as surveillance networks, advanced health care delivery, environmental monitoring, industrial process control, managing inventory, monitoring product quality and disaster areas. The application of WMSN is not limited to just these.

7.3.1 Surveillance

Surveillance can be used for public places, events, private properties, parking lots, remote areas, borders or battlefields [11]. It can locate and communicate the available parking spaces near any required area. Audio sensors can be used to further augment the system. WMSN can advance the surveillance technology by extracting non redundant information on the visual sensor node, decreasing the bulk of data required to be transmitted.

7.3.2 Environmental Monitoring

This can encompass monitoring of hazardous areas, animal habitats and monitoring buildings [27]. These areas can be remote and inaccessible, the energy efficient operations are mandatory to continue monitoring over a long period of time. Sometimes, the other sensors are integrated with the WMSN, the camera only gets activated when activity is detected by the other sensors. As an example, video

sensors are already being used by oceanographers to determine the evolution of sandbars [13].

7.3.3 Advanced Healthcare Delivery

Telemedicine sensor networks can be incorporated with multimedia sensor networks to provide health care [13]. Typical sensors include blood pressure, pulse oximetry, breathing activity and ECG. Advanced health care for the elderly can be incorporated by using motion sensors to monitor the activity of the patients.

7.3.4 Industrial Process Control

Information such as images, pressure or temperature can be gathered and processed to support industrial manufacturing process such as automobiles, food products and pharmaceuticals. Defects can be identified. This will make the visual inspection of processes easier and much more flexible. Round the clock service can be provided.

7.4 Testing And Validation

The testing for this setup was done in the lab to gain full control over the camera parameters. The visual sensor node was tested by placing it at variable distances from the base station mote in the lab. Once the pictures were successfully received at the lab computer, the picture reception quality and network efficiency was tested by placing the visual sensor node outside the lab. Relay mote was used in when the distance between the visual sensor node and the base station node exceeded a certain threshold. In such an arrangement, using the relay mote increased network efficiency. Next, for the car detection algorithms, the detection algorithm was made

more accurate by using template matching. The images were masked with a binary mask to decrease false detection. The set up was run for a model intersection with cars standing at the traffic signal.

7.5 Conclusion

The field of WSN has a lot of potential to further develop and become stable enough to support many applications. The purpose of the project was to establish transmission of multimedia information over a wireless sensor network. Traffic monitoring application was also presented to prove the potential of the project. All the objectives of the project are achieved.

APPENDIX A- nesC CODE FOR MICAz MOTE IN VISUAL SENSOR NODE

BlinkToRadio.h

```
#ifndef BLINKTORADIO_H
#define BLINKTORADIO_H
enum {
    AM_BLINKTORADIO = 6,
    TIMER_PERIOD_MILLI = 25,
};
typedef nx_struct BlinkToRadioMsg {
    nx_uint8_t buffer[90];
}
BlinkToRadioMsg;
#endif
```

BlinkToRadioC.nc

```
* Program edited to be used as an edge node at MCS, NUST for WMSN
* @author Hamna Anwar and Hamza Tariq Khan
* @date 28-05-2012
*/
#include <Timer.h>
#include "BlinkToRadio.h"

module BlinkToRadioC {
    uses interface Boot;
    uses interface Leds;
    uses interface Timer<TMilli> as Timer0;
    uses interface Packet;
    uses interface AMPacket;
    uses interface AMSend;
    uses interface SplitControl as AMControl;
    uses interface Receive;
    uses interface UartByte;
    uses interface StdControl;
    uses interface UartStream;
}
implementation {

    uint16_t length;
    uint8_t buffer[90];
    uint8_t temp[3];
    uint16_t counter;
    uint8_t i;
    message_t pkt;
    uint8_t alter = 0;
    char ack1[] = "abc";
    char str[] = "str\r\n";
    bool busy = FALSE;

    void setLeds(uint16_t val) {
        if (val & 0x01)
```



```

    call Leds.led0On();
else
    call Leds.led0Off();
if (val & 0x02)
    call Leds.led1On();
else
    call Leds.led1Off();
}

event void Boot.booted() {
    length = 90;
    call AMControl.start();
    call StdControl.start();
}

event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) { // call Timer0.startPeriodic(TIMER_PERIOD_MILLI); }
    else { call AMControl.start(); }
}

event void AMControl.stopDone(error_t err) {}

event void Timer0.fired() { }

event void AMSend.sendDone(message_t* msg, error_t err) {
    if (&pkt == msg) {busy = FALSE;}
}

async event void UartStream.sendDone(uint8_t *buf, uint16_t len,error_t err) {
    call UartStream.receive(buffer,length);
}
async event void UartStream.receiveDone(uint8_t *buf, uint16_t len, error_t err) {
    counter++;
    if (!busy) {
        BlinkToRadioMsg* btrpkt =
            (BlinkToRadioMsg*)(call Packet.getPayload(&pkt, sizeof(BlinkToRadioMsg)));
        if (btrpkt == NULL) {
            return;
        }
        for ( i = 0; i<length ; i++)
        {
            btrpkt->buffer[i] = buffer[i];
        }
        if (call AMSend.send(0x0001,
&pkt, sizeof(BlinkToRadioMsg)) == SUCCESS) {
            busy = TRUE;
        }
    }
    call UartStream.receive(buffer,length);
    call Leds.led1Toggle();
}

async event void UartStream.receivedByte(uint8_t byte) {
}

```

```

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
    call Leds.led0Toggle();
    if (len==3)
        call Leds.led2Toggle();
    call UartStream.send(payload,len);
    return msg;
}
}

```

BlinkToRadioAppC.nc

```

#include <Timer.h>
#include "BlinkToRadio.h"
configuration BlinkToRadioAppC {
}
implementation {
    components MainC;
    components LedsC;
    components BlinkToRadioC as App;
    components new TimerMilliC() as Timer0;
    components ActiveMessageC;
    components new AMSenderC(AM_BLINKTORADIO);
    components new AMReceiverC(AM_BLINKTORADIO);
    components PlatformSerialC;

    App.Boot -> MainC;
    App.Leds -> LedsC;
    App.Timer0 -> Timer0;
    App.Packet -> AMSenderC;
    App.AMPacket -> AMSenderC;
    App.AMControl -> ActiveMessageC;
    App.AMSend -> AMSenderC;
    App.Receive -> AMReceiverC;
    App.UartByte -> PlatformSerialC;
    App.UartStream -> PlatformSerialC;
    App.StdControl -> PlatformSerialC;}

