ROBOCOP – SURVEILLANCE ROBOT
USING IROBOT, NODEJS AND BEAGLEBONE

SREEJITH UNNIKRISHNAN, SHREEJA KUMAR, JOHN KIM

CSE 291 – EMBEDDED SYSTEMS – UCSD
DECEMBER 2015

INTRODUCTION

For our final project, we built a Surveillance Robot which performs intrusion detection. The robot will be connected to a secure home network where it will act as a host, and several client sensors will be attached to key break in points. When an intrusion is detected through one of the client's vibration sensors, the data will be automatically sent to the core server and the robot will begin scouting to the location which the data was sent from. Once it reaches the break in location, it takes a photo of the break in point and send the image to owner's email address. Also, the robot can be manually controlled through the web application.

OBJECTIVES

The main objectives of our project is as follows,

- Create a surveillance system that is easy to use and control
- Support innovative alert system in case of emergencies
- Leverage the connectivity capabilities of Beaglebone and create a small connected sensor network
- Interface the webcam to have a visual input
- Use the irobot create to move towards the sensor nodes in case of emergencies
- Should be reliable, fully customizable and can be controlled over web interface

IMPLEMENTATION

After stating the objectives, we did a survey of the available technologies and equipment. We then agreed on the motive, final outcome we expected based on our objectives and set out to work.

The initial task was to decide on the system architecture. The system can be implemented using a variety of technologies and components. However, we stuck to the minimum set of components that we had and devised a robust, easy to use system architecture. In a world where all gadgets are connected to each other, it made sense to make the sensor network and robot communicate among themselves and make intelligent decisions.

1. Components

- **IRobot create:**

  IRobot create is one among the core components of the overall system. It provides easy to use serial API to control the robot and issue commands. It also has a wide array of sensors that helps in motion. We control the robot using a python script running on the beaglebone through serial console.

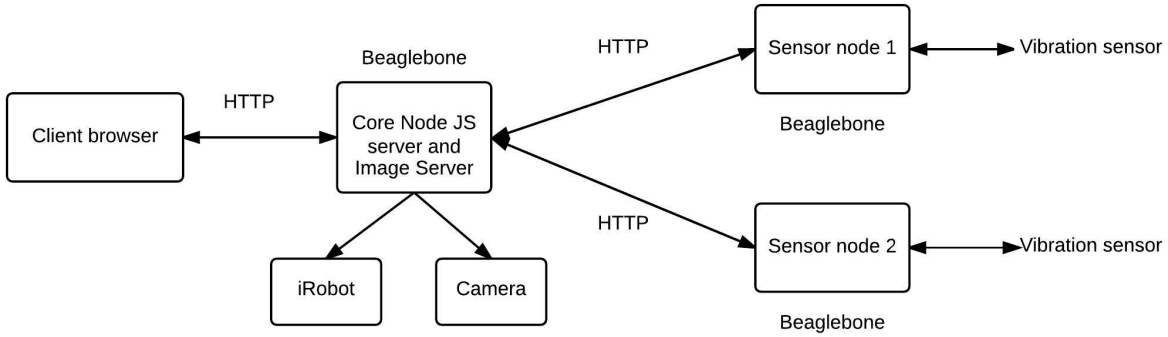
- **Beaglebone:**

  Beaglebone black is used for the project. One beaglebone resides on the iRobot running the web server as well as controlling the whole system. It is connected to two sensor nodes which is also implemented using beaglebones. We extensively used the serial communication feature of the beaglebone to control the robot as well as the Wi-Fi connectivity through which we established HTTP socket connections to user's web interface as well as the sensor clients.

- **Piezo electric sensors:**

  The piezo electric sensors are used to detect any kind of motion happening at the key break in points. The sensor when connected to GPIO pins via a serial resistance provide high input at idle state. When a break in happens the piezo sensor becomes a closed switch, making the signal in a low state. The sensor provides simple interface to detect vibration and costs less.

2. System architecture



The general system architecture is shown in the diagram above. Now we can look at each part in detail and see how the communication between them happens.

