Abstract

The project’s goal was to design, build, and successfully operate a motion tracking robotic arm. The challenge was developing a program that could read the pixels from the image and return coordinates of an object in the image. The next phase was to write a code to communicate to the Arduino board that powers the servos to move accordingly. The program used for image processing was Intel’s OpenCV. After trial and error, OpenCV's facial recognition program was used because of its accuracy. Once facial recognition was possible the next phase was to communicate to the Arduino board. To achieve serial communication between the computer and the Arduino board, the program Processing, developed at MIT, was used. Processing was chosen after failed attempts at using the programming language C++. Algorithms were designed to convert pixel locations into degrees of motion the arm would move. After there was successful locating and physical tracking of a face, the construction of a mount for the laser pointer and base to hold the arm steady were completed. The arm is attached to a wooden base board so it remains secure while moving to keep it situated in position. A drilled out servo body with a hole drilled through it acts as the holder for the laser pointer. The robot successfully detects a face and the arm follows it. If given a longer period of time, coding using C++ instead of Processing may have improved the motion of the arm due to processing’s memory limitations.

Introduction

Robots have become one of mankind’s greatest tools since they were first crafted decades. Just like all other human made tools there's always room for advances. One of the many challenges encountered when designing a robot is how is it going to sense its surroundings. Over the past few years advancements in computer science has increased a computer’s ability to analyze video images. The next step would be to integrate this technology into a robot. A robot that can view the world through the lens of a camera and be able to understand what it is looking at is the next step for robotics.

Just how can a computer look at an image and understand anything on it like a human is capable of? A digital picture is a collection of very small dots of different colors, known as pixels. The goal is for the computer to examine these dots, and based on information stored in the computer, attempt to make out what the image is. After the image is processed and the object is targeted a program is used to relay signals to the servos in the robot. This allows the camera to follow the object to watch where it goes without human intervention. Security cameras are great examples of this technology in action .These cameras are controlled by motion sensors. When they detect motion, the camera analyzes the images then moves the camera appropriately to center the activity on the screen. This ensures that any movement is fully in the frame and nothing is missed.

To demonstrate this technology the projects objectives were to design, build, and successfully operate a motion sensing robotic arm. Using a static placed camera and a LynxMotion Robotic Arm, a program needed to be created to relay information from the camera to the arm. This was done using two different programs. One was to process the images received from the camera, the other was to communicate to the Arduino board that controlled the arm. Attached to the arm a pointing device will follow a moving object within the camera’s field of view.

LynxMotion Arm and Arduino Board

The company LynxMotion builds robotic arm components and parts (LynxMotion Robotic Arm). A completed arm that was previously built for a different application was used for this project. Because the application varied from the original intended design, modifications had to be made before beginning.  Initially, the robotic arm had additional servos, joints, and a gripper on the end that was not necessary.  We removed the unnecessary components, as seen in the figures 1 and 2 below.

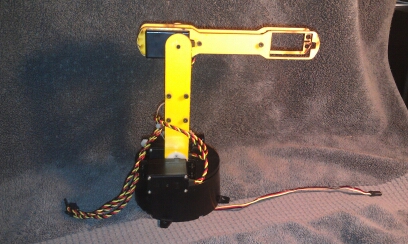


Figure 2 - Modified LynxMotion Arm

Figure 1 - Original LynxMotion Arm

Removing the additional parts also decreased the strain on the needed servos tremendously by reducing added weight.  The arm is capable of moving in the *x* and *y* direction.  The *x* direction rotates about the black base seen in the photos above, and the*y* direction operates by the top joint and visible servo.

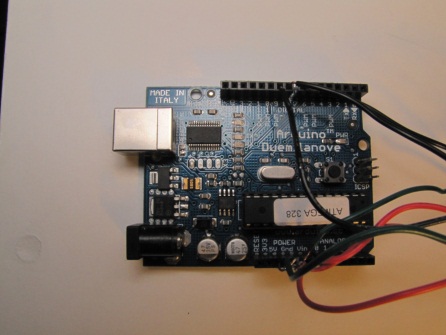
 Provided with the LynxMotion arm was also an Arduino board. The Arduino board is responsible for receiving signals from the computer and translating that to motion into the servos controlling the robotic arm (Arduino). A picture of the Arduino board is found in figure 3.