```

APPENDIX B- nesC CODE FOR RELAY MICAz MOTE

RelayStationP.nc

```
// Adapted from basestationP by the University of California to be used as a radio relay
#include "AM.h"
#include "Serial.h"
module RelayStationP @safe() {
  uses {
    interface Boot;
    interface SplitControl as RadioControl;
    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Receive as RadioSnoop[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;
    interface Leds;
  }
  implementation
  {
    enum {
      RADIO_QUEUE_LEN = 23,
    };
    message_t radioQueueBufs[RADIO_QUEUE_LEN];
    message_t * ONE_NOK radioQueue[RADIO_QUEUE_LEN];
    uint8_t radioIn, radioOut;
    bool radioBusy, radioFull;
    task void radioSendTask();
    void dropBlink() {
      call Leds.led2Toggle();
    }
    void failBlink() {
      call Leds.led2Toggle();
    }
    event void Boot.booted() {
      uint8_t i;
      for (i = 0; i < RADIO_QUEUE_LEN; i++)
        radioQueue[i] = &radioQueueBufs[i];
      radioIn = radioOut = 0;
      radioBusy = FALSE;
      radioFull = TRUE;
      call RadioControl.start();
    }
    event void RadioControl.startDone(error_t error) {
      if (error == SUCCESS) {
        radioFull = FALSE;
      }
    }
    event void RadioControl.stopDone(error_t error) {}
    uint8_t count = 0;
    message_t* ONE receive(message_t* ONE msg, void* payload, uint8_t len);
    event message_t *RadioSnoop.receive[am_id_t id](message_t *msg, void *payload, uint8_t len) {
```

```

    // return receive(msg, payload, len);
}
event message_t *RadioReceive.receive[am_id_t id](message_t *msg, void *payload, uint8_t len) {
    message_t *ret = msg;
    bool reflectToken = FALSE;
    atomic
    if (!radioFull)
    {
        reflectToken = TRUE;
        ret = radioQueue[radioIn];
        radioQueue[radioIn] = msg;
        if (++radioIn >= RADIO_QUEUE_LEN)
            radioIn = 0;
        if (radioIn == radioOut)
            radioFull = TRUE;
        if (!radioBusy)
            { post radioSendTask();
              radioBusy = TRUE; }
    }
    else dropBlink();
    if (reflectToken) { //call UartTokenReceive.ReflectToken(Token);}
    return ret;
}
}
task void radioSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr, source;
    message_t* msg;
    atomic
    if (radioIn == radioOut && !radioFull)
        { radioBusy = FALSE;
        return;}
    msg = radioQueue[radioOut];
    len = call RadioPacket.payloadLength(msg);
    addr = call RadioAMPacket.destination(msg);
    source = call RadioAMPacket.source(msg);
    if (source==1)
        addr = addr + 1;
    else
        addr = addr - 1;
    id = 0x06;
    call RadioPacket.clear(msg);
    call RadioAMPacket.setSource(msg, source);

    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
        call Leds.led0Toggle();
    else
        { failBlink();
        post radioSendTask();}
}
}
event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
        failBlink();
    else
        atomic

```

```

    if (msg == radioQueue[radioOut])
    {
        if (++radioOut >= RADIO_QUEUE_LEN)
            radioOut = 0;
        if (radioFull)
            radioFull = FALSE;
    }

    post radioSendTask();
}
}

```

RelayStationC.nc

```

configuration RelayStationC {
}
implementation {
    components MainC, RelayStationP, LedsC;
    components ActiveMessageC as Radio;
    MainC.Boot <- RelayStationP;
    RelayStationP.RadioControl -> Radio;
    RelayStationP.RadioSend -> Radio;
    RelayStationP.RadioReceive -> Radio.Receive;
    RelayStationP.RadioSnoop -> Radio.Snoop;
    RelayStationP.RadioPacket -> Radio;
    RelayStationP.RadioAMPacket -> Radio;
    RelayStationP.Leds -> LedsC;
}

```

APPENDIX C- nesC CODE FOR BASE STATION MICAz MOTE

BaseStationP.nc

```
/*
 * BaseStationP bridges packets between a serial channel and the radio.
 * Messages moving from serial to radio will be tagged with the group
 * ID compiled into the TOSBase, and messages moving from radio to
 * serial will be filtered by that same group id.
 * Modified at MCS, NUST to accept payload bytes from the serial link rather than serial packets
 */

#include "AM.h"
#include "Serial.h"

module BaseStationP @safe() {
  uses {
    interface Boot;
    interface SplitControl as SerialControl;
    interface SplitControl as RadioControl;
    interface AMSend as UartSend[am_id_t id];
    interface Packet as UartPacket;
    interface AMPacket as UartAMPacket;
    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Receive as RadioSnoop[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;
    interface StdControl;
    interface UartStream;

    interface Leds;
  }
}

implementation
{
  enum {
    UART_QUEUE_LEN = 22,
  };

  message_t uartQueueBufs[UART_QUEUE_LEN];
  message_t * ONE_NOK uartQueue[UART_QUEUE_LEN];
  uint8_t  uartIn, uartOut;
  bool    uartBusy, uartFull;
  uint8_t  radioIn, radioOut;
  uint8_t  buffer[3];
}
```

```

uint16_t length;
task void uartSendTask();

void dropBlink() {
    call Leds.led2Toggle();
}

void failBlink() {
    call Leds.led2Toggle();
}

event void Boot.booted() {
    uint8_t i;
    length = 3;

    for (i = 0; i < UART_QUEUE_LEN; i++)
        uartQueue[i] = &uartQueueBufs[i];
    uartIn = uartOut = 0;
    uartBusy = FALSE;
    uartFull = TRUE;

    call RadioControl.start();
    call SerialControl.start();
    call StdControl.start();
}

event void RadioControl.startDone(error_t error) {
    if (error != SUCCESS)
        call RadioControl.start();
    else
        call UartStream.receive(buffer,length);
}

event void SerialControl.startDone(error_t error) {
    if (error == SUCCESS) {
        uartFull = FALSE;
    }
}

event void SerialControl.stopDone(error_t error) {}
event void RadioControl.stopDone(error_t error) {}

uint8_t count = 0;

message_t* ONE receive(message_t* ONE msg, void* payload, uint8_t len);

event message_t *RadioSnoop.receive[am_id_t id](message_t *msg, void *payload,uint8_t len) {
    return receive(msg, payload, len);
}

event message_t *RadioReceive.receive[am_id_t id](message_t *msg, void *payload, uint8_t len) {
    return receive(msg, payload, len);
}

```

```

message_t* receive(message_t *msg, void *payload, uint8_t len) {
    message_t *ret = msg;

    atomic {
        if (!uartFull)
            {
                ret = uartQueue[uartIn];
                uartQueue[uartIn] = msg;
                uartIn = (uartIn + 1) % UART_QUEUE_LEN;
                if (uartIn == uartOut)
                    uartFull = TRUE;

                if (!uartBusy)
                    {
                        post uartSendTask();
                        uartBusy = TRUE;
                    }
            }
        else
            dropBlink();
    }

    return ret;
}

uint8_t tmpLen;

task void uartSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr, src;
    message_t* msg;
    atomic
        if (uartIn == uartOut && !uartFull)
            {
                uartBusy = FALSE;
                return;
            }

    msg = uartQueue[uartOut];
    tmpLen = len = call RadioPacket.payloadLength(msg);
    id = call RadioAMPacket.type(msg);
    addr = call RadioAMPacket.destination(msg);
    src = call RadioAMPacket.source(msg);
    call UartPacket.clear(msg);
    call UartAMPacket.setSource(msg, src);

    if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
        call Leds.led1Toggle();
    else
        {
            failBlink();
            post uartSendTask();
        }
}

