0TTER
0g Tracking and Targeting Experiment in Real-Time

By: Brandon Bussjaeger
Under: Dr. Foaad Khosmood

Background

The 0TTER experiment was designed for NASA's Microgravity University program. This program selects teams from colleges across the US and directs them in building an experiment to be flown in a Zero-G plane. This plane completes a parabolic arc every few minutes, giving the passengers and experiments inside a simulated low-gravity environment for 20-30 seconds at a time. The experiment will be flown on August 5th, 6th, and 7th out of Johnson Space Center in Texas. Our team selected by NASA consisted of the following:

Christian Hume	4th year EE	Team Lead and Electronics
Brandon Bussjaeger	4th year CSC	Lead Programmer
Jenna Becker	3rd year ME	CAD Design and Mechanical Assembly
Bodin Rojanachaichanin	3rd year ME	CAD Design and Mechanical Assembly
Sara Lillard	3rd year AERO	TEDP Writeup and Logistics
Dr. John Oliver	EE Department Professor	Faculty Supervisor
Robert L. Hirsh	JSC Employee	NASA JSC Contact

The intent of the experiment was to track a moving object in real-time while both the object and tracking system were free-floating in microgravity. The exact specification of the experiment is below.

> "Develop and demonstrate technology to allow a **floating experimental rig** to continually determine range/bearing to a target. The target could be a special color/shape (i.e. have a "fiducial marker"), but preference would be given to a target point that can be selected at the start of each parabola airplane, and then tracked from that time forward. In addition to **finding and tracking the relative location of the target point**, the experiment must visually verify the accuracy of the solution by **continually controlling a "laser pointer" to point at the target** during the entire parabola. This would be accomplished by having a motors move the laser pointer, or by thrusters that could re-orient the vehicle, or a combination of both. The pointing accuracy desired would be better than 1 degree of angular. Since it must be demonstrated inside an airplane, assuming a **target range of about 3 meters**, this would mean the laser pointer would need to **stay within 5cm** of the designated target as the experiment "floats" during an entire 20 seconds of a 0-g parabola. Of course, obtaining even smaller errors in pointing accuracy is better. " - Robert Hirsh

My part was specifically developing a software and hardware solution to track an object in front of our rig. This involved two major pieces:

1. Utilizing some computer vision processing to locate an object in front of the rig. This tracking had to be completed in real time since the object being tracked would constantly be moving. Tracking would involve interfacing with some sensor, such as a webcam, and process that data on a laptop.
2. Send the location of the object to a mechanical setup to aim the laser pointer at the target, proving that we were tracking the object. This communication between software and hardware was implemented in real time as well.

Architecture and Design of Software

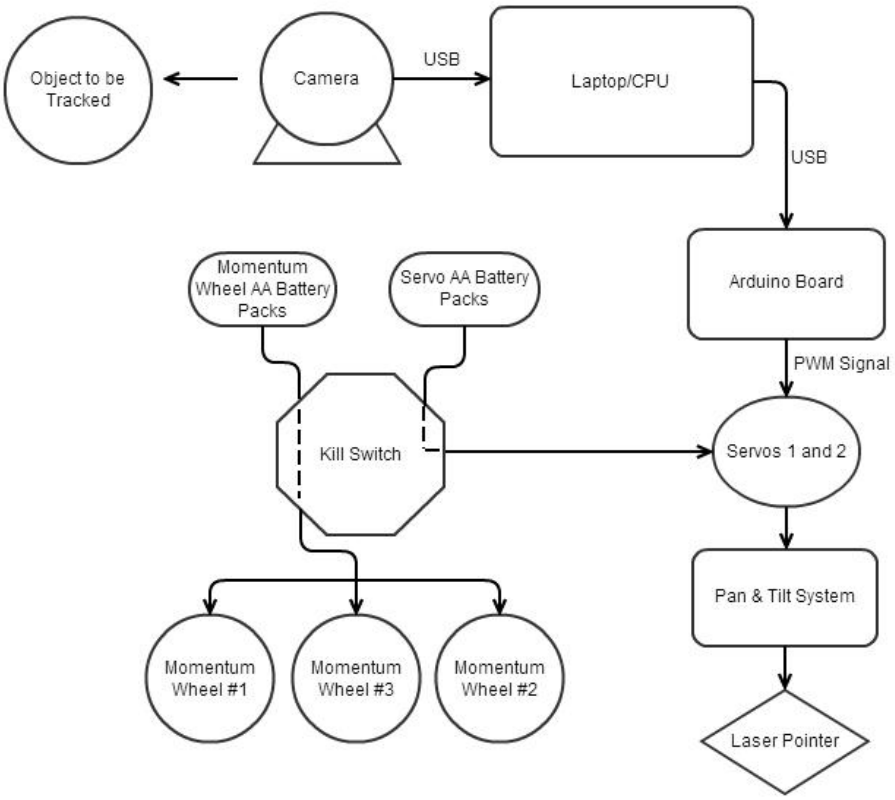
The software design had to fit a wide range of requirements. A powerful enough sensor was required to detect a tennis ball sized object at 10 feet away, accurate up to an inch. This data would have to be processed in real time. A control loop would have to be run to move some actuator, which in turn would move the laser pointer to the object.

