3D Tracking of Multiple Balloons Using Computer Stereo Vision

KARLO DUMBOVIĆ
Technische Universität Wien
dkarlo2@gmail.com

Abstract

This project is a part of the wider project described in [1], which focuses on interactive audience participation in live concerts, and in original suggests the use of only one balloon as an interaction media. The goal of this project is to build a system that is able to estimate 3D world positions of several balloons during the concert, and to calculate them in real time. The system uses two static cameras placed in front of the audience. In every frame it needs to detect each of the tracked balloons in both camera images. After detection, using their estimated 2D image position it calculates their 3D world position. Once the system has calculated the positions, it sends them using the OSC protocol to a named IP address and port. The system is expected to be robust enough to work in different kinds of conditions that may occur during the live concert, such as fogginess and dynamic lighting. The system is made in C++ using the well-known computer vision library OpenCV.

1. Introduction

Audience participation in live concerts has been yet covered in terms of using some additional technical devices such as mobile phones ([2] and [3]) or other purpose built gadgets ([4]). The idea of this project is to make it possible for the audience to participate without any extra devices. This would take away their responsibility regarding the installment and use of such a technical device and allow them to freely enjoy the concert. A media that the audience will use to interact are several balloons going around the concert room from one person to another. An example of a balloon bouncing over the room can be seen in Figure 1.

In order to make this possible we need to build a system that can track each of the balloons in real time. To make an information that one balloon provides more complex we will track its 3D position in the room, instead of a basic 2D. Data containing the balloons’ positions will be constantly forwarded to another module, which will use it as the audience’s response and incorporate it in the sound of music. The aim of this project is making such a tracking system, while the rest can be found in [1].

We would need the system to be robust enough to be able to work in an imperfect conditions, such as in a presence of fog or very fast dynamic lighting, both of which are very feasible to happen on a live concert.

There are various approaches in tracking objects when considering the type of a sensor whose job is to detect them, such as Global Positioning System, camera, infrared technology, radar, etc. One of the goals of this project is for system to be as cheap and simple as possible, while preserving its power to be precise and robust enough. Following these guidelines, the
clear choice was to use cameras, especially because today there is a great variety of cheap but good web cameras.

Estimating balloon’s position in 3D can be done using only one camera, but it would require us to know correct details about the balloon real size and to be able to precisely determine its size in the image. While the first condition can be met without much trouble, the second one would demand very accurate balloon detection. This is something we should not expect, considering the conditions that the balloon could be placed in.

Better approach is to use two equal cameras, placed parallel one to another with some small distance apart, and pointing to the same direction - towards the audience. This system is called computer stereo vision and is a general approach for extracting 3D information from two digital images [5]. By capturing balloon’s positions in both camera images it is possible to know its 3D position in the room without knowing its size or some environment-special characteristics.

We can’t expect such cameras to be perfectly placed, i.e. their image planes to be on the same plane and X-axes on the same line. Also, their lenses might be distorted. To correct these imperfections we need to perform a calibration procedure when first using the system after at least one of its cameras was relocated or replaced.

We have already said that to calculate balloon’s 3D world position we need to know its 2D positions in both camera images. Estimating the 2D image position is a multi-step process that needs to be accurate and as least as possible subjected to different noise coming from the environment. Accuracy is important because small misdetections can have a large influence on calculating the 3D world position.

The first step is a motion detection. It is a procedure that uses the images already seen in previous camera frames and eliminates from the present image the part of the scene that remains static, such as walls and other objects stationed in the concert room. An important assumption when using this technique is that the balloon is (almost) always moving, what should be expected regarding the idea of this project. Motion detection is useful because it gets rid of the stationary part of the scene, which can include round shapes that are very balloon-looking, what could easily mislead the tracking system.

After we have made the problem a bit easier by eliminating the stationary part of the scene, we need to detect balloons in the rest. In images, balloons can obviously be interpreted as circles. This fact guides us to use some circle detection technique, which will give us the data about their positions and sizes.

