Team Lux

Jonah Lou, Jenny Hoac and Chris Woodall

EC535 :: Embedded Systems
Professor Coskun
Spring 2013

Abstract

The Laser Turret uses a webcam to track objects and then shoots a laser in the direction of the tracked object. The goal of is to process the images taken by the webcam and, using the results that comes from processing the images, control the servo motors appropriately to direct the beam of laser at the object. In the time given, we were able to accomplish the goals that we set for ourselves. Using only the gumstix board, we were able to successfully interface with the webcam and process the images taken. We were also able to provide the proper pulse width modulations to the servo motors to correctly aim the laser at the target that the webcam processed.

Introduction

The main goal of this project was to be able to implement some sort of target detection. The inspiration to make a laser turret out of it came from the popular game, Portal. The job of a Portal turret is to detect motion and, upon detecting motion, target the moving object with a laser. Motion detection is important for security measures – to be able to track intruders – as well as for novelty reasons (ie. detecting a human's movements can be used as a game control). Integrating motion detection with a laser is also useful for security measures as well as for military application, since the laser can be substituted for any type of target indication device.

There are two main components that make up the Laser Turret. The first component is the webcam, which is used in finding the moving target. The second component includes the servo motors, which drive the laser that points at the moving target. The servo motors are clamped onto a stand and attached to the top of the motor is the webcam. Integrating the webcam with the gumstix board was rather challenging as finding the proper software that could interface between the two was difficult. However, uvccapture turned out to be the best at providing the proper drivers that the gumstix could understand and control the webcam with. After the libraries were installed, processing the images became the next step. The processed image provided the coordinates of the detected object (a number between 0 and 320 on the x-axis and 0 and 240 on the y-axis). Then the coordinates are converted into the proper inputs for the servo kernel module. The servo kernel module "pwmservo", is a character driver at "/dev/pwmservo" and receives the coordinates through file system writes. The format for the servo inputted coordinates are "pNUM" or "tNUM" where NUM is an integer from 200 to 410, which map between 0 degrees and 180 degrees. The servo signals are sent using the PWM peripheral on the PXA270, but are sent in bursts of 4-5 every 100ms to prevent overheating.

Upon project completion, we were able to take images and process them within one second. We allocated another second for the servo motors to move the laser to the targeted area. After all the implementation, we realize that the servo motors do not need one second to move to the next targeted area. This could be one area that we could improve upon in the future.

Design Flow

Quantity	Component	Market Price PPU	Description
2	Servo Motors	$10.00	
1	Gumstix	$150.00	
1	Physics Experiment Stand and Clamps	$0.00	Gifted from BU Physics Department
1	Laser Pointer	$10.00	
1	HD Webcam C270	$40.00	720HD webcam from Logitech
1	Protoboard	$3.00	Provides a small circuit board for easy wiring of the gumstix to the servo motors and power lines
3	.1" Header (3 pin)	$1.00	
2	Metal L pieces	$0.00	Scavenged scrap metal

Table : Parts Table

The components listed above are all the parts needed to create the Laser Turret. The protoboard is a small board used to easily connect the gumstix pins to the proper servo wires. The .1" headers are also used to ease the process of wiring the gumstix board. The metal pieces are attached to the servo motors to provide a base for the laser to rest on. The two servo motors provide a pan tilt system for the pointing of the laser. The lab stand and clamps are used to hold the servo motors in place and the webcam is mounted on top of everything. Shown below are Figure 2: Block Diagram and Figure 1: Protoboard with headers which demonstrate how the hardware and software interface together.