Sensor

The sensor chosen was a HD webcam. This webcam, a (model name here) gave us a high resolution image able to separate a small object at a distance from background noise. The next requirement that entered the system was the rate in which the object detection must be done at: 30 frames per second, or 33ms.

Vision Framework

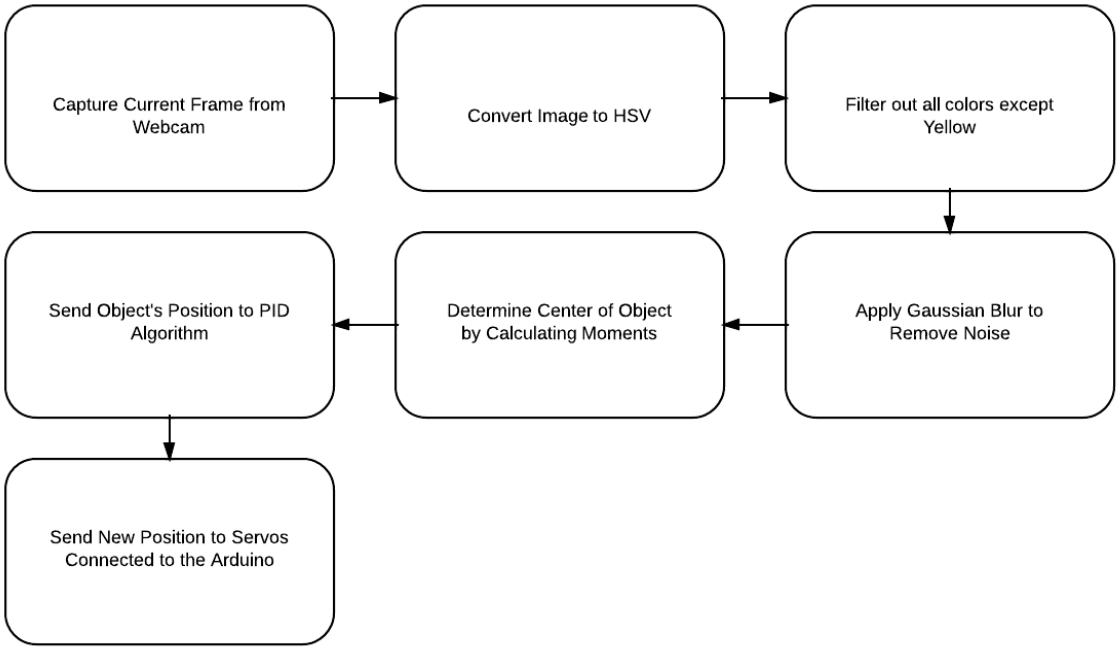
OpenCV, an open source computer vision library was used in conjunction with the webcam to collect images from the webcam over USB and process them. OpenCV provides the functionality to interface easily with most webcams, as well as performing operations on the image such as color filtering, blurring, image moment calculation, basic shape detection, and providing a simple GUI. OpenCV was run under Windows 7 using the Microsoft Visual Studio 2010 environment.



Outline of Software

The program runs within the Microsoft Visual Studio 2010 environment. The software interfaces between the webcam and Arduino Uno controller to move the servos. This seven step process is outlined below.

1. Capture an image form the webcam. The webcam is initialized once using *cvCaptureFromCAM(0)*. Subsequent calls to *cvQueryFrame()* return the current frame the webcam has in it's buffer. This will attempt to poll faster than the webcam can capture a new image.
2. Convert the captured image from a RGB format to HSV using *cvCvtColor()*. Converting to a Hue/Saturation/Value style image allows for easy color filtering in the next step.
3. Apply a filter to the colors in the HSV image using *cvInRangeS()*. To detect yellow, the code only keeps the pixel location of any pixel with a hue between 20 and 24 (out of 180). The saturation and value components of the HSV image are also used. These are calibrated at startup. This results in a binary image with a white pixel where there was a color in our range, and a black pixel otherwise.
4. A gaussian blur in a 5x5 grid on each pixel is run over the binary image to remove any noise using *cvSmooth()*.
5. The moments of the image are calculated with *cvMoments()*. The moments represent where the "center of mass" of the binary image is. That is, it will give the average location of all of the white pixels on the screen. For this, it will give us the center of the object we are tracking.
6. With the position of the tracked object relative to the center of the camera, a PID algorithm is used to determine a new position for the servos to point the laser pointer at the target. A slightly modified version of the Arduino PID library in C++ is used.
7. Finally, the new servo positioned determined by the PID algorithm is sent to the Arduino through a USB serial communication. The Arduino is set up to only read absolute X and Y servo values and set the servos to that position.



