ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Automatic Control Laboratory

Vision and State Estimation for an Automated Billiard Player

Semester Project

Constantin Stamatiadis

21. December 2018

Supervisors: Dr. Joe Warrington, Dr. Nikos Kariotoglou

Supervising Professor: Prof. Dr. John Lygeros

Abstract

IfA started a project with the aim of developing a fully-autonomous billiard playing robot. The project is in its early stages. Up until now, the following tasks have been addressed: implementation of the AI through Approximate Dynamic Programming, Cue Control, and Vision and State Estimation. In this work, we focus on the latter. First, a ceiling camera was set up. Its video feed was feed into a custom image processing pipeline used to estimate the current state of the game. The pipeline first preprocesses the incoming images (remove distortion, correct perspective), then extracts the ball location through different detectors, to finally merge the measurements through a filter and give a state estimate. Second, the camera system was used to find the dynamic properties of the game, as these are essential to obtain a good model. The friction between the balls and the tablecloth was measured, as wells as the coefficient of restitution for ball collisions with other balls and with the cushions. Finally, a cue camera was set up, with the aim of getting a "player's eye" view of the game. A depth camera was used in order to get ball distance measurements. The camera was located through an ArUco marker, which was also exploited to find the angular position of the cue.

Contents

List of Figures

Chapter 1

System State Estimation

In this chapter, we will explore the entire pipeline used to extract the current state of the system. Before entering into the algorithmic part, it is worth to briefly talk about the hardware setup that we decided to use. When it comes to the actual ceiling mounted camera, the main challenge was to select a camera with a wide enough Field of View (FoV), in order to be able to capture the entire snooker table, and a high enough resolution, to reduce the pixel to mm conversion factor and hence achieve higher measurement accuracy. These requirements led to choosing the ZED by Stereo Labs. Given that this camera is able to output images at a resolution of 2.2K with a diagonal FoV of up to 110°. This device is actually a stereo camera and hence has depth measurements capabilities. However, these are not exploited for the projected, mostly because the snooker table setup is too flat for the camera to be able to produce any meaningful measurements. The camera was positioned in such a way that it would be able to fit the entire table within its FoV. The vision system runs on a dedicated computer, an Intel NUC. The different blocks of the pipeline were all implemented as separate ROS nodes, as can be seen in Figure 1.1, which shows the ROS Graph. This algorithm could have been implemented also without ROS. However, we decided to opt for this solution due to its modularity and the fact that it would ease the communication between different parts of the future system. In the following sections of this chapter, the different blocks will be analysed, giving also explanations of what ultimately led to all the design choices.

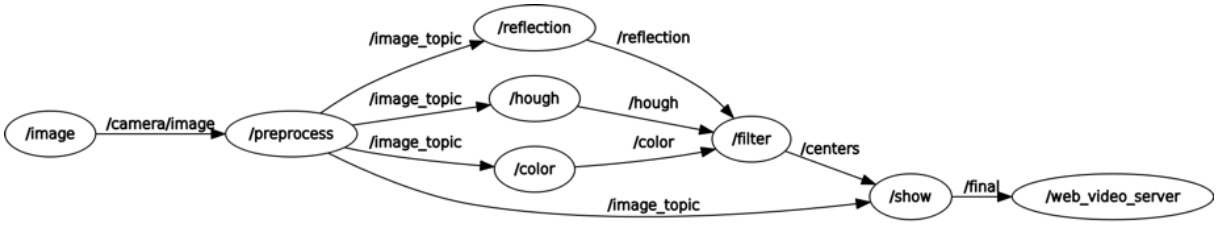

Figure 1.1: ROS Graph

1.1 Video Feed

The *Image* node is used to access the camera and start streaming the raw images. The camera was set to 2.2K in order to have as many pixels as possible per ball. At this particular resolution, it is only possible to operate at 15 Frames per Second (FPS). The low frame rate is not a problem for our implementation, as it is only concerned with

finding the system state when there is no motion. As mentioned in the introduction of this chapter, the camera is a stereo camera. Meaning that one image consists of the two images, one for each element of the stereo pair. These are stacked together horizontally. This particular feature was not of particular interested for the pipeline. Hence, the image is immediately split and only the one which captures the entire table is published by the node. This is done in the *Image* node rather than in the *Preprocess* one in order to reduce the computational overhead. Which allows maintaining a higher message publishing rate along the pipeline.

1.2 Preprocessing

This section deals with all the image processing steps that are required to obtain an image with desired characteristics. As expected the raw image suffers from barrel distortion, Figure 1.2. Hence it was necessary to calibrate the camera. However, even the calibrated camera still wasn't producing fully satisfactory results, as can be seen in Figure 1.3. Due to the fact that the camera plane is not aligned with the table surface. The parallel edges intersect at infinity on the image plane. This was solved by applying a perspective transformation, Figure 1.4. Moreover, also the edges of the table were extracted.

1.2.1 Camera Calibration

Camera calibration consists of finding the intrinsic parameters as well as the distortion parameters [15]. The intrinsic camera matrix provides information on how to map between camera coordinates and pixel coordinates in the image frame, according to the following equation:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{1.1}$$

Where f_x and f_y are the camera focal lengths, while c_x and c_y are the optical centers expressed in pixel coordinates.

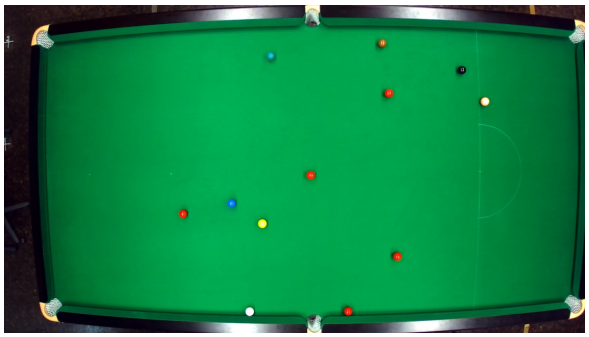

Figure 1.2: Raw Image

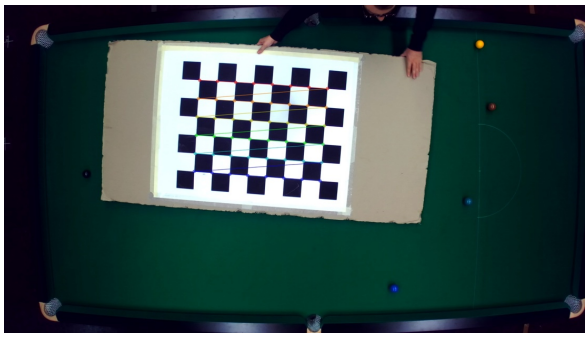

Figure 1.3: Calibration

The distortion parameters allow correcting for both radial distortion, which manifests in a barrel effect, as well as tangential distortion, caused by the fact that the image taking lenses are never perfectly parallel to the image plane. The correction occurs according to the following equation:

$$x_{corrected\_radial} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{1.2}$$

$$y_{corrected\_radial} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{1.3}$$

$$x_{corrected\_tangential} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{1.4}$$

