TEK5030 - Velocity Tracking

Benjamin W. Lea (benjamwl) and Tony Nguyen (hpnguyen)

1 Introduction

The project's application is measuring the velocity of a tracked object that's moving around in a frame. Estimating the velocity of a moving object in 2D-space is not particularly difficult; find the distance covered in a specified interval of time, and divide by that interval. But that method fails if the object moves along the z-axis. Therefore, the goal of this project is to create a program that can measure the velocity of an object moving in 3D-space.

In order to do this properly, we needed the program to extract information not just about where the object is along the x- and y-axis, but along the z-axis as well. This means that measuring depth is necessary. In order to limit the project's workload based given the limited time we had on the project, we decided on an important assumption; the program should only measure the velocity of one (or two objects) at a time.

The basic measurements can be done with just one camera, but only if you disregard the z-axis. In a frame with only (x,y)-coordinates we can measure the changes in pixel-difference in two consecutive frames. One might think that a large pixel-difference indicates a large movement change i.e. large velocity. However, since a single frame lack depth the velocity has no relative scale to compare it against. We propose a solution by adding a second camera, which opens up the possibility of measuring the velocity of an object that's moving forwards or backwards in front of the cameras.

The idea behind adding a second camera is to utilize epipolar geometry. The two cameras are now connected to each other and using basic epipolar geometry, we can calculate the disparity of a point in 3D between the cameras. Using the disparity further, we can estimate the depth of the point on the image. This depth allows us to calculate the velocity in 3D-space.

Before we can use the stereo camera we must first calibrate it in order to retrieve the optimal homogeneous transformations between the two camera frames. An ideal stereo camera has the two camera's coordinate frames aligned along the x-axis. However, this is not necessarily the case for our stereo camera, so we must therefore perform stereo-rectification to solve this issue.

2 Method

2.1 Camera Setup

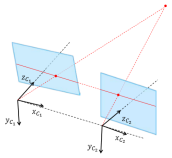

Figure 1: Non-ideal camera setup

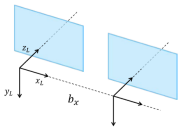

Figure 2: Ideal camera setup

Figure 2 shows a non-ideal camera setup, where the two cameras' coordinate systems are not placed in parallel. We want to estimate the relationship between the two cameras in order to find how they should adjusted to be in an ideal setup, shown in figure 2. The first step of that process is to capture pairs of images of a chessboard. This gives us information about the cameras' positions relative to the chessboard, which can be used to find the cameras position relative to each other. We can then find the camera matrices and distortion parameters using a stereo calibration function from OpenCV, and they are used to estimate the disparity-to-depth matrix Q using the stereo rectification function from OpenCV. The Q-matrix is later used to estimate the depth.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -{}^{L}c_u \\ 0 & 1 & 0 & -{}^{L}c_v \\ 0 & 0 & 0 & f_u \\ 0 & 0 & \frac{1}{b_x} & \frac{{}^{R}c_u - {}^{L}c_u}{b_x} \end{bmatrix}$$

Here defines the values $-{}^{L}c_u$ and $-{}^{L}c_v$ the left camera center, f_u the focal length, b_x the baseline and ${}^{R}c_u$ the right camera center along the x-axis.

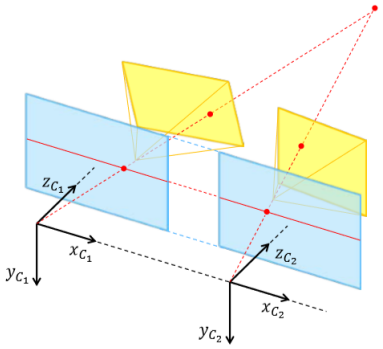

Figure 3: Rectified stereo setup

The result is a system shown in figure 3, where the orientation between the cameras are in an ideal setup as in figure 2, which means we can use the epipolar geometry.

2.2 Velocity estimation

The first step is to make the program find the object it should track. When we were developing the program, we used objects that should be easy to separate from the background, typically objects that have a single, strong colour. In order to find the object the program should track, we needed to filter out the background from the pixels belonging to the object. In order to do this, we used Otsu's method to determine the threshold of the image. Due to the possibility of the object having holes and irregularities, we use the morphological operation closing in order to make the object more even.

This thresholded image is now a binary image. The next step is to create a bounding box around the object. First we find the contours of the detected object, which is now a set of points on the image that corresponds to the location of the object. Using this contour set further, we can determine the highest, lowest, farthest left and right points to create a minimal rectangle around the object based on these points. This rectangle is now our bounding box and we will use this to track the object. We assume that the objects that will be tracked are larger than a specified threshold.

Now that we have the tracked object with bounding box, we must measure the depth of it. To do so first, we first find the disparities between the frames of the two cameras using Semi-Global Block-Matching (SGBM).

