CS231A Final Project Report: Pool Safety System

Ben Gilboa
CS231A Winter 2018 Stanford University
gilboab@gmail.com

Code repository: https://github.com/gilboab/PoolSafetySystem

Abstract

This project explores computer vision algorithms and techniques that can be used in a pool safety system design. It does not design the entire system. However, it experience different ways for achieving certain tasks required by such a system. The report described the experiments that were conducted and the results. It also explains why one method should be used over the other.

1. Introduction

The project goal is to explore and design key portions of a safety and maintenance system for privet pools. Owning a pool in the back yard is a source of joy and fun during the summer but it is also a source of concern throughout the year. The worst concern is having kid drowning in the pool when it is not monitored. The most common solution to this problem today is building a fence around the pool with a gate that a child can't open. Building these type of fences cost between $1000 and $2000 and it has negative appearance on the look of the back yard. Other solutions are to cover the pool or to have an underwater electronic detection system based on computer vision that is priced way above $1000. There are low cost solutions that detect water movement but their performance in terms of false alarms is poor. Other concerns that pool owners often face are related to pool maintenance. Small animals often fall into pool and end up dead in the pool skimmers. Ducks sometimes decide to make home in certain pools and cause a lot of mess. Excessive amount of leaves and branches often falls from trees during autumn and winter and overload the pool cleaning system causing equipment wear out. There are many other bad things that can happen and result by pool owner paying hundreds or thousands of dollars.

The objective of this project is to explore and design portions of a low cost computer vision based system that can potentially address most of the concerns raised. The main idea is to explore a system that is based on one or two cameras that are positioned in the back yard out side of the pool. The envisioned system can provide online stream video of the pool. It is continuously detecting certain events based on computer vision techniques. Once an event is detected the system can either notify the owner by sending a notification to owner phone or turn on an alarm in case of detecting life hazardous event. Building the entire envisioned system is outside the scope of this project. This project focus is the detection portion of objects in the pool as proof of concept or pre-work phase before architecting the actual system. Therefore, the project assumes several degrees of freedom, some of which it tries to resolve. For example, single camera vs two cameras system. The decision regarding the number of cameras will affect the system dramatically in terms of size, cost, ease of use and the type of algorithms that are used to design it. This is one of the tradeoffs that the report discusses.

2. Problem Statement

Given an image of a pool the main problem is detecting objects in the image and deciding whether a certain object is inside or outside the pool. Solving this problem in a robust and reliable way is the main focus of my project. There is no intention to detect under-water or submerged objects. It is assumed that once an object (person) falls into the pool there is enough time while it is still on the surface inside the pool boundaries for reliable detection by a system that continuously captures the scene. The system is meant to monitor the pool while not in use and provide a warning of the presence of the object when not expected. There are systems that help life-guards in public swimming pools to detect people in danger. This is not the objective in this case. Therefore, the problem is primarily to monitor the pool and detect the event of an object entering the pool by detecting it inside the pool knowing it was not there before.

3. Technical Approach

There are several key steps to achieve this goal.
Given an image that contain a swimming pool, the first step is to detect the boundary of the pool. The second step is to detect objects in the image. The third step is to detect for each object whether it is inside the pool or not.

I explored many of the provided pointers for datasets. However, I decided to get the dataset from google search since there were not enough images with pools available as

part of the datasets and much less with annotation. The dataset that I am using for this project is constructed from images found in the internet that contain outdoor swimming pools with or without humans in them. For the third step where I checked the decision criteria of both single camera and stereo of two cameras, I needed an image of the same scene taken from two different angles. To achieve that I took the images myself using iPhone standing on tripod and controlled using remote control.

4. Experiments

4.1. First step – detect pool as polygon

For this step I have used both techniques from the class and other techniques that I explored. The objective is to have a solid algorithm that can detect the boundaries of a pool. In phase one the algorithm need to work well for empty pool and in phase two it needs to perform for pools with objects. In the lack of enough annotated images for pools, my dataset is taken from the internet and tested manually by comparing the boundaries of the polygon found by the algorithm to the boundaries of the pool. As a simplification for this project I started by exploring rectangle shaped pools.

Figure 1: example of rectangle pool

The first method explored to detect the boundaries of the pool was using Canny edge detector. I applied different gaussian filters and different configuration of the canny edge detector from OpenCV. Using the output of the Canny edge detector I tried to fit the straight lines that construct the boundaries of the pool using Hough Transform. The edge detector performed as expected but it was not easy to find the right setting for the Hough Transformation and get definite lines.

