Project Number: MXC 0551

ROBOTIC DART LAUNCHER PROJECT

A Major Qualifying Project Report
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
as evidence of completion of the requirements for the
Degree of Bachelor of Science
By

Francisco M. Sanchez

Date: April 27, 2017

Professor Michael J. Ciaraldi, Major Advisor

Table of Contents

Table of Figures

Abstract

In the game of darts, players compete by aiming darts at the bullseye and other sections of the dart board, however to most untrained dart throwers, the variance between shots is very high. A robotic system that integrates computer vision to detect and locate the dart board, and a mechanical dart launcher will have better accuracy than a human in most cases. This projects seeks to create a low cost robotic dart launcher that anyone can replicate and build on. It will utilize OpenCV libraries in python, and an Arduino microcontroller connected to servos to control the dart launching and aiming of the robotic system.

Introduction

Dart boards represent an interesting target for computer vision, due to their simple and intuitive design, which humans can easily interpret. However, in order for a computer to understand each section of a dart board, computer vision algorithms must be applied. While at first it seems like the only tasks the computer vision algorithms must perform are to detect the dart board, and mathematically find the center of the smallest convex circle that that bounds the dart board, the camera perspective distorts the orthographic projection, and results in the need for more robust algorithms.

Darts, on the other hand represent a task which is easier for a robotic system to handle, than a human. This is simply because of the repeatability of a mechanical system, compared to variance between darts thrown by a human. Darts, in and of themselves are fairly consistent projectiles, and as a result, if the system that delivers power to the darts is consistent, so too will their trajectory. As a result, the actual dart launching aspect of this robotic system is fairly simple, mechanically, and not as complicated to develop as the control system logic and computer vision.

The purpose of this robotic dart launcher project is to create a robotic system that utilizes computer vision to find a dart board target, and determine the bull's eye location. With this information, the computer vision program will communicate with the robotic system that controls the mechanical dart launcher, so that the trajectory of the dart launcher lines up with the center of the dart board target.

One of the motivations going into this project, is the need for simplicity where possible, so that anyone can attempt to recreate this project on their own, with minimal cost and only a small amount of tweaking within the code. Further motivation for this project came from a recently

completed computer vision class, which introduced various topics and methods of processing images. With this knowledge, I was eager to apply them to a real-world task, where I could see very clear results. Having played around with darts before, especially in the weeks leading up to this project, I was interested in seeing how well a fairly simple robotic dart launcher would be able to do, if it implemented computer vision to find its target.

Background

Darts are believed to have existed since the medieval times, where archers would throw shortened arrows at targets in a pub to practice their aim. It is believed that they eventually started using a cross-section of a log as a target, with the age rings serving as a marker for accuracy. Eventually, the log would crack which is thought to have evolved into the radial sections of the modern dart board. In 1896 a carpenter, named Brian Gamlin, invented the current numbering system which features 20 at the top followed by 1, then 18, etc. This numbering system was devised specifically to penalize inaccuracy. The darts themselves started to resemble more modern darts at this time, with a wood barrel, a metal point on one side, and feathers on the other. By the 1900's darts, as game, evolved into its modern adaptation.

Computer vision is a field which seeks to automate the tasks that a human can easily do with their own visual system. Computer vision requires that a computer first acquire a visual representation of the world, through the use of a camera or other digital images. After acquiring the digital image, the computer must process the image, so that it may analyze the image using computer vision algorithms. And finally, using the analysis, the computer must be able to extract data from the real-world to make decisions and interactions. Some examples of processes that utilize computer vision, are the automatic categorization of objects, facial recognition and biometric security, or environment mapping and motion estimation.

There are various different algorithms within computer vision to extrapolate information from a digital image that can be used to make informed decisions, and each algorithm has its pros and cons for analyzing certain scenarios. For instance, the Hough Transform is good for detecting arbitrary simple shapes, however, for more complex ones, it may give a rougher approximation, or take too much processing time. As a result, for more complex object recognition, and feature

detection algorithm such as Scale-Invariant Feature Transform may be of better use, although it will require more training to make a robust detector.

OpenCV is a commonly used computer programming library that is centered on real-time computer vision. It is available across various operating systems, and different languages including C++ and python. OpenCV was initially launched in 1999 as part of and Intel Research initiative, with the goals of advancing vision research, spreading vision processing knowledge through a common infrastructure, and advancing vision-based commercial applications. In 2009, OpenCV 2 was released in October, featuring new functions and better implementations, while also moving development to an independent Russian team, supported by corporations. Additionally, in 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org which maintains both developer and user sites. OpenCV has a great deal of documentation and tutorials, making learning how to extrapolate information from an image, easier to grasp, especially since it is open source.

