Advanced Engineering Projects
Semester One Report

Real Time Object Tracking Camera Using Computer Vision

Stefan Prestrelski

Advisor: Mr. Eric Neubauer (Period 4)
Torrey Pines High School

Objectives:

To design a camera system controlled by multiple embedded systems that can track and follow objects based on a specified color using computer vision. The system analyzes real time video feed from a camera and rotate the camera on a set of servos to keep the object in the center of the frame.

This camera is a critical component of my yearlong Engineering Capstone project, which I plan to mount on an autonomous vehicle that I will design during the second semester. The camera will track objects as the vehicle is moving, which will serve as a prototype for a self-driving vehicle.

Introduction:

Real time object tracking using computer vision is among the most complex and challenging projects for computer engineers. It can be as simple as tracking a ball across a field (which by itself is already very complex) to facial recognition or applications in national security and defense. Real time object tracking involves several different but interconnected systems from the camera to the computer to a display. All of these systems must work in sync for real time object tracking to work.

In this project, the camera is mounted on a set of servos to be able to rotate to keep the object it is tracking in the center of the frame. The servos are controlled by a set of embedded systems using an Arduino microcontroller, which in turn receives inputs from a Raspberry Pi microprocessor that processes the video feed from the camera.I did a lot of research on cameras but I could not find any available blueprints for object tracking cameras that fit my needs, so I had to design the electronics and write the majority of the code (C++) from scratch.

In terms of image processing and tracking, there is a lot of open-source code shared by computer engineers in open collaboration platforms such as OpenCV, which provides the backbone computer code that I was able to adapt and modify specifically for this project.

Materials and Hardware:

- Raspberry Pi 3 Model B Motherboard (by Raspberry Pi)
- Arduino Uno Microcontroller (by Sunfounder Electronics)
- Wide angle fish-eye camera lense for Raspberry Pi (by SainSmart)
- Servo x2 (by Sunfounder Electronics)
- Jumper wires (by various companies)
- Battery Pack (used only as camera mount)
- Masking Tape
- Servo Mounts
- Breadboard

Design:

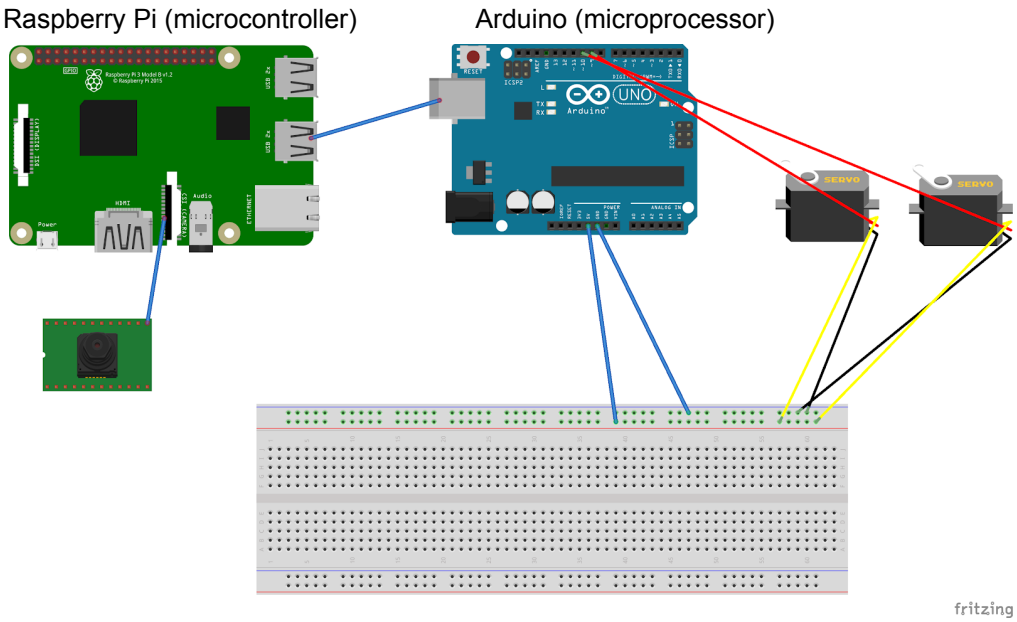

Figure 1: The above schematic shows the electronic connections of the camera (middle left) to the Raspberry Pi microprocessor (for image processing) and Arduino microcontroller, which control the movement of the servos (the two black squares in the middle right), mounted on a breadboard (lower middle)