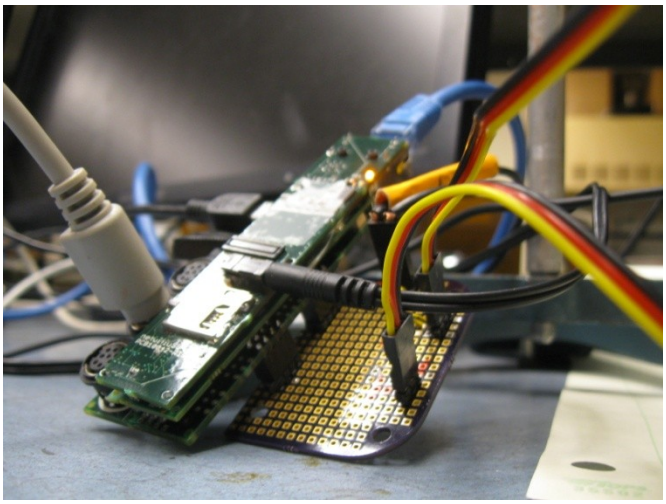

Figure : Protoboard with headers

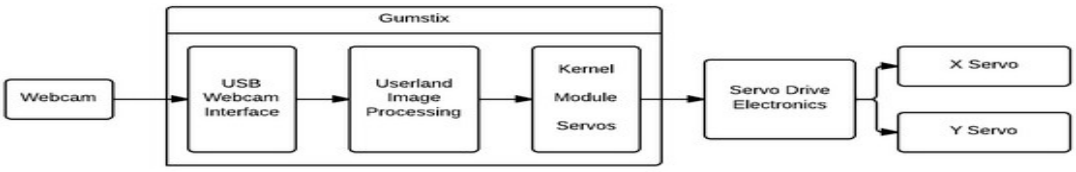

Figure : Block Diagram

Work Distribution

Christopher Woodall worked on the servo device driver, which runs the two servo motors that drive the direction of the laser. He also made the framework that combined all of the components as well as the mechanical parts of the Laser Turret.

Jonah Lou worked on the image processing code, which detects objects by comparing the current image with a base image and calculates the correct area to target.

Jenny Hoac worked on the user space code, which does calibration, accesses the two separate components, and converts the image processing output into the proper servo device driver inputs.

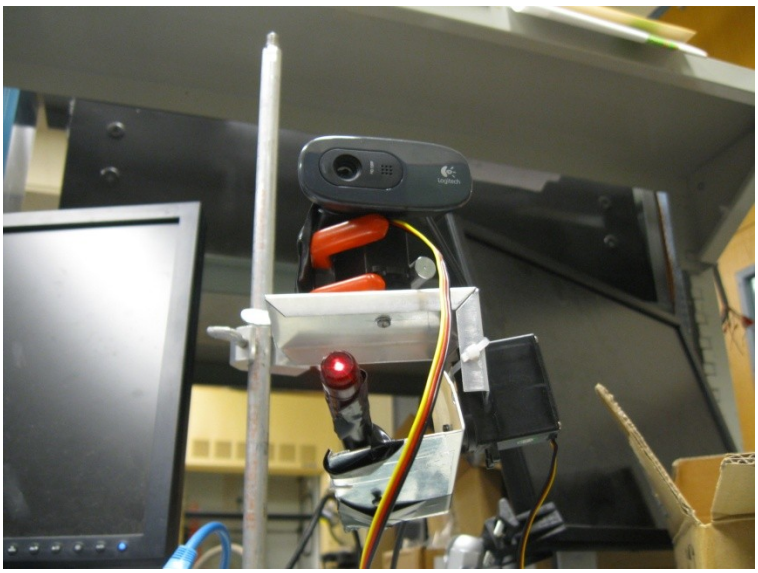

Figure : Completed Laser Turret

Project Details

Image Processing

Research

Since we did not have much knowledge of a motion detection algorithm, we did research in this matter. We found that the best method to implement is to compare every current image with a base image that is taken.
In addition, we had initially wanted to use OpenCV as it offers us a lot of capabilities in terms of computer vision. However, we were unable to compile the libraries necessary for Open CV. After researching several image librarys, we decided on CImg [1] because it is all contained within one header file (*CImg.h*). Since CImg works with C++, our image processing algorithm is written in C++.

Webcam

The camera that is used for the Laser Turret is the HD Webcam C270. In order to get this webcam to interface with the gumstix, you need to download the proper libraries. [2] The link to the latest libraries can be found in the reference section. It is also possible to find pre compiled versions of the library for the gumstix. The libraries need to be installed in a specific order:

```
v4l1-compat.ko
v4l2-common.ko
compat_ioctl32.ko
videodev.ko
uvcvideo.ko
```

After installing these drivers onto the gumstix, it will recognize when the webcam is connected and add a /video1 into the /dev/ folder. After you verify that this folder has been made automatically, the webcam driver has been installed properly.