$$y_{corrected\_tangential} = y + [2p_2 xy + p_1(r^2 + 2y^2)] \tag{1.5}$$

Where x and y are the pixel coordinates in the original image and $x_{\text{corrected}}$ and $y_{\text{corrected}}$ are the coordinates of the output image. While k_1, k_2, k_3, p_1, k_2 are the distortion parameters. All these parameters are found by using a planar checkerboard of know size. The calibration algorithm finds the corners of the checkerboard and is then able to find all the parameters by solving the perspective equation (1.1). This process requires to take multiple shots of the grid at different angles. As the algorithm tries to find the parameters with the best fit to the provided images. In Figure 1.3 the extracted corners are plotted on top of the checkerboard. Once all the values are found it is possible to undistort the image. As mentioned in the introduction to this section, the results were still not satisfactory, as can be seen in Figure 1.4. This was due to the fact that the camera plane is not parallel to the table itself. It can be seen in the same figure, that after undistorting the image, towards the edges of the table, the balls look ellipsoidal. We tried to solve this by using more complex distortion model within OpenCV, without any improvements. We also tried to use a different toolbox, namely Kalibr [10], which was developed by the Autonomous System Lab at ETH and supposedly delivers more accurate parameter than OpenCV. Unfortunately, also this didn't help in solving the distortion related issues.

Figure 1.4: Undistorted Image

Figure 1.5: Image after Homography

1.2.2 Homography

In order to overcome the limitations of simple calibration. It was necessary to apply another transformation, in particular, a *Homography* [4]. This is a perspective transformation, meaning that it has the same effect as rotating the camera would have. A homography allows transforming any quadrangle into any other quadrangle. In this particular case, the quadrangle from Figure 1.4 had to be transformed into a rectangle. Which was successfully achieved, as can be seen in Figure 1.5. A homography basically consists in finding the mapping between four points in the original image to four points in the desired final image. The equation is the following:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1.6}$$

Where (x',y') are the desired pixel coordinates and (x,y) are the coordinates on the original image. Four points are required in order to find a non-trivial solution for the H matrix. Since it has 8 DOF, the ninth parameter would be scale, which however is arbitrary. For the same reason, the four points have to be non-collinear. The points on the original image were found by extracting the lines corresponding to the edges and successively finding their intersection. The correctness of the applied transformation was verified through the detectors explained in the following sections. In particular, we checked whether the centroid of the ball would have a constant x or y component when moved along the sides of the table. This was indeed the case.

1.2.3 Line Extraction

Finding the internal sides of the table is not really a task related to how the image looks. However, it is an important step for the successive parts of the algorithm. As the sides correspond to the border of the space occupied by the balls. Knowing them allows removing false positives that occur during the actual ball detection. The line extraction was done by means of the Hough transform [7], that relies on representing lines in normal form: $x\cos\theta + \sin\theta = \rho$. The algorithm works as follows:

1. Convert the image to grayscale
2. Apply Gaussian Blur to remove noise
3. Find edges (usually through Canny Edge Detector)
4. Each edge point (x,y) votes for plausible line parameters (θ,ρ), which are stored in a voting array
5. Find the values (θ,ρ) for which maxima occur in the array. These values correspond to actual line parameters.

1.2.4 ROS Implementation

It is important to note that the *Preprocess* node in ROS does not always find all these parameters, but rather assumes that they have all been found. This is also valid for the lines, since they don't change, given that the camera, as well as the table, are fixed. So the node simply applies the transformations mentioned in the previous subsections, in order to publish the image in the desired format to the detectors.

1.3 Detectors

The detection of the balls was done with different detectors. In the beginning, we tried to use a single detector. But none of the possible solutions seemed to provide the desired robustness. Hence we decided to use several detectors, which are fused together by the means of a filter. As mentioned in the previous section, the extracted lines are used in all the detectors to delimit the table and remove false positives. Each detector was implemented as a ROS node. All of which subscribe to the preprocessed image and all publish the position of the detected balls.

1.3.1 Hough Circles

This detector is very similar to the algorithm that was used for line extraction in section 1.2.3. The working principle is basically the same. The edges have to be extracted. Once this is done, rather than trying to fit a line, a circle is fitted. The parametrisation used by the Hough Circle transform is the following: $(x-a)^2 + (y-b)^2 = r^2$. Hence in this case the parameter space (a,b,r) is three dimensional. Where (a,b) is the center of the circle, while r is the radius. When it comes to the latter, the search space can actually be reduced if its size is known. Which is true for the kind of detection we were trying to do. The rest of the algorithm is identical, a voting array is populated and its maxima are selected as the circle parameters. The main issue with this detector was its instability, probably related to the fact that the balls were slightly distorted, hence not allowing to perfectly fit a circle.

1.3.2 Color Blob

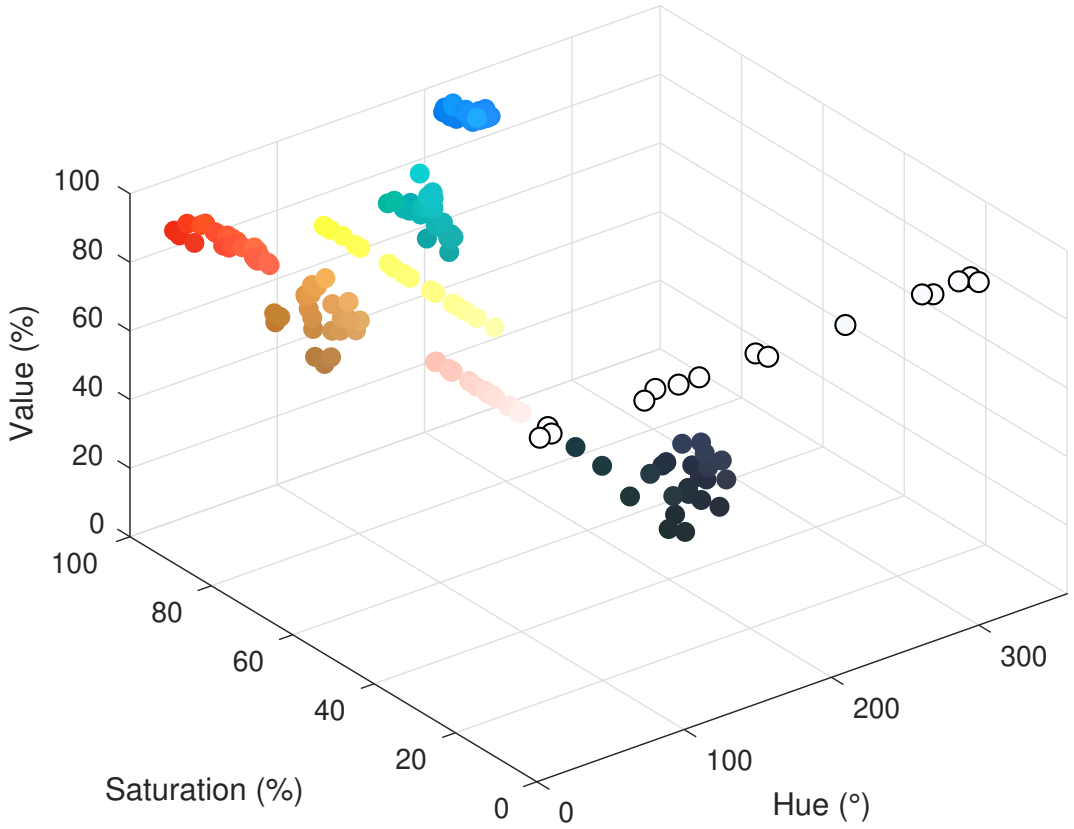

Figure 1.6: HSV Color detections

This blob detector finds balls based on their colour. This option is viable in our scenario, given that the colours of the balls are known. The colour information could be extracted in the RGB colour space. However, this colour space is additive, meaning that colours are obtained by linearly combining the red, green and blue values. This makes things