- **Core Server:**
  This is the central part of the system. It coordinates sensor information, serves web interface and images, and controls the iRobot and webcam connected. The server is written using NodeJS and Express. The central server that runs on port 5000 uses Socket.IO framework to exchange event messages over HTTP sockets to sensor clients and user's web interface. The events are broadcasted over the connected sockets as required (eg., when intrusion is detected). The server forks child processes to control the robot as well as to take the picture. If the system has internet connectivity it will email an alert to user along with the image taken by the webcam. The Socket.IO events are unique for sensor nodes as well as client web interface to avoid possible conflicts. We used the serial module for NodeJS to communicate with the iRobot. We partitioned the code to be executed when the user needs just an image, but not an email, when an alert happens etc. The system also controls the robot when the user needs to drive the robot remotely via the web interface.

- **Web interface:**
  The web interface is the main medium though which user interacts with the system. Because of that we have tried to make it elegant and simple to use. Bootstrap and JQuery were used to make the front end straight forward and intuitive. It connects to the core server on the robot and displays the status of both sensor nodes, and the overall system status. When an alert occurs as informed by the server, the web page displays the warning message with a red symbol to catch user's attention. The web page will display the image captured by the robot when the intrusion had taken place. The user can reset the alert status which in turn clears the alert message, image and the danger sign. Apart from this core functionality, the system allows user to request a picture from the robot's camera and this will get displayed on the web page. The interface also provides buttons to drive the robot remotely. Coupled with image reception, the system would enable the user to drive the robot relatively easily.

- **Sensor node:**
  We used piezo electric vibration sensors for detecting intrusion or break in. The beaglebone analyses the GPIO pin connected to the sensor for anomalies. The voltage produced by the piezo sensor was in micro volt ranges which was not enough to trigger an input signal to beaglebone's GPIO pin, and hence we used a series resistor to drive the voltage to high levels. The NodeJS client running on the sensor node runs a python daemon that checks the GPIO pins. When a break in attempt is detected, it writes the event to a file. The server monitors the file and an interrupt is generated whenever the file is changed. The Node client then registers a disturbance with the core server via Socket.IO event. To minimize the possible errors in detection, the python daemon checks the time when an event has happened and compares the last occurred disturbance. Only when the time difference crosses a threshold, new disturbance signal is triggered. This minimizes the false positives.

- **IRobot:**
  IRobot is controlled by the Beaglebone and is used to move to the location of disturbance when a break in happens. It can also be controlled remotely using the remote drive feature.

- **Webcam:**
  The webcam attached to Beaglebone takes picture when the robot reaches the break in place. It is also providing images when the user explicitly needs a picture as requested via the web interface.

- **Vibration sensor:**
  The vibration sensor is a piezo electric switch that is open when the sensor is idle and low when it detects a vibration. Being piezo electric, the generated current is extremely small and requires a series resistance to drive the GPIO pin of beaglebone.

3. Protocols and Interfaces

- **NodeJS and express:**
  We extensively utilized NodeJS and HTTP sockets to host the web server and client interfaces. NodeJS provide API for forking child processes and it was used to start python daemons to control the robot and monitor the GPIO pins for piezo sensors.

- **Socket.IO:**
  Since most of the communication was event based it made sense to make sure that the socket communication triggers events. We used Socket.IO NodeJS interfaces to communicate and receive events. It was used for communication between core server and sensor nodes, as well as between web interface and core server.

  The various Socket.IO events that are used to transfer event information are given below;
  - Connect events which establishes the connection between the http client residing on sensor nodes and server.
  - Event to update the status of sensor1 and sensor2 on the webpage.
  - Events to handle sensor disturbances. This event executes the python script which makes the irobot move to the location of disturbance and capture an image. This information is then emitted to the webpage.
  - Events to drive the robot remotely via the web interface. There is one event per drive options.
  - Event to capture image when requested by the user calls the python script that does the job. This image is then emitted to the webpage.

- **Serial UART interface:**
  We used serial communication for beaglebone to issue commands to iRobot. For this purpose, we used UART4 port of beaglebone and used the iRobot API to control the robot.

DIFFICULTIES AND SOLUTIONS

We faced a bit of difficulties while designing the system and most importantly while integrating all parts together. The first difficulty was for ensuring proper asynchronous and real time event information communicated between core server and user`s web interface. It is always desired to inform the user as soon as possible. We tried to implement various socket based solutions, and most of the time, the code became too complicated to maintain. We then used Socket.IO for event notification which proved to be very fast and reliable, while maintaining minimum code footprint.