Testing Procedure:

1. Connect the Arduino and webcam via USB to the laptop.
2. Open Microsoft Visual Studio, selecting the most recent project.
3. Power on the servo motors and gyros via the main power switch on the frame.
4. Run the Visual Studio project. You do not need to recompile.
5. Take the yellow ball and hold it a few inches from the webcam. The webcam should move slightly and begin tracking the ball.
6. Back the ball up a few steps 5 feet away and hold at chest level, then begin metric. The score for each column should reflect both the percentage of time the laser is on the ball, and how quickly an error is corrected.

On the grading rubric, a 5 indicates that the laser pointer stays on the ball >90% of the time. If the pointer ever leaves the ball, it is for a small moment of time and quickly recovers. A 4 indicates that the laser pointer may leave the ball, but always recovers within a second or two. A 3 indicates that the laser consistently leaves the ball until the direction is changed, or the ball slows down. The laser should be on the ball during the start and end of the test (never permanently lost tracking). A 2 indicates that the ball was tracked for part of the test, but lost tracking partway through. A 1 indicates that the ball was lost during the start of the test.

Keeping the ball at chest level, move the ball from your right shoulder to left with a travel time of one second to each shoulder.	**5**	**4**	**3**	**2**	**1**
Keeping the ball centered horizontally, move it up and down from waist to chest level with a travel time of one second each way.	**5**	**4**	**3**	**2**	**1**
Move the ball in an X using the same area and speed covered in the previous two tests.	**5**	**4**	**3**	**2**	**1**
Hand the ball off from one hand to another, outstretching your arms each time.	**5**	**4**	**3**	**2**	**1**
Toss the ball from one hand to another, keeping your hands two feet apart (shoulder width). This is faster than the program is calibrated for, so some loss is expected. Recovery time should be low after tracking is lost.	**5**	**4**	**3**	**2**	**1**
Toss the ball a few feet into the air. This is faster than the program is calibrated for, so some loss is expected. Recovery time should be low after tracking is list.	**5**	**4**	**3**	**2**	**1**

Initial Test Results using Direct Control Only:

The testing procedure outlined above was used to determine the difference between using a direct control (Proportional only) algorithm versus a PID control algorithm. First the direct control was used, and the test above was run.

- Slow horizontal test: 5
- Slow vertical test: 5
- Slow X pattern test: 5
- Medium horizontal test: 4 – Laser had to catch up at ends
- Fast horizontal test: 3 – The laser either had to catch up or lost the ball each time
- Fast vertical test: 2 – The laser tracked the ball to the top, but did not follow it down

During the slow tests, the tracking performed well. However, as the tests increased in speed and acceleration or directional change, the laser would not keep up with the ball. During the fast vertical test, the ball was lost while being thrown in the air.

Second Test Results using PID Control:

I implemented the Arduino open source PID library into the control portion of the code. This is run on the laptop and absolute servo position is sent to the Arduino after processing. To calibrate the PID values, I used a standard calibration procedure.

This standard procedure took three steps. To simulate the ball moving rapidly, I placed identical objects on a table. I would cover one object at a time with a white sheet, forcing the tracking code to move to the other object. I could then simultaneously reveal the covered object and hide the current one to switch targets.

First, the proportional value was dialed in. I slowly increased the proportional constant until the laser slightly overshot the ball when moving from one object to another. Then the integral constant was increased slowly until the laser overshot the ball by about 15%, which in this setup was about 4 inches. Finally, the derivative constant was increased until the laser no longer overshot the edge of the ball. Then the same tests were run.

- Slow horizontal test: 5
- Slow vertical test: 5
- Slow X pattern test: 5
- Medium horizontal test: 4 – Laser had to catch up at ends
- Fast horizontal test: 4 – The laser either had to catch up at the ends
- Fast vertical test: 3 – The laser tracked the ball to the top and bottom multiple times.

There was a slight improvement during the quicker tests after implementing the PID control. Fortunately, the ball will be “free floating” in microgravity during it's experiment, which will not require the tracking software to make quick adjustments.

Conclusions

Overall, the vision tracking software and control algorithm performed very well. The system runs at a constant 30 frames per second, which is the maximum allowed by the webcam. If I had to change part of the system to increase performance, I would change the servo motors to have a faster response time, and run a webcam with a higher frame rate. In addition, moving the vision computation from the CPU to the GPU would allow more features to be added, such as a more powerful blurring calculation or circle detection, without dropping the frames processed each second below 30.

I am confident that when put in microgravity during the experiment at the end of July, it will successfully track the ball in a weightless environment.