Route and non-metric path navigation such as “go down the hall, turn left at dead end, stop at the second intersection” is more natural for behavior-based robots. Such procedures are known as ‘topological path planning’, which have been utilized and developed for several decades. The difficulty of this technique is defining locations; hall, dead end, intersections, etc… One of the techniques to accomplish this is known as artificial land marking, which involves a camera mounted on a robot. The drawback to this however, is the high computation power needed to process the feed from this system. Our approach for this project is to bypass the high computing cost by using multiple stationary cameras. This system won’t require much computation power and calibration to construct the topology land marks of the given environment.

I. Overview

The goal of our project is to construct a topological map using an arbitrary amount of cameras. This map could then be used as a navigational tool to move around in the environment. In order to accomplish this, we utilized a system of cameras connected to client computers. These clients would process the graphical information it receives from the cameras, and then send it to a central server which would consolidate the information and build the topological map.

For proper experimentation, we constructed a maze and setup consumer grade cameras to obtain views from within the structure. We also constructed a robot to move around the maze to be tracked by our cameras. Using this system, we were able to successfully construct topological maps out of an arbitrarily designed maze, as well as successfully incorporate up to 9 cameras into the system.

II. Hardware Components

The hardware setup for our project consisted of three client computers and one server computer. The clients were typical desktop machines, with 9 consumer grade cameras evenly distributed between them. A laptop connected via a router took the role as a server. Additionally, each of our group members used their own laptops to help administer the client computers via remote desktop, since there was only one set of keyboard, mouse and monitor connected to the clients.
For our maze, we used pre-cut Styrofoam poster board connected by blue masking tape and book clamps. To minimize interference of reflections, shadows and other lighting conditions, we also laid down a matte cloth as the ground of our maze. (See Figure 1)

Finally for experimentation and tracking, a simple RC-car covered with a black paper mask was used within the maze. Later on, a turtle-type circular controlled robot was made for more accurate tracking and demonstration of the system.

III. Design

Our system consisted of three parts; the server, the client and the robot. The client took care of camera management and object recognition, and the server took care of constructing the map from the information sent by the clients. We spent a lot of time getting the clients and server to network and communicate with each other. As this was meant to be a design for motes communicating with a central server, we designed for minimal computational power (on the client side) as well as minimal packet sending between the clients and server. While this was being done, we also built a robot to serve as our tracking object.

IV. Networking and Communication

Our networking infrastructure was based in part by Rene Nyffengger’s Socket() class. We used this to learn how to use the windows API and eventually created our own wrappers for our Send (sendTO) class. Rene’s RecieveByte function was still utilized for our system.
For our packets, we decided on using a header with four bytes and consisted of the following: 1 byte represented a camera’s unique ID, 2 and 3 byte was for the length of the packet and 4 byte was for the packet/package type.

Each camera received a unique camera ID when it connects to the server, which allowed us to scale to as many cameras we may need for a given environment.

This came to be one of the major challenges in this project. Along with properly sending and receiving packets, we had trouble running multiple threads and getting sockets to run with each other. For example we ran into multiple issues when sending packets. The process was either too slow in sending or when sending too quickly, the server would crash because it would grab more or less than one packet. We remedied the problem by increasing packet length from one byte to two (we could only store up to 255 but packets turned to be greater in size than that).

We also tried to send the first frame of each camera to the server for visual representation. However because of the insistent streaming, we opted to save the frames locally and then transfer them to the server via USB. Another issue we faced was socket blocking. This caused the client to hang until the server responded to its request for a unique camera ID. This was fixed by fixing the socket type to non-blocking.

Finally we had the issue of having the server running multiple threads. This caused the issue when they context switched. We implemented a critical section so that the server doesn’t try to grab from the data queue ay the same time it is trying to write to it.