harder since chrominance and luminance data are mixed together. Hence, we decided to implement the colour tracking in HSV, which is the de facto standard for this task. In this colour space, colours are represented in a manner that is more comprehensible for humans. The Hue (H) refers to the dominant wavelength (i.e the pure colour it resembles). This value is measured in degrees and goes from 0°to 359°. The Saturation (S) provides information about how white colour is. It is measured in percentage. A saturation of 100% means that the color is pure, its other tints will have a lower saturation. Finally, the Value (V), gives information with regards to how dark colour is. As Saturation, it is measured in percentage. A value of 100% corresponds to black.

The actual detection is based on colour segmentation (Algorithm 1). In order to this, the colour values of the balls were analysed, to find suitable colour ranges. These were exploited to create a mask for each of the 8 colours. The masks were obtained by thresholding the images with the desired ranges .Since there always tends to be some noise, the masks were eroded and then dilated, before undergoing binary thresholding. Finally, to actually segment the balls, all the contours in the image are found. The contours are filtered out based on their area. Since the ball size is known, only contours having area within certain bounds are selected. Then the centroid of the contours is extracted and taken as the midpoint of the ball. As can be seen in Figure 1.8, this procedure produces very well defined contours. This is probably the detector with the highest degree of robustness among the implemented ones. However, it was not sufficient on its own, since it fails to correctly separate balls of the same colour if they are too close, as shown in Figure 1.9. The measured colour values used for detection are shown in Figure 1.6. HSV is usually represented through a cone or a cylinder, nevertheless, the figure shows that the colours are well separated.

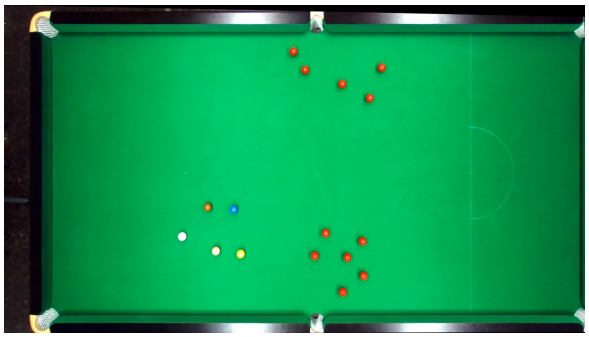

Figure 1.7: Table

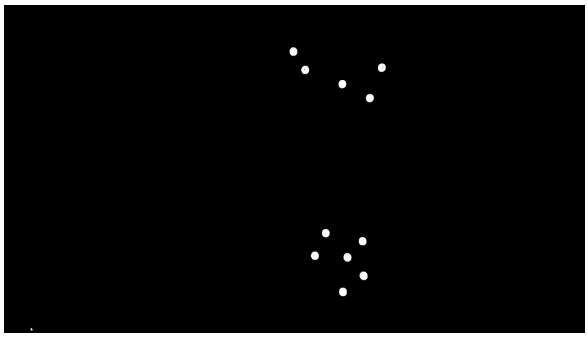

Figure 1.8: Mask Red

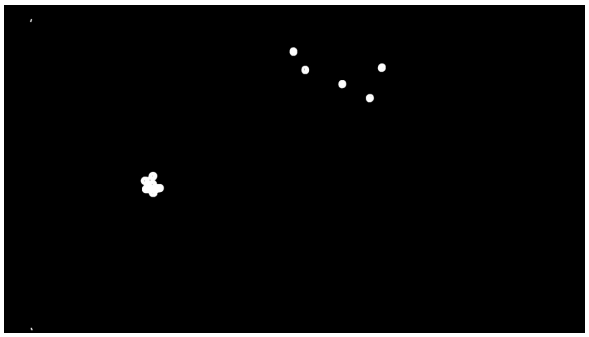

Figure 1.9: Not separable contours

Figure 1.10: Reflection

Algorithm 1 Color Blob

Data: Image, n_colors
Result: centers

```
hsv = BGR_to_HSV(Image)
for i ← 0 to n_colors do
    mask = inRange(hsv,l_bound(i),u_bound(i))
    erode(mask)
    dilate(mask)
    thr = threshold(mask,bound)
    cnts = findContours(thr)
    for c ← 0 to len(cnts) do
        if area(c) > lower && area(c) < upper then
            (x, y) = findCentroid(c)
            centers.append((x, y))
        end
    end
end
```

1.3.3 Reflection Detection

This detector is rather peculiar and exploits the fact that snooker balls are very reflective [5] . This property is exploited by binary thresholding the grayscale image, obtaining the output shown in Figure 1.10. One can see that certain balls appear in a more distinct way, these are white, pink and yellow. This happens simply because of the brightness of these colours. Similarly to what was done for the prevision detector, also here the contours are found and then used to extract the centroid. This might sound like a not very sound detection method. However, it works quite well, given that the reflection spans over almost the entire surface of the ball. Making its centroid a good estimate of the actual ball position.

1.4 Filtering

The system state estimation is completed by merging the measurements of all the detectors through filtering and colour assignment (Algorithm 2). The filter was implemented based on the one proposed by [2] and Wojke [14] for realtime tracking of humans. Their problem was different as they were trying to track humans in motion. When we implemented the filter we were not interested in tracking the balls during motion, but only once they reached a static configuration. However, in their filter, they also used a metric to decide whether a detection was valid or not. The decision is taken by keeping a list of *active humans* (i.e detected ones) and and a *to be confirmed* list. Incoming measurements were then matched to the previously stored ones by means of a Kalman filter with constant velocity assumption. We took over the idea of keeping a list of *active balls* and *to be confirmed* ones and adapted it to out specific use case. The actual implementation was done as follows. The filter receives all the ball detections from the detectors. These are then compared to the list of currently *active balls*. The matching is done by checking whether the distance between the centroids is below a certain threshold (radius). If this is the case, the *active ball* is confirmed and its position is update by averaging the centroid with the ones of all matches. All the *active balls* which didn't match any of the incoming measurements are

then discarded. Once this first comparison step is completed, the incoming measurements that didn't match any of the *activate balls* are compared to the list of *to be confirmed balls*. The process stays the same, with the only difference that in case of a match, the *to be confirmed ball* changes status to *active*. Also, in this case, all the balls that didn't match any of the incoming measurements are discarded. All the new detections which didn't match any ball in the two lists are stored as *to be confirmed*. After this filtering process we have a list of *balls*. However, until here they have no colour associated with them. Since all the detectors, even the colour blob one, only output position. In order to assign a colour the same thresholded images as in section 1.3.2 are used. This is done by checking whether around a given centroid there are enough pixels of a certain colour (color_check in the algorithm). If this condition is verified the ball is assigned to that specific colour. This last assignment step also allows discarding any leftover false positives.