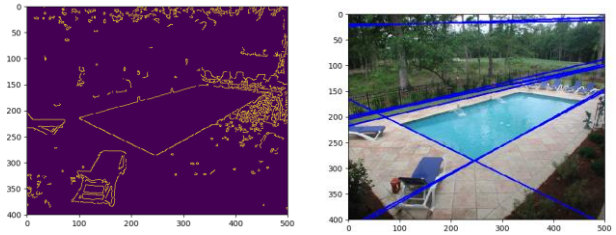

Figure 2: Canny edge detector and Hough transformation for lines

This technique does not achieve the desired result. Even for a single image it was very hard to adjust the parameters to get a decent detection of the pool area. As can be seen in figure 2, Hough Transform misses pool boundary lines and detects wrong lines.

The second method use segmentation techniques like the ones from PS4. K-mean and mean-shift algorithms. Playing with number of clusters and bandwidth the results that segmentation techniques achieved are not good enough for my purpose. Any shade, reflection or occlusion impacted the segmentation result and detecting only part of the pool.

The third method used rely on the fact that pools are usually in shades of blue. The technique mask the image with certain shades of blue as upper and lower thresholds. Then change using OpenCV threshold function to convert to binary image. Using the binary image, I searched the image for finding contours. Out of all detected contours I choose the largest one assuming the pool is the largest object in the image. The last step was to approximate the largest contour to using OpenCV ApproxPolyDP function to get a polygon with the desired number of sides. In this case since the pool was rectangular I optimized the polygon to a shape with 4 sides. The above technique works well on rectangular pool images without occlusions as can be seen in figure 3.

Figure 3: Pool boundary detection in the second method

For pools that are not rectangular 4 sides approximation of a polygon does not work well. In case of occlusions the result is also not satisfying as in figure 4.

Figure 4: 4 sides approximation to-non rectangle pool

To better support non-rectangular shapes and occlusions I increased the number of vertexes of the polygon significantly. To overcome occlusions specifically I used convex hull approximation. The convex hull includes portions that are not part of the exact polygon in the polygon. However, designing a safety system, I tradeoff false alarm over miss detects.

Figure 5: pool detection using final algorithm

The results achieved on my dataset are not perfect but are good enough for this stage. There are several assumptions that can be taken in the actual system that can simplify the pool boundary detection and lead to robust results. For example, during the system setup it is reasonable to assume that the pool is empty. It is also reasonable to assume that the system can run for several iterations and present the results on the owner smartphone asking the owner whether the pool detection is correct.

4.2. Second step – object detection

For object detection I use pre-trained convolutional neural network. I installed TensorFlow by google and started working with the object detection API. I examined how the different pre-trained models work and detect object on images of pools. The default model that object detection API tutorial is using "ssd_mobilenet_v1_coco" runs very fast but its performance is not good enough as can be seen in figure 6. It often misses many obvious objects. It also classifies object wrong in many cases. The second model that I tried was "faster_rcnn_resnet50_coco". There is a significant improvement in the performance as can be seen in figure 7 in the expense of longer run-time. The 3rd model that I tried "faster_rcnn_inception_resnet_v2_atrous_coco" is much heavier than the other two but it provides very good results. I decided to use this model for the project.

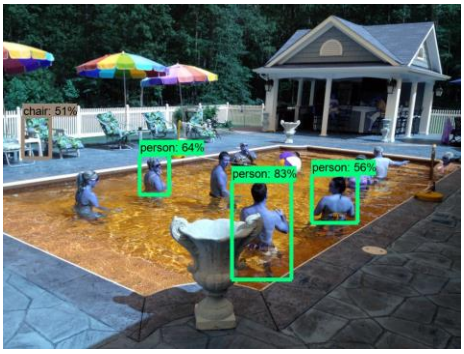

Figure 6: object detection using ssd_mobilenet_v1_coco model

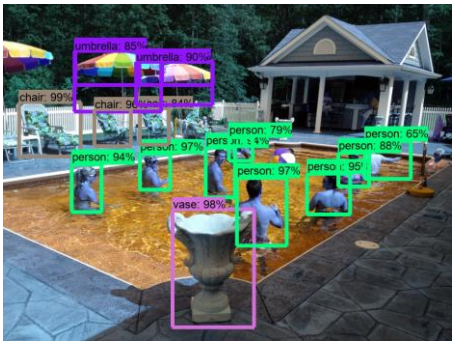

Figure 7: object detection using faster_rcnn_resnet50_coco model

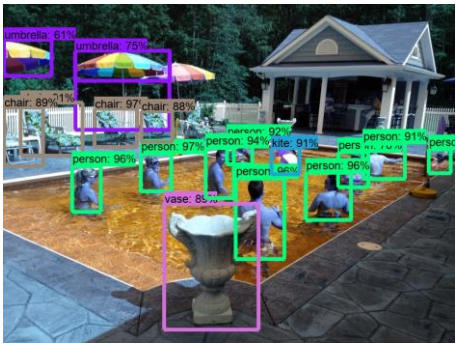

Figure 8: object detection using faster_rcnn_inception_resnet_v2_atrous_coco model

4.3. Third step – location of object

The third step of the project is to use the pool area detected in the first step as polygon and the objects detection in the second step and build the decision algorithm whether the object is inside the pool or outside of the pool.

