Spring 2010

ECE 400

Electrical/Computer Engineering Design Project

PROJECT F: WIRELESS SMART CAMERA NETWORK

May 6. 2010

Team Members

Ravi Arimilli
Bryan Hunter
Lancelot DaSilva
Dhaval Patel

Team Supervisor

Dr. Qi Hairong
Target Localization Using Wireless Camera Sensor Network

Bryan Hunter, Dhaval Patel, Ravi Arimilli, Lancelot DaSilva
University of Tennessee at Knoxville
bhunter6@utk.edu, dhaval1988@gmail.com, rarimill@utk.edu, ldasilva516@gmail.com

Abstract

Vision provides humans with unmatched capabilities to sense the world, and yet it has not been exploited in a sensor network setup. This paper outlines the design and construction of such a network, which implements a target localization algorithm in a real-world setting.

1. Introduction

The goal of this project was to develop a system using networked smart camera nodes that collaborate with each other to achieve a better understanding of the real world. This project included four components: 1) image acquisition and processing; 2) collaborative processing among camera nodes through wireless communication; 3) system assembly; and 4) application development.

2. Design Approach

The first objective of the design was to find an algorithm that properly utilized the network of cameras. A target localization algorithm using certainty maps was chosen. Information from each camera was integrated in order to generate a result that localized multiple targets.

The image acquisition and processing was done individually on each node in the system. Each node consisted of a camera and a processor used to generate its own description of the situation based on its captured images. The information obtained from each camera was then merged together to generate an overall description of the site.

The system required a way to send information between nodes to achieve a result that could not be achieved by a single camera. In order for these nodes to collaborate in real-time, a wireless network was integrated into the system design as the communication medium.

A key component to get the network up and running was the system assembly. Integrating cameras, processors, and a wireless network was necessary for an accurate and useful implementation of the target localization algorithm. In addition, proper placement and calibration of all of this equipment was required.

One useful addition to the design process was a graphical user interface (GUI) running on a separate machine that provided real-time feedback on how the system was performing. The GUI allowed the displaying of certain processes within the algorithm and quantified results generated by the system.

3. Algorithm Outline

The algorithm that we employed in our wireless camera network was a target localization algorithm utilizing certainty maps. The algorithm fuses information from several different cameras in order to localize targets within a room. This information is displayed on a certainty map, which is the overhead view of a room showing the areas of the room that have been clarified as having no objects so far. Certainty maps narrow down the areas in which objects could be located by clarifying areas where objects are certainly not located.

![Figure 1. Illustration of the construction of a certainty map at a visual sensor node. (a) Visual hulls of objects in 3D; (b) Projection of 3D cones onto a plane parallel to the ground; and (c) The local certainty map constructed from this information.](image-url)
3.1. Purpose

The purpose of this algorithm is to gain information from a network of cameras working cooperatively that cannot be obtained from each of the cameras individually. Collaborative processing is used to localize targets without using highly complex image processing algorithms that can be time and power consuming. Since communication is the most expensive process in terms of power, simple image processing is done on board so that only small amounts of information are sent out.

3.2. Segmentation

Rather than using complex and computationally expensive algorithms to extract targets out of images, simple image segmentation was used. A background image without any targets is first acquired in order to represent the image of an empty room. This image is stored in memory for later use. When the targets are added to the room, another image is taken. This new image is subtracted from the background image in order to highlight the targets.

A few image-processing techniques are needed to “clean up” the subtracted image. First, Otsu’s thresholding algorithm is used to convert the image from color to black and white. The algorithm dynamically chooses an optimal threshold based on the image’s intensities in order to highlight the target. Otsu’s approach works better than arbitrarily choosing a threshold, because it works for any environment and requires no prior testing.

![Figure 2. Image Segmentation. (a) Background image; (b) New image with target present; (c) Result after subtracting and Otsu thresholding new image from background; (d) Image after performing morphological filtering; (e) Certainty map created from the filtered image.](image)

3.3. Vector Acquisition

The field of view (FOV) of a camera is the angular area that a camera portrays in its images. The only area that can be clarified by a particular camera is inside its own FOV.