```



```

}

event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
  if (error != SUCCESS)
    failBlink();
  else
    atomic
      if (msg == uartQueue[uartOut])
        {
          if (++uartOut >= UART_QUEUE_LEN)
            uartOut = 0;
          if (uartFull)
            uartFull = FALSE;
        }
    post uartSendTask();
}

event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
  if (error == SUCCESS)
    call Leds.led0Toggle();
  call UartStream.receive(buffer,length);
}

async event void UartStream.sendDone(uint8_t *buf, uint16_t len,error_t err) {
}

async event void UartStream.receiveDone(uint8_t *buf, uint16_t len, error_t err) {

  am_id_t ids;
  am_addr_t addr,source;
  message_t pkt;
  ids = 0x06;
  call RadioPacket.setPayloadLength(&pkt, len);
  memcpy(call RadioPacket.getPayload(&pkt, len), buf, len);
  addr = 0x0002;
  source = 0x0001;
  if (call RadioSend.send[ids](addr,&pkt, len) != SUCCESS)
    call Leds.led2Toggle();
}

async event void UartStream.receivedByte(uint8_t byte) {
}
}

```

BaseStationC.nc

```

configuration BaseStationC {
}
implementation {
  components MainC, BaseStationP, LedsC;
  components ActiveMessageC as Radio, SerialActiveMessageC as Serial;
  components PlatformSerialC;

  MainC.Boot <- BaseStationP;
  BaseStationP.RadioControl -> Radio;
}

```

```
BaseStationP.SerialControl -> Serial;  
BaseStationP.UartSend -> Serial;  
BaseStationP.UartPacket -> Serial;  
BaseStationP.UartAMPacket -> Serial;  
BaseStationP.RadioSend -> Radio;  
BaseStationP.RadioReceive -> Radio.Receive;  
BaseStationP.RadioSnoop -> Radio.Snoop;  
BaseStationP.RadioPacket -> Radio;  
BaseStationP.RadioAMPacket -> Radio;  
BaseStationP.UartStream -> PlatformSerialC;  
BaseStationP.StdControl -> PlatformSerialC;  
BaseStationP.Leds -> LedsC;  
}
```

APPENDIX D- C CODE FOR MICROCONTROLLER ATmega644 in AVR Studio 5.0

```
/*
Adapted from FaceAccess, final project for ECE 4760 under Professor Bruce Land at Cornell
University Electrical and Computer Engineering
*/

#define F_CPU 16000000UL

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>

volatile unsigned char PCLK;

#include "uart.h"
#include "uart.c"
#include "MatlabLib.h"
#include "MatlabLib.c"
#include "flashmem.h"
#include "flashmem.c"
#include "camlib.h"
#include "camlib.c"
#include "twimaster.c"

#define begin {
#define end   }

// #define CNTS_FLOP 100

FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

void initialize(void);

void transitionA(void);
void transitionB(void);

volatile unsigned int transCnt; //timer for debouncer
unsigned char led; //light states
unsigned char input;

// state machine constants

#define PageSize 528
#define FaceSize 64
#define ActualFace 48

char numFrames;
```

```

unsigned char curFace;
unsigned char* FrameRead;

//*****
//timer 0 compare ISR
ISR (TIMER0_COMPA_vect){
  //Decrement the times if it is not already zero
  if (transCnt>0) transCnt--;
}

//*****
//Reads the desired number of pages at face
void sendFacetoMatlab(unsigned char face){
  uint16_t junk = 0;
  unsigned char coon;
  FrameRead = (unsigned char*) malloc(PageSize);
  for (int i=0; i<ActualFace; i++)
  {
    int pageNum = (face*FaceSize) + i;
    readFlash(FrameRead, PageSize, pageNum, 0);
    for (int tea = 0; tea < PageSize; tea++)
    {
      coon = FrameRead[tea];
      printf("%c",coon);
      //if (((tea+1)%66) == 0)
      if (((tea+1)%88) == 0)
      {
        printf("%c",junk>>8);
        printf("%c",junk);
        junk++;
        _delay_ms(9.2);
      }
    }
    //dumpFrame(FrameRead, PageSize);
  }
  free(FrameRead);
}

//*****
//Transition to SEND items to MATLAB
void transitionB(){
  setUp(1, ActualFace);
  sendFacetoMatlab(curFace);
  CLRBIT(PORTA,5);
  curFace++;
}

//*****
//Transition to take a picture
void transitionA(){

```

```

        takePicture(curFace);
        while(isBusy());
        CLRBIT(PORTA,4);
    }

//*****
//Set it all up
void initialize(void)
{
    //set up the ports
    DDRA=0xff; // PORT A are LEDS on 1,2,3 and 4
    DDRB=0xa8;
    DDRC=0x00;
    DDRD=0x80; //PORT D.7 is LED

    /*
    //set up timer 0 for 1 mSec ticks
    TIMSK0 = 2; //turn on timer 0 cmp match ISR
    OCR0A = 250; //set the compare reg to 250 time ticks
    TCCR0A = 0b00000010; // turn on clear-on-match
    TCCR0B = 0b00000011; // clock prescalar to 64

    //init the task timer
    //transCnt = CNTS_FLOP;
    */

    unsigned char f;
    f = 3;
    while(f != 0)
    {f = camera_init();}
    /*for (int ko = 0; ko<150; ko++)
    {_delay_ms(500);
    SETBIT(PORTA,7);
    _delay_ms(500);
    CLRBIT(PORTA,7);}

    camera_write(0x28,0x11); */

    spi_init();

    //init the UART -- uart_init() is in uart.c
    uart_init();
    stdout = stdin = stderr = &uart_str;

    //crank up the ISRs
    //sei();

    //printf("Starting");

    // Show initiation

```

```

while(isBusy());
chipErase();
while(isBusy());

// if (f==0)
//      CLRBIT(PORTA,0);
curFace = 5;
}

/*int main(void){
    initialize();
    int8_t buffer[] = "abc/0";
    char junk1;
    while(1){
        for (int i = 0; i<3; i++)
        {
            loop_until_bit_is_set(UCSR0A, RXC0);
            if ( (buffer[i] != UDR0) )
                break;
            else
                if(i ==2)
                    {

                SETBIT(PORTA,7);SETBIT(PORTA,6);SETBIT(PORTA,5);SETBIT(PORTA,4);
                    while(isBusy());
                    eraseFace(curFace);
                    while(isBusy());
                    takePicture(curFace);
                    while(isBusy());
                    sendFacetoMatlab(curFace);

                CLRBIT(PORTA,7);CLRBIT(PORTA,6);CLRBIT(PORTA,5);CLRBIT(PORTA,4);
                    }
                }
        }
    }
}

int main(void){
    initialize();
    unsigned char f;
    int8_t buffer[] = "abc/0";
    char junk1;
    while(1){
        for (int i = 0; i<3; i++)
        {
            loop_until_bit_is_set(UCSR0A, RXC0);
            if (i ==2)
            {

