Pew Pew Final Report

Craig Hiller, Kevin Wu, Leo Kam, Christopher Hsu

I. Introduction

THIS project will create an automated NERF turret that can detect, track, and fire at a target (human face). The project will model the detection and aiming of a target as a state machine governed by the combination of RGBD camera data and other sensor inputs to correctly rotate the turret and incline the NERF gun to aim at the target. The goal will be to accurately detect a target and fire accurately for maximum effect.

II. Project Requirements

The general requirements that can be measured to determine the success of our turret are as follows:

- The turret platform supports $45°$ rotation.
- The turret platform supports an incline angle between $0°$ and $30°$.
- The turret is able to perform facial recognition.
- The turret can accurately hit a stationary target (human face) within the range of 3 meters.
- In the case of communication error, turret supports graceful failure without damaging the turret.

III. System Components

A diagram illustrating the system components for this project is shown in Figure 1.

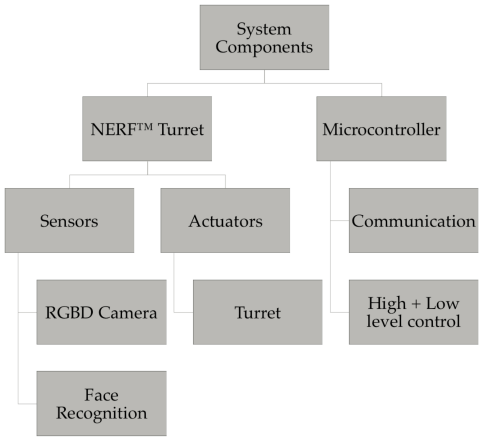

Fig. 1. System components hierarchy

In general, the system is divided into two overarching components: (1) the NERF Turret and (2) the Microcontroller. Each component can be subdivided further as shown in the diagram. In general, the NERF Turret component is responsible for providing the actuators that control the rotating platform - a turntable made of bearings, motors, and a winch -, an electronically triggered NERF gun mounted on the platform, and the RGBD camera sensor that performs the face recognition. On the other hand, the microcontroller component is responsible for providing the controller api calls that appropriately handle the high and low level control of the system. The microcontroller is connected to the actuators via wired connections, and to the sensors using a WiFi connection via the Adafruit CC3000 chip.

IV. NERF Turret Hardware

The descriptions we have for our hardware in the NERF turret are described below.

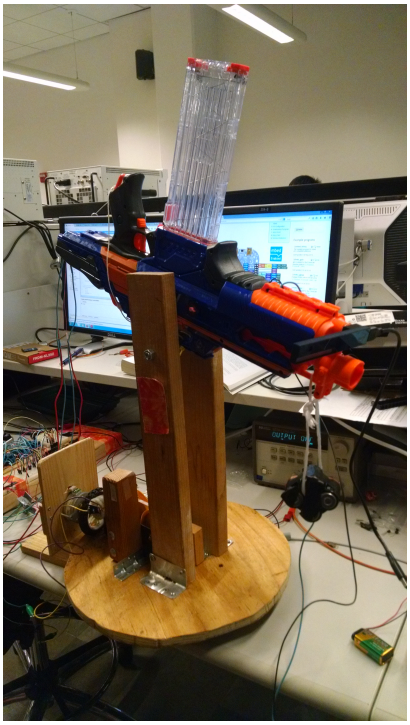

Fig. 2. The actual NERF gun and turret.

A. NERF Gun

The NERF gun and the motors on the turret are the actuators in our system. We selected the CS-18, which was chosen since it was not only relatively lightweight, but also entirely electronically controlled which allowed us to swap out the switches of the firing mechanism. Figure 3 below shows the result of disassembling the gun to allow electronic firing.

Currently, we have disassembled the NERF gun and reverse-engineered it to the point where we can electronically trigger it

Fig. 3. The CS-18 that has been modified to allow external control.

to fire a NERF dart. In addition, we have obtained two relays to replace the trigger switches in the gun in order to connect it to the microcontroller. Relays are used since they are more easily controlled via a GPIO pin on the microcontroller.

B. Turret Motors

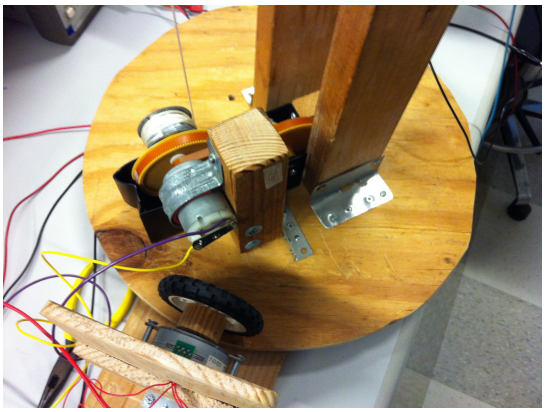

Fig. 4. The turret for the NERF gun along with the two driving motors.

