## 5.7 IR Sensors

The purpose of this subsystem is to detect and track the amount of people are in a room. If there are no people in the room the subsystem is supposed to turn the lights off. This saves power and extends the life of the lights in the room. It will also be a good opportunity to show students another application of electrical engineering concepts. In the previous teams design they accomplished this functionality by using a combination of IR sensors and pressure sensors. However, there were problems with the implementation that which lead us to redesign the subsystem.

### 5.7.1 Fall Subsystem

The previous design consisted of 4 IR sensors in the four corners of a box. [11] Each of the sensors working in sync to track an IR beacon that would be placed inside the model room. The sensors would track the position of the beacon as it was moved around the model room.

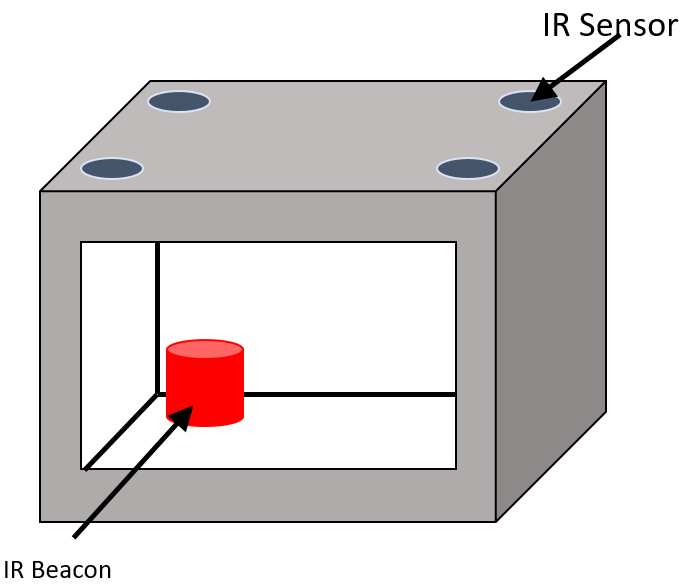


Figure 48: Concept Model of Previous IR Sensing Design

This interaction was then represented by a GUI on the web server.

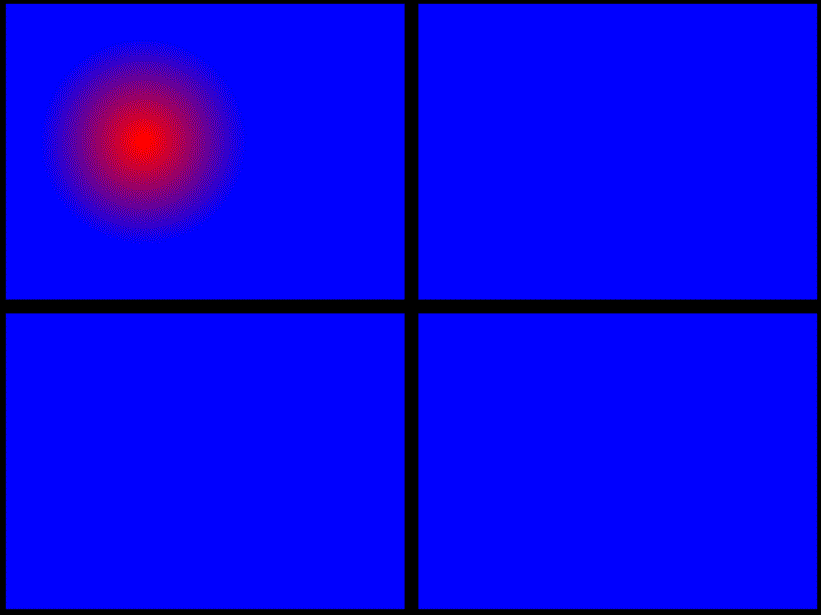


Figure 49: Visual Representation of IR Sensors Reading IR Beacon

Each quadrant of the picture represents the vision of one of the IR sensors. In the above example the top right most sensor is reading an IR signal, which is represented by the red cluster. If the IR beacon was to move the sensors would track it as it moved across the model room. While this design is the cheapest and most simple approach there were a few drawbacks that caused this semesters group to redesign the subsystem.