```

```

switch (UDR0)
{
case 99: // take pictures
    SETBIT(PORTA,7);
    while(isBusy());
    eraseFace(curFace);
    while(isBusy());
    takePicture(curFace);
    while(isBusy());
    sendFacetoMatlab(curFace);
    CLRBIT(PORTA,7);
    break;
case 101: // green off red on
    SETBIT(PORTA,1);SETBIT(PORTA,2);SETBIT(PORTA,4);
    CLRBIT(PORTA,0);CLRBIT(PORTA,3);CLRBIT(PORTA,5);
    break;
case 102: // green on red off
    SETBIT(PORTA,0);SETBIT(PORTA,3);SETBIT(PORTA,5);
    CLRBIT(PORTA,1);CLRBIT(PORTA,2);CLRBIT(PORTA,4);
    break;
case 103:
    camera_write(0x4E,0x00); //
    break;
case 104:
    camera_write(0x4E,0x20); //
    break;
case 105:
    camera_write(0x4E,0x40); //
    break;
case 106:
    camera_write(0x4E,0x60); //
    break;
case 107:
    camera_write(0x4E,0x80); //
    break;
case 108:
    camera_write(0x4E,0xA0); //
    break;
case 109:
    camera_write(0x4E,0xC0); //
    break;
case 110:
    camera_write(0x4E,0xF0); //
    break;
case 111:
    camera_write(0x28,0x01); //
    break;
case 112:
    camera_write(0x28,0x11); //
    break;
case 113:
    f = camera_write(0x12,0x80); // soft reset
    if (f) return f;
    _delay_ms(2);
}

```

```

(176x144) f = camera_write(0x14,0x20); // QCIF frame size

if (f) return f;
_delay_ms(2);
f = camera_write(0x12,0x24); // (default setting) AGC

enable, YCrCb mode, no AWB

if (f) return f;
_delay_ms(2);
f = camera_write(0x13,0x01); // (default setting) 16 bit

format (see datasheet)

if (f) return f;
_delay_ms(2);
f = camera_write(0x11,(0x1C)<<1); // reduce fps
if (f) return f;
_delay_ms(2);
f = camera_write(0x29,0x00); // set to camera master

mode

if (f) return f;
_delay_ms(2);
f = camera_write(0x38,0x89); //89
if (f) return f;
_delay_ms(2);
f = camera_write(0x4E,0x60); //
if (f) return f;
_delay_ms(2);
f = camera_write(0x20,0x04); //
if (f) return f;
_delay_ms(2);
break;
case 114:
junk1 = camera_read(0x05);
junk1 = junk1 + 60;
camera_write(0x05,junk1); //
break;
case 115:
junk1 = camera_read(0x05);
junk1 = junk1 - 60;
camera_write(0x05,junk1); //
break;
case 116:
junk1 = camera_read(0x06);
junk1 = junk1 + 15;
camera_write(0x06,junk1); //
break;
case 117:
junk1 = camera_read(0x06);
junk1 = junk1 - 15;
camera_write(0x06,junk1); //
break;
case 118:
junk1 = camera_read(0x00);
junk1 = junk1 + 5;
camera_write(0x00,junk1); //
break;
case 119:

```



```

        junk1 = camera_read(0x00);
        junk1 = junk1 - 5;
        camera_write(0x00,junk1); //
        break;
    case 120:
        junk1 = camera_read(0x10);
        junk1 = junk1 + 50;
        camera_write(0x10,junk1); //
        break;
    case 121:
        junk1 = camera_read(0x10);
        junk1 = junk1 - 50;
        camera_write(0x10,junk1); //
        break;
    case 122:
        camera_write(0x12,0x20); //
        break;

        default:
            break;
    }
}
if ( (buffer[i] != UDR0) && i<2)
    break;
}
}

```

APPENDIX E- MESSAGES FROM BASE STATION MOTE TO VISUAL SENSOR NODE

Table E.1 Control Messages

Serial Number	Message Byte Value	Description
1	0x63	Initiates image capture and transfer
2	0x65	Switches on the green lights and shuts down the red
3	0x66	Switches on the red lights and shuts down the green
4	0x67	Sets AEC/AGC voltage reference value to 1.3v
5	0x68	Sets AEC/AGC voltage reference value to 1.5v
6	0x69	Sets AEC/AGC voltage reference value to 1.6v
7	0x6A	Sets AEC/AGC voltage reference value to 1.7v
8	0x6B	Sets AEC/AGC voltage reference value to 1.8v
9	0x6C	Sets AEC/AGC voltage reference value to 1.9v
10	0x6D	Sets AEC/AGC voltage reference value to 2.0v
11	0x6E	Sets AEC/AGC voltage reference value to 2.1v
12	0x6F	Stop AEC and AGC
13	0x70	Restart AEC and AGC
14	0x71	Reset the camera
15	0x72	Increase contrast
16	0x73	Decrease contrast
17	0x74	Increase Brightness
18	0x75	Decrease Brightness
19	0x76	Increase gain
20	0x77	Decrease gain
21	0x78	Increase exposure
22	0x79	Decrease Exposure
23	0x7A	Stop Automatic White Balance