We have stated earlier that it’s not acceptable to completely rely on the balloon detection as is, i.e. circles detected in the previous step. That’s why we need to use some object tracking algorithm. Here are two sentences taken from [6], which briefly describe the object tracking problem.

In its simplest form, tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. Tracking objects can be complex due to:

- *loss of information caused by projection of the 3D world on a 2D image,
- noise in images,
- complex object motion,
- *nonrigid or articulated nature of objects,
- partial and full object occlusions,
- *complex object shapes,
- scene illumination changes, and
- real-time processing requirements.

The items marked with a star do not present a real problem in our case, because the tracked objects are simple balloons.

To have a better look of what the system outputs, we can make a visualization of the recent balloon trajectory, which will immediately present observable results to the user.

The final output of the system in every frame is a 3D world position of each balloon.
This data is instantly being sent via OSC protocol [7] to the receiving module, which is going to, as already mentioned, use it for further processing.

In Figure 2 we can see a common setting of the tracking system, including two web cameras on the top of the TV, two balloons, a chessboard used for calibrating, and a laptop where the system is run.

A reasonable choice when implementing the given task is to use the recognized computer vision library OpenCV [8]. The chosen program language for the implementation is C++.

2. State of the Art

In this chapter we will discuss some existing techniques that can be used in solving both balloon recognition and 3D positioning problems.

2.1. Motion Detection

Motion detection generally means differentiating between the foreground part of the image, which is moving, and the background, which stands still. Here are two OpenCV approaches that offer background subtraction:

- Class BackgroundSubtractorMOG, which implements the algorithm that uses an improved adaptive background Gaussian mixture model for background subtraction described in [9];

As we can see, both of them use a mixture of Gaussian as the model for describing the background, what is the most effective approach nowadays.

Other motion detection methods are based on the calculation of optical flow. Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene (as cited in [12]). OpenCV functions that implement optical flow algorithms are as follows:

- calcOpticalFlowPyrLK, which implements a sparse iterative version of the Lucas-Kanade optical flow in pyramids described in [13];
- calcOpticalFlowFarneback, which computes a dense optical flow using the Gunnar Farneback’s algorithm described in [14];
- createOptFlow_DualTVL1, which implements “Dual TV L1” Optical Flow Algorithm described in [15] and [16].

These methods don’t give us the exact pixel set that represent moving parts of a scene. They can only be used to get an estimate of the motion.

2.2. Circle Detection

The shape of a balloon can best be approximated by a circle. The most known and reliable circle detection method is circle Hough transform (CHT) described in [17]. OpenCV offers the HoughCircles function, which applies a modified Hough transform on a grayscale image and returns centers and radii of detected circles.

2.3. Object Tracking

In order to keep a good track of the balloons’ positions, we need some algorithm that can deal with possible noise, which induces false detections. Such algorithms are called object
tracking algorithms and there is a broad spectrum of them, as seen in Figure 3.

The most interesting for this project are statistical point tracking methods because the noise can be easily built into the model of the motion. The other reason is that there is no need of a complicated object representation, since a balloon can be simply represented by its center point.

Now that we have narrowed down the number of interesting methods, we can make a simple overview of the most used statistical tracking methods:

- **Kalman filter**, which keeps track of the estimated state of the system and the variance or uncertainty of the estimate [18];
- **Particle filter**, which uses a genetic type mutation-selection sampling approach, with a set of particles (also called individuals, or samples) and their weights to represent the posterior distribution of some stochastic process given some noisy and/or partial observations [19];
- **Joint probability data association filter**, which is considered as the most widely and successful strategy for multi-target tracking under data association uncertainty [20];
- **Multiple hypothesis filter**, which is another considerable option in the context of multi-target tracking [21].

When it comes to particle filtering, the most common approach for implementing it is the Condensation algorithm, which is described in [22] and [23]. Same as the other statistical point tracking methods, it consists of prediction and correction steps that are being performed in every iteration of the algorithm. Most generally, the prediction step samples a new set of particles from the old one using the particle weights as probabilities and applies presumed motion to the new set. The correction step then assigns new weights to the particles using the given measurements.