Arduino microcontrollers are open source hardware devices that are frequently utilized for building digital and interactive devices for the hobbyist user community. The Arduino microcontrollers features digital and analog input/output pins, a USB-port for serial connection to host computers, an ATmega chip which controls what signals the board sends out, and much more. Given the analog and digital input/output pins on the Arduino board, each microcontroller can control various servos, motors, and other actuators, as well receive signals from potentiometers, encoders, and other sensors. Further, Arduino microcontrollers start at around $20 for a genuine board, and $5 or less for a Chinese-replica, although given that the hardware specs are open-source, it should be noted, that they will theoretically perform identically. The price of an Arduino microcontroller, coupled with its various input/output pins makes it especially good for simple robotic tasks and projects, and allows most users to easily replicate.

Given the motivation to make this project easily repeatable by anyone, the low price of entry of the Arduino, and the open-source documentation of OpenCV and Arduino libraries, made them a clear choice. While other dart throwing robotic projects have been made, such as Technical University of Munich's in which they programmed a KUKA Light-Weight-Robot to mimic the actual throwing pattern of a human's dart throw, the cost of entry to recreate this project is extremely high as you must have a 6 degree of freedom arm. Additionally, the Technical University of Munich's project utilized AprilTags, a form of 2D barcodes to detect the dartboard location, instead of using computer vision to detect the actual dartboard itself.

Methodology

The hardware that was used to complete this project included, a computer, anArduino Uno microcontroller, 3 hobby servos (Tower Pro MG995R), a webcam, 4 AAA batteries, a battery holder, miscellaneous wire, surgical tubing, and metal scraps. The construction of the launcher was fairly simple, with two pieces of aluminum C channel acting as the guides for the dart, two pieces of aluminum L channel, attached at the front of the guides on either side, with each end attached to a single piece of surgical tubing. This construction resembles a crossbow. One of the servos was mounted on the opposite end of the L channel guides to hold the surgical tubing back while the robotic system was aiming. The webcam was mounting on the underside of the aluminum C channel, and lined up with the dart trajectory. Another servo was mounted near the center of gravity of the dart launcher construction, in order to rotate the launcher up and down. This servo was fixed onto an L bracket, which was in turn attached to the final servo which controlled lateral rotation of the launcher. Each component was wired into the Arduino Microcontrollers as shown in *Figure 1.* below, and the webcam was plugged into the host computer.

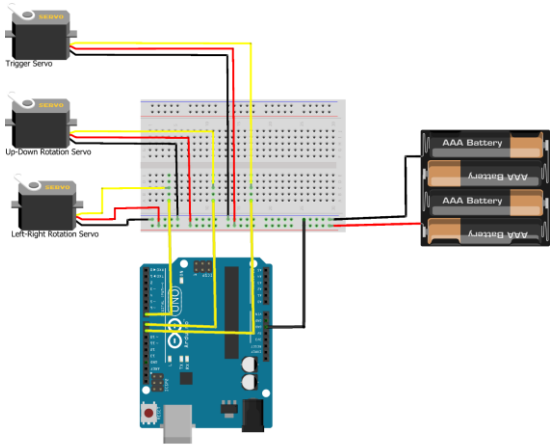

Figure 1. Arduino Wiring Example: *each servo was wired into the Arduino, each on a digital pin, and connected to an external battery pack for power.*

As mentioned before, the computer vision processing was done using OpenCV in the python language. The first method used to find the dartboard was to attempt to utilize the built-in Hough circles function. This function takes in a grayscale image and given slope differences in color, attempts to identify circles in the image. Before running the Hough circles function, the image was blurred and transformed to grayscale as pre-processing. In theory this method should find any circles present in the image however, since a dart board is not a solid color, and instead is divided into various disparate sections, this method struggled to adequately identify the dartboard's location.

The second method, which is what I ended up using was a combination of contour identification, and mathematical transforms to both locate the dartboard and its true center. The preprocessing for this method requires a transformation into a binary image, which only has two colors, black and white. This was done by making the image grayscale and then setting a threshold to transform it into a binary image. Once in a binary image format, I could run the build in find contours function to find each unique contour in the image. Using these contours, I was able to create bounding rectangle, and ellipses approximations for each contour. Then by selecting the 2nd largest contour, which should in most cases be the dartboard, given that it is hung on a plain wall, I was able to locate the dart board.