APPENDIX F- MATLAB GUI CODE

```
function varargout = WMSN_final(varargin)
%WMSN_FINAL M-file for WMSN_final.fig
%   WMSN_FINAL, by itself, creates a new WMSN_FINAL or raises the existing
%   singleton*.
%
%   H = WMSN_FINAL returns the handle to a new WMSN_FINAL or the handle to
%   the existing singleton*.
%
%   WMSN_FINAL('Property','Value',...) creates a new WMSN_FINAL using the
%   given property value pairs. Unrecognized properties are passed via
%   varargin to WMSN_final_OpeningFcn. This calling syntax produces a
%   warning when there is an existing singleton*.
%
%   WMSN_FINAL('CALLBACK') and WMSN_FINAL('CALLBACK',hObject,...) call the
%   local function named CALLBACK in WMSN_FINAL.M with the given input
%   arguments.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WMSN_final

% Last Modified by GUIDE v2.5 16-May-2012 08:57:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @WMSN_final_OpeningFcn, ...
                  'gui_OutputFcn', @WMSN_final_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMSN_final is made visible.
function WMSN_final_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
```

```

% handles  structure with handles and user data (see GUIDATA)
% varargin unrecognized PropertyName/PropertyValue pairs from the
%         command line (see VARARGIN)
c=date;
set(handles.date,'String',c);
guidata(hObject, handles);
set(hObject, 'toolbar', 'figure');

% Choose default command line output for WMSN_final
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

ha=axes('units','normalized',...
        'position',[0 0 1 1]);
uistack(ha,'bottom');
I=imread('4.jpg');
hi=imagesc(I)
colormap hsv
set(ha,'handlevisibility','off',...
    'visible','off')
set(hi,'alphadata',0.5)

% UIWAIT makes WMSN_final wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global folder;
global paclen;
global pixelbytes;
global relpix1;
global relpix2;
global relpos1;
global relpos2;
global ext;
global image1;
global data;
global r;
global imge;
global background;
folder = 'WMSN/';
ext='.bmp';
paclen = 103;
pixelbytes = 88;
relpix1 = 10;
relpix2 = 97;
relpos1 = 98;
relpos2 = 99;
image1 = zeros(1,25344,'uint8');

% --- Outputs from this function are returned to the command line.
function varargout = WMSN_final_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

% handles  structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in exitbutton.
function exitbutton_Callback(hObject, eventdata, handles)
% hObject  handle to exitbutton (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles  structure with handles and user data (see GUIDATA)
close(gcf)

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject  handle to togglebutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles  structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1
button_state = get(hObject,'Value');
global data;
global r;
if (button_state == get(hObject,'Max'))
    data = tcpip('169.254.185.41','remoteport',10002,'localport',9090);
    set(data, 'InputBufferSize',33000,'Timeout',0.1,'OutputBufferSize',4)

    pause(.1)
    fopen(data)
elseif (button_state == get(hObject,'Min'))

    fclose(data)
    delete(data)
    clear data

end

% --- Executes on button press in background_button.
function background_button_Callback(hObject, eventdata, handles)
% hObject  handle to background_button (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles  structure with handles and user data (see GUIDATA)
global imge;
global background;
background=imge;

% --- Executes on button press in image_processing.
function image_processing_Callback(hObject, eventdata, handles)
% hObject  handle to image_processing (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles  structure with handles and user data (see GUIDATA)
global background;

```

```

global imge;
global binaryImage;

set(handles.cars_background,'String',0);
%cla(handles.cars_background,'reset')
%cla(handles.golmol,'reset')
guidata(hObject,handles)
axes(handles.golmol);
grid off;
drawnow
imshow(imge,[16 240]);
axis off;
guidata(hObject, handles);
axis square;
binarymask = imread('mask_final.bmp');
b=imge;
b(~binarymask)=0;
originalImage=b;

background(~binarymask)=0;

binaryImage=imsubtract(originalImage,background);

binaryImage=imadjust(binaryImage);
binaryImage=medfilt2(binaryImage);

level=graythresh(binaryImage);
binaryImage = im2bw(binaryImage,0.39); %hamza changed it from 0.11
binaryImage = imfill(binaryImage,'holes');
binaryImage = bwareaopen(binaryImage,10);
binaryImage=imclose(binaryImage,strel('rectangle',[1 5]));
binaryImage=imerode(binaryImage,strel('rectangle',[2 9])); %hamza changed it from [2 7]
binaryImage=imdilate(binaryImage,strel('disk',1));
s=binaryImage;

a=regionprops(s,'BoundingBox');
box=cat(1,a.BoundingBox);
[m n]=size(box);
minimum=min(box(:,2));

for i=1:1:m
    p(1,i)=box(i,3)*box(i,4);
    p(2,i)=((box(i,2) + (box(i,4)/2))^3.1)/10000;
    %p(2,i)=((box(i,2)-80)^2)/100;
    %p(2,i)=log(log(log(box(i,2)^2)));
    p(3,i)=p(1,i)/p(2,i);
end
dent=[box p']
[len b]=size(dent);
for i=1:1:len

    if dent(i,b)<0.8
        [ab cd]=size(s(ceil(dent(i,2)) : ceil(dent(i,2))+ceil(dent(i,4)-1), ceil(dent(i,1)) :
        ceil(dent(i,1))+ceil(dent(i,3)-1)));
    end
end

```

```

        s(ceil(dent(i,2)) : ceil(dent(i,2))+ceil(dent(i,4)-1), ceil(dent(i,1)) : ceil(dent(i,1))+ceil(dent(i,3)-
1))=zeros(ab,cd);
    end
end

hold on;
labeledImage = bwlabel(s,4); % Label each blob so measurements can be made

% Get all the blob properties. Can only pass in originalImage in version R2008a and later.
blobMeasurements = regionprops(labeledImage, originalImage, 'all');
numberOfBlobs = size(blobMeasurements, 1);

% bwboundaries() returns a cell array, where each cell contains the row/column coordinates for an
object in the image.
% Plot the borders of all the coins on the original grayscale image using the coordinates returned by
bwboundaries.

hold on;

boundaries = bwboundaries(s);
numberOfBoundaries = size(boundaries);
for k = 1 : numberOfBoundaries
    thisBoundary = boundaries{k};
    % plot(thisBoundary(:,2), thisBoundary(:,1),'g', 'LineWidth', 2);
end
guidata(hObject, handles.golmol);
fontSize = 14; % Used to control size of "blob number" labels put atop the image.
labelShiftX = -7; % Used to align the labels in the centers of the coins.
blobECD = zeros(1, numberOfBlobs);
hold off;
for k = 1 : numberOfBlobs % Loop through all blobs.
    % Find the mean of each blob. (R2008a has a better way where you can pass the original
image
% directly into regionprops. The way below works for all versions including earlier
versions.)
    thisBlobsPixels = blobMeasurements(k).PixelIdxList; % Get list of pixels in current blob.
    meanGL = mean(originalImage(thisBlobsPixels)); % Find mean intensity (in original image!)
    meanGL2008a = blobMeasurements(k).MeanIntensity; % Mean again, but only for version
>= R2008a

    blobArea = blobMeasurements(k).Area; % Get area.
    blobPerimeter = blobMeasurements(k).Perimeter; % Get perimeter.
    blobCentroid = blobMeasurements(k).Centroid; % Get centroid.
    blobECD(k) = sqrt(4 * blobArea / pi); % Compute ECD
- Equivalent Circular Diameter.
% % Put the "blob number" labels on the "boundaries" grayscale image.
% text(blobCentroid(1) + labelShiftX -3, blobCentroid(2)-7, num2str(k), 'FontSize', fontSize,
'FontWeight', 'Bold', 'Color',[.6,.3,.4]);

end
hold off;
set(handles.cars_background,'String',numberOfBlobs);
guidata(hObject,handles);

% --- Executes on button press in receivedata.