1.4.1 Value Transform

Up to this point, all the nodes only extract pixel coordinates, rather than millimetre values. Hence it was necessary to set up a dedicated transformation. This was also done based on the extracted lines, as they provide information with regards to where the surface of the table starts and ends. The image has a left-handed coordinate system with origin in the top left corner of the image. Thus, in agreement with the AI model, it was decided to choose the top left corner of the table as its origin. The pixel coordinates of this point are subtracted to all the detected balls since they represent an offset. Finally, two multiplicative coefficients were found to convert from pixels to mm. This was done by knowing the distance of the parallel table sides in both the image plane and on the physical table. Yielding final coefficients of $(1.75816mm/px, 1.65525mm/px)$ respectively for x and y. These transformed values are the ones which are then published by the node and used by the AI system to decide what move to choose next.

1.4.2 Ground truth measurements

In order to validate the results of the filter, its output was compared to the ground truth values. These were obtained by placing a millimetre grid on top of the snooker table (Figure 1.11) and taking measurements by hand.

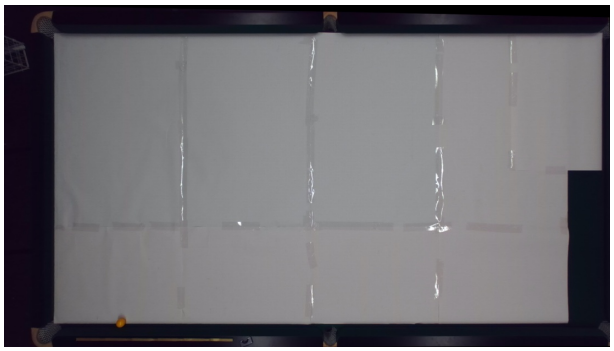

Figure 1.11: Ground truth measurements

Algorithm 2 Filtering

Data: d
Result: active_b

```
a = active_b
t_b = to_be_determined
empty(active_b)
empty(to_be_determined)
for i ← 0 to len(a) do
    center = a(i)
    n = 1
    flag = 0
    for j ← 0 to len(d) do
        if dis(center,d(j)) < bound && not_detected(d(j)) then
            flag = 1
            center += d(j)
            n++
        end
        if flag then
            tmp.append(center/n)
        end
    end
end
for i ← 0 to len(t_b) do
    center = t_b(i)
    n = 1
    flag = 0
    for j ← 0 to len(d) do
        if dis(center,d(j)) < bound && not_detected(d(j)) then
            flag = 1
            center += d(j)
            n++
        end
        if flag then
            tmp.append(center/n)
        end
    end
end
for i ← 0 to len(d) do
    if not_detected(d(i)) then
        to_be_determined.append(d(i))
    end
end
for i ← 0 to len(tmp) do
    if color_check(tmp) then
        active_b.append(tmp(i))
    end
end
```

The measurements (Figure 1.12 and Figure 1.13) revealed that the that the distortion towards the edges of the table was indeed influencing the overall accuracy. In these regions, the error for both x and y reaches values close to ± 3 mm. While in more central, not affected by distortion, areas the error is in the $[-1, 1]$ range. An attempt to recalibrate the measurement system was made. However, this was unsuccessful, given that there was no tendency of the algorithm to systematically over or underestimate the measurements.

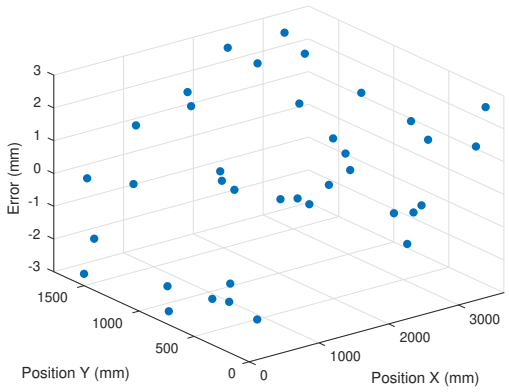

Figure 1.12: Error X axis

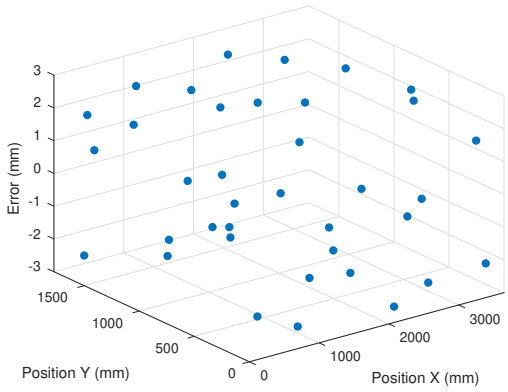

Figure 1.13: Error Y axis

1.5 Visualisation

The final two nodes don't serve the state estimation algorithm but are intended to provide visual feedback for the user. The *Show* node plots the detected centroid as well the table sides on the image. Since the Vision computer is going to be mounted on the ceiling. The final image is published as a message itself and can be visualised through the *Web video server* node, after having established an SSH connection to the computer. In Figure 1.14 the balls detected by the filter are displayed on the table image. This corresponds to what would be seen through the *Web video server*

1.6 ROS Performance

When it comes to the performance of the actual algorithm, there seem to be two main bottlenecks. The *Preprocess* node is not able to publish images at the same frequency as it is receiving them from the camera. Rather than working at 15Hz, it drops to about 9 Hz. This is most likely caused by the high resolution of the images and by the fact that different steps have to be taken to get a satisfactory image, as explained in section 1.2. The second bottleneck is the *Color Blob* detector. Which has a frequency of about 2 Hz. The reason for this drop is related to the number of operations that have to be executed since a thresholded image has to be obtained for each of the 8 colours. The other detectors manage to work at 7.5/8 Hz, hence at a frequency comparable to one of the incoming preprocessed images. However, due to the *Color Blob*, the overall publishing frequency of the filter is of about 2 Hz.