Calculating the FOV of a camera from a picture it took is simple. If the area viewed in the picture ranges from straight left of the camera to straight right of the camera, the FOV is 180 degrees. If this is not the case, trigonometry is used to determine the FOV by calculating the difference between the maximum angular excursion of a picture with a 180 degree field of view and the maximum angular excursion of the actual picture from the camera.

Starting with the segmented image, the image is viewed as two halves divided at the center of the image, which is considered the 90 degree mark. This makes the geometry for calculating the angles of the target boundaries very simple. Wherever the image is highlighted, angles are calculated based on the image width, the FOV, and the location of the highlighted area. The image width fully occupies the FOV. Each half of the image can now be manipulated as if it were a right triangle. The distance in pixels from the edge of the image is used as a ratio to calculate an angle between the FOV and 90 degrees. These calculated angles now represent the vectors that were shown in Figure 1.

3.4. Certainty Map Construction

Geometry is also used to generate certainty maps and update them. A fully black certainty map is used as the base map, signifying that nothing has been clarified as empty space as of yet. Each camera calculates its own angles and knows its own position within the map. Scanning the latest updated certainty map, the ratio of the number of rows and columns from the camera position is used to set pixel values. If the current pixel is white, meaning the pixel has already been clarified as a non-object (empty space), nothing is changed, and the next pixel is examined. If this pixel falls inside of the vectors calculated before, the pixel is left black. See Figure 3 [1] for an illustration of this process.
3.5. Collaboration

Having each camera send its own certainty map over the network would not meet the standard of a low power consumption wireless sensor network. Instead of sending certainty maps back and forth, camera positions and certainty map angles are the only pieces of information that are sent out. A centralized processor, or master node, fuses all of this information together into a single certainty map, reducing the total transmitted power by several orders of magnitude, as shown in Table 1.

| Power Consumption per Byte (hypothetical) | 1 µW |
| Number of Angles (4 bytes per angle) | 2 | 4 | 6 |
| Power Consumption (µW) | 8 | 16 | 24 |
| Image Size (1 byte per pixel) | 176x144x3 | 240x240x3 | *320x240x3 |
| Power Consumption (µW) | 7.60E+04 | 1.73E+05 | 2.30E+05 |

*Resolution of Cameras Used in Project

Table 1. Hypothetical Power Consumptions of Sending Angles Compared to Images

3.6. Target Counting using Region Growing

In addition to localizing targets, a way of counting the number of targets represented on the final certainty map was desired. A region-growing algorithm was implemented for this purpose.

Region growing requires two scans of the image. The first scan classifies pixels as it goes across and down the image. When a black pixel, which is a target pixel, is reached, that pixel’s previously traversed neighbors are examined. If any of its previously examined neighbors have already been classified, the pixel assumes the class of one of those pixels. Otherwise, the pixel is classified into its own unique class. This process is done iteratively over the whole image.

![Region-Growing Example](image)

The second pass scans the entire image again. On this pass, every time a target pixel of one class bordering a target pixel of one or more other classes is found, all containing classes are combined into a single class. Once finished, each class represents a target. A simple example of region growing is presented in Figure 4.

3.7. Integration of Algorithm into System

In order to integrate the overall algorithm into our wireless camera network, a master and slaves approach was taken. One of the cameras represents a master node, while the rest are considered slave nodes. Figures 5 and 6 display flowcharts describing the master’s and slave’s algorithms. Each slave node is responsible for acquiring background images, segmentation, and calculating certainty map angles when requested by the master.
The master coordinates the tasks of the slaves and performs the same initial image processing tasks as the slaves. However, once the master determines its own certainty map angles, it begins certainty map integration. After initially creating a certainty map based on its own certainty map angles, the master receives certainty map angles from all of the slaves and fuses them together into the final result. It then calculates the total number of targets using the region growing method described above.