Algorithm

The way the algorithm works is it takes a base image and stores it. Every subsequent image that is taken will be subtracted from the base image. From this, the area with the highest contrast will be the area that is targeted. However, just subtracting the images brought up a problem. In the real world, there is a constant matter of difference in lighting; as a result of this, there is simply too much noise to deal with. This problem can be solved by first changing the images into grayscale [4] before taking the difference between the images [5]. After differencing the images, a threshold is applied to the images so that only the areas with the most change will be detected. After the image has been properly processed, a 20 by 20 pixel grid is swept across the image to find the grid with the highest amount of white pixels. The white pixels in this image indicate the contrast between the base image and the current image. This being said, the grid with the highest amount of white pixels means that this grid is the area at which there is the most change. The x and y coordinates of the center of that grid will then be the target for the laser to point at. The series of images below shows the development of the various stages of the algorithm.

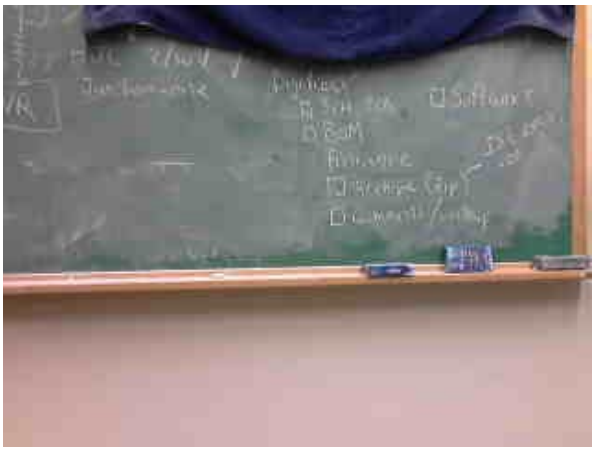

Figure : image1

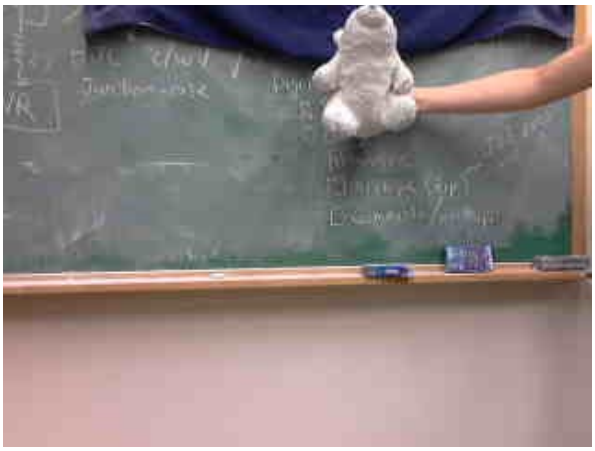

Figure : image2

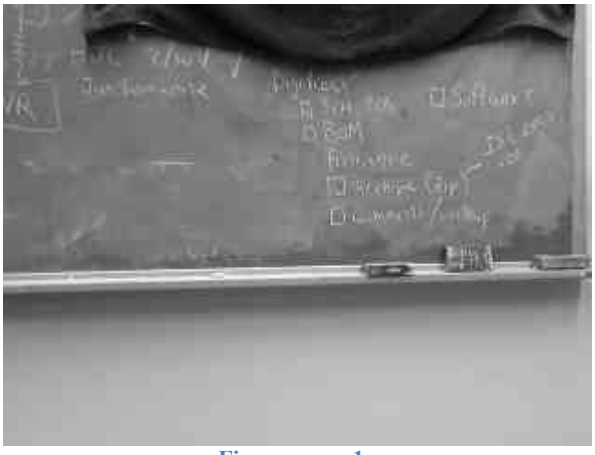

Figure : gray1

Figure : gray2

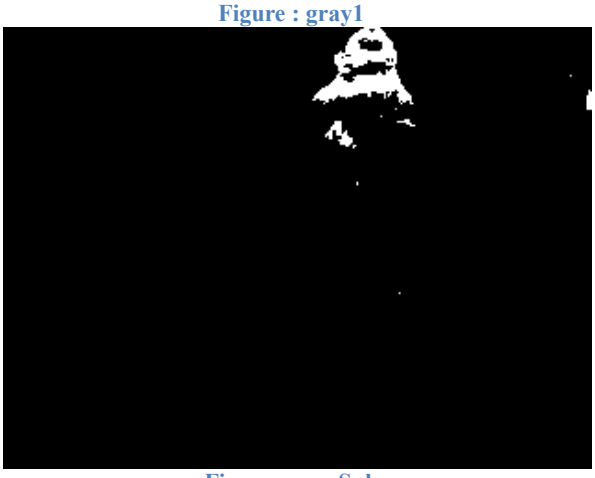

Figure : graySub

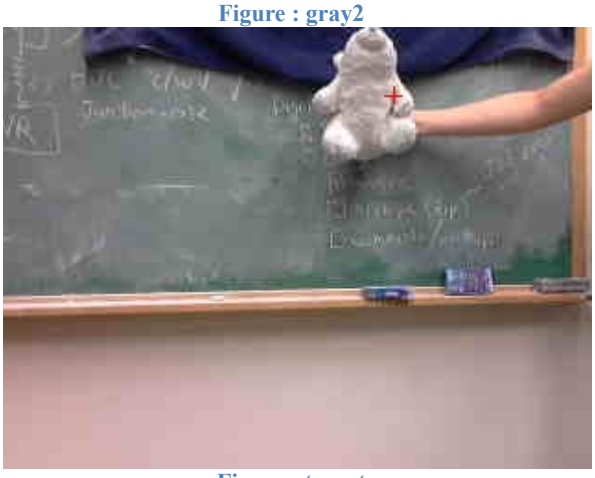

Figure : target

Here, Figure 4: image1 is the base image and Figure 5: image2 would be one of the subsequent images taken. To start the algorithm, both are converted into grayscale, as shown. The difference and threshold is then applied and returns an image such as Figure 8: graySub. Finally, the target is found, as shown in Figure 9: target.

Target Coordinates to Servo Module

Originally, we had thought that we would need to take the rectangular coordinates obtained from image processing and convert them into polar coordinates. This would mean coming up with a way to calculate the distance the target object is from the camera. However, once both image processing and servo driving was completed, we realized that the servo