### 2.4. Stereo Camera Calibration

Camera calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images [24]. It consists of estimating camera’s parameters, both intrinsic and extrinsic. These parameters specify how is a point defined by its world 3D coordinates projected onto the camera image plane, where the points are defined by their pixel 2D coordinates. *OpenCV* function `calibrateCamera` performs a stereo calibration and uses the approach described in [24]. The overall description of the algorithm for calibrating cameras in *OpenCV* can be found in [25]. The most important thing to note about the algorithm is that it uses a set of reference points in multiple images of a chessboard in order to estimate cameras’ parameters.

More about different camera calibration methods can be found in [26].

### 3. Methodology

This chapter contains a detailed explanation of the used approach. It is divided into several structural units for a better overview, similarly as in the previous chapter (Chapter 2).

#### 3.1. Motion Detection

For a motion detector, *OpenCV* class `MOG2` has been chosen. It uses the same algorithm as the class `BackgroundSubtractorMOG2`, which was mentioned in the previous chapter (Chapter 2). The advantage of using the `MOG2` class is that it’s a part of the *OpenCV* OCL module [27], which contains a set of classes and functions that implement and accelerate *OpenCV* functionality on OpenCL compatible devices.

Optical flow algorithms for motion detection were not considered because they don’t provide us with the foreground of an image, but only with it’s flow estimate. Having the exact foreground is crucial, because that is a part of an image where we expect the balloons to appear. We could use optical flow algorithm to estimate a direction of the balloon move-
ment, but it’s not necessary since that problem is dealt with in another way.

Using the MOG2 class is pretty straightforward. To initialize it, we need to set the number of Gaussian mixtures in the background model. The best value for this number can be determined empirically. In every frame the class is given a new image from the camera, and it outputs the foreground of the given image. The foreground image has all background pixels set to black and is used in the further processing.

3.2. Balloon Detection

As said before, balloons can obviously be interpreted as circles. Circle Hough transform is used to detect circles. The exact procedure goes as follows:

- Convert the foreground image to gray;
- Binarize the image in a way that all non-black pixels become completely white;
- Perform a morphological opening on the binarized image in order to eliminate noise;
- Use that image as a mask for getting a new foreground of the original image taken from the camera;
- Add a small black border around the image so that circle detection could detect circles on the edges of the image;
- Apply Gaussian blur to the image;
- Call the HoughCircles function to get a vector of circles’ centers and radii.

When implementing these steps, the problem of defining values of many different parameters comes along, such as blur kernel size and sigma, various threshold values, erode and dilate window size, circle minimum and maximum radius, etc. To get the desired outcome, what is a good circle detection, one has to tune them properly. There isn’t any particular algorithm to do that, so empirical approach based on as much as possible different video sequences is the best option. Something about parameter tuning will be also covered in Chapter 3.9.

3.3. Balloon Measurements

Because circle detection in a cluttered environment, which might also contain dynamic lighting, is not in every frame reliable enough, we have to define a structure that represents a measurement of the balloon’s state and has some additional parameters that describe a strength of the measurement.

Balloon measurement creation is initiated by detecting a circle. Then, for each of the tracked balloons, we first calculate the level of similarity between the balloon and the circle. This similarity is estimated as the Pearson’s correlation coefficient between mean RGB colors of a part of the image belonging to the circle and an image of the balloon that was
taken and stored in the tracking system before starting it, such as the one in Figure 4. Another factor that influences the strength of the measurement is whether the circle center lies inside a small rectangle region defined around the last known position of the balloon. If not, the strength is decreased. We can’t rely on the perfect circle detection in each frame, but we can expect that the correct circle will be detected a sufficient number of times in a small time interval. Because of that, we will keep the measurement stored for some small number of frames after the moment it was created. Also, we will decrease the strength of the measurement in each frame, and finally delete it from the memory when it gets too old.

3.4. Balloon Tracking

Although the tracking methods reviewed in chapter 2.3 would suggest that one of the latter two should be used to track multiple balloons, the particle filter approach was chosen because of its simplicity and a straightforward data association model that is in our case being used to match a measurement to a balloon. For each balloon, there is a separate particle filter constructed that tracks it.