Also, interfacing webcam from Beaglebone proved to be a bit difficult initially. We tried to use OpenCV, but the little board took a bit of time to capture image and saving it to disk. Then we researched a bit and came across a library called fswebcam which was widely used for raspberry pi. We tested on Beaglebone and in our testing it was able to capture and save the image to the disk in around 1.2 seconds on average. We tried a loop test, continuously taking pictures and found that the library was consistent taking images every 1.2 seconds for a straight 100 count marathon. We then wrote a nifty 3 liner in python that can capture the image and save it disk.

The next task was to have the emailing facility ready. The email functionality was straight forward using the ssmtp library for Linux. However, having attachments was the main issue. ssmtp was not up to the task. We had to search around for a good command line library to mail with attachments. Finally, our search went onto mpack library. We then wrote a python program combining both mpack and fswebcam to capture an image and mail it to the user.

We used normal http based image transfer to web interface from the server. However, that proved to be very slow and sometimes image was downgraded in quality. Then we created express based server that serves static files. The solution was easy but the path to it was a bit tedious. We integrated the express server with the NodeJS core server to ensure that the system worked in unison.

On the web interface side, we need to display images on an event basis. Usually the browsers tend to cache images, due to which dynamically altering the src property of image field using jQuery was not effective during our tests. We did a bit of fiddling with JQuery and used random number generator to force load the image when the event happened. This approach proved to be successful.

While interfacing the vibration sensor, we initially tried to plug it directly into GPIO considering the fact that the generated voltage will be adequate for the Beaglebone to detect the disturbances. However, we were not able to detect any signal unless the sensor was hit really hard. Upon monitoring in an oscilloscope, we found that the generated voltage was extremely low. We then put a series resistance to load the sensor and raise the input voltage to the board. Using this method, we were able to reliably measure the signals.

Secondly, the vibration sensors oscillate between high and low signals till the oscillations last. This was like around 300 milliseconds. To ensure that the false positives do not cause false alarms again and again, we recorded the time stamp for last sensor trigger and compares the latest one with the last time. Only when the time difference is more than a threshold, we triggered the event again. This made a lot of difference in ensuring the reliability of the sensor nodes.

EVALUATION AND PROJECT SUCCESS

Considering our stated project objectives, we were able to achieve all of it, although given time we would have loved to add quite a few features more to it. The system is stable and provides an easy to use and interactive front end. The Beaglebone was able to control the robot, monitor the sensors, transfer images, email the user successfully.

We would have implemented dynamic sensor location management if we had a suitable sensor like GPS. Within our limitation we had to hardcode the sensor locations. We also feel that there was memory limitation considering the platform. Given that we run a relatively busy web server with image capture and robot control, a board like Raspberry Pi 2, with more powerful processor and double the RAM with half the cost, would have made a great difference. The system was static waiting for image to be captured. Using Chrome developer tools, we measured the image the image request duration was like 1.8 seconds long, out of which 1.2 seconds approximately were used by the board to capture and save the webcam image.

Also, we could have made the robot autonomous, given that we had a bit more time and sensors to have positioning of sensor nodes.

We wanted to have some sort live streaming from the webcam, but could not make it happen. We tried using open CV however the streaming made the system extremely slow. The live video capture taxed the CPU a bit and we did not want to sacrifice the system response times just for video. We also tried to use Linux ffmpeg and avconv libraries to stream video over a UDP channel however it was of no help. This will be one of the future work we would have taken up given enough time.

TEAM EFFECTIVENESS & COMPOSITION

Sreejith Unnikrishnan:

- Design of NodeJS core server running on beaglebone.
- Socket.IO events design for web interface and sensor nodes.
- Web application design using JQuery and Bootstrap.
- Python script for webcam control.
- Python module for emailing alert messages with image attachments.
- Evaluation of different video streaming protocols.

Shreeja Kumar:

- Design of the sensor side Socket.IO events and HTTP client.
- Software for controlling the iRobot using Beaglebone.
- iRobot module for remote drive, as well as motion towards sensor nodes.
- Interface design for vibration sensor and HTTP client running on sensor node.

John Kim:

- Software on Beaglebone sensor node monitoring piezo sensor.
- Circuit for sensor monitoring.
- Algorithm minimizing the false triggers by the sensor.
- Integration with sensor web client.