motors only take in a pan value and a tilt value – the pan value corresponds to a coordinate in the X axis and the tilt value corresponds to a coordinate in the Y axis. This means that we can approximate the target linearly. We take the coordinates outputted by the image, which range from 0 to 320 on the X axis and 0 to 240 on the Y axis and map them with a number between 200 and 410 in the X axis (for pan) and 200 and 410 in the Y axis (for tilt).

While we gain calculation ease from doing this, we trade off the ability to have images with too much depth. Any depth discrepancy will result in an offset when targeting. Additionally, we add in the extra task of calibration. Before we can run our code, we have to find the min and max pan and tilt values that will correspond to the real world limits of our image. We also realized upon testing that the image taking in from the camera is mirrored from what it is in real life. For example, during one of our test runs, we received values that mapped (0, 0) to (p330, t316) and (320, 240) to (p255, t380).

Main control code

The program is first fed in the calibration coordinates (panMin, tiltMin) and (panMax, tiltMax). Then every time the code is called, it takes an image and runs the image through the image processing code. Once image processing is complete, it transforms the xy coordinates obtained from image processing and transforms the coordinates into pan and tilt values that fall under the calibrated range. Finally, it opens the servo module and writes in the calculated pan and tilt values in a form readable for the module (first t[number] then p[number]).

Servo Module

Hardware Setup

In order to change where the laser was pointing we needed a way to actuate it. At first we were going to actuate a mirror, but the scheme of directly actuating the laser pointer proved easier to mount and was more accurate. We utilized a pan tilt scheme, as seen in Figure : Pan Tilt System, for two reasons: it has a simple construction and the basic structure was already built earlier in the semester as a side project. The mathematics for a flat plane intersecting the spherical coordinate system requires some additional math than an X-Y scanner setup, but in our application could be mitigated by good calibration and mapping.