3.8. Potential Problems

Although the algorithm is very simple and efficient, a few problems can occur. As the number of targets increases, the number of cameras also needs to increase. Too few cameras will lead to a certainty map that is not fully clarified.

Also, using an unsophisticated approach like image subtraction can lead to errors. Intensity changes in the background may appear as objects after thresholding. This problem may occur because the target is blocking some light going to the background. Using elevated camera placement to avoid the blocking of overhead lighting may sometimes fix the problem. Other times, this error is unavoidable with the lack of sophistication of background subtraction.
3.9. Improvements for Algorithm

Many improvements can be made to the target localization algorithm employed in the system. First, the algorithm required multiple scans of the image. This approach is more efficient than many of the more complex algorithms, but it can still be optimized through better hardware and software techniques.

Second, a dynamic approach to certainty map construction could have been implemented in the system, so any camera could act as the master or a slave in a single simulation. Instead of updating the certainty map in a predefined order, a polling method could have been used to update the map on each iteration with the camera that would clarify the map the most. This would achieve the final result without having to use all of the cameras, theoretically reducing transmission power usage even more. However, it was found empirically that with relatively few cameras, this method would actually increase power consumption. On the other hand, if many cameras were employed in the system, the dynamic approach would much more efficiently clarify targets.

Finally, more image processing could have been employed to reduce false image segmentation as described above. Edge detection could have been used to match edges in the background and find the edges of the target. However, given the system design and insufficient processing power of the nodes in the system, more image processing would have been detrimental. However, techniques like this could be successfully implemented in a system with nodes having more processing power.

4. Networking

To establish a communication link between the various camera nodes, sockets were used. Sockets are a software endpoint that establishes bidirectional communication between a server program and one or more client programs. The socket associates the server program with a specific port on the machine it is running on, so any client program can open a socket to that port on the server to communicate with the server program. A simple client – server model was used as the architecture of the networking protocol. The master camera, which can be seen transmitting radio signals in Figure 8, functioned as the server while the other cameras and the GUI served as clients. The 802.11b PCI card shown in Figure 7 provided the mobile sensor platforms with the wireless capability to network the multiple cameras together.

![Figure 8. Network Configuration of System](image)

To run the recognition counting program, the master program must be executed first. If the master program is not running, the slave program and the GUI will be unable to establish a socket connection with the master. The master program first creates a socket with the socket() system call, and then it binds the socket to the address of the master MSP using the bind() system call. The master node then employs the system call listen(), which listens for connections from clients on port 80. The listen() call will fail if the port is not available, which means the port number cannot be reserved or is already in use. Due to the prior network configuration on the MSPs, port 80 was the only available port for socket communication. Connection requests from the clients are then accepted with the accept() system call, which blocks (pauses the program) until a client connects to the server, and a new file descriptor is returned.
However, before the client nodes can connect to the master, there are several commands that need to be executed to ensure a secure and reliable connection between the camera nodes. Similarly to the master node, the client node must first establish a socket with the socket() system call. After the creation of the socket, the hostname of the master node, e.g. msp6.nomads.utk.edu, is passed to the function gethostbyname(char *name), which returns a pointer to a hostent C language structure containing information about that host. The connect() function is then called by the client to establish a connection to the master node.

Once the required steps have been executed on both the client and master nodes, bidirectional information transmission can begin. The socket file descriptor is the socket descriptor to used to send and receive data, whether it is the one returned by socket() or the one returned by accept(). A char buffer and the write() and read() system calls are then used to send and receive data, respectively, over the network. The function read() uses the new file descriptor returned by accept(), not the original file descriptor returned by socket(). The read() call will block until there is something for it to read in the socket, i.e. after the slave or master node has executed a write(). It will read either the total number of characters in the socket or 255, whichever is less, and return the number of characters read.