The data association algorithm works on the next principle: for each balloon, its strongest measurement is added to the measurements list of the corresponding filter, but keeping in mind that the same measurement can’t be matched with more than one balloon. So, if two balloons have the same strongest measurement, the measurement will be associated with the balloon with which it has the stronger bond. This algorithm applies that in every frame there is only one new measurement added to the measurements list of each filter.

Having the separate measurements for each filter, we can perform the above-mentioned Condensation algorithm. We will set the number of particles to 1000. Each particle, besides its weight, stores its $x$ and $y$ coordinate, the amount of speed in both $(x$ and $y$) directions, and the amount of acceleration in both directions. The presumed motion of the particles, which is being applied in the prediction step, uses a model of the expected balloon movement. The basic elements of the model are:

- small downward gravity acceleration;
- velocity reduction due to air resistance;
- random hits from the audience with a greater probability on the edges of the image;
- Gaussian noise considered in each aspect.

In the correction step, we need to assign appropriate values to the particle weights. Let’s define the goodness of a measurement for some particle as a factor of the measurement’s strength and the value $N(p; \mu, \Sigma)$ of 2D Gaussian probability density function in point $p = (\text{particle}_x, \text{particle}_y)$ with parameters $\mu = (\text{measurement}_x, \text{measurement}_y)$ and $\Sigma$ equal to some empirically chosen value that describes how good do the measurements generally estimate the real state. Then, a particle weight is set equal to the goodness of the best measurement for that particle. After determining all of the weights using this principle, they are being normalized.

Another problem may arise in the case of balloon tracking that we haven’t covered yet. It is that a balloon might go out of the camera’s sight for a short period of time, and then unexpectedly appear somewhere else in the image. In order to catch the new state of the balloon as soon as possible one extra step was added to the algorithm. In each iteration, 10% of the particles are being replaced with the same number of randomly created particles. These new particles are then able to discover an eventual new position of the balloon redirect the filter state estimation if necessary.
Finally, each of the balloons’ positions is estimated as a weighted average of all particles’ positions belonging to the corresponding particle filter.

3.5. Computer Stereo Vision

After we have done all the preceding tasks separately on both camera images, we need to calculate each of the balloons’ 3D world position. Prerequisite for doing that is that we have calibrated the cameras. Stereo camera calibration algorithm was already described in Chapter 2.4. Having the calibration, we can perform the steps called undistortion and rectification. In OpenCV we have to use the `initUndistortRectifyMap` function. Exact details about how this function behaves can be found in [29]. The function calculates a transformation data that is in each frame used as a parameter for the `remap` function. We give this function the camera image as another parameter and then it performs the aforementioned steps of undistortion and rectification on the image using the transformation data.

After we have undistorted and rectified the camera images, they are aligned so that they share the same image plane and X axis. Now it’s pretty straightforward to determine all 3 world coordinates of the balloon, given the balloon’s pixel positions in both images. Please notice that it’s also straightforward to correctly match the same balloon from one camera image to another because each particle filter knows exactly which balloon it’s tracking.

Using the nomenclature from Figure 5 and assuming that the world’s origin and axes are aligned with the left camera’s origin and axes, we can state the following:

\[
X = x_L \frac{b}{x_L - x_R}, \quad (1)
\]
\[
Y = y \frac{b}{x_L - x_R}, \quad (2)
\]
\[
Z = f \frac{b}{x_L - x_R}, \quad (3)
\]

where \( y \) in theory has the same value for both images.

3.6. Smoothing Balloon’s Trajectory

Trajectories of the balloons’ movements that we get after we have finished with the previous step still have some noise. In order to eliminate that noise, we will choose already mentioned Kalman filter. Kalman filter is a simple but effective choice, since we can presume that every small part of a balloon’s trajectory is linear. To be precise, here we only deal with the acquired 3D balloon positions, and no longer think of 2D pixel positions that were different for each camera.