The depth of the images is defined along the z-axis from the two camera-frames. We have now found the depth map of the images and plotting the 2D-points of the tracked object into the depth map we get the depth of the object. This yields a distance between the object and camera-frame. Now given this depth of the different pixels we can convert the point from 2D to 3D. Note that these points that we are trying to find are estimates.

$$\begin{aligned} X &= (u - c_u)\tfrac{b_x}{d_u} \\ Y &= (v - c_v)\tfrac{b_x}{d_u} \\ Z &= f_u\tfrac{b_x}{d_u} \end{aligned}$$

Now we have the 3D-coordinates of the object in both the previous frame and the current frame. Previous frame can be at any arbitrary time, but here we have chosen it to be every 30th frame because our cameras are recording at 30 frames per second (FPS). This means we are measuring the difference in one second between the previous and current frames. Now this yields the Euclidean distance the tracked object has moved between the two frames. To compute the Euclidean distance we can use the following formula:

$$Distance = \sqrt{(X_{curr} - X_{prev})^2 + (Y_{curr} - Y_{prev})^2 + (Z_{curr} - Z_{prev})^2}$$

To measure speed, one can use the generalized and averaged velocity of the object using

$$V = \tfrac{\triangle s}{\triangle t}$$

where V is velocity, $\triangle s$ and $\triangle t$ is respectively the distance and time between the current and previous frame.

We have now estimated the velocity of a moving object using a stereo camera setup.

3 Results

Examples of objects we tracked are shown in figure 4 and 5.

Figure 4: Orange ball

Figure 5: 3D printed bullet

Figure 6: Our setup

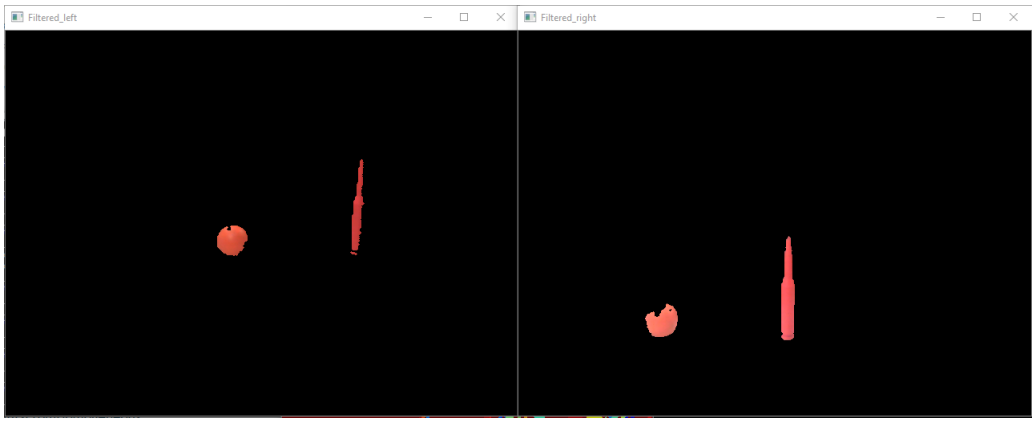

Figure 7: Screenshot of an colour-filtered image

Figure 7 shows an example of what it looks like when the colour filtering and the morphological operation closing is executed.

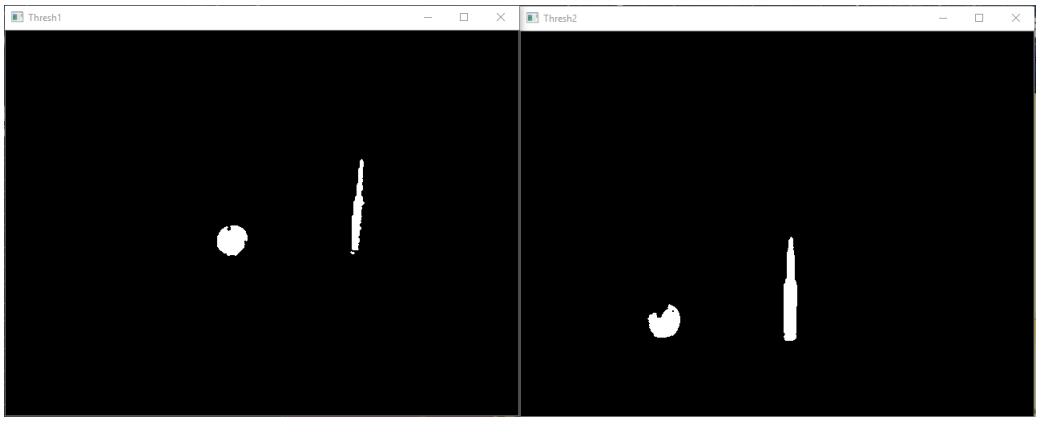

Figure 8: Screenshot of a thresholded image

Figure 8 shows an example of what it looks like when the colour filtered image is thresholded.