One of the design challenges was successfully transmitting the certainty maps between the master and the GUI nodes. Due to the large number of bytes being sent across the network, frequent packet/information loss occurred. To fix this problem, the protocol was modified to precede each transmission with a header containing its size in bytes so that the entirety of the information could be read before the program continued.

5. Using Qt for the Graphical User Interface

Qt is a cross-platform graphical user interface (GUI) framework. In order to design the particular GUI desired for this system, the following were laid out in a single window: a log of operations area (text box), an area to display certainty maps with their labels, and the following buttons: “connect to master”, “take background picture”, “take actual picture and generate certainty maps”, and “quit” (see Figure 9).

Figure 9. Graphical User Interface

The QWidget class was used to implement the window. The QLabel class was used to implement the log area. The QWidget class was used to implement the certainty map display area, which contained QLabels to contain the certainty map pictures and QLabels to contain the corresponding labels for the certainty map pictures. The QPushButton class was used to implement each of the four buttons. The certainty map display area and the log area were both given horizontal and vertical scroll bars (which show when necessary) by placing each inside a QScrollArea.

To lay out the log of operations, certainty map display area, and buttons in the desired fashion, the QGridLayout class was used. This class was also used within the certainty map area to lay out the certainty map pictures and their labels in the desired fashion.

The only four events of consequence to be accounted for were the clicking of any of the four buttons. The buttons that were not allowed to be clicked at any given time (because the operation they initiated was currently in progress or not allowed at the current time) were grayed out. (The “quit” button was always enabled.) Thus, upon opening the GUI initially, only the “connect to master” and “quit” buttons were enabled. After the “connect to master” button was clicked, it was grayed out, and the “take background picture” button became active. (After this point, the “take background picture button” always would be active.) After the “take background picture” button was clicked, the “take actual picture and generate certainty maps” button became active. (After this point,
Lastly, the event handlers for the events of consequence needed to be written and tied to the appropriate events. This was done using the Qt signals and slots mechanism, which allows the tying of a slot (event handler) to a signal (event). Each slot is a function that is not allowed to take any arguments and not allowed to return any values. The slot for the “connect to master” button opened up a connection over a TCP socket to the master MSP. The slot for the “take background picture” button told the master to make all slaves and itself take a background picture simultaneously. The slot for the “take actual picture and generate certainty maps” button told the master to get the slaves to take their actual picture simultaneously, generate certainty map angles, send them to the master, and have the master send pictures of certainty maps to the GUI as they arrived. The “quit” button told the master to tell all slaves and itself to end the program, and it quit the GUI.

6. Qt Challenges

The primary problems with developing the GUI in Qt were never having used Qt before, and the lack of instruction tailored to the specific GUI application that was desired. The learning documentation had many examples and plenty of coverage of the concepts of Qt, and the reference documentation had voluminous information about classes and functions, but to use them both effectively, one already had to know what functions and classes to use to achieve a desired end result.

The primary means of overcoming this problem was trial and error. Namely, this involved writing simpler programs that allowed mastery of particular subsets of Qt necessary to implement the necessary superset of Qt in the final version of the GUI for the camera system.

A problem encountered along the way in the design and implementation of the GUI was the discovery that no updates to the GUI coded in event handlers were applied until the event handler was complete. This caused a problem in the connect to master event handler, which contained a loop that was supposed to receive a message each time a slave MSP connected to the master, which it was supposed to display in the log area. This event handler problem caused all the intermediate connection messages to be displayed all at once after all nodes had connected, rather than once each time a node connected, rendering this information useless. Because displaying this intermediate connection information was not vital, the solution chosen was simply removing this loop inside the event handler. If displaying this information were desired, it would be necessary to make the Qt GUI multithreaded, having a primary thread handle receiving events, and a secondary thread listen for messages from the master.