Using the bounding boxes that the TensorFlow object detection algorithm identifies, I first tried very simple decision criteria. Comparing the bottom left and bottom right corners of the bounding box to the polygon area. If any of these two corners is located inside the polygon it is assumed to be inside the pool. For single camera this decision criteria this detector performance is reasonable, and I am not sure if it can be improved much further. Figures 9 and 10 show the objects detected as inside the pool in red and the ones outside the pool in blue.

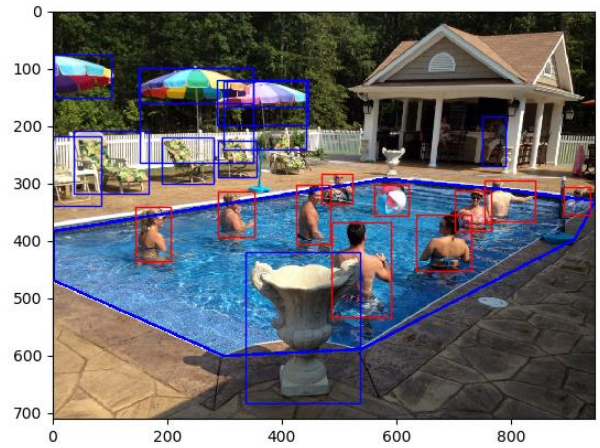

Figure 9: decision results

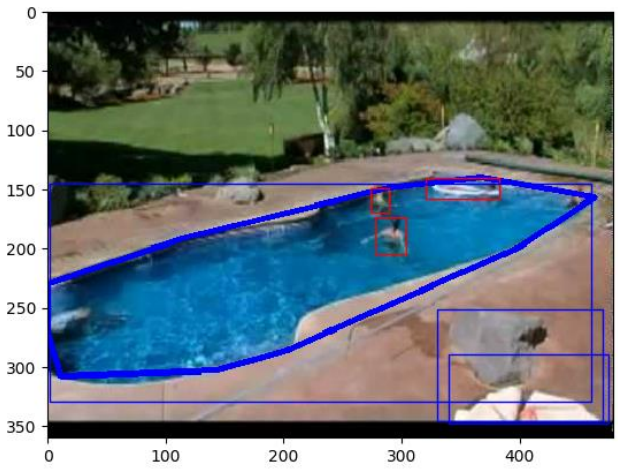

Figure 10: Second example of decision results

At this point I achieved a 3 stages pipeline that is finding the pool boundaries, detecting the object in the image and deciding whether each object is located inside or outside of the pool.

One clear drawback or flaw in this simple decision algorithm is the fact that it has no 3D understanding.

As can be seen in figure 11, it fails to detect an object that flies above the pool. The pink ball is positioned about 20" above the water level and about 10" above the ground level and still being detected as inside the pool (red bounding box). It means that a bird flying over the pool can trigger the system.

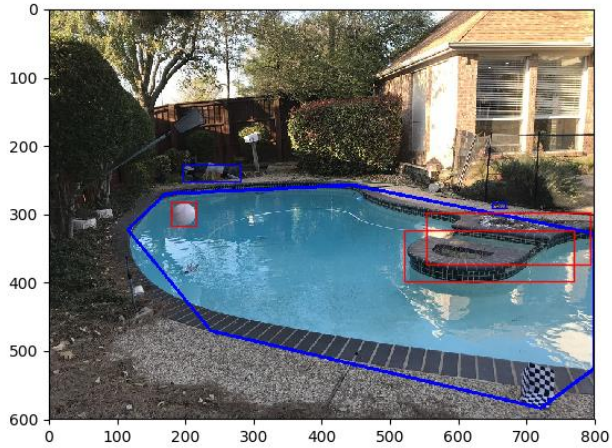

Figure 11: wrongly detecting flying object as inside the pool

To cope with this problem I explored Structure from Motion using 2 cameras located in 2 different angles. The objective of the FSM is to find the 3D point that corresponds to the object detected in the image. To use the non-linear triangulation the camera projective matrix is needed for both cameras. In addition, the RT transformation between the two reference simplify the process I used same world reference system by having the calibration checkerboard structure at the same position in the two images.