```

```

function receivedata_Callback(hObject, eventdata, handles)
% hObject  handle to receivedata (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles  structure with handles and user data (see GUIDATA)
global folder;
global paclen;
global ext;
global image1;
global pixelbytes;
global relpix1;
global relpix2;
global relpos1;
global relpos2;
global imge;
global background;
folder = 'WMSN/';
ext='.bmp';
paclen = 103;
pixelbytes = 88;
relpix1 = 10;
relpix2 = 97;
relpos1 = 98;
relpos2 = 99;
image1 = zeros(1,25344,'uint8');
data = instrfind;
p = [];
pos = 0;
count = 0;
countpack = 1;
relpix1;
%% receive packets
flushinput(data)
pause(0.1)
flushinput(data)
fprintf(data,'%s','abcd')
pause(.1)
fprintf(data,'%s','abcd')
pause(.1)
fprintf(data,'%s','abcd')
pause(.1)
pause(2.8)
while(get(data,'BytesAvailable')==0)
    fprintf(data,'%s','abcd')
    pause(.1)
    fprintf(data,'%s','abcd')
    pause(.1)
    fprintf(data,'%s','abcd')
    pause(.1)
    pause(2.8)
end
pause(5.1)
packet1 = fread(data);
flushinput(data)
%% count number of escape/delimiter byte instances
for i = 1:length(packet1) - 1

```



```

    if (packet1(i)==125 && (packet1(i+1)==94 || packet1(i+1)==93))
        count = count + 1;
    end
end
%% remove escape and delimiter bytes and parse packets
packet = packet1;
for i = 1:length(packet) - count
    if (packet(i)==125 && packet(i+1)==94)
        packet(i) = 126;
        packet = [packet(1,1:i) packet(1,i+2:end)];
    elseif (packet(i)==125 && packet(i+1)==93)
        packet(i) = 125;
        packet = [packet(1,1:i) packet(1,i+2:end)];
    end
end

%% count number of packets
for i = 1:length(packet) - 6
    if (packet(i)==126 && packet(i+1)==126 && packet(i+2)==69 && packet(i+3)==0 &&
        packet(i+4)==0)
        countpack = countpack + 1;
    end
end
packet2 = [packet 126];

%% remove any garbage/corrupted extra bytes within packets
for i = paclen:paclen:(countpack*paclen)
    while (packet2(i)~=126 || packet2(i+1)~=126)
        packet2 = [packet2(1,1:i-1) packet2(1,i+1:end)];
    end
end
%packet3 = packet2(1,1:(countpack*paclen))
packet3 = packet;
%% extract data bytes from the packet structure

for i = 1:paclen:(((countpack-1)*paclen)+1)
    pos = (packet3(i+relpos1)*256) + packet3(i+relpos2);
    if (pos<countpack)
        image1(1,(pos*pixelbytes)+1:(pos*pixelbytes)+pixelbytes) = packet3(1,i+relpix1:i+relpix2);
        p = [p pos];
    end
end

%% rearrange and save/display image
neteff = size(p,2)/288;
set(handles.inefficiency,'String',neteff);
guidata(hObject, handles);
imge = reshape(image1,176,144)';
axes(handles.golmol);
grid off;
drawnow
imshow(imge,[16 240]);
axis square;
axis off;

```

```

guidata(hObject, handles);
axes(handles.img2);
grid off;
drawnow
imshow(imge,[16 240]);
axis square;
axis off;
guidata(hObject, handles);

```

```

filename = datestr(now,'dd-mm-yyyy-HH-MM-SS');
filename = [folder filename ext];
imwrite(imge,filename,'bmp');

```

```

% --- Executes on button press in match_template.
function match_template_Callback(hObject, eventdata, handles)
% hObject handle to match_template (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global imge;
global sum;
data = instrfind;

```

```

set(handles.cars_template,'String',0);
%cla(handles.cars_template,'reset');
axes(handles.img2);
%imshow(imge,[16 240]);
binarymask = imread('mask_final.bmp');
b = imge; %problem with n/8 and n/9 and pics/21 and pics/22 and pics/13 pics/16

```

```

b(~binarymask)=0;

```

```

c=b;
c = imadjust(c);
level = 1.4 * graythresh(b); %2.55 %5.55 for n/10
d = im2bw(c,0.68);
s=d;

```

```

cars_lam = 0;
hud = 0;
a = imread('templates_final/ulu4.bmp');

```

```

cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
while ((max_cc > 0.5) && (hud < 50)) %0.52

```

```

corr = [ (ypeak-size(a,1)) (xpeak-size(a,2)) ];
try
e = c(corr(1)-8:corr(1)+size(a,1)+22,corr(2)-4:corr(2)+size(a,2)+6); %-8 +22 -13 +13
%figure,imshow(e); %+8 +22 -4 +4

```

```

d(corr(1)-8:corr(1)+size(a,1)+22,corr(2)-4:corr(2)+size(a,2)+6) = 0;
cars_lam = cars_lam + 1;

catch
if (corr(1)<0 && corr(2)>0)
d(1:size(a,1)-corr(1),corr(2)+3:corr(2)+size(a,2)) = 0;
end
if (corr(2)<0 && corr(1)>0)
d(corr(1):corr(1)+size(a,1),1:size(a,2)-corr(2)) = 0;
else
d(1:-corr(1)+size(a,1),1:-corr(2)+size(a,2)) = 0;
end
end
hud = hud + 1;
end
%f = b(corr(1)+105:corr(1)+size(a,2)+105,corr(2):corr(2)+size(a,1));

cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
end

cars1 = 0;
hud = 0;
a = imread('templates_final/ulu1.bmp');

cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
while ((max_cc > 0.36) && (hud < 50) && ypeak>75 && ypeak<144) %0.52

corr = [ (ypeak-size(a,1)) (xpeak-size(a,2)) ];
try
e = c(corr(1)-8:corr(1)+size(a,1)+22,corr(2)-13:corr(2)+size(a,2)+13); % -8 +22 -13 +13

d(corr(1)-8:corr(1)+size(a,1)+22,corr(2)-13:corr(2)+size(a,2)+13) = 0;
cars1 = cars1 + 1;

catch
if (corr(1)<0 && corr(2)>0)
d(1:size(a,1)-corr(1),corr(2)+3:corr(2)+size(a,2)) = 0;
end
if (corr(2)<0 && corr(1)>0)
d(corr(1):corr(1)+size(a,1),1:size(a,2)-corr(2)) = 0;
else
d(1:-corr(1)+size(a,1),1:-corr(2)+size(a,2)) = 0;
end
end
hud = hud + 1;
end
%f = b(corr(1)+105:corr(1)+size(a,2)+105,corr(2):corr(2)+size(a,1));

cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
end