7. Experimentation

7.1. Phase 1 - CMUcam

The goal of this experiment phase consisted of having each CMUcam communicate with the Tmote Sky, Xbee, or TI wireless to send and receive data from the wireless network. The CMUcam provides a simple vision capability to small embedded systems in the form of an intelligent sensor. CMUcams consist of a small video camera and a microcontroller with a serial interface. The system architecture is shown in Figure 10 and Figure 11 [2]. It consists of three I/O interfaces, named GPIO Header, Serial Port on uart_0, and TTL Serial Port on uart_1 in Figure 11.

![Figure 10. CMUcam system architecture.](image-url)
7.1.1 Tmote Sky

The first part of the experiment was connecting a CMUcam to a Tmote Sky. The Tmote Sky (shown in Figure 12) is a wireless sensor platform on a board. It has an MSP430 based board with an 802.15.4 compatible radio chip. The layout of the 10-pin expansion is shown in Figure 13 [2].

As shown in Figure 14, the GPIO must be connected in the order shown in Table 2.

<table>
<thead>
<tr>
<th>CMUcam</th>
<th>TMOTE SKY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Enable</td>
<td>ADC3</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>SCK</td>
<td>I2C_SDA</td>
</tr>
<tr>
<td>MISO</td>
<td>ADC2</td>
</tr>
<tr>
<td>CAM Reset</td>
<td>I2C_SCL</td>
</tr>
<tr>
<td>CS</td>
<td>ADC1</td>
</tr>
<tr>
<td>RX2</td>
<td>UART0TX</td>
</tr>
<tr>
<td>MOSI</td>
<td>ADC0</td>
</tr>
<tr>
<td>TX2</td>
<td>UART0RX</td>
</tr>
<tr>
<td>AUX POWER</td>
<td>AVCC</td>
</tr>
</tbody>
</table>

Table 2. CMUcam and Tmote Sky pin connection

After all the necessary connections were made, sending and receiving data on the UART port was tested. Nodes were synchronized at 9600bps with no parity bit. When sending data over the radio, it was observed that the transition from UART mode to radio mode took more than a few minutes, and as a result, the interrupts were being missed. This was due to the fact that the radio and
UART0 expansion bus is physically shared with the SPI data bus that connects the radio to the microcontroller. Therefore, the system is only half duplex instead of full duplex.

### 7.1.2 Xbee Series 2

The Xbee Series 2 is an embedded solution providing wireless end-point connectivity to devices. It uses the 802.15.4 Zigbee stack and provides a simple to use serial command set as a wrapper around the Zigbee stack.

![An Xbee 1mW Wire Antenna](image1.png)

**Figure 15. An Xbee 1mW Wire Antenna**

To connect an Xbee and a CMUcam, the Xbee Explorer Regulated was purchased. The Xbee Explorer Regulated (shown in Figure 16) takes care of 3.3V regulation, signal conditioning, and basic activity indicators.

![Xbee Explorer Voltage Regulated](image2.png)

**Figure 16. Xbee Explorer Voltage Regulated**

The next part consisted of using an Xbee Series 2 with a CMUcam. This was done by removing the level shifted serial jumper (shown in Figure 17) and connecting the pin of the Xbee Explorer Regulated (shown in Figure 16).

![CMUcam serial port layout](image3.png)

**Figure 17. CMUcam serial port layout**

The CMUcam and Xbee pins were connected as shown in Table 3.

<table>
<thead>
<tr>
<th>CMUCAM</th>
<th>XBEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5V</td>
<td>5V</td>
</tr>
<tr>
<td>LOGIC OUT STX</td>
<td>DOUT</td>
</tr>
<tr>
<td>LOGIC IN SRX</td>
<td>DIN</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

**Table 3. CMUcam and XBEE configuration**

Then both the CMUcams and the Xbees were synchronized at 9600bps with no parity bit. The system was able to transmit and receive data successfully. However, due to receiving the CMUcams too late because they were on backorder, we moved the implementation of the algorithm onto the MSPs.

