SHaZam the Magic Lamp: IR-Based Gaze Tracking and Light Direction

Chaim Halbert, Dexter Scobee, and Edward Zhao
EE149A/EE249A Project Milestone Update

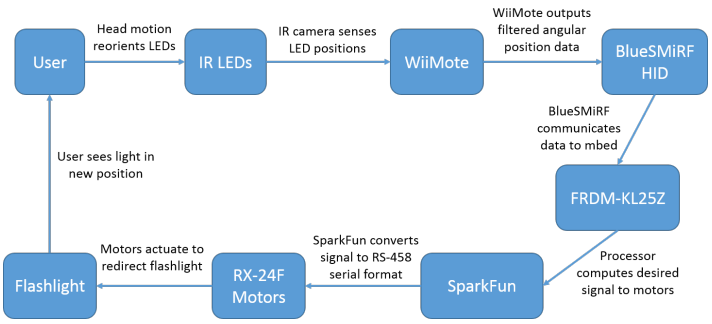

Fig. 1: SHaZam ConOps. This diagram represents the interconnections between components for the SHaZam system

I. Project Vision

The goal of this project is to design a lamp that will redirect its light to follow a user's gaze. The magic lamp will consist of a standard adjustable desk lamp with a microcontroller embedded in its base and servo motors connected to its light source. Additionally, the lamp will utilize a WiiMote to track IR LEDs affixed to the user's face or hat. The WiiMote will communicate relative positioning data to the microcontroller via Bluetooth, which will in turn direct the motors to aim the light source. The controller will behave according to a state-machine that models states such as "Track" or "Stay." Figure 1 shows the logical connections between system components and how they are intended to interact.

II. Project Components

A. Hardware

The hardware consists of a gimbal-mounted flashlight connected to a system controller, in turn connected wirelessly to a Wiimote, which tracks sensor bars mounted on the user's hat.

The gimbal is a pair of Dynamixel RX-24F smart servos, which are daisy-chained and connected to the system controller via RS-485 serial link. To convert the RS-485 to 3.3V serial UART, we used an SP3485 breakout board.

The system controller is a Raspberry Pi running Raspbian Linux in "headless" configuration, without a monitor, keyboard or mouse. These can be added for development and debugging, or an ethernet connection can be used to log in remotely via SSH.

The system controller links to the WiiMote via a USB Bluetooth dongle. (Somehow, we had issues with newer Wiimotes that included MotionPlus, so we used an older Wiimote without it.) The WiiBrew web site [1] was a valuable resource in understanding the Wiimote hardware and in interfacing the Wiimote with the system controller.

[1] http://wiibrew.org/wiki/Wiimote

The hat has two Wii "sensor bars" mounted to it. These bars are not sensors at all, but actually contain a simple collection of IR LEDs at either end of each. The specially-designed camera in the Wiimote picks up and tracks these four points of invisible light. For more information about the sensor bars' asymmetrical configuration, see the **??** section.

B. Design changes

Although the overall structure and linkages of our system diagram have not changed since the beginning of our project, virtually every block was modified by the end of development.

1) Gimbal: Our original design called for traditional servo motors to drive the gimbal's motion, but we decided to change to advanced, serially-controlled servos. These Dynamixel RX-24F motors (see Figure 2) provided several advantages, the primary one being ease of control.

Unlike traditional servo motors which are commanded to set their position using PWM, our smart servos could be commanded via RS-485 serial link. This provides more timing flexibility in our control logic, since this makes PWM interrupt routines unnecessary. The smart servos also allowed us to set the speed of the movements over the serial link, offloading logic from our main control loop to the motors.

Also, the new servos were daisy-chainable, an impossibility with PWM. This allowed us to use a single serial output, as opposed to multiple, independent PWM outputs, each with their own interrupt routines. This resulted in a subtantial time savings in development, and greater reliability.

A disadvantage was that RS-485 is quite different from serial UART, in that it uses 200 mV differential signaling over two wires and is half-duplex, whereas our TTL UART operates at 3.3V and is full-duplex. We used a breakout board for the SP3485, which accomplishes both level conversion and protocol translation, with the aid of an RTS signal from our system controller.