Figure 3 - Arduino Board

The servos controlling the arm are then connected to the Arduino board. The Arduino board was then connected to the computer using a USB cable.  Processing was the computer coding language used for this project.

Computer Coding and Integration

To begin writing the code for blob detection, the program OpenCV was chosen. OpenCV is a project created by an Intel Research initiative to advance CPU-intensive applications with image processing as part of a series of projects including real-time ray tracing and 3D display walls (OpenCV). Included with the documentation for the program, were basic examples of a blob detection library. This library analyzed an area of similarities in an image based on color.

Before writing code for blob detection, a code that would detect color was created.

This method of image tracking proved to be very ineffective. Written in OpenCV, the camera sees vivid colors in its display.  By clicking on an object of a distinct color on the computer screen, the program defines it to be the color to track. It then will continually follow the object while placing a red dot at its approximate center.  The red dot would be the ideal location to point the laser pointer once the robotic arm is integrated with the camera color tracking program. Very good lightning was needed to detect the different colors.  Also, overly reflective surfaces were not seen by the program as an object, but as just a light source. These limitations caused the tracking to be inconsistent and unreliable, and therefore impractical in real world applications. A screenshot of the color tracking program is seen in figure 4.

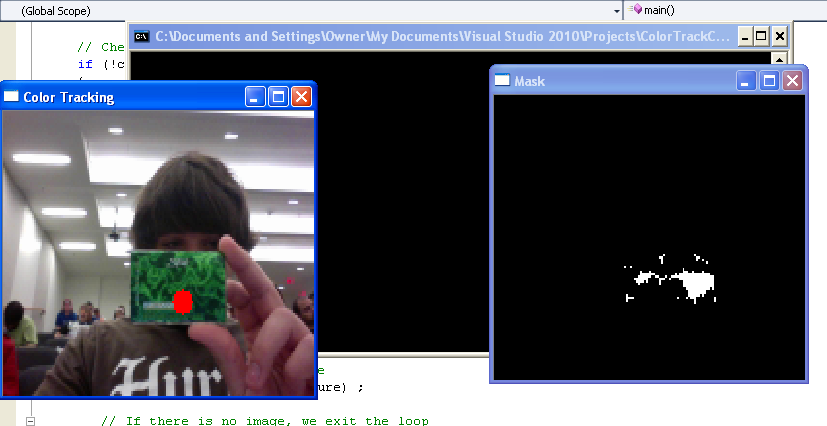


Figure 4 - Color Tracking Program

After achieving basic motion detection the next objective was to work with serial communication between a computer and an Arduino board. The coding language used for this was C++, and the Arduino coding library. The goal was to construct a basic application that will provide live servo control from a computer. A screenshot of this application is seen in figure 5.

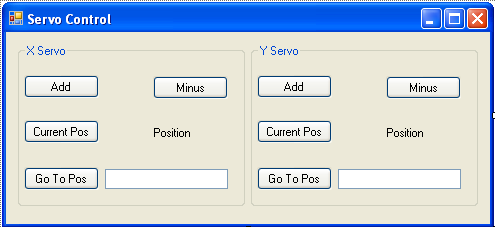


Figure 5 - Real Time Application

The code controlling the Arduino board is a statement that takes the data from a serial input and processes it based on one of three specific cases. One of these cases reads the position of the servo, another adds a single degree to the position of the servo, and finally one subtracts a degree from the position of the servo. All of these commands result in real time controlling of the LynxMotion robotic arm.

The next objective was to integrate the code controlling motion tracking with the code controlling communication to the robotic arm. This would complete the overall goal to have the arm follow a moving target. However, a serious problem occurred while trying to complete this. Each situation had its own unique environment behind the camera that would need to be analyzed so that the arm could accurately follow the target. Another problem was integrating the OpenCV to communicate with a serial connection. Standard ways of serial communication in C++ would not work for this case because they are written in a type of system C++ and not a standard (STD) type of C++ which OpenCV is capable of understanding.

To solve the first problem, the original servo application was put into a configuration module for the servos. The algorithm designed to control the Arduino board for motion of the arm was as follows.

((Xposition)/(ScreenWidth/(ConfigXmax-ConfigXmin)).