The next step was to find the center of the dartboard, however, given that the camera perspective does not match the orthographic project for a circle, it was a little more difficult than just finding the center of the ellipse from the contour. At first I believed that the reason the expected center did not match up with the real center was that the camera had lens distortion, a known phenomenon that causes straight lines not to appear straight on a camera that hasn't been calibrated. However, even after calibrating the results, were off, as can be seen in *Figure 2*. and

Figure 3. Further difficulty was encountered when, I realized that for each ellipse detected by my program, there were two potential true centers for the dart board. As a result, the solution was to detect a center within the contour of the dartboard. This luckily could be done with the Hough circle function, and resulted in a fairly accurate center.

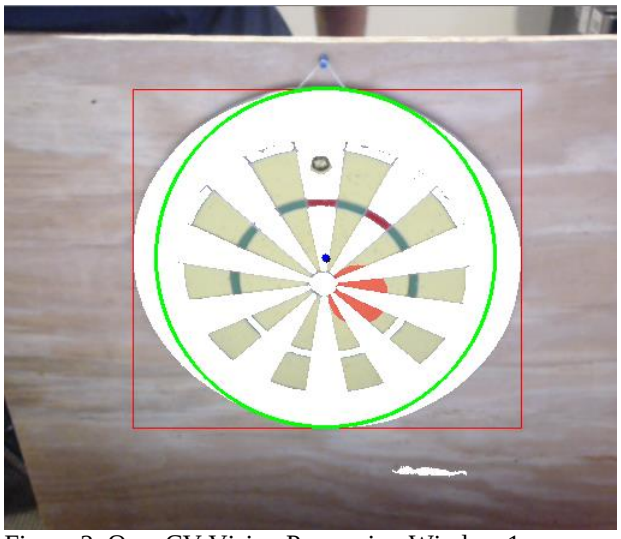

Figure 2. OpenCV Vision Processing Window 1: *detected contour of the dartboard, overlaid with bounding box, circle approximation, and misaligned center*

Figure 3. OpenCV Vision Processing Window 2: *estimated ellipse of the dartboard (white), overlaid with bounding box, circle approximation, and misaligned center*

In order to communicate between the Arduino microcontroller which was controlling the mechanical system, and the host computer running python with OpenCV, the serial port over COM5 was utilized. This required opening a serial connection from python to COM5, and having the Arduino listen to its serial port via USB. The python code, was written to send single character commands to the Arduino, each of which corresponded to a command, which was either a direction in which to move the dart launcher, or the fire command to release the trigger. In order to accurately move the dart launcher, the Arduino would have each of the rotational servos change their angle by 2 degrees at a time, for each time it read in a command. In order for the python program to no overshoot the target, a 100 ms delay was added. Finally, once the center of the dartboard was

within the threshold of the center of the camera, which was aligned with the dart launcher, the trigger command would be sent, releasing the surgical tubing and firing the dart.

Results

While the mechanical system was fairly simple, it's accuracy was not too great. This may have been due to various things, such as variance in the dart wings, which could affect its trajectory depending on its resting orientation on the launcher, or variance in pulling back the surgical tubing. The latter problem was partly rectified by marking the center of the surgical tubing with sharpie. Further, it was hard to align the camera with the expected trajectory of the dart, because as it travels, the dart follows a parabolic fall. This meant the camera would be aligned only for certain distances. The optimal distance found through testing was 8ft away from the target, with a preferred angle between +/- 15°.

Additionally, while the computer vision processing worked well to approximate the location of the dartboard and its bullseye, it did not always work robustly. There were times, where the code would throw an error for some reason or another that I could not trace back, especially since restarting the program without changing anything else solved the problem each time. Further, although the computer vision program is robust enough to detect a dartboard of various styles against a fairly plain wall, if there is significant clutter in the image, it can mistake a different large object for the dartboard.

Overall, when the programs were running in ideal conditions, the robotic dart launcher accomplished its task of having more accuracy than a normal human dart thrower. With the majority of the darts thrown landing in the bullseye or surrounding ring. However, if counting points as you would in a game of darts, the inability of the robotic system to favor the direction of 20 or other high numbers, made its performance somewhat more varied. However, on a simple concentric dart board with rings as sections, this robotic system did great. The performance by

percentage can be seen in *Figure 4.* and *Figure 5.* Each performance was recorded with the robotic dart launcher system approximately 8ft from the dartboard, and with 100 shots with varying angle to the dartboard. 40 shots were taken head-on at a 0° angle to the dartboard, and then 10 shots were taken at each angle from the following, +5°, -5°, +10°, -10°, +15°, and -15°.