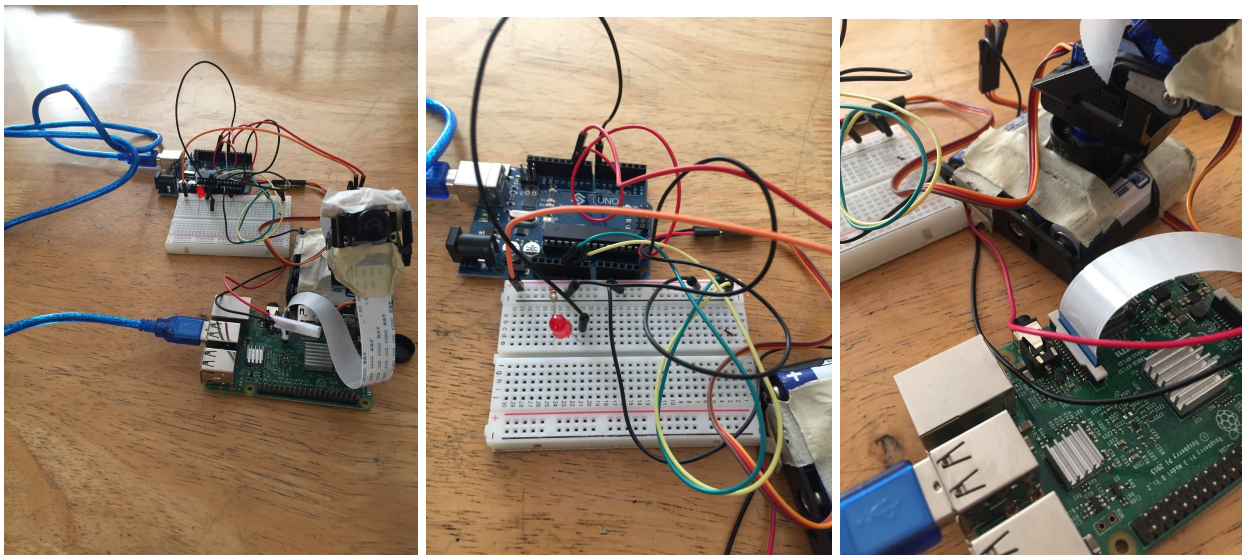

Figure 2: Left to right: a) Overview of camera, arduino, and raspberry pi; b) Arduino and breadboard; c) Raspberry Pi and camera mount.

Algorithms:

Overview of object detection algorithm:

1. Camera sends image to computer (Raspberry Pi)
2. Computer analyzes image. Looks for objects of specific size, color, appearance, etc
3. If desired object is detected, Computer sends output (Message to Arduino)
4. Arduino moves servos based on message received
5. Repeat sequence from Step 1

Overview of object detection code:

1. Computer loads in a single frame from camera video feed
2. Frame is blurred using a Gaussian Blur filter to remove noise that could interfere with image processing
    a. cv2.GaussianBlur
3. Computer scans all pixels in blurred frame (cv2.inRange). All pixels outside of desired color range are set to black, while those within the color range are set to white.
4. Image is eroded (cv2.erode) to further remove noise
5. Image is dilated (cv2.dilate) to reset object to original size
6. Find object contours (cv2.findContours) to determine if there are any objects in color range
7. If object found, determine where it is in the frame by looking at where its center is
8. Based on where center is, update servo position variables.
9. Send updated servo positions to arduino over the USB serial connection (arduino.write)
10. Arduino sets servo position based on the received position message
11. Next frame is loaded and process repeats

Overview of arduino code:

1. Import Servo library
2. Initialize servos to set position (center horizontal, 25 degrees down)
3. Loop:
    a. Read message from Raspberry Pi using Serial Monitor
    b. Set new servo position based on message

Argparse Arguments:

Argument	Description
-x	Horizontal Screen Size (default 480px)
-y	Vertical Screen Size (default 360px)
-s	Minimum Object Size (default 10px)
-f	Camera Framerate (default 32 frames/sec)

These arguments can be applied when the software is run to change the parameters that the software uses to detect objects.