2) Bluetooth: Originally, we were going to use a BlueSMiRF Gold to connect to the WiiMote. However, the Gold only connects via serial data endpoints, while the WiiMote requires an HID interface. So, the BlueSMiRF Gold did not work.

To resolve this problem, we purchased a BlueSMiRF HID, which has the same hardware as the BlueSMiRF Gold, but has different factory-flashed firmware for HID capabilities. This also did not work, because the BlueSMiRF HID was designed to operate as an HID slave device, such as a mouse, keyboard or joystick; it could not operate as an HID host, to control such HID devices.

In the end, we used a USB-Bluetooth dongle.

3) System controller: Although we originally intended to use the ARM mbed FRDM-KL25Z Freedom board, we decided to change platforms to the Raspberry Pi model B.

Fig. 2: SHaZam Hardware. From left to right, IR LED assembly for user wear, WiiMote with internal IR camera, Dynamixel 2-axis motor assembly with attached flashlight

We did not use the Pi initially because the lab already provided each of our group members with a free, personal ARM mbed processor. This allowed us to work independently, parallelizing development. In comparison, the Pi cost $35. Also, PWM control requires Linux kernel programming on the Pi, or a separate daughterboard with its own microprocessor, such as the Arduino-compatible Pi Alamode (another $35). We deemed this too complex and expensive.

However, circumstances changed later in the development. By switching to the serial servos, we eliminated the need for kernel development or a daughterboard. Just like with the ARM mbed, we could now control all the hardware with a single system controller. Also, when we exchanged the BlueSMiRF for the Bluetooth dongle, the easily-installed support for the dongle in Linux gave the Pi a clear advantage over the ARM mbed, which in contrast required compiling and integrating C++ code. The Pi was simpler.

The Pi also offered new capabilities not available on ARM mbed. With the Pi, we could change languages from C++ to Python, and we could develop and test directly on the Pi itself. This meant no more waiting for compilation and flashing every time we made a change. It also gave us an interactive Python shell to test out snippets of code prior to integration.

Next was the ethernet connection on the Pi. Connected to a LAN, multiple members could work simultaneously on the Pi via SSH. Also, with an internet connection, we could use package managers to quickly install and test pre-built third-party modules for new Linux and Python functionality. The internet also gave us tighter integration with GitHub for version tracking and merging our code modifications. The confidence that we could quickly revert our changes allowed rapid progress even as the deadline approached.

C. Software

We have created Makefiles (tested on Mac) to enable program compilation without internet access, and to allow us to use our own IDEs. We also have hello world programs for:

- Blinking the mbed's LED
- Receiving debugging messages over serial-USB
- Sending and receiving data via serial to the Bluesmirf Gold

We also have the capability to communicate with the WiiMote via Bluetooth connection to a laptop, and we have obtained a "USB2Dynamixel" adapter which allows interfacing a computer to the Dynamixel motors to prototype motor control software. In terms of data filtering and motor positional control software, our peripherals already provide a useful suite of functionalities. The WiiMote performs blob tracking and filtering onboard to provide smooth angular position estimates for the IR LEDs (there actually is no mode in which the WiiMote will transmit raw pixel data). Similarly, the Dynamixel motors are smart servos, so the control logic to maintain a position is internal to the servo and we only need to provide the position to be held.

D. Modeling and Algorithms

We have developed a finite state machine that models the behavior we want for our project, as well as mathematical models detailing our gaze tracking algorithm.

For our state machine, we have two main states, "On" and "Off." Within the On state, we have two sub state machines, one dedicated to pitch, and one dedicated to yaw, to control the respective servo motors. Inside the pitch and the yaw submachines we have three further substates; "Calibrate," "Stay," and "Track". Figure 3 illustrates the flow between these states within the pitch sub state machine. Within the Calibrate state, we want to standardize our measurements before beginning any tracking. This state will act as an initialization state and will only execute once. Then, depending on the angle of the user's head, we will either move to and remain in the Stay state or advance through to the Track state, which begins moving the servo motor in accordance to the user's head movements. We have designed guards to avoid actuating the motors when the user may not intend for his motion to be tracked by SHaZam. To this end, we set minimum and maximum limits on the change in desired lamp orientation (based on change in user head position) to which the system will respond. Note that these are not instantaneous changes, but the change over a fixed, short period of time (on the order of 0.2 seconds).