Section	Percentage in Given Section
Bullseye	23%
Surrounding Ring	32%
Other Sections	45%

Figure 4. Dartboard Performance Radial Sections: *percentage of darts lading in each section after 100 dart launches from 8ft*

Section	Percentage in Given Section
10	21%
9	57%
8	19%
7	3%
6-1	0%

Figure 5. Dartboard Performance Concentric Sections: *percentage of darts lading in each section after 100 dart launches from 8 ft.*

Conclusion and Improvements

The results of this robotic system show that it is indeed possible to create a low cost robotic dart launcher that almost anyone can recreate, and will be fairly accurate when aiming at a dartboard against a plain wall. In fact, the total cost of the major components for this system come to less $100, not including the computer, which could in fact be replaced with a micro-computer such as a Raspberry Pi. Further, this project demonstrated how seemingly easy visual tasks for humans can become a lot more complicated due to computer limitations. Nevertheless, this project accomplished what it set out to do.

Some ideas for improvements and further development into this project include adding an ultrasonic range finder sensor, adding additional motors and mechanical subsystems to automatically reload or prepare the launcher, and changing the computer vision program to better account for clutter.

The first of these improvements would help the robotic system determine the distance to the dartboard, and if the dart trajectory was carefully mapped or simulated, it could offset its vertical angle to compensate for distance changes.

The second improvement, would require a more mechanically inclined person or group, to devise a method of retrieving the surgical tubing after release and resetting the trigger servo in tandem. One such approach I came up with that could be used, is a motor with a spool attached to it, to winder back the surgical tubing. Then once the trigger servo is in place, it could release the string attached to the surgical tubing, so that it can freely launch the dart. For an automatic loading system, one could use a magazine like contraption to load darts from the top.

Finally, the last improvement involves spending more time investigating further computer vision algorithms to more consistently detect the dartboard. One such possible avenue would be to train the computer vision program to a generic dartboard for feature detection. However, one of the caveats would be making sure not to over constrain the model, so that it can still work with different dartboard brands, etc.

All of these potential improvements could help make this robotic dart launching system more consistent and robust, although, some may increase the cost to replicate this project, which would be something to keep in mind.

References

"Arduino Language Reference." Arduino - Reference. N.p., 2017. Web. 27 Apr. 2017. N

"Fritzing." Fritzing Fritzing. N.p., n.d. Web. 27 Apr. 2017. <http://fritzing.org/home/>.

NumPy Community. "NumPy Reference Release 1.12.0." (n.d.): n. pag. 16 Jan. 2017. Web. 27 Apr. 2017.

"OpenCV API Reference." OpenCV API Reference — OpenCV 2.4.13.2 Documentation. N.p., 26 Apr. 2017. Web. 27 Apr. 2017.