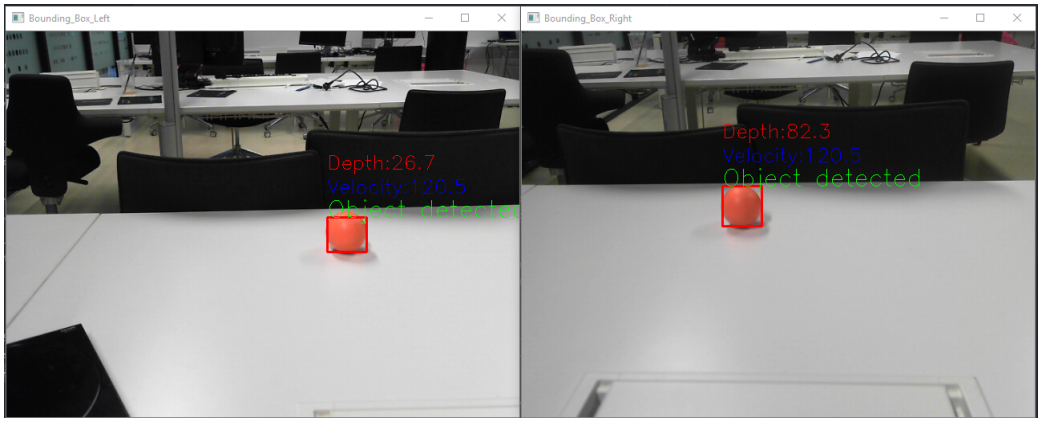

Figure 9: Screenshot the ball's velocity and depth being estimated

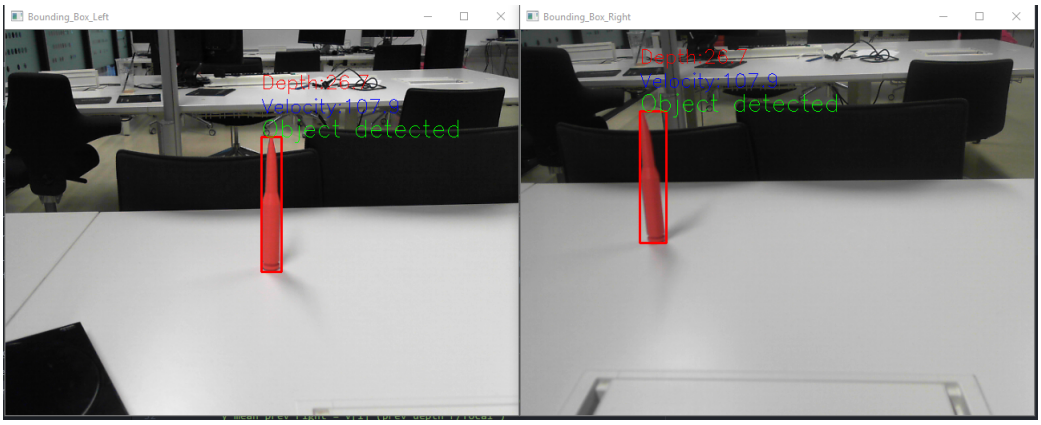

Figure 10: Screenshot the bullet's velocity and depth being estimated

In figure 9 and in figure 10 we can see examples of the rolling ball's and a still-standing bullet's depth and velocity being estimated, respectively. This is where some of the issues of our program comes to light. The depth estimation seems to be really jumpy, due to the unstable disparity calculation. The velocity estimation is of course directly related to disparity, but this seems to be more stable when the object is moving. The velocity unit is in pixels per second.

3.1 Discussion

In our application we have tried to create a velocity tracking program that measures the velocity of a tracked object. This proved to a difficult, but interesting task. Here is a list of positives and negatives that we could take from the project.

Pros	Cons
Maintain tracking of object	Unstable depth calculation
Fast calculation	Bounding box limitation
Estimation of velocity	Not optimal feature extraction
Practical purposes	Naive approach

The program is able to maintain fairly good tracking of the object, given that the color filtration is adjusted to fit the object. It executes the calculations fast and measures velocity fairly well. The program can have practical purposes.

However, the program suffers from glaring issues. The estimates from the depth map does not correspond to the actual depth, which directly affects velocity estimation. There are certain practical limitations as well; the object has to be of a certain size for the program to be able to track the object, and the way objects are filtered are currently done manually; the program will be more useful if that process was executed automatically. There might exist more sophisticated methods and accurate ways of measuring the velocity compared to ours.

4 Conclusion

By using stereo camera setup, we have a created a velocity tracking program by estimating the Euclidean distance between two 3D points between two time

steps. Our methods, however, yielded erroneous results, given instability of the disparity map and the not optimal stereo camera calibration.