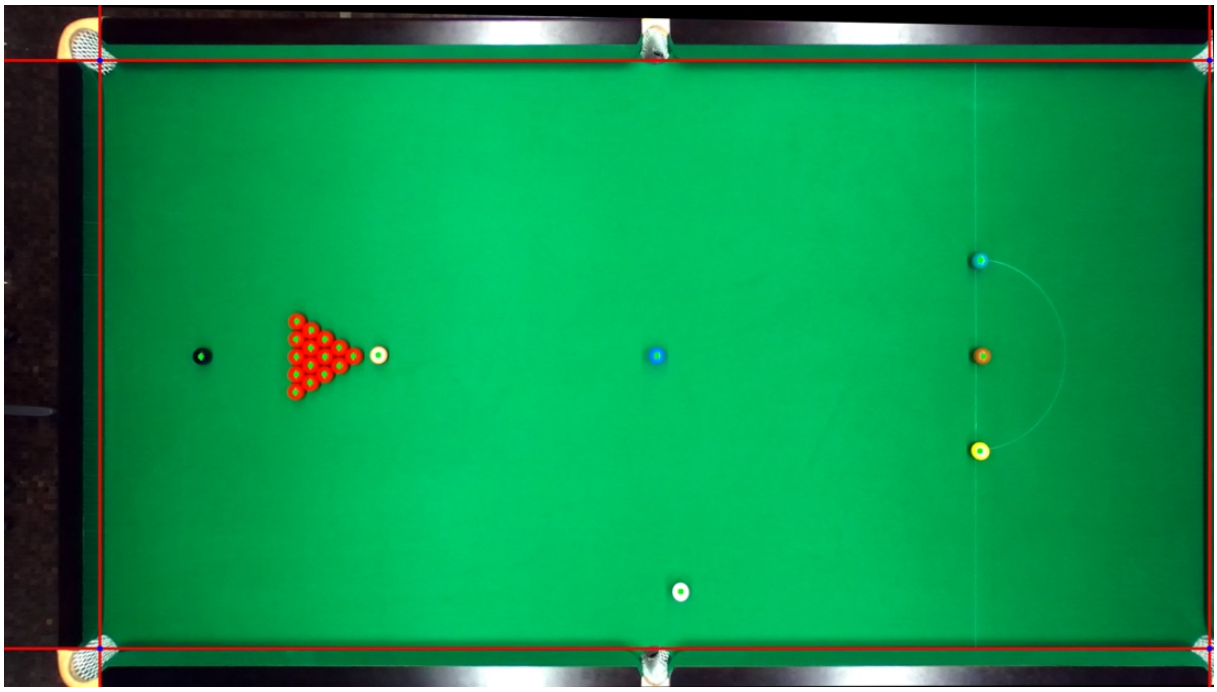

Figure 1.14: Filter output

Chapter 2

Table Parameters

In this chapter, we will focus on how to exploit the vision setup to extract some of the dynamical parameters of snooker. The interactions between the balls and the table were studied, by finding the friction in both the sliding and rolling regime [6] [13]. As well as the coefficient of restitution of the ball cushion collisions. Moreover, the impact between the cue ball and the target balls was studied, in order to extrapolate the coefficient of restitution [12] [3]. A similar analysis was completed by Mathavan et al. [9] in 2009. Mathavan suggests using a high-speed camera in order to be able to capture the dynamics as good as possible. Thus, we decided to operate the ceiling mounter camera at a higher frame rate. The ZED allows the FPS to be increased up to 100, this however comes at the cost of having to work at a lower resolution (Wide VGA, 768x480 px). In order to use the camera to take the required measurements with the desired settings, all the preprocessing steps described in section 1.2 had to be done again (i.e calibration and homography). The ball detection pipeline that was previously implemented was not suitable for this task. Since the balls will be in a different position on a frame to frame basis, it would make little sense to use the filter from section 1.4. Instead, we simply tried to run the Color blob detector on each frame. This detector was used since it is the one that provides the best results. Initially, we were doing a real-time tracking but immediately realised that this was not going to work. Due to the time required to do the detection some of the frames were lost. Hence, we decided to first capture all the images and then analyse them offline.

2.1 Friction

Before starting to analyse the obtained data, it is important to explain what kind of models have to be taken into consideration when trying to establish the interaction between the balls and the table. The simplest model is the one in which both the ball and the table are considered rigid. A more complex, but more correct model for the snooker table, is the one in which the surface is assumed to be deformable [6] [9]. Below a brief explanation of both models will be provided.

2.1.1 Rigid sphere on a rigid horizontal plane

We consider a rigid sphere of mass M and radius r, with initial velocity v_0. The motion of the sphere undergoes two distinct phases. In the first one, the frictional force acts on the sphere it the direction opposite to its centre of mass (C_m) , as can be seen in Figure2.1

(a). In this situation, the sphere is rolling and slipping at the same time. The dynamics are given by the following set of equations [6]:

$$\sum F = Ma_{cm} \qquad -f = Ma_{cm} \tag{2.1}$$

$$\sum \tau = I_{cm}\alpha \qquad f_r = I_{cm}\alpha \tag{2.2}$$

Where a_{cm} is the acceleration of the centre of mass, while I_{cm} is the moment of inertia about the axis through the centre of mass and τ is the torque. The slipping friction f between the table and the balls is given by [6]:

$$f = \mu_k Mg \tag{2.3}$$

Where μ_k is the friction coefficient. Coupling this equation with the previous ones allows us to find the relationship between the acceleration of the centre of mass and the friction coefficient:

$$a_{cm} = -\mu_k Mg/M = -\mu_k g \tag{2.4}$$

After a certain time the angular speed of the sphere will increase, while the linear one will decrease. Until $v_{cm} = r\omega$, which is when the balls start to roll without slipping. If we assume a perfectly rigid surface the rolling would continue indefinitely, with both v_{cm} and ω staying constant. However, in a real scenario, at least one of the surfaces is deformable, which introduces also a rolling friction coefficient.

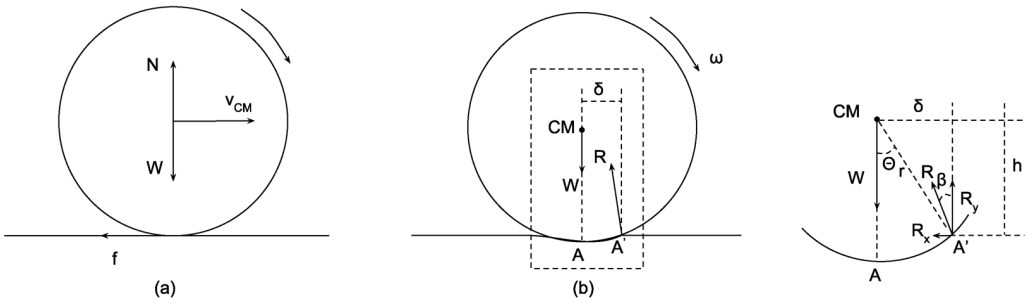

Figure 2.1: Rigid ball and surface (a), rigid ball on deformable surface and force distribution (b). Adapted from Hierrezuelo and Carnero [5]

2.1.2 Rigid sphere on a deformable horizontal plane

This scenario is shown in Figure 2.1 (b). Due to the deformation of the horizontal plane, the normal force won't be applied to point A, but to point $A^{'}$ instead [6]. Hence it will have a horizontal (R_x) and a vertical component (R_y). This leads the ball movement to be governed by the following equations [6]:

$$-R_x = Ma_{cm} \tag{2.5}$$

$$R_y - W = 0 \tag{2.6}$$

$$R_x h - R_y \delta = I_{cm}\alpha \tag{2.7}$$

By taking into account the moment of inertia of a sphere, it can be shown that

$$\frac{R_x}{R_y} = \frac{5}{7}\frac{\delta}{r} \tag{2.8}$$

where $\frac{\delta}{r} = \sin(\theta)$, assuming a small deformation, both β and θ will be small and we obtain

$$R_x = R\beta = R_y\frac{5}{7}Mg\theta \tag{2.9}$$

This leads to a constant rolling resistive force R, which is independent from velocity and purely determined by the surface deformation. Moreover, R_x is the main cause for deceleration.