3.7. Plotting the Trajectory

In order to have a better look into what the system outputs, I decided to plot the balloons’ trajectories. The easiest but still effective way to do it is using a simple gnuplot-cpp library that can be found on [31]. It requires to have Gnuplot [32] installed on a local machine.

3.8. OSC Sender

The final step of the overall process is to send the tracking data using the OSC protocol. A simple set of C++ classes for packing and unpacking OSC packets that I used to send the data can be found on [33].
3.9. Parameter Tuning

For some of the above described methods I have mentioned that they require certain parameters to be set when implementing them. This is actually a common case that can be applied to the most of the aforementioned methods. There isn’t any specific recipe for how to do that in our case. In general, there are many well-defined methods for parameter tuning, which necessitate some kind of an automated evaluation of a proposed value set. However, due to the nature of our problem, such an evaluation is unable to be implemented successfully. Instead, to tune the parameters, the one should try out different values and choose the best one based on a self-estimate of the experiment realization.

The system is implemented in a way that it reads all parameters from a text file, so that the performance can be tested without recompiling.

4. Results

The above described tracking system gives very decent results when testing in an environment without influences of a fog or very fast light changes. The tracker can smoothly track balloons’ trajectories, what is visible through a plotting interface. If the lighting suddenly changes, the tracker gets a little confused, but it adjusts fast and continues to track the balloons correctly. The system was also tested in a low-light setting, which can be compared to a certain amount of fogginess, and its performance was satisfying, meaning that it could track the balloon almost as good as in the perfect conditions.

Let’s present some of the results taken as screenshots during an execution of the tracking program.

In Figure 6 we can see the appearance of the foreground image returned by the motion detector in a case when there is a small light change in the environment. Figure 6a shows the original camera image, while Figure 6b shows its foreground.

Figure 7 shows the typical-frame foreground difference between two motion detector models that have a different number of Gaussian mixtures. Figure 7a shows the foreground for the number of Gaussian mixtures set to 3, while Figure 7b shows the number of mixtures set to 10.

The next set of figures concentrates on the balloon detection part. Figure 8 shows an example of a good circle detection. In Figure 8a we can see how the mask of the foreground image returned from the motion detector looks like. Figure 8b shows how did the morphological opening affect the mask. Finally, in Figure 8c the detected circles are shown.

Particle filter performance can be seen in Figure 9. Figures 9a and 9c show the tracking status, while 9b and 9d show random 100 particles of the particle filter for each tracked balloon.

Figure 10 shows balloon trajectories for a pair of simple motions. In Figure 10a we can see a near-linear motion, while in Figure 10b there is a circular motion presented. In Figure 10c balloon was moved away from and towards the cameras, together with some motion along X-axis. The coordinate system is rotated to get a better look into the motion along Z-axis.

The finally presented result, seen in Figure 11, represents input and output of the overall tracking system. There is a screenshot of a console receiving OSC data from the tracking program (Figure 11c), together with the tracking status of both camera images (Figures 11a and 11b). Due to some irrelevant circumstances, cameras were not calibrated in this case, but they were placed precisely enough to show the desired result.

The system is undependable on any specific type of a camera as long as the ratios of frame sizes of paired cameras are the same. Because of that, the above presented results were made filming with various cameras in different environments.

5. Discussion

Motion detection results in Figures 6 and 7 demonstrate its importance as a prequel to
Figure 6: Motion detection with light changes.

Figure 7: Foreground images of typical frames.

Figure 8: Circle detection steps.
Figure 9: Particle filter performance.
(a) Near-linear motion.

(b) Circular motion.

(c) Towards the cameras and back.

Figure 10: Different balloon trajectories.
Figure 11: Input and output of the tracking system.
the circle detection. Because of the balloon’s constant movement, it is always in the foreground of the video. On the other side, steady surroundings keep being in the background. These two factors make the balloon to stand out, what further makes it possible to easily detect it. Sudden light changes cause the surroundings to be classified as a foreground, what makes it harder to detect the balloon, but only for a little while. Increasing the number of Gaussian mixtures makes the model more robust to the small moving of the crowd, while there are no significant disadvantages regarding the detection speed or any other. Before bringing the balloon into the scene, it is always a good practice to let the model learn as much as it can about the static part of the environment.