### 7.2 Phase 2 - MSP

The MSP (shown in Figure 18), which stands for Mobile Sensor Platform, uses a mini-ITX board with a 600 MHz processor, a wireless card, and a webcam. This pre-built platform allowed us to implement our camera sensor network algorithm.
Before testing, much Linux troubleshooting on the MSPs was necessary to deal with the various problems that came up. The MSPs tended to be finicky about connecting to the wireless network upon startup. A reboot would fix this under most circumstances. The wireless card on one of the MSPs was malfunctioning, and reboots would not fix it. It was necessary to issue commands in the console, such as ifconfig, iwconfig, and iwpriv, to figure out if the wireless network interface was running properly and attempting to connect to the wireless network properly. The problem was solved after a few iterations of this process.

Another of the MSPs had wireless card driver issues that could not be resolved using the aforementioned methods. To solve this problem, it was necessary to make an image of the hard drive of the Linux installation on one of the properly functioning MSPs and clone it onto this problematic MSP. The Linux command used to create and write hard drive images for this purpose was dd.

Using extra webcams purchased online required a newer version of Linux than was on all but one MSP. The Linux installation on this MSP was cloned onto all the others (after backups of the original installations were made). While the extra webcams now worked, for some reason the wireless networking did not work on these reimaged MSPs anymore. The problem could not be resolved on the reimaged MSPs, so the reimaged MSPs were reverted back to the Linux installations each one had previously.

In the second phase of this project, the implementation was done on 6 MSPs. After much experimentation over the period of the final two months with 6 MSPs in a 10ft x 10ft room, it was found necessary to use a larger room to expand the field of view of the cameras and to use more cameras to clarify a larger portion of the room. Two laptops were added to the system and effectively used as slave MSPs because they could support more USB webcams than MSPs could. In total, 14 webcams were used for the simulation in a 15ft x 20ft room. There were 5 cameras placed along the 20ft wall, with adjacent cameras 3 1/3 ft from each other. 2 cameras were used along the 15 ft wall, with adjacent cameras placed 5 ft from each other. The setup of the room is shown in Figure 19.

Each webcam was assigned a camera ID and a two-dimensional camera position. The display computer running the GUI would connect to the master MSP. First the master would listen for connections from the slave nodes, one by one in a predefined order, and inform the GUI each time a slave connected. Second, the GUI would tell the master to signal all slaves and itself to take a background picture simultaneously. Figure 20 shows a sample background picture.
GUI can tell the master to tell all slaves and itself to quit the program.

Three successful simulations were run with 1, 2, and 3 targets. The final certainty map from each of these simulations is shown in Figure 21 (a) through (c).

In general, placing more cameras close to each other clarified more uncertain area in the room.

8 Conclusion

In this paper, the design of a robust system composed of MSPs and webcams that can detect the position of objects in a room using distributed processing on the MSPs and a minimum of communication bandwidth over TCP sockets was presented. The certainty map algorithm employed by the system makes the MSP webcams work cooperatively to narrow down the areas that definitely do not contain objects, leaving the areas containing objects marked as such. Because only certainty map angles are transferred between the master and slave MSPs, the bandwidth consumption is very low.

There are some improvements to the system that can be made in the future. Implementing the system on smaller inexpensive cameras, such as the CMUcam, would be paramount. Also, incorporating a different wireless network using Xbees or Tmotes would make the system more versatile and deployable by eliminating reliance on the setup of an internet connection. Finally, making the GUI multithreaded, allowing display of intermediate status information, would aid in diagnosing problems and seeing in vivid detail how the components of the system interact with each other.

9 Incorporation of Class Concepts Into Project

In order to help with our design process, we used several key strategies learned in class. We learned that it was imperative to keep the team coordinated and up to date, so we created a website to help with communication. The website made it easy to update files and keep everyone informed of who was doing what. One section of the website contained a Gantt chart showing our progress. The Gantt chart was also discussed in class. In addition, we took the advice of making a video for our presentation. Although making a video required much more effort than putting together slides, we felt that having complete control over the entire presentation was a good idea. Finally, we took many general concepts learned in class, like separating the work and the system design approach, and applied them to our design process.
References