The ConfigXmax and ConfigXmin variables are set variables of the minimum and maximum movement the arm can do while still in the frame. The calibration application is seen in figure 6.

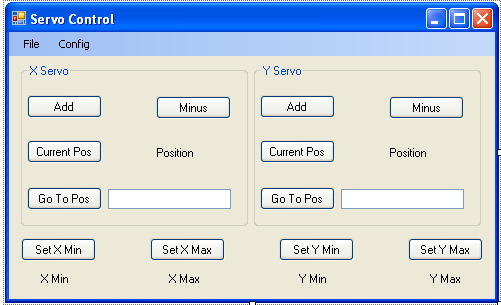


Figure 6 - Calibration Application

The only way to solve the second problem would be to re-write the application using another type of coding language. For this, the language of Processing was selected. Processing is an open source programming language developed at MIT Media Lab. It is used for integrated development environment (IDE), electronic arts, and visual design communities with the purpose of teaching the beginnings of computer programming in a visual context (Processing). Using processing did not eliminate using OpenCV. Rather, it acted as a substitute language for C++.

The new language allowed for two significant upgrades to the code. The first significant update affected the motion tracking. Originally, motion tracking used OpenCV’s blob detection. This proved to be very good in theory, but during application picked up too many minor movements, causing the arm to overreact and not pick up on the main target. Because of this, blob detection proved to be too inaccurate to use. To correct this, blob detection was changed to a more advanced component of OpenCV, facial detection. A screenshot the facial detection program running can be seen at figure 7.



Figure 7 - Facial Recognication Program

Both of these methods were significant improvements over the first method of tracking used, color detection.

The second update to the code was within the Arduino board coding itself. Rather than taking in a serial number and only being able to increase or decrease the servo position by one, the new code could take in 2 individual numbers. One number for the ID of the servo, and one number for the direct position of the servo to be set to. This allows the arm to react much faster to movement and send direct locations based on the same motion algorithm as before. Having direct locations also allows us to overcome part of the delay with the Processing coding language. Because Processing is a java derivative it is not meant for high memory allocations. It has great delay when running code as big as the code needed for this project to be successful. A way to fix this would be to adapt a custom serial communication library within a more powerful coding environment, such as C++. This would allow for instant reaction time of the arm and not a delay within the loop of the main program. Doing so would accomplish a smooth constant tracking of motion in the frame. Currently, it takes a few moments to find the new position and calculate where to move to. Although this is not ideal, it sets a solid framework for advancement in the future.

Mechanical Integration

*Laser Mount*

A critical component of the motion tracking arm is the laser pointer. The LynxMotion arm required several modifications to accept the laser. The arm originally had components to allow movement in the x and y directions and a rotating gripper attached that could open and close. The gripper was not necessary for this application, therefore it was removed and replaced with a mount to securely support the laser.

The laser mount was designed to be attached to the arm where a gripper servo was formerly mounted. It was imperative that the mount could be attached to the arm in the same fashion that gripper servo was. The mount also had to be lightweight and allow access to switch the laser on and off.

It was found that the servo was a very good template to base the mount on; the remaining issue was what material to use in the mount and how to manufacture it. The first design called for a near exact replica of the servo to be printed with a rapid prototyping machine. The replica would have a half inch hole going through the center longitudinally and two threaded holes perpendicular to the main hole. The laser was to be inserted through the center hole and secured with two setscrews. However, to design the piece and print it out would take more time than was allowable.

The next design was to be bent from a flat piece of aluminum stock ¾” wide and 1/8th of an inch thick. The mount would resemble the Greek letter omega, but with four 90 degree bends, as shown in Figure 8. Like the first design, this mount would have a hole down the center. Hose clamps were to be utilized to secure the laser body in the mount. Unfortunately, the tools to accurately bend the stock were unobtainable.



Figure 8 - Aluminum Laser Mount

The final design proved to be the easiest to produce. It required a hole to be drilled longitudinally through an existing servo body. The electronics were all removed from the inside of the servo and a hole was drilled straight through using a drill press and a 9/16 inch drill bit. The laser was inserted through the servo body, a hose clamp was tightened around the laser outside of the servo body, and a zip tie was fastened around the switch that activated the laser beam. The zip tie put the switch in the on position which would allow the beam to be switched on and off by twisting the body of the laser. Twisting the body to the right would complete the circuit with the laser’s batteries; conversely, twisting to the left would remove the batteries from the contact and deactivate the beam. The important part was the laser beam could be locked on without anyone physically holding down a button.