2.1.3 Measurements

As one might expect the model introduced by equation (2.9) is rather difficult to implement. Hence, we decided to exploit the relationship between a_{cm} and μ_k given by equation (2.4). Meaning that the only thing needed in order to determine the friction coefficient was the value of the deceleration along the table. To obtain this value, the ball speed had to be determined first. Originally, we wanted to do so by simply looking at the position change of the ball in each frame. However, the ball detection seemed to suffer under the low resolution at which the camera was being operated. Leading us to use this equation to determine the speed:

$$v_i = \frac{p_i - p_{i-k}}{t_i - t_{i-k}} = \frac{\Delta p_{(i,i-k)}}{\Delta t_{(i,i-k)}} \tag{2.10}$$

It can be seen that the current position was compared to the position k step ago, with k varying between $[10, 15]$. The value k can be considered as a smoothing factor, that allows compensating for lack of measurement accuracy. For each measurment k was chosen to be the smallest possible value which would produce a reasonable speed plot, as the one that can be seen in Figure 2.2. We knew that this the sort of speed representation was what we wanted to obtain thanks to reference [9]. The plot also reveals that the speed can be modeled rather well through a piece-wise linear function. Hence, we decided to do so. This then made the task of finding the deceleration straightforward, since it only required to extract the gradient of the speed. This was leading to a constant friction model. The other nice thing about the speed plot is that based on the steepness of the line segment it is possible to determine whether the ball is rolling or slipping. When the line segment is very steep we can assume that the ball is slipping. It is also interesting to note that when the ball hits the cushions, it starts to slide for a short period of time, due to the fact that the collision violates the $v_{cm} = r\omega$ assumption.
With the procedure described above, we obtained dimensionless slipping friction in the range $0.1199 - 0.1647$ $(1.175 - 1.614\frac{m}{s^2})$ and dimensionless rolling frinction in the range $0.0085 - 0.0116$ $(0.0833 - 0.114\frac{m}{s^2})$. The values we obtained are comparable to the ones reported by Mathavan et al. [9], that obtained the following ranges $0.178 - 0.245$ and $0.0127 - 0.0129$ for the slipping and rolling friction respectively. The value reported by Mathavan et al. lead to a slipping friction which is $15 - 20$ times bigger than the rolling one, while in our case the factor is $10 - 20$.
Given that the rolling friction range was rather big, we also tried to explore whether there was some sort of correlation between speed and friction. We didn't succeed and as suggested in 2.1.2 the friction is most likely constant. However, the quality of our data

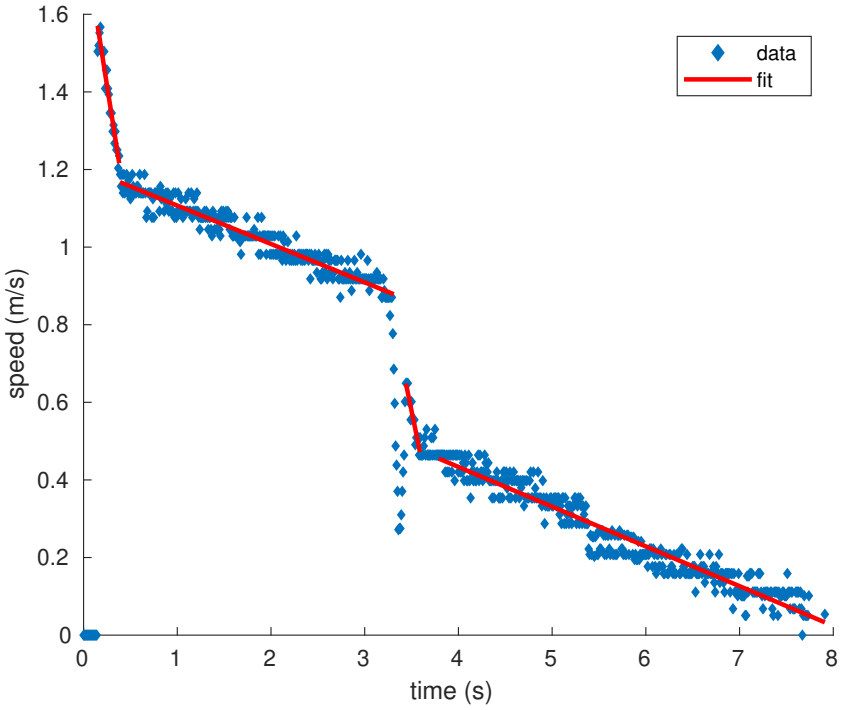

Figure 2.2: Speed Plot Single Ball

doesn't allow us to narrow down the ranges. We also tried to see whether the directional nap of the table cloth would influence the friction coefficients by any means. But no such correlation was detected

2.2 Interaction with cushions

The interactions between the balls and the side cushions was studied in order to find the coefficient of restitution. The latter is defined as follows:

$$e = \frac{v_{ra}}{v_{rb}} \tag{2.11}$$

Where v_{rb} and v_{ra} are the relative speeds before and after the collision. In this scenario, given that the side cushion doesn't move, both values correspond to the ball's speed component perpendicular to the cushion.
To find the values the same speed plots as in the previous section were used. Figure 2.3 and 2.4 both refer to the same shot. The first one is the speed plot, in which 3 bounces are visible. Which is visually confirmed by the second image, on which the position over time is plotted on top of the snooker table.
From Figure 2.5 it can be deduced that the relationship between the rebound and incident speed is linear. Hence a line was fit to all the measured values, yielding a coefficient of restitution of 0.8100. Which is very close to the value of 0.818 reported by Mathavan et al. In their paper, they suggest that the values are more closely fit by a second order polynomial. They attribute this to the fact that at higher speeds the collisions are more strongly influenced by the deformation of the table sides. We also proceed in fitting a second order polynomial. Which indeed ended up having slightly lower residuals.

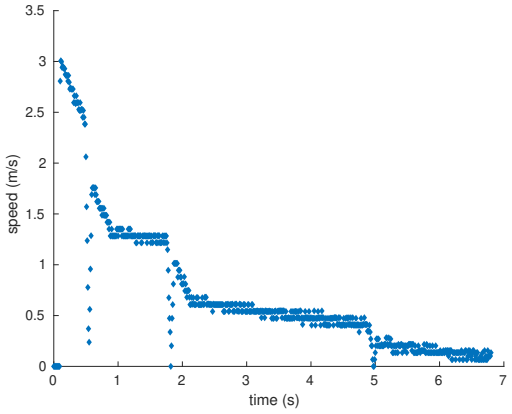

Figure 2.3: Speed plot

Figure 2.4: Ball-Cushion collision

However, we believe that the actual relationship should be linear, as the second order fit seems to be very much influenced by one single point. Which probably could be considered as an outlier. It can be seen that in our plot there are too many incident speed values above $3\frac{m}{s}$, this is due to the filtering procedure we had to use. This led to the loss of information if the collision occurred almost immediately, which unfortunately is the only scenario in which such high speeds were measurable.