A pair of motors were used to control the turret; one for vertical motion, and one for horizontal.

Our turret is mounted on a turntable, and spinning this turntable allows our NERF gun to rotate side to side. To control the turntable, we went with a stepper motor mounted with a wheel that lies on the rim of the turntable. By driving this motor, our wheel pushes our turntable, causing it to spin and allowing us to move the NERF gun. We decided to use a stepper motor because we felt that the control it gave us made it ideal for this role. We did not need a lot of torque since our turntable spun quite easly, so we could get away with using the stepper motor. Additionally, the stepper lets us have precise control of how many degrees it turned each time we sent it a signal from our microcontroller, which allows us to precisely aim our NERF gun.

As for up and down motion, we ended up using a DC motor that through gearing, had a very high torque. Because we mounted our NERF gun on a pivot, we wanted a motor strong enough to be able to pull the gun up and down. Since the NERF gun has a decent weight to it, we needed a pretty high torque motor in order to be able to move the gun. The motor has a strong string that it can wind up that is tied to the back of the gun, allowing us to be able to aim the gun higher or lower.

C. Electronics

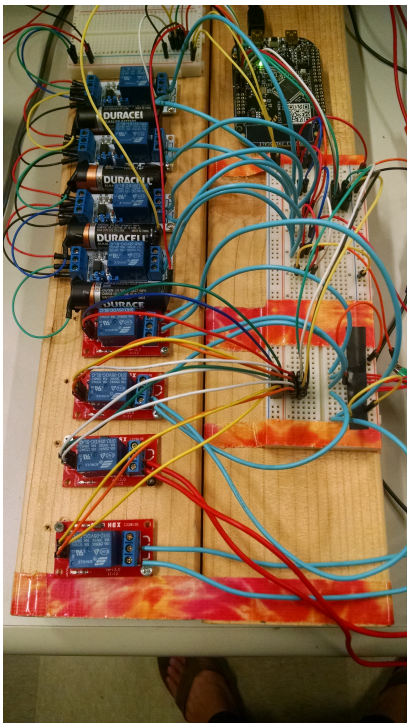

Fig. 5. The electronics mounted on a two by four. These are used mainly to control the turret firing and rotation.

In order to control our motors and the firing of the NERF gun, we relied on a series of relays that lets the MBED mincrocontroller control when to send power to the various components of the NERF gun. We ended up using eight relays in total; four relays to control the DC motor, two relays to control the stepper motor, and two relays that dealt with the firing of the NERF gun.

We needed to create an H-Bridge in order to give our microcontroller the ability to switch the direction of the voltage driving the DC motor, allowing us to wind and unwind the string that controlled the up down motion of the NERF gun. An H-Bridge requires using four switches, hence four of our relays. The stepper motor could be controlled by senting voltages across the four wires that were attached to it, so we ended up using two of our relays here.

The final two relays were used to control the firing of the NERF gun. In order to fire, we had to have two things happen– the flywheels inside the gun had to be spinning, and the plunger had to push the NERF bullet into the flywheels. Both of these events could be done by connecting a pair of wires inside the gun for each functionality, so we had our two relays each control one of these functionalities.

V. Camera and Controller

A. RGBD Camera, Face Recognition, and Server

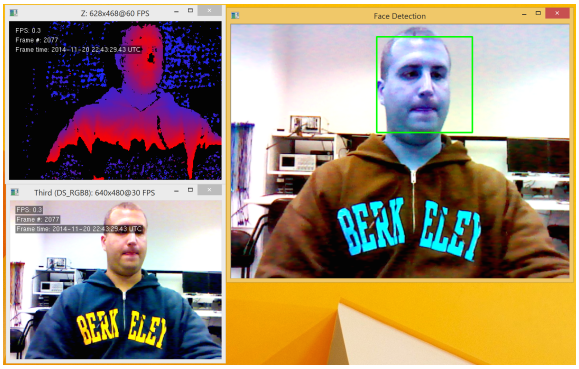

Fig. 6. Going counter-clockwise from the top right - Depth Map, RGB Image, and Face Detection

We have decided to use an Intel RealSense 3D Camera to handle face recognition. This camera was chosen since it has a small form factor and provides RGBD information at 30 frames per second. The RGB information is important in order to perform face recognition, and the depth information will be useful to allow the NERF gun to properly target the face once it is detected.

A limitation of this camera is that it must be connected to a Windows 8 computer via USB3, but it only came with a short cable. To get around this limitation, we ordered a 2 meter USB3 extension cable.

On the software side, we have C++ code that reads from the camera and uses OpenCV to detect faces. Figure 6 shows a demo of this behavior. Once the code figures out where the face is, it sends a POST request containing the x and y coordinates of the face on the camera picture, the depth of the face, and a timestamp. The POST request is send to a node.js server running on Heroku, which our microcontroller can then pull these readings from.