We have also derived a mathematical model that will allow us to reconstuct the state of the LED configuration (which reveals where the user is looking) by obtaining angular position measurements from the WiiMote.[2] The data provided are x- and y-angle measurements, corresponding to the estimated angular position of the LEDs in two orthogonal planes. Figure 4 shows the geometry of reconstructing the state in one of these measurement planes. Note that the state of the LEDs in a plane has three degrees of freedom (shown in Figure 4 as u, v, and ψ), so three angular measurements are needed to unambiguously reconstruct the state. Once the state in the first plane is known, however, the range data (v) is known for the second plane as well, so only two angular measurements are needed to reconstruct the state relative to the second plane.

III. Path Forward

At this point in the project, we believe that we have all of the basic components to successfully implement our design. With the new BlueSMiRF module in hand, we will move forward with creating and testing the necessary software to collect data from the WiiMote on the mbed processor. In parallel, we will use the USB2Dynamixel to move forward prototyping motor control from a laptop. Once motor control functionality is established, the SparkFun board should enable the same control from the mbed,

[2]This model is an expansion of work done by Johnny Chung Lee (Google, formerly CMU). See http://johnnylee.net/projects/wii/

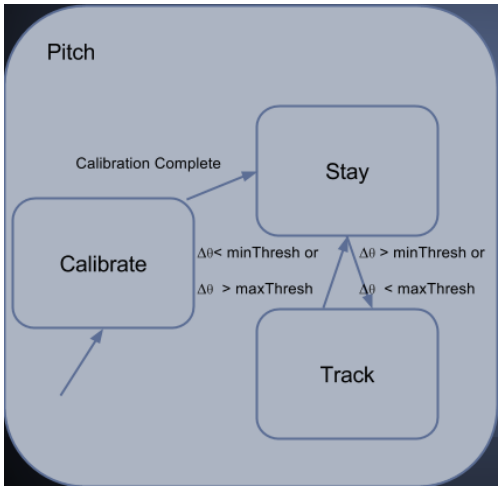

Fig. 3: State Machine. The state machine controlling the motors' pitch orientation is designed with three states: "Calibration," "Stay," and "Track." Note that there will be thresholds to prevent undesired tracking. The control for yaw follows an identical model

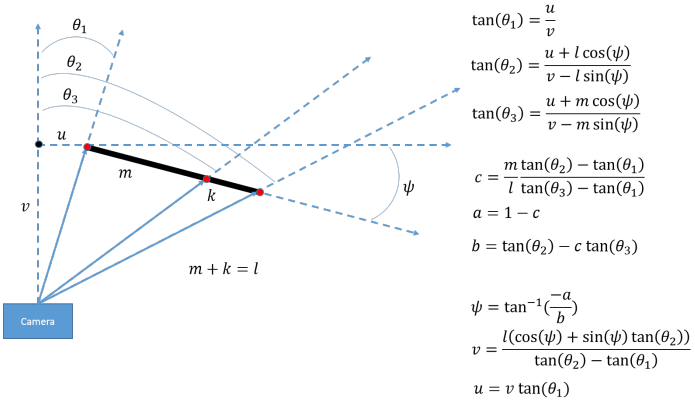

Fig. 4: State Reconstruction. By obtaining angular position measurements of three LEDs with known offsets, it is possible to fully reconstruct the state in a plane. Range data (v) obtained from one plane of measurement can be applied to aid state reconstruction in the other

serving as a means to convert data from the mbed UART into the RS-458 format required by the Dynamixel motors.

In addition to continuing to develop our electronics interfaces, we will also iterate on the designs of user-facing components. We will look to create a more comfortable but less mutable fixed LED array for the user to wear so that the distances between LEDs can be better characterized. Finally, the motor assembly will be mounted on a lamp-like frame to enable the desired experience for the user.