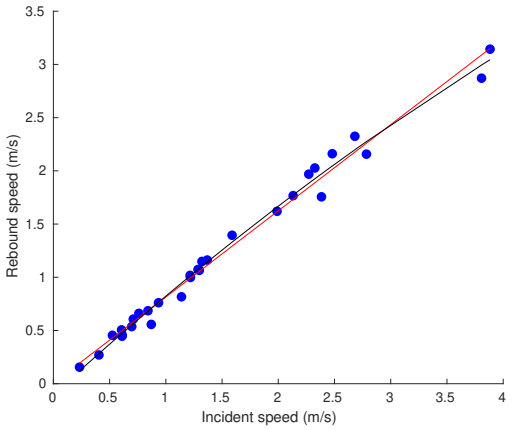

Figure 2.5: Coeff. of Restitution Ball-Cushion

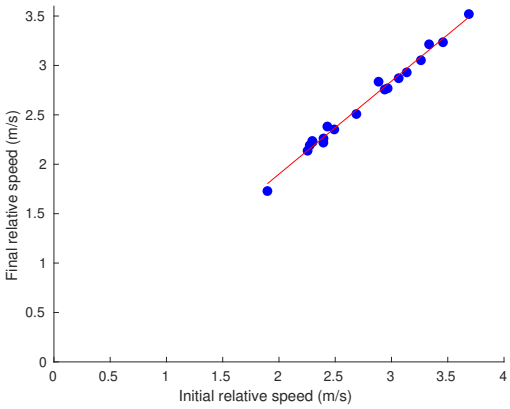

Figure 2.6: Coeff. of Restitution Ball-Ball

2.3 Impact between balls

As mentioned in the introduction also the interaction between the cue ball and the target balls was analysed, in order to find the coefficient of restitution between them. Before it was relatively easy to extrapolate the value, given that the cushions are fixed. Things are slightly more complex when looking at collisions between two balls, as both are moving. Since we were only interested in finding the coefficient of restitution, it was not necessary to define a global reference frame and defining the speed components of the two balls in it. As can be seen in Figure 2.7 , we decided to assume that the collisions occur in the direction orthogonal to the motion of the target ball. A local reference frame was set up based on this choice. Thus, in order to complete the measurements the cue ball speeds

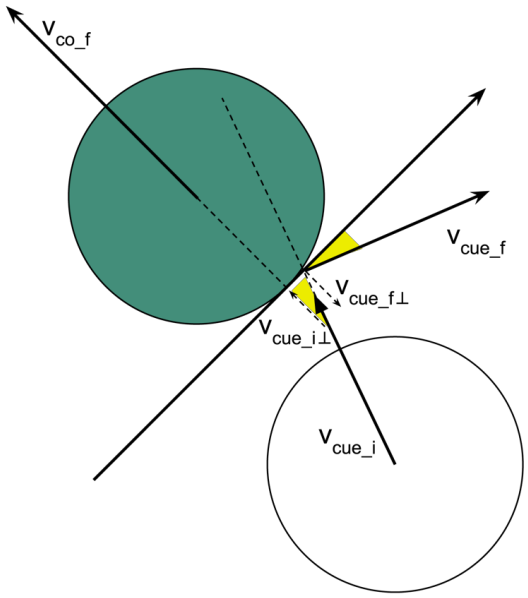

Figure 2.7: Ball-Ball Collision

before and after the collision had to be expressed in the above-defined reference frame. The required angle measurements were obtained by fitting lines to the ball displacement data. Once their slope was found, it was possible to simply calculate the desired angle. These values were then used in combination with the speed plots to extrapolate the desired values. This was done using equation (2.11) and expanding it as follows

$$e = \frac{v_{rf}}{v_{ri}} = \frac{v_{co\_f} - v_{cue\_f}}{v_{cue\_i} - v_{co\_i}} = \frac{v_{co\_f} - v_{cue\_f}}{v_{cue\_i}} = \frac{v_{co\_f} - v_{cue\_f\perp}}{v_{cue\_i\perp}} \tag{2.12}$$

The numerator represents the final relative speed and is obtained by taking the difference

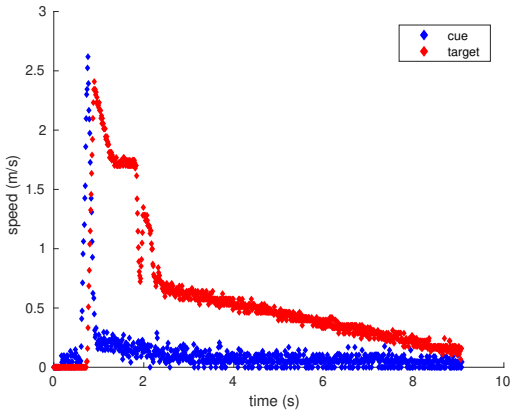

Figure 2.8: Ball-Ball collision speeds

Figure 2.9: Ball-Ball collision plot

between the target ball's speed, which is by default expressed in the correct reference frame, and the cue ball's speed component orthogonal to the collision plane. While at

the numerator we only have the initial orthogonal component of the cue ball speed, given that the other ball is assumed to be still before the collision occurs.
In Figure 2.8 the speed of the cue ball and the target ball are shown before and after the collisions. While in Figure 2.9 the path of the two balls is plotted on the snooker table.
Also, in this case, the final coefficient of restitution was found through linear regression. Which this time was the best possible fit. The final value was of 0.947. Such a high value did not come as a surprise since we assumed the collisions would be very close to being perfectly elastic. The linear fit plot is shown in Figure 2.6.

Chapter 3

Cue Camera System

In the last part of the project, we focused on setting up a cue mounted camera, with the aim of a "player's eye" perspective. For this specific task, we decided to choose the *Intel Realsense D435* due to its depth sensing capabilities. In Chapter 1, we saw how preprocessing the images slowed down the detection pipeline. This is not a problem with the *Realsense* since all the preprocessing is done directly within the camera and the ready to use image is provided to the user. Out of the box the camera is calibrated, and if necessary the parameters can be adjusted through dedicated software. Also, the stereo images are directly processed on board, hence alongside with the undistorted image also the depth map is provided. Both of which were exploited to set up a second ball detection algorithm.
To detect the position of the cue camera, we decided to mount an ArUco marker on top of the linear motor. The marker was not only exploited for finding the camera but also to determine the cue angle. This information, together with the desired hitting angle provided by the AI, was used to set up a *Human in the Loop* system, which shows the user how to align the linear motor in order to be able to take the shot exactly as planned.

3.1 Cue Camera ball detection

In this section, a brief introduction to depth sensing will be provided, before talking about the actual implementation of the ball detection.

3.1.1 Stereo Vision

Depth from stereo (i.e. Stereo Vision [8]), deals with the task of recovering the 3D structure from images, assuming that for both cameras intrinsic parameters are known. The idea is inspired by the human vision system. Our eyes also represent a *"Stereo System"*. The images detected by our left and right eye are not perfectly aligned, there is a horizontal displacement called *disparity*, which is exactly what allows us to perceive depth. The set up in Figure 3.2 is a simplified case, that however perfectly suits our needs, given that the camera is implemented in the same manner. The following equations can be found from similar triangles