```

```

hud = 0;
cars5 = 0;

a=imread('templates_final/ulu3.bmp');
cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
while (max_cc > .55 && hud <5 ) %0.52 or 0.55 %0.62 for n/11 %% 0.57 IS THE STANDARD

    corr = [ (ypeak-size(a,1)) (xpeak-size(a,2)) ];
    try
        e = c(corr(1)-3:corr(1)+size(a,1)+3,corr(2)-3:corr(2)+size(a,2)+3); %-6 +9 -2 +6
        %figure,imshow(e);
        d(corr(1)-3:corr(1)+size(a,1)+3,corr(2)-3:corr(2)+size(a,2)+3) = 0;
        cars5 = cars5 + 1;
    catch
        if (corr(1)<0 && corr(2)>0)
            d(1:size(a,1)-corr(1),corr(2)+3:corr(2)+size(a,2)) = 0;
        end
        if (corr(2)<0 && corr(1)>0)
            d(corr(1):corr(1)+size(a,1),1:size(a,2)-corr(2)) = 0;
        else
            d(1:-corr(1)+size(a,1),1:-corr(2)+size(a,2)) = 0;
        end
        hud = hud + 1;
    end
    %f = b(corr(1)+105:corr(1)+size(a,2)+105,corr(2):corr(2)+size(a,1));
    cc = normxcorr2(a,d);
    [max_cc, imax] = max(cc(:));
    [ypeak, xpeak] = ind2sub(size(cc),imax(1));
end

hud=0;
cars7=0;
a=imread('templates/misc-temp.bmp');
cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
while (max_cc > 0.65 && hud <5 ) %0.52 or 0.55 %0.62 for n/11 %% maybe: && ypeak>23 &&
ypeak<40

    corr = [ (ypeak-size(a,1)) (xpeak-size(a,2)) ];
    try
        e = c(corr(1)-2:corr(1)+size(a,1)+3,corr(2)-5:corr(2)+size(a,2)+3); %-6 +9 -2 +6
        %figure,imshow(e);
        d(corr(1)-2:corr(1)+size(a,1)+3,corr(2)-5:corr(2)+size(a,2)+3) = 0;
        cars7 = cars7 + 1;
    catch
        if (corr(1)<0 && corr(2)>0)
            d(1:size(a,1)-corr(1),corr(2)+3:corr(2)+size(a,2)) = 0;
        end
        if (corr(2)<0 && corr(1)>0)
            d(corr(1):corr(1)+size(a,1),1:size(a,2)-corr(2)) = 0;

```

```

else
    d(1:-corr(1)+size(a,1),1:-corr(2)+size(a,2)) = 0;
end
hud = hud + 1;
end
%f = b(corr(1)+105:corr(1)+size(a,2)+105,corr(2):corr(2)+size(a,1));
cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
end
%figure, imshow(d)

hud = 0;
randomcars = 0;
a=imread('templates_final/ulu2.bmp');
cc = normxcorr2(a,d);
[max_cc, imax] = max(cc(:));
[ypeak, xpeak] = ind2sub(size(cc),imax(1));
while (max_cc > 0.5 && hud < 5 && ypeak > 1 && ypeak < 60) %0.62 or 0.7 %0.85 for n/11 %0.79 for
pics

    corr = [ (ypeak-size(a,1)) (xpeak-size(a,2)) ];
    try
        e = c(corr(1)-4:corr(1)+size(a,1)+4,corr(2)-4:corr(2)+size(a,2)+4);
        %figure,imshow(e);
        d(corr(1)-4:corr(1)+size(a,1)+4,corr(2)-4:corr(2)+size(a,2)+4) = 0;
        %d(corr(1):corr(1)+size(a,1)-40,corr(2)+3:corr(2)+size(a,2)-40) = 0;
        randomcars = randomcars + 1;
    catch
        if (corr(1)<0 && corr(2)>0)
            d(1:size(a,1)-corr(1),corr(2)+3:corr(2)+size(a,2)) = 0;
        end
        if (corr(2)<0 && corr(1)>0)
            d(corr(1):corr(1)+size(a,1),1:size(a,2)-corr(2)) = 0;
        else
            d(1:-corr(1)+size(a,1),1:-corr(2)+size(a,2)) = 0;
        end
        hud = hud + 1;
    end
    %f = b(corr(1)+105:corr(1)+size(a,2)+105,corr(2):corr(2)+size(a,1));
    cc = normxcorr2(a,d);
    [max_cc, imax] = max(cc(:));
    [ypeak, xpeak] = ind2sub(size(cc),imax(1));
end

cars_lam
cars1
cars7
randomcars

axes(handles.imge2)
sum=cars_lam+cars1+cars5+randomcars+cars7;
grid off;
drawnow

```

```

imshow(имге,[16 240]);
axis square;
axis off;
guidata(hObject, handles.имге2);
set(handles.cars_template,'String',sum);
guidata(hObject,handles);
if (sum>=0 && sum<5)
    fprintf(data,'%s','abed')
    pause(.1)
    fprintf(data,'%s','abed')
    pause(.1)
    fprintf(data,'%s','abed')
    pause(.1)
end
if (sum>=4)
    fprintf(data,'%s','abfd')
    pause(.1)
    fprintf(data,'%s','abfd')
    pause(.1)
    fprintf(data,'%s','abfd')
    pause(.1)
end

% --- Executes on button press in transmit_data.
function transmit_data_Callback(hObject, eventdata, handles)
% hObject   handle to transmit_data (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
data = instrfind;
global sum;
if (sum>=0 && sum<6)
    fprintf(data,'%s','abed')
    pause(.1)
    fprintf(data,'%s','abed')
    pause(.1)
    fprintf(data,'%s','abed')
    pause(.1)
end
if (sum>=6)
    fprintf(data,'%s','abfd')
    pause(.1)
    fprintf(data,'%s','abfd')
    pause(.1)
    fprintf(data,'%s','abfd')
    pause(.1)
end

% --- Executes on button press in ib.
function ib_Callback(hObject, eventdata, handles)
% hObject   handle to ib (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abtd')

```

```

pause(.1)
fprintf(data,'%s','abtd')
pause(.1)
fprintf(data,'%s','abtd')
pause(.1)

% --- Executes on button press in db.
function db_Callback(hObject, eventdata, handles)
% hObject handle to db (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abud')
pause(.1)
fprintf(data,'%s','abud')
pause(.1)
fprintf(data,'%s','abud')
pause(.1)

% --- Executes on button press in ic.
function ic_Callback(hObject, eventdata, handles)
% hObject handle to ic (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abrd')
pause(.1)
fprintf(data,'%s','abrd')
pause(.1)
fprintf(data,'%s','abrd')
pause(.1)

% --- Executes on button press in dc.
function dc_Callback(hObject, eventdata, handles)
% hObject handle to dc (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','absd')
pause(.1)
fprintf(data,'%s','absd')
pause(.1)
fprintf(data,'%s','absd')
pause(.1)

% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject handle to reset (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abqd')
pause(.1)
fprintf(data,'%s','abqd')
pause(.1)

```

```

fprintf(data,'%s','abqd')
pause(.1)

% --- Executes on button press in noawb.
function noawb_Callback(hObject, eventdata, handles)
% hObject handle to noawb (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abzd')
pause(.1)
fprintf(data,'%s','abzd')
pause(.1)
fprintf(data,'%s','abzd')
pause(.1)

% --- Executes on button press in ie.
function ie_Callback(hObject, eventdata, handles)
% hObject handle to ie (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abxd')
pause(.1)
fprintf(data,'%s','abxd')
pause(.1)
fprintf(data,'%s','abxd')
pause(.1)

% --- Executes on button press in de.
function de_Callback(hObject, eventdata, handles)
% hObject handle to de (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abyd')
pause(.1)
fprintf(data,'%s','abyd')
pause(.1)
fprintf(data,'%s','abyd')
pause(.1)

% --- Executes on button press in ig.
function ig_Callback(hObject, eventdata, handles)
% hObject handle to ig (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abvd')
pause(.1)
fprintf(data,'%s','abvd')
pause(.1)
fprintf(data,'%s','abvd')
pause(.1)