B. Controller

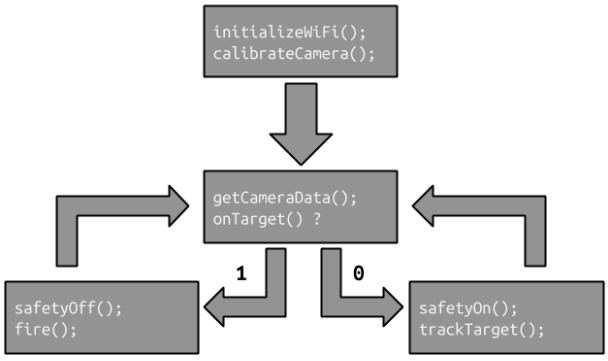

Fig. 7. Control flow graph of the turret controller

Finally, we have our MBED, which we are using to drive the actual aiming and firing of the NERF gun. In order to determine where to point the gun and when to actually shoot, we are relying on the measurements that are stored on our node.js server. To get these, we have decided to use WiFi for our connection scheme. While WiFi might not be the optimal choice if we wanted to track and shoot the target as fast as possible, we found it was the quickest way for us to satisfy our requirements. We used the Adafruit CC3000 chip in order to achieve this. Our MBED would send a GET request to the node.js server, which would return the coordinates of the target, which we would then use to aim and fire the gun.

Figure 7 shows a high level view of the software ran on the MBED to drive the turret. In the very beginning of the controller program, various initialization are performed before the turret starts operating: web connection is set up for getting camera data and camera is calibrated to ensure accuracy. Next, the MBED gets the camera data and checks if the turret is aiming at the target with some tolerance, if yes, it fires the NERF gun, if no, it adjusts the turret's orientation to try to track the target. Afterward, the MBED gets the camera data from the node.js server and checks if the turret is aiming at the target, and the cycle repeats ad infinitum.

VI. Real Time Networks and Timing

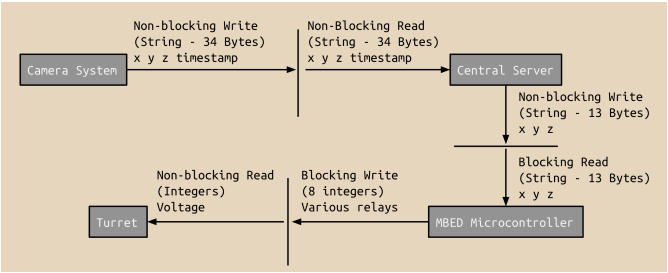

Fig. 8. Networking interactions between the camera, server, and MBED

A large part of getting our system to work involved getting our modules to communicate with each other in real time. To do this, we had to design our communication network to be able to control the NERF gun within a reasonable window of time to allow it to actually hit people.

Our communication network consists of three main entities talking to one another–the camera, the central sever, and the MBED. The interactions between these systems can be seen in the diagram. Most of the inter system communication was done using WiFi–we found that it was the easiest solution for us to implement that still gave us performance good enough to meet our criteria. We found that the latency of the WiFi components were roughly 100 milliseconds, which we found was perfectly reasonable for our system. Also, we made sure that we made as much of the communication non blocking as possible, so that our reads and writes would not hang if we could help it.

We ended up able to get our getting the communication between the camera and server to be a non blocking write to

optimize our network, while our MBED pulls from the server in a non blocking read, which follows the observer design patter (camera writes, MBED observers, so we don't get any conflicts). The only blocking communication we had was the actual control of the turret with a blocking write to control the MBED.

VII. Realtime Behavior

There were a few techniques that we used in the class to try to get as best performance out of our system as possible. We programmed in hysteresis into our system for aiming the NERF gun, so that we wouldn't have this wobbling back and forth cycling effect where our system can't reach the exact position that it wants to. We also had a delta for our firing window in order to give us some cushion about when we could shoot, which also helped our system achieve it's target more quickly.

VIII. Future Goals

While we did manage to meet our requirements with the robot we presented, there are a few areas of improvements that we can work on if we were to continuing developing our project.

On the hardware side of things, we found that our wiring and packaging of all the components was not done very well–the components have a very large footprint, with wires spanning all over the place. We sometimes found that our wiring, because of the packaging, was limiting how much our turret could rotate. Additionally, the wiring schematic was not particularly planned out to minimize the footprint initially, so a smarter packaging system would improve our system greatly.

Additionally, we want to look at speeding up our target acquisition and firing so the turret could perhaps track a moving target. With our current motor and WiFi setup, we can find and hit a static target pretty easily, but our system does not move fast enough to track a moving target. Methods to improve this would be to try a faster network protocol, such as using wired connection instead of wireless, or optimizing our code in order to speed up facial recognition or controlling the movement of the robot. We could also modify our algorithm to make the turret move fast if our target is further away in order to lower the number of times we need to poll the camera.