*Wooden Base*

 A critical step towards the completion of the project included constructing a secure mounting base. This acted as a mount for the arm and camera, as well as provided a place to hide the electrical components, including wires and Arduino board. The base is 15”x12”x2.25” and is open in the back to allow for computer connection wires and power supply cords to exit from under the base. The mounting base is seen in figure 9.

Figure 9 - Mounting Base

It was critical when creating the base that it be stable and secure. When running the program, it must first be calibrated to the corners of the area of view. If there is even the slightest shift in any direction it will directly affect the accuracy of the robot when operating. In order to ensure that the accuracy was not compromised, both the robotic arm and the camera must be mounted firmly.

Secondly, the base provides a cosmetic improvement. Wires, boards, and power supply are all concealed underneath of the base. This makes it more aesthetically pleasing, professional, and most of all eliminates excess wires. Excess wires are likely to interfere with the operation of the LynxMotion arm while in motion. If this were to occur it would have the same effect as not having a securely mounted robotic arm and camera, and would negatively affect the accuracy of the pointing device.

Conclusion

The construction of a mechanical arm designed to actively point a laser at a moving target turned out to be a truly interdisciplinary task. The arm, mounting base, and the laser mount were all closely tied to mechanical engineering. Coding a program to follow an object and return it’s coordinates represents computer programing. The two separate systems had to be integrated to form the final product, a motion tracking arm.

There were many setbacks while designing this system. It was found that facial detection was the most effective way to track a human. Color detection proved to be too finicky to be used in a real world application. Blob detection was ineffective because it would track too many different changes and become unstable.

The integration of the laser pointer and the robotic arm is proof that a simpler solution is generally the best. Rather than design and manufacture a new part, a servo body was modified slightly to house the laser and secure it to the LynxMotion arm.

If the arm was simply set on a table, the motion of the arm would move it from its zero point and cause the system to be inaccurate; therefore a mounting base had to be designed. The role of the base turned out to be two-fold, it provided protected mounting space for the electronics, as well as provide the foundation needed to maintain the zero point of the arm.

The result of all these different pieces integrated into one system was a LynxMotion arm that could track a human and point a laser at that person. The system was not perfect however. Because the system used a static camera, it could not track a person a full 360 degrees. The processing was also very slow. Every time the target moved it would take approximately a second for the arm to catch up. The result was a choppy movement that was slightly inaccurate.

A possible application of this system would be a motion tracking spotlight. Current motion activated lights broadcast light over a large distance and can be a nuisance for neighbors. A motion tracking spotlight would focus the light directly in front of the person whom triggered the light, thus lighting their path without scattering light through neighboring homes. This technology can also be applied to security cameras. The camera would be programed to follow motion and ensure the action remains in frame.

References

[1] "Arduino." Internet: <http://www.arduino.cc/>, Feb. 12, 2012 [April. 9, 2012]

[2] "Axis Communications." Internet: <http://www.axis.com/products/video_motion_detection/index.htm>, April. 17, 2012 [April. 17, 2012]

[3] "OpenCV." Internet: <http://opencv.willowgarage.com/wiki/> , Mar. 16, 2012 [April. 9,2012]

[4] "LynxMotion Robotic Arms." Internet: <http://www.lynxmotion.com/c-27-robotic-arms.aspx> , 2010 [June 6, 2012]

[5] "Processing." Internet: <http://processing.org/> , [June 6, 2012]

Motion Tracking Robotic Arm

Group I.D: 025-03

Faculty Advisor: Dr. Ani Hsieh

Group Members: Justin Agren (Section 003)

Stephen Calabro (Section 040)

Jared Pagats (Section 025)

Zachary Palanker (Section 025)

Section 025 & 040

Faculty Instructor: Dr. Eli Fromm

Teaching Fellow: Ghasideh Pourhashem

Teaching Fellow: Nicholas Vacirca

Section 003

Faculty Instructor: Dr. Kambiz Pourrezaei

Teaching Fellow: Ghasideh Pourhashem