```



```

% --- Executes on button press in dg.
function dg_Callback(hObject, eventdata, handles)
% hObject   handle to dg (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abwd')
    pause(.1)
    fprintf(data,'%s','abwd')
    pause(.1)
    fprintf(data,'%s','abwd')
    pause(.1)

% --- Executes on button press in freeze.
function freeze_Callback(hObject, eventdata, handles)
% hObject   handle to freeze (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abpd')
    pause(.1)
    fprintf(data,'%s','abpd')
    pause(.1)
    fprintf(data,'%s','abpd')
    pause(.1)

% --- Executes on button press in unfreeze.
function unfreeze_Callback(hObject, eventdata, handles)
% hObject   handle to unfreeze (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abod')
    pause(.1)
    fprintf(data,'%s','abod')
    pause(.1)
    fprintf(data,'%s','abod')
    pause(.1)

% --- Executes on button press in v13.
function v13_Callback(hObject, eventdata, handles)
% hObject   handle to v13 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abgd')
    pause(.1)
    fprintf(data,'%s','abgd')
    pause(.1)
    fprintf(data,'%s','abgd')
    pause(.1)

% --- Executes on button press in v18.
function v18_Callback(hObject, eventdata, handles)
% hObject   handle to v18 (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abkd')
pause(.1)
fprintf(data,'%s','abkd')
pause(.1)
fprintf(data,'%s','abkd')
pause(.1)

% --- Executes on button press in v19.
function v19_Callback(hObject, eventdata, handles)
% hObject handle to v19 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abld')
pause(.1)
fprintf(data,'%s','abld')
pause(.1)
fprintf(data,'%s','abld')
pause(.1)

% --- Executes on button press in v16.
function v16_Callback(hObject, eventdata, handles)
% hObject handle to v16 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abid')
pause(.1)
fprintf(data,'%s','abid')
pause(.1)
fprintf(data,'%s','abid')
pause(.1)

% --- Executes on button press in v17.
function v17_Callback(hObject, eventdata, handles)
% hObject handle to v17 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;
fprintf(data,'%s','abjd')
pause(.1)
fprintf(data,'%s','abjd')
pause(.1)
fprintf(data,'%s','abjd')
pause(.1)

% --- Executes on button press in v15.
function v15_Callback(hObject, eventdata, handles)
% hObject handle to v15 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
data = instrfind;

```

```

fprintf(data,'%s','abhd')
pause(.1)
fprintf(data,'%s','abhd')
pause(.1)
fprintf(data,'%s','abhd')
pause(.1)

% --- Executes on button press in v21.
function v21_Callback(hObject, eventdata, handles)
% hObject   handle to v21 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abnd')
    pause(.1)
    fprintf(data,'%s','abnd')
    pause(.1)
    fprintf(data,'%s','abnd')
    pause(.1)

% --- Executes on button press in v20.
function v20_Callback(hObject, eventdata, handles)
% hObject   handle to v20 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
    data = instrfind;
    fprintf(data,'%s','abmd')
    pause(.1)
    fprintf(data,'%s','abmd')
    pause(.1)
    fprintf(data,'%s','abmd')
    pause(.1)

```

BIBLIOGRAPHY

- [1] David Gascón, "802.15.4 vs ZigBee"
<http://www.sensor-networks.org/index.php?page=0823123150>, November 17, 2008.

- [2] *www.xmos.com/published/uart-tutorial*

- [3] "Serial Peripheral Interface Bus" Wikipedia, 30 June 2012

- [4] AN10216-01 I2C Manual

- [5] "TCP/IP" *<http://searchnetworking.techtarget.com/definition/TCP-IP>*, October 2008

- [6] "Common Intermediate Format" Wikipedia, 21 May, 2012

- [7] "EEPROM", Wikipedia, 25 May, 2012

- [8] "Static Random Access Memory", Wikipedia, 26 June, 2012

- [9] "Flash Memory", Wikipedia, 3 July, 2012

- [10] "Binary Image", Wikipedia, 2 December, 2011

- [11] Sven Zacharias and Thomas Newe , "Technologies and Architectures for Multimedia-Support in Wireless Sensor Networks," in *Smart Wireless Sensor Networks*. Hoang Chinch and Yeh Tan, Ed. Rijeka: InTech, 2010.

- [12] Purushottam Kulkarni, Deepak Ganesan, Prashant Shenoy and Qifeng Lu, "SensEye: A Multitier Camera Sensor Network" in *Proc. ACMMM*, 2005, pp. 229-238.

- [13] Ian Akyildiz, Tommaso Melodia, and Kaushik Chowdhury, "A survey on wireless multimedia sensor networks." *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 51, pp. 921-960, Mar. 2007.

- [14] C3088 CMOS Camera Datasheet

- [15] ATmega644 Datasheet, AVR

- [16] 45DB321D-SU Datasheet, AVR

- [17] MDA100CB Datasheet, MEMSIC

- [18] MICAz Mote Datasheet, MEMSIC

- [19] Farshchi, Shahin, Nuyujukian, Paul H, Pesterev, Aleksey Mody, Istvan Judy and Jack W, "A TinyOS-enabled MICA2-based wireless neural interface," *IEEE Trans. Biomed. Eng.*, vol. 53, pp. 1416-1424, Jul. 2006.

- [20] Tia Gao, Dan Greenspan, Matt Welsh, Radford R. Juang, and Alex Alm, "Vital Signs Monitoring and Patient Tracking Over a Wireless Network." in *Proc. ICICTH*, 2005, pp. 102-105

- [21] Vijay Devabhaktuni and James Haslett, "Introduction to Theory and Applications of Self Organizing Wireless Sensor Networks," in *Proc. International Conference on Wireless Communications*, 2004, pp. 74-78.

- [22] MIB600 Ethernet Gateway Datasheet, MEMSIC
- [23] "TINYOS Documentation Wiki " <http://docs.tinyos.net>, 1 August, 2011
- [24] "Mote-PC serial communication and SerialForwarder (pre-T2.1.1)"
http://docs.tinyos.net/tinywiki/index.php/Mote-PC_serial_communication_and_SerialForwarder, 5 March 2012
- [25] "Face Access"
http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/index.htm,
- [26] "Wireless Visual Sensor Networks for Urban Traffic Management," white paper, NIST Technology Innovation Program.
- [27] Stanislava Soro and Wendi Heinzelman, "A Survey of Visual Sensor Networks," *Advances in Multimedia*, vol, 2009, pp. 1-22, 2009.