Appendix A: Python OpenCV Processing *(source code)*

```
import numpy as np
import cv2
import cv2.cv as cv
import serial
import time

#capture video from webcam
cap = cv2.VideoCapture(0)

#distortion array fix
mtx = np.array([[  1.90638142e+03,   0.00000000e+00,   3.18730134e+02],
       [  0.00000000e+00,   2.58955491e+03,   2.39657683e+02],
       [  0.00000000e+00,   0.00000000e+00,   1.00000000e+00]])
dist = np.array([[  7.00490318e-01,  -1.20221782e+01,   6.22422810e-02,
         -2.09269518e-02,  -1.67269544e+02]])
h = 480
w = 640

#create matrix from distoration arrays
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),1,(w,h))

#open serial port to Arduino
ser = serial.Serial('COM5')

#find center of capture, which is alligned with dart launcher
h = (int)(cap.get(3)/2)
k = (int)(cap.get(4)/2)


while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()

    #undistort each frame
    frame = cv2.undistort(frame, mtx, dist, None, newcameramtx)

    x,y,w,h = roi
    #crop each frame
    frame = frame[y:y+h, x:x+w]

    #convert image to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    #creat a binary images from grayscale
    retval, image = cv2.threshold(gray, 127, 255, cv2.cv.CV_THRESH_BINARY)

    #show binary image
    cv2.imshow('binary',image)

    #find contours in the binary image using built in function
    contours, hierarchy = cv2.findContours(
        image,
        cv2.cv.CV_RETR_LIST,
        cv2.cv.CV_CHAIN_APPROX_SIMPLE
    )


    #arrays for holding points, centers and areas of each contour that gets analyzed
    true_centers = []
    centers = []
    radii = []
```

```
    pt1 = []
    pt2 = []
    fill = []
    areas = []

    for contour in contours:
        area = cv2.contourArea(contour)

        # there is one contour that contains all others, filter it out
        if area < 200:
            continue

        #keep track of each contour and its respective area measurement
        fill.append(contour)
        areas.append(area)

        #create a bounding rectangle for each contour
        br = cv2.boundingRect(contour)
        #save points to recreate rectangle on overlay
        pt1.append((br[0], br[1]))
        pt2.append((br[0]+br[2], br[1]+br[3]))
        #save the radii of the circle that would be bound by this rectangle
        radii.append(min(br[2], br[3])/2)

        #save the true center of this dartboard
        true_centers.append((br[0]+ (int)(br[2]/2), br[1]+ (int)(br[3]/2)))

        #save the center of the contour shape itself, approximation of center
        m = cv2.moments(contour)
        center = (int(m['m10'] / m['m00']), int(m['m01'] / m['m00']))
        centers.append(center)

    #print("There are {} circles".format(len(centers)))

    maxV = 0
    maxIndex = 0
    maxIndex2 = 0

    #find the indices of the two max areas of contours
    for i in range(len(areas)):
        print('index', i, 'area', areas[i], "maxIndex", maxIndex, "max value", maxV)
        if areas[i] > maxV:
            maxV = areas[i]
            maxIndex2 = maxIndex
            maxIndex = i
    print(maxIndex)
    fill[maxIndex] = None

    #create an overlay frame for highlighting the expected trajectory of launcher on
the image
    overlay = frame.copy()
    cv2.circle(overlay, (h, k) , 10, (0, 0, 255, 50), -1)

    #if there were contours detected
    if(len(centers)):

        for n in range(len(centers)):
            #do not do anything if not the second largest contour
            #the 2nd largest contour should be the dartboard
            if n != maxIndex2:
                continue
            #fit an ellipse to the rectangle
            ellip = cv2.fitEllipse(fill[n])
```

```
            #draw on the contour of the dartboard and the ellipse that was
approximated
            cv2.drawContours(frame, fill, maxIndex2, (255, 255, 255), 1)
            cv2.ellipse(overlay, ellip, (125, 125, 0), -1)

            #draw the bounding rectangle around the dartboard
            cv2.rectangle(frame, pt1[n], pt2[n], (0, 0, 255), 1, 8, 0)

            #draw the true center of the dart board
            cv2.circle(frame, true_centers[n], 3, (0, 0, 0), -1)

            #send signals to Arduino microcontroller depending on location of the
center
            if((true_centers[n][0]-h)**2 + (true_centers[n][1]-k)**2 > 10) {
                if(true_centers[n][0]-h > 0) {
                    ser.write('L')
                }
                else if(true_centers[n][0]-h < 0) {
                    ser.write('R')
                }
                if(true_centers[n][1]-k > 0) {
                    ser.write('U')
                }
                else if(true_centers[n][1]-k < 0) {
                    ser.write('D')
                }
                time.sleep(0.1)
            }
            #once center is alligned with trajectory, release the trigger
            else {
                ser.write('T')
            }

            #draw center of the contour by moment
            cv2.circle(frame, centers[n], 3, (255, 0, 0), -1)

            #add the overlay of the center of trajectory
            cv2.addWeighted(overlay, 0.5, frame, 0.5, 0, frame)


    # Display the resulting frame
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture, close serial connection
cap.release()
cv2.destroyAllWindows()
ser.close()
```

Appendix B: Arduino Robot Control *(source code)*

```
#include <Servo.h>

String readString;
//pins for each of the servos attached to the Arduino
int leftRightPin = 7, upDownPin = 8, triggerPin = 9;
//Servo classes for each servo, to send commands individually
Servo leftRightServo;
Servo upDownServo;
Servo triggerServo;

void setup()
{
  // Start the hardware serial port
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  //attached the servos
  upDownServo.attach(upDownPin);
  triggerServo.attach(triggerPin);
  leftRightServo.attach(leftRightPin);
}

void loop()
{
   //read in serial information from python program
   while (Serial.available()) {
    if (Serial.available() >0) {
      char c = Serial.read();  //gets one byte from serial buffer

      //rotate launcher left
      if (c == 'L') {
        leftRightServo.write(leftRightServo.read()+2);
      }
      //rotate launcher right
      else if (c == 'R'){
        leftRightServo.write(leftRightServo.read()-2);
      }
      else {
        //rotate launcher up
        if(c == 'U') {
          upDownServo.write(upDownServo.read()+2);
        }
        //rotate launcher down
        else if(c == 'D') {
          upDownServo.write(upDownServo.read()-2);
        }
        //release trigger and set back up after one second
        else if (c == 'T') {
          triggerServo.write(0);
          delay(1000);
          triggerServo.write(180);
        }
      }
    }
  }
}
```