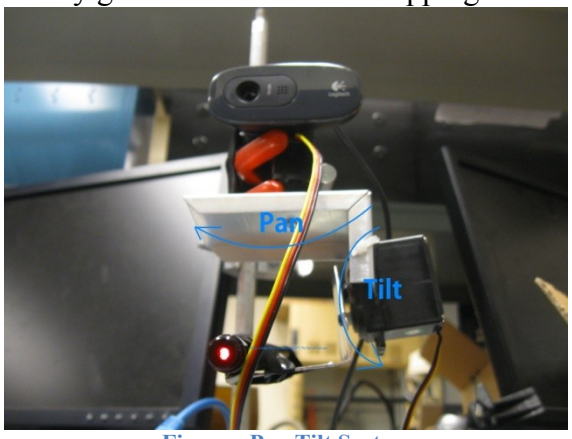

Figure : Pan Tilt System

RC Servomotor Signals

The servo motors we have are standard RC servomotors which take signals like the one in Figure : Standard RC Servo Signal. The signals are essentially a pulse width modulation with a period, or frame rate, of 20 ms and a pulse width between 1 ms and 2 ms where the pulse widths determine the angle of the servo, in our case it was over a range of 180 degrees. There are three solutions for implementing the servo signaling: (A) using the PWM peripheral (PWM0 and PWM1) on the PXA270; (B) timer based soft pwm; and (C) using a dedicated microcontroller to generate the signals which accepts angles over UART, SPI or I^2C from the gumstix. Which solution is appropriate is determined by the specific needs of the system. For example, in this case (B) was out of the question because in order to get the resolution we wanted we would have had to use OS timer interrupts on the order of 1us or less, which brings Linux to a halt.

Figure : Standard RC Servo Signal [6]

Figure : Actual servo drive signal with scale at 100ms/div, and .5V/div

In practice we implemented (A), which utilized the PWM outputs on PWM0 (pin 16) and PWM1 (pin 17). However, there were problems with the minimum frame rate of the PXA270's PWM peripheral, which ran on a 13MHz clock. This 13MHz clock could be divided by 64 and then counted up to 1024 to yield a frame rate of 5ms. After doing some research and experimentation [7] we found that the servomotors will go to the right position for a little while with a 5ms frame rate, but run hot and stop working on the order of minutes. The solution we implemented was to have an OS Timer on Match 1 running at about 10ms which toggles the PWM signal on and off allowing 4-5 5ms frames through every 100ms. This was a solution arrived at experimentally and which prevents the servomotors from running hot and from jittering. The result is we have 10bit resolution from 1ms (200) to 2ms (410), giving 210 discrete positions, which is more than enough for our needs.

Interfacing With the pwmservo Kernel Module

The pwmservo kernel module interacts with userland as a character driver with major mode 61 and minor mode, and is positioned at /dev/pwmservo. PWM servo only has one method of interaction, which is write - a status readout was never implemented but could be useful. The basic interactions are catalogues in Table 1, and should be delimited with a newline and sent as two separate write system calls for pan and tilt.

Function	Command	Value	Example
Pan Servo Control	p	Integer value from 200 to 410	p300
Tilt Servo Control	t	Integer value from 200 to 410	t393

Table : pwmservo write commands

Summary

Our laser turret is able to detect a target and point the laser in the target's direction in a 2 second timeframe. While we were able achieve the general motion detection, target aiming functionality that we put forth, we still have some parts that we can definitely improve on.

First of all, the 2 second pause is due to the fact that we want to give the program enough time to process the images and drive the motors to the desired position. We could definitely have sped up this wait time but we did not want to risk overwhelming the servo motors – we wanted a smooth laser to target transition.

Our implementation also required us to have tight setting constraints. Since the images were converted to gray scale, objects that were vastly different in color from the background but had the same shade in grey scale may not have been detected. Therefore, motion detection worked best when there was high contrast between the target and the background. In addition, our calibration technique required the image backdrop to be fairly flat and the target to be fairly close to the background in order to simulate the flatness of the actual image. Otherwise, the spot we actually hit would have been slightly offset from the spot we wanted to hit.

Ideally, we would have wanted our laser turret to work instantaneously and in any setting but we had limited resources and implemented something that proved to work well regardless.

References

[1] The CImg Library. http://cimg.sourceforge.net/
[2] Latest Drivers for V4L. http://git.linuxtv.org/media_tree.git
[3] Convert RGB image to grayscale using CImg library. http://sorj.de/?p=168
[4] The Cog Shop. “Motion Detection and Segmentation. http://www.ai.mit.edu/projects/cog/VisionSystem/motion_detection.html
[5] Robot Platform. “Hobby Servo Motor Tutorial”. http://www.robotplatform.com/knowledge/servo/servo_tutorial.html
[6] Standard RC Servo Signal. http://upload.wikimedia.org/wikipedia/commons/7/7b/ServoPwm.png
[7] Servo Protocol. http://www.rcheliwiki.com/Servo_protocol