Circle detection shown in Figure 8 quite depends on the motion detection, because better motion detection means clearer boundary of the balloon. Morphological opening plays a significant role too. It helps eliminate noise in the foreground image and keeps larger objects untouched. As seen before, the motion detection can easily return more of the foreground than just a balloon. This is the reason why the circle detection can’t work just on the binary mask of the foreground, even after applying the morphological opening. Like said in Chapter 3.2 we need to take the masked part of the original image first and then perform the circle detection on that, where we have the information about each pixel’s color.

Particle filter shown in Figure 9 works very satisfying when the measurements are good enough. With a lack of good measurements it tends to push the estimated balloon state towards the lower middle part of the image. This is a consequence of the motion model, which presumes gravity and that people push the balloon always to the opposite side from where they stand on. Another noticeable thing is that the particles always form a big cluster on the place of the balloon, with exception of a small part of them, which are placed randomly around the image. These particles are the ones that are in charge of detecting sudden changes of the balloon’s position. Finally, observing Figures 8c and 8d we can conclude that when a view of the (green) balloon gets obstructed, particles start to spread. The cause of that behavior are integrated random movements in the motion model and a lack of good measurements, which would keep them focused on a certain position.

Real tracking performance can be seen in the trajectory plots in Figure 10. The first thing to point out are smooth changes of the position, what means that the noise gets well eliminated. Now let’s look into the correctness of the trajectories. The first plot (Figure 10a) is a result of moving the balloon nearly along the X-axis in one direction and a little back. On the plot there is also some shift along the Z-axis in the beginning of the trajectory, what can be a consequence of one camera not seeing the balloon when it was on the edge of another. The second plot(Figure 10b) shows an almost perfect circle, which was also the real balloon’s trajectory. This proves that Kalman filter, which was used for smoothing the motion, is a good choice, although it’s basically a linear filter and the overall motion wasn’t. The third plot (Figure 10c) demonstrates the correct calculation of Z coordinate when moving the balloon towards the camera and back.

The output presented in the console in Figure 11 also shows smooth transitions of the balloon position between consecutive frames. Having in mind that the x coordinate values go from 0 to 3 from left to right, and the y coordinate from 0 to 1.65 from top to bottom, looking at the console output we can state that they are correct.

The speed of the system was tested on a laptop that operates on Intel(R) Core(TM) i3 CPU with 2 × 2.53 GHz. The number of balloons used in the test was 3. At first, the system was very slow and not usable, so an extra effort needed to be taken. The solution was the most obvious one, to immediately shrink the image from the camera to have some predefined area before processing it further. An important thing when resizing it is to preserve the pixel ratio. Fixing this predefined area to
some number that is small enough for the system to work in (nearly) real time (in our case 50000) gave good results, which were shown in Chapter 4. That meant that no further steps needed to be taken. System frame response time is in average 60 to 70 ms, what makes it about 15 frames per second. Please notice that this is not a top-shelf processor nowadays and that we only use one core, so that the running time can certainly be much faster.

I would like to address some problems that may arise when trying to run the implemented system. The familiar ones are that the user doesn’t have a correctly installed OpenCV library or a properly installed Gnuplot. Before running the system, the user should be advised to look up the installation guides and follow them accurately.

6. IMPROVEMENTS

When considering improvements of the system concerning the matter of the running time, the first thing that comes to mind are some implementation-specific techniques, such as multithreading. The prediction and correction steps of each particle filter could be run in parallel, what would apparently speed up the overall computation time.

Another running-time improvement point could be using the fact that OpenCV implements a GPU module, which offers some functions that work only with certain types of graphic cards. These functions operate faster than the usual ones, and some of them can for example be used for the motion detection.

On the other hand, some of the improvements regarding the methodology used could be dealing with contours instead of circles, normalizing the pixel colors in the image, physically marking balloons with distinct features and detecting them in the image, etc.

REFERENCES