4.3.1 Calibration

Using a home-made checkerboard, I calibrated both cameras and found the projective camera matrices. Each mark on the checkerboard is a square of 2" by 2". To calibrate a projective camera we need find 11 unknowns from at least 6 points (each point provides 2 constraints). To get a better accuracy I used 20 points. The points location on the image were found manually. Figures 12 and 13 show the images with the checkerboard that I used for calibration along with the points used for calibration. The intersection of X Y and Z dimensions represent the world reference system. I have learned the hard way that camera calibration must be very accurate in order to give a

reasonable camera matrix. In my first attempt I used a corner from a shipping box where planes that are not 100% perpendicular along with only 1" squares and the location of the checkerboard in the image was at the corner and too far. The calibration results using this first attempt were very poor. In the second attempt I significantly improved the accuracy of the checkerboard, I increased its size by factor of 2 and I positioned it in a center location close to the camera. This procedure gave results that I consider good enough. To test the calibration matrix , instead of taking actual measurements of the scene I defined a virtual cube in a certain position above the pool in the size of 30" by 30" by 30", I used camera matrix to calculate the location of the cube in the image and marked it on the image. As can be seen in figures 14 and 15 the cube is clearly seen as a cube in the image.

Figure 12: Image 1 calibration

Figure 13: Image 2 calibration

By receiving clear cube on both images I conclude that the calibration result is satisfying.

Figure 14: Image 1 calibration test

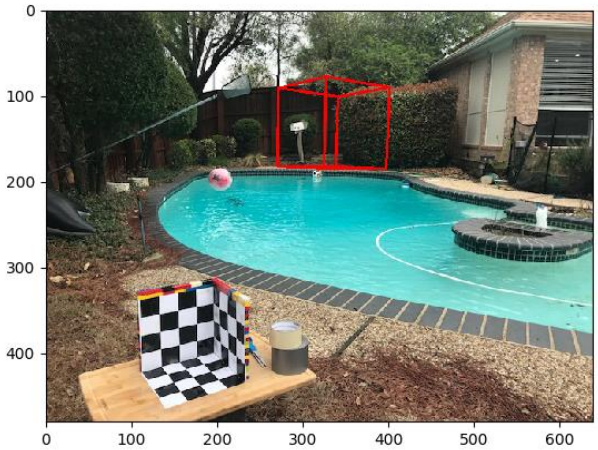

Figure 15: Image 2 calibration test

4.3.2 SFM Results

With the camera matrices I manually matched points on the pink ball and ran non-linear triangulation. The 3D points found by the triangulation clearly positioned the ball above the water level (water level is positioned 16" below the world reference system origin). This result proves that if the pool safety system have at least 2 calibrated cameras positioned at certain distance from each other it can detect and find the 3D position of the objects. Given the pool boundary in the images, the system will be able to find the pool boundaries in 3D and have more accurate decisions.

5. Conclusion

Considering the objectives of the pool safety system that I want to achieve as presented in the problem statement, the experiments that I did paved to define the architecture of

the system. Surprisingly the pool boundary detection is more difficult task than I initially anticipated Because of different pool shapes, shade that covers part of the pool, occlusions and reflections on the water, it is not easy to design a good detector that finds the exact pool boundaries for every case. One method that I haven't explored in this project is a deep learning method for pool boundary detection. This method can potentially generate good results if provided with large enough accurate and annotated dataset. If not deep learning than it might be that segmentation techniques with different features work well as well. As for detectors and fitting techniques, I believe my experiments showed that the chosen heuristic approach gave better results. Since pool boundary detection is a one time process in the life of the system done during the initial installation and the fact that certain assumptions can be made as described in section 4.1 the results that are currently not perfect should bot be a gating factor for the system design.

For object detection CNN is definitely the right way to go. The system will not run on a PC or server. Due to the mission critical nature of the system operation, it can't rely on cloud computation either. Therefore it will run on some microprocessor. From my short interaction with TensorFlow it seems to be working fine on smartphones for example. Nevertheless, optimizing the object detection portion of the system for accuracy and run-time is a key challenge for the system. I also believe that dedicated dataset containing pools, objects in the pool, etc; is important and can provide better results.

After exploring both single camera and two cameras and dealing with camera calibration, I believe that a single camera is the better option for this system. It will keep cost low and ease of use and installation low as well. To overcome the main problem of detecting an object that flies or passes above the pool I believe that a certain threshold can be used. Meaning that the system will require a certain object to be detected inside the pool for X number of seconds before initiating the alarm or sending notification. In addition, based on the object that was detected the system can tell whether this is a person or other object and not to trigger the alarm unless human being is in danger (I did not explored the classification accuracy of object detection algorithms in this project)

6. References

[1] – Open CV user guide: https://docs.opencv.org/2.4.13.2/doc/user_guide/user_guide.html

[2] – TensorFlow documentation and programmers guide: https://www.tensorflow.org/programmers_guide/

[3] – Chapters 2,3,10 from R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, 2003.

[4] Reference system available in the market: http://coraldrowningdetection.com/