V. Camera Setup and Object Recognition

One of the first challenges we faced was properly setting up the camera array for the assorted group of cameras we obtained. We had to install drivers on each of the clients for each different type of camera. The client computers we had were unable to support more than two cameras per USB bus, which sometimes resulted in inconsistent frame placement. Additionally, every time a camera was plugged or unplugged into another computer, the cameras had to be re-calibrated, which was a rather time consuming process.

The object recognition algorithm was composed of simple background subtraction techniques. We applied a median to filter and color thresholding to remove shadows and slight movement. Once filtered, we ran an edge detector in order to pre-compute the edges in our background. These edges were used for bisecting the given camera view into regions which would then be used as nodes in the topological map. If the robot appeared in view or disappeared, we checked to see how close it was to an edge. If it was within a certain distance, and was detected closer to any other edge in view, we bisected the line there. Our ‘motes’, or sensed nodes were setup in a tree structure. We would start with one and for each bisection, the mote would also be bisected. The leaves of the tree represented active motes where the branches were bisections of the mote above it. (See Figure 2)
For synchronization, all cameras had to keep track of when an identifiable object, in this case the RC car or robot we built, appeared in a given camera’s view. The event in which the robot moved from one bisection to another, as well as when it exited was also recorded. Throughout this exercise, the actual system time, which was synced between each computer was also kept track of.

Once information was processed by a client, it would send packets to the server. We had it set to send in a minimal manner in order to minimize the load on the server.

**VI. Map Construction**

The server was responsible for building the topological map. The code would process packets that the client sent. Depending on what type of packet it was, the server would write the data to the mote class. After the mote class was modified, the complex is then written to. For visual feedback, a 3d representation of the complex is shown in a window which changes according to the different bisections (model packets) and regions that are active (observation packets).

If a bisection occurred, old data was thrown out. That means so say that if mote A was connected to mote B, but mote B was bisected, we wouldn’t know anything about the connection between motes A.
and the resulting motes of the bisection of mote B. The way that mote A’s regions connected to the bisected regions in mote B would be unknown, hence we no longer connect them.

Mote connections were made once the region being observed was “closed”, which means that two bisectings would have to see the robot at the same time, and then both send a packet saying that they no longer see the robot. Our setup allowed us to see which nodes were currently active by marking them as red simultaneously on the complex window. Once neither was marked red anymore, a line was drawn between the nodes, thus connecting them. (See Figure 3)

![Figure 3: Server side windows showing generated 3d complex and motion tracking.](image)

This visual representation was written in matlab by Edgar Lobaton. This code essentially reads from the complex and simplex files produced and uses them to create a map using plot points and 3d graphics.

**VII. Robot Motion**

To serve as our tracking object, we designed a turtle-type Bluetooth controlled robot. It was built using a Luminary board LM3S8962 microcontroller board with two servers, three sonar sensors, a voltage regulator and a BlueSmirf module for communication. (See Figure 4)
The robot was programmed in LabVIEW 8.6 and compiled/uploaded to the microcontroller using LabVIEW Embedded 1.1. Along with direct control, the robot was primarily implemented to do a random walk within the maze. The sonar modules were set so that the robot would avoid hitting the walls as it traversed the terrain. Overall movement was designed around a simple state machine implementation with the three sonar modules as the input. (See Figure 5, 6)
Figure 5: Robot design schematic from Luminary board to components.
VIII. Conclusion

This project was a very valuable learning experience for our team. Because of the complexity of the system, we were able to get a better understanding of many unique systems and implementations. Computer networking, computer vision, embedded systems with state machine implementation and sonar tracking were the most prevalent components needed for our system to achieve its goal.

It is satisfying to know that this system could have many possible practical applications. It can accommodate any number of cameras and clients, which means we can theoretically be applied to any pre-existing camera setup, such as the cameras watching hallways in a building. This setup also would overcome a lot of costly computational power compared to embedded system navigation such as a camera mounted robot. A possible next step of this project would be to implement a feedback system where we could program our robot to navigate to specific areas of the maze based on the complex it generated. This same implementation could also be used for personal navigation by allowing a person being tracked on a camera to query for directions from the server.