The first problem was that in accuracy of the sensors. Each sensor had a difficult time point pointing the exact location of the IR beacon because the distance being read was inconsistent. The values would jump randomly which would throw off the positioning algorithm on the raspberry pi. Another issue was that there were multiple dead spots in the model room where no signal would be read. These issues stemmed from the fact that the IR sensors were not designed for such an application. The sensors were designed to be used at a longer distance and are incapable of reading IR signals at such a precise distance.

### 5.7.2 Subsystem Requirements

The design of this subsystem was to simulate the detection process of a person entering a room and being kept track of. But more important than the detection process was to show how that process worked to kids. While most of the other subsystems in this model room were designed around teaching ECSE students engineering concepts, this design was more geared towards outreach to kids. Therefore, the clarity and the **simplicity** of the solution was important. The goal of the entire subsystem was to track an object as it moved around the model room. This process had to be precise enough so that it was accurately tracking the object. Thus, a minimum error of **0.1 inches** was set to ensure accurate tracking. The system was also required to be **cheap** as the professor wanted it to be replicable.

### 5.7.3 Benchmarking and Selection of Components

There were 3 solutions that were considered and researched. A laser grid was suggested as a potential solution of from modern occupancy sensing setups. The second solution was a solution suggested by the previous team. The last solution was to use computer vision to analysis a video feed of an IR camera. Each one was then compared on 4 characteristics of complexity, price, precision, and reliability.

Table 1: IR Sensor Concept Selection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Solution | Complexity | Price (cheapness) | Precision | Reliability | Total |
| Laser Grid | 4 | 4 | 2 | 1 | 11 |
| Additional IR Sensors | 2 | 4 | 4 | 3 | 13 |
| Computer Vision | 3 | 3 | 5 | 5 | 16 |

The IR sensor could be used to create a laser grid system that could detect when an object entered and exited a section of the room. This had the drawback of being not very precise because you can’t track the object once it gets into the room, and that it is quite difficult to tell if an object is entering or exiting.

The Additional IR Sensors is a solution proposed by the team last semester and it improves the accuracy by increasing the resolution, but it would require an IR beacon and an algorithm to track it. This would greatly increase the complexity of the design as well as the increasing the chance for error. The sensors would also need to carefully calibrated and secured.

The computer vision solution is using an IR camera to detect where the objects are in the room via an algorithm. It simplifies the design down from an array to a single camera, and scored the highest on the selection matrix.

There were two notable pieces of hardware that were selected for this project. The first being the IR camera. Since the market is incredibly small for IR camera’s specialized for the raspberry pi, there is only one camera commercially available that achieves the desired functionality of being able to capture IR light in a picture. This is the “Raspberry Pi No-IR Camera” [13]. Then a simple cheap IR filter was selected. [12]

### 5.7.4 Design

For the hardware portion of this fairly straight forward. The only thing to do was to connect an IR camera with the main raspberry pi using the built-in camera port. Then, a simple IR filter was placed over the IR camera lens to block out all visible light, and only except IR light. This simple design was chosen as it is fairly cheap and easy to replace for the purposes of replacing the camera in the future to show how an IR camera differs for a regular camera.

The computer vision design had a few major requirements that were considered heavily when designing the software for this subsystem.