The two screen size arguments (-x, -y) adjust the size of the screen in pixels. Smaller screen sizes run faster but have less precision in detecting objects. I found that resolutions above 640x480 slowed down the object detection considerably and thus were not useful for this application

The object size argument (-s) sets the smallest possible radius in pixels an object needs to be before it will be detected. Smaller sizes are more likely to detect objects but are also more likely to detect background clutter as objects and thus introduce more noise.

The camera framerate argument (-f) sets how fast the camera sends images to the computer in frames per second. Higher framerates lead to greater responsiveness but take up more computer resources. Above a certain framerate (in the case of the pi, 32), the increased resource usage overtakes the faster image loading speed, negating its benefits.

Results and Conclusion:

Under optimum conditions (even lighting, clean background), the camera had no issue tracking the objects. But under less-than-optimal conditions (uneven, excessively bright, or low lighting), the camera would sometimes not detect the object, even when the object in question was only a foot in front of it.

Color based object tracking works best in a controlled environment simply due to the number of things that can affect the color that the camera “sees”. A situation where this would work well is in a factory. In a factory, you can control lighting, the size of the objects you are trying to monitor, and the placement of the camera. A camera could be placed above a conveyor belt to monitor the size and/or color of different objects passing under it, and count/sort the objects.

Given enough computing power, color-based object tracking could work in even the most uncontrolled environments with the computers being able to clean up any clutter that shows up, but since a lot of applications limit the size, and by extension the power, of a computer, color-based object tracking is not practical for applications in uncontrolled environments such as outdoors or in a cluttered environment.

Next Steps:

In the next semester, I would like to further develop the image processing capability of my programming, incorporating some or all of the listed features:

- If no object is found for 5 seconds, pan area to look for objects. If no object is found, return to the rest position
- If an object is found, turn on LED. If the object is lost and the camera is panning, LED blinks. When panning stops, turn the LED off.
- Modify code to be able to read QR codes instead of detecting color. Color detection is heavily dependent on lighting, but QR codes can be read in almost any different lighting. The camera will be able to read a QR code and print its message/link and stop processes for people to read/copy message

In addition, I plan to design an autonomous vehicle (either line-following or obstacle-avoidance). with the camera mounted on top of the vehicle to track objects as the vehicle is moving, which will serve as a prototype for a self-driving vehicle.

Computer Codes:

Raspberry Pi code (Python)

```
# USAGE
# python object_movement.py --video object_tracking_example.mp4
# python object_movement.py
# import the necessary packages
from collections import deque
from imutils.video import VideoStream
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import argparse
import cv2
import imutils
import time
import serial

arduino = serial.Serial('/dev/ttyACM0', 115200)
horiz = ""
vert = ""

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video",
	help="path to the (optional) video file")
ap.add_argument("-b", "--buffer", type=int, default=32,
	help="max buffer size")
ap.add_argument("-f", "--fps", type=int, default=32, help="Framerate")
ap.add_argument("-x", "--horizontal", type=int, default=480, help="Horizontal screen size")
ap.add_argument("-y", "--vertical", type=int, default=360, help="Vertical screen size")
ap.add_argument("-s", "--size", type=int, default=8, help="Object Size")
args = vars(ap.parse_args())

# define the lower and upper boundaries of the "green"
# ball in the HSV color space
greenLower = (29, 86, 6)
greenUpper = (64, 255, 255)
# initialize the list of tracked points, the frame counter,
# and the coordinate deltas
counter = 0
(dX, dY) = (0, 0)
```

```
direction = ""
screenx = args["horizontal"]
screeny = args["vertical"]
# if a video path was not supplied, grab the reference
# to the webcam
if not args.get("video", False):
	print "grabbing video"
	#vs = VideoStream(src=0).start()
	camera = PiCamera()
	camera.resolution = (screenx, screeny)
	camera.framerate = args["fps"]
	rawCapture = PiRGBArray(camera, size=(screenx, screeny))
# otherwise, grab a reference to the video file
else:
	vs = cv2.VideoCapture(args["video"])
# allow the camera or video file to warm up
time.sleep(2.0)
# keep looping
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
	frame = frame.array
	# handle the frame from VideoCapture or VideoStream
	#frame = frame[1] if args.get("video", False) else frame
	# if we are viewing a video and we did not grab a frame,
	# then we have reached the end of the video
	if frame is None:
	print "broke while loop"
	break

	# resize the frame, blur it, and convert it to the HSV
	# color space
	#frame = imutils.resize(frame, width=600)
	blurred = cv2.GaussianBlur(frame, (11, 11), 0)
	hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
	# construct a mask for the color "green", then perform
	# a series of dilations and erosions to remove any small
	# blobs left in the mask
	mask = cv2.inRange(hsv, greenLower, greenUpper)
	mask = cv2.erode(mask, None, iterations=2)
	mask = cv2.dilate(mask, None, iterations=2)
	# find contours in the mask and initialize the current
	# (x, y) center of the ball
	cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
	cv2.CHAIN_APPROX_SIMPLE)
```

```
cnts = cnts[0] if imutils.is_cv2() else cnts[1]
center = None
# only proceed if at least one contour was found
if len(cnts) > 0:
# find the largest contour in the mask, then use
# it to compute the minimum enclosing circle and
# centroid
c = max(cnts, key=cv2.contourArea)
((x, y), radius) = cv2.minEnclosingCircle(c)
M = cv2.moments(c)
center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
# only proceed if the radius meets a minimum size
if radius > args["size"]:
# draw the circle and centroid on the frame,
# then update the list of tracked poi
centerx = int(x)
centery = int(y)
if(centerx > (screenx*2/3)):
        print("right")
        horiz = "right"
elif((centerx >= (screenx/3)) and (centerx <= (screenx*2/3))):
        print("horizontal center")
        horiz = "center"
else:
        print("left")
        horiz = "left"
if(centery > (screeny*2/3)):
        print("down")
        vert = "down"
elif((centery >= (screeny/3)) and (centery <= (screeny*2/3))):
        print("center")
        vert = "center"
else:
        print('up')
        vert = "up"
arduino.write(horiz + "," + vert + "\n")

# show the frame to our screen and increment the frame counter
#cv2.imshow("Frame", frame)
#key = cv2.waitKey(1) & 0xFF
counter += 1
# if the 'q' key is pressed, stop the loop
#if key == ord("q"):
```

```
		#	break
		rawCapture.truncate(0)
# if we are not using a video file, stop the camera video stream
if not args.get("video", False):
		vs.stop()
# otherwise, release the camera
else:
		vs.release()
# close all windows
#cv2.destroyAllWindows()
```

Arduino Code (C++)

```
#include <Servo.h>
Servo myservo;
Servo moservo;
String inByte;
String onByte;
String dir;
String ver;
int pos = 90;
int pis = 25;
void setup() {

  myservo.attach(9);
  moservo.attach(10);
  myservo.write(pos);  //initialize servos
  moservo.write(pis);
  Serial.begin(115200);
}
void loop()
{
  if(Serial.available())  // if data available in serial port
          {
          inByte = Serial.readStringUntil(','); // read data until newline
          onByte = Serial.readStringUntil('\n');
          dir = inByte;   // change datatype from string to integer
          ver = onByte;
          if(dir == "left"){
          pos += 5;
          }
          if(dir == "right"){
          pos -= 5;
          }
          if(ver == "up"){
          pis -= 5;
          }
          if(ver == "down"){
          pis += 5;
          }
          myservo.write(pos);    // move servo
          moservo.write(pis);
          //Serial.print("Servo1 in position: ");
          //Serial.println(pos);
```

```
        //Serial.print("Servo2 in position: ");
        //Serial.println(pis);
        }
}
```