Table 3: Requirements Table for Computer Vision Solution

|  |  |  |
| --- | --- | --- |
| Requirement name | Meaning | **Solution** |
| Safe | Power consumption will not exceed 2.1A | **Limit processing power** |
| Work in parallel with system | Take up a little processing power as possible to avoid throttling | Limit CPU usage to **one worker thread** and **one server thread** |
| Precise Tracking | Must track object within margin of error of 0.1 inches | Min resolution of 170x130 or 0.1 inches per pixel (Actual 320x240 or **0.045 inches per pixel**) |
| Time Precise Tracking | Must keep track of object as it moves | Min frame update rate of **8 fps** |

In accordance with the power supply subsystem the power consumption of the raspberry pi was not to exceed 2.1A, therefore it was important to use as little CPU power as possible. So, it was important to only analyze a frame of the video when necessary and keep processing time as a whole to a minimum.

The picture that the IR camera would take would be 320 by 240 pixels which is roughly 0.045inches of coverage per pixel when you take into account the that the view of the IR camera is 14.5inches by 10.8inches when placed on the roof of the box. The minimum object position update limit was set to 8 per second. Therefore, the computer vision algorithm had to analysis at least 8 frames in a second.

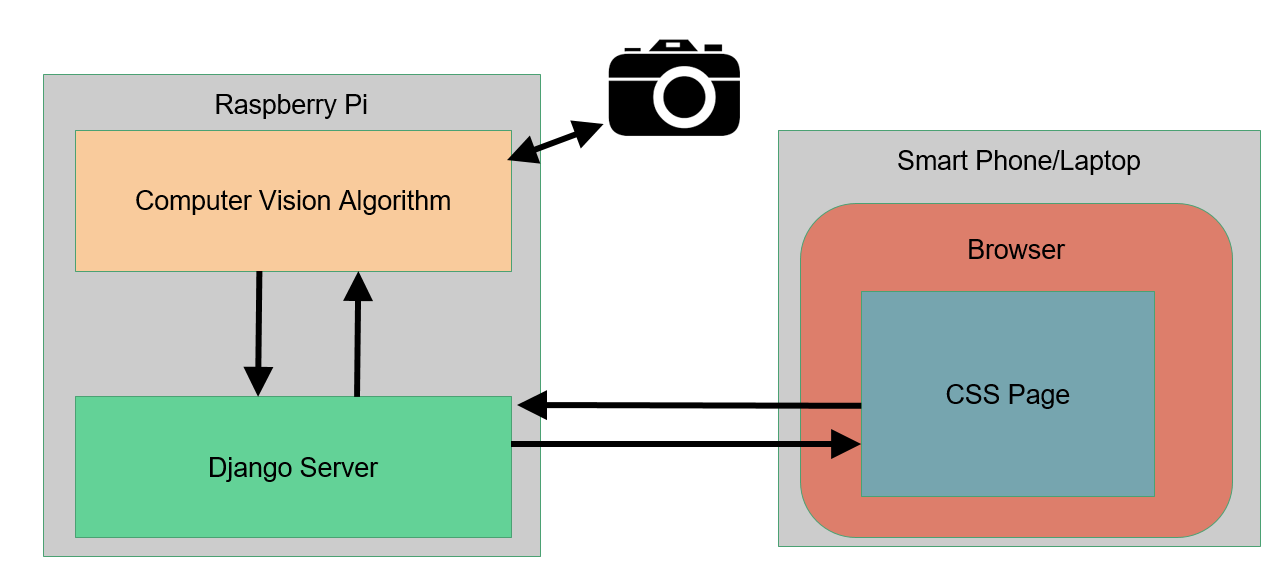


Figure 50: General Overview of Software Flow

The general overview of how a user would interact with the model smart conference room in occupancy mode is show above. The very first step is the ajax call that the CSS page makes to the Django server. The CSS page makes this ajax call 10 times a second. Then the Django server will ask the computer vision algorithm for the position of the IR beacon. The algorithm then requests a new frame from the IR camera. Then when it gets back the frame the algorithm analyzes the picture for the center of an IR beacon. After it finds the center it sends the coordinates back to the Django server. If the algorithm is unable to detect a beacon in the box it sends back a (-1,-1) as the position. Lastly the Django constructs a JSON response with the x and y coordinates stored as the data and sends them back to the CSS page. The page will then update the display with the position of the IR beacon.

The computer vision algorithm itself was something that a lot of time was put into designing. The algorithm is done in three main steps. Step zero is to first get the last frame that was taken by the camera. It does this by using a 3rd party software called PiCamera and a wrapper class that was written by the team to utilize this library. After the last frame is captured and readied into memory the algorithm starts its process. Step one is to apply a Gaussian blur to the image.[14] This reduces the noise in the image which will in turn result in fewer errors and faster computation. Step 2 is to apply a color filter that will turn the image into a black and white image.

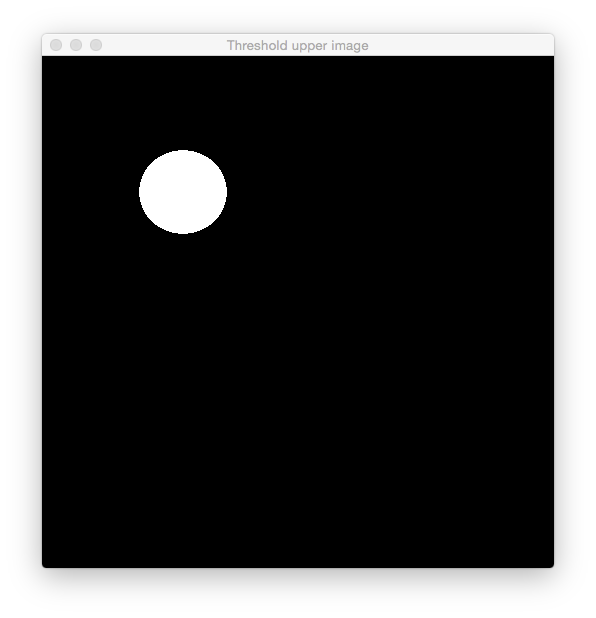


Figure 51: Image after a color mask was applied.

This process essentially looks for pixels within a certain color range and rejects any pixels that do not fit into this range. Through testing the color range was found to be between the values of (0,100,100) and (45,255,255). This mask is then applied to the image and now the algorithm only has to analyze black and white pixels further increasing speed. The third and final step is to apply a blob detection algorithm with looks for the blobs of pixels with the same color. [15] In this case, it is looking for blobs of white pixels which signify that an IR beacon is there. After a match is found the center of the blob is returned as two integers to the Django server.

### 5.7.5 System Evaluation

This subsystem was tested by running the code a long side a debugger which was able to capture how much time each process was taking. These results were compared to the requirements set forth for the software.

Table 18: Testing Results

|  |  |
| --- | --- |
| Requirement | Test Result |
| Current doesn’t exceed 2.1A | Current doesn’t exceed 2.1A during algorithm |
| Min resolution of 170x130 or 0.1 inches per pixel | Algorithm runs at 320x240 or 0.045 inches per pixel |
| Min frame update rate of 8 fps | 10-20fps with server and computer vision algorithm running |

The current was analyzed by attaching an amp meter to the supply of the raspberry pi and measuring how much current was being draw when it was running the server and the computer vision algorithm. During this test, the current never went over 2.1A.

The next two requirements were tested by running a subroutine inside the main program and tracking the number of frames per second the computer vision algorithm was getting. Initially the computer vision algorithm was only getting around 1-2 fps. However, after a lot of optimization, such as offloading tasks to a specific thread to be done once and then having the result stored in memory, the test was run again. This time the computer vision algorithm could analyze around 10-20fps. The variation in the result was because of the variance in color of different frames taken. I.E. sometimes the computer vision algorithm will have to analyze more pixels to get the result of where the IR beacon is.

### 5.7.5 Open Issues

The algorithm itself varies based on what type of IR beacon is being used and how the beacon is being pointed at the camera. This affects what color comes the computer vision algorithm recognizes. To fix this issue permanently change the color mask in the computer vision algorithm to accept basically everything that is not black.