$$\frac{f}{Z_p} = \frac{u_l}{X_p} \tag{3.1}$$

$$\frac{f}{Z_p} = \frac{-u_r}{b - X_p} \tag{3.2}$$

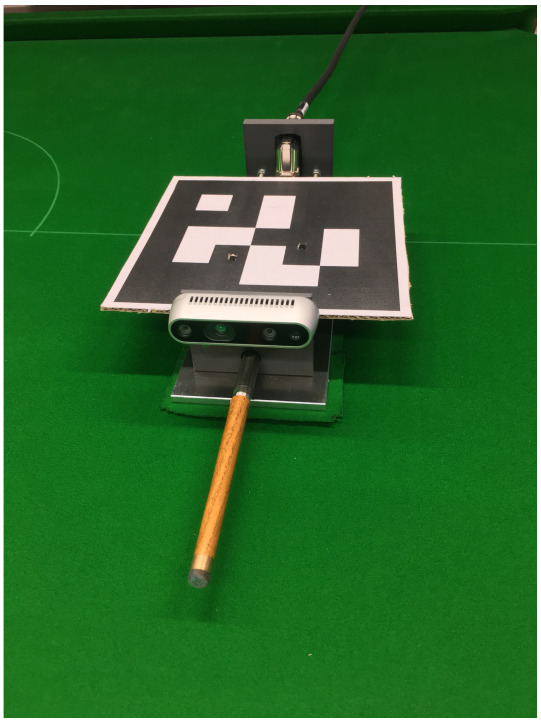

Figure 3.1: Cue Camera

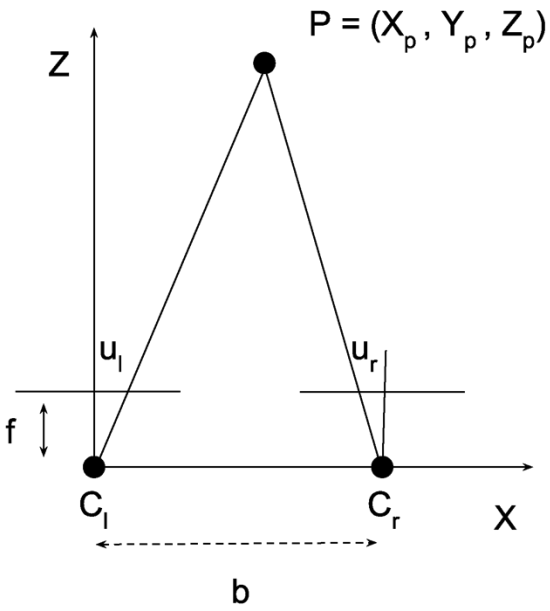

Figure 3.2: Stereo Setup

Combining the two equations it is possible to extrapolate the depth values.

$$Z_p = \frac{bf}{u_l - u_r} \tag{3.3}$$

It is important to note that $u_l - u_r$ is the disparity, while b is the baseline, which corresponds to the distance between the two camera centres and f is the focal length, which for simplicity is assumed to be the same.

3.1.2 Cue algorithm

The algorithm for ball detection through the cue camera builds upon previous parts of the project. It is indeed structured in a very similar to the ceiling camera one. When it comes to the actual detectors, as in 1.3.1 the Hough Circles transform was used. Due to the fact that now the camera does not have a top view anymore, the balls do not have the same size all over the image plane. This was taken care of by simply allowing circles to have a wider range of radiuses. Additionally, also the colour blob detector (Algorithm 1) was reused. Also, this detector had to be slightly changed. The main issues were caused by the fact that by seeing the balls from the side, the lower half would be in the shade of the upper half. Hence the HSV colour ranges had to be changed to take care of this. In some cases, two distinct ranges were needed, i.e. two separate masks, which were combined to find the ball through contour extraction. Moreover, for both detectors, the size of the radius was stored alongside the ball coordinates.
The filter (Algorithm2), was reused in order to obtain as accurate as possible detection estimates. Given the different camera pose, the algorithm had to be slightly adjusted. With the top camera it was sufficient to merge detections based on their proximity in the image plane. Since the real world surface, we were dealing with was parallel to the latter.

This was not valid anymore. Hence, rather than combining the images solely based on their image plane distance the depth information was exploited as well. First, we checked whether the distance between the balls was smaller than the radius of the bigger one. Whenever this condition was met, we checked if the pixel coordinates themselves were close enough for a match. If this was true, the balls were merged.

Figure 3.3: Cue detection

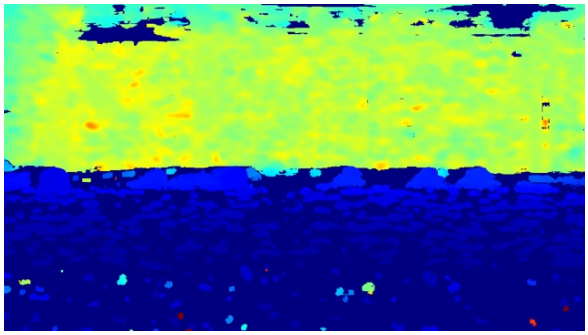

Figure 3.4: Depth

When we were dealing with the ceiling camera, the working environment was controlled, since we were able to easily assess whether a ball was on the table or not. In this scenario, things were slightly more difficult, given that at each camera displacement a different part of the table would be visible. The depth measurement was used to partially mitigate the false positives problem, by simply ignoring points above a certain distance. This fix wasn't good enough, hence the table surface was extracted through colour detection, Figure 3.6. This was done exactly as for the *Color Blob* detector. This information was exploited by fitting a square around the balls and checking that at least one of the corners was on the table. If this condition was verified, the ball would be considered active. The results of the ball extraction can be seen in Figure3.3, while in Figure 3.5 the same image is provided with distance measurements, which were obtained through the depth map in Figure 3.4.

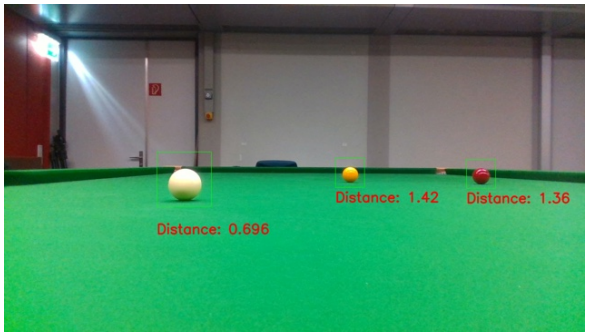

Figure 3.5: Detections with distance

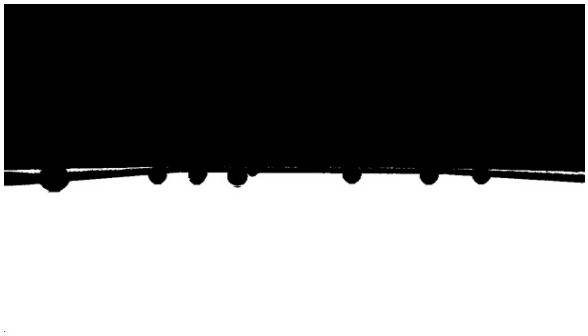

Figure 3.6: Table detection

3.1.3 Issues

Different problems did arise while implementing the algorithm. The first one was caused by the height of the cue camera. Due to the fact that it is not placed high enough, based on how the balls are configured on the table occlusion might arise. This can become a serious issues if one tries to do some sort of sensor fusion between the two cameras. Especially

during the late stages of the game, when the number of balls on the table starts to reduce and due to how they are positioned on the table. It might become extremely hard to extract enough meaningful measurements, even more so if some occlusions occur.
We noticed that the depth error was getting significant for distances beyond $2m$, it would reach values of more than 10%. This caused some issues for the detector since it would occasionally occur that balls are merged, due to the erroneous distance measurements, even though they were not actually close enough. This high error turned out not to be an all too surprising factor. In the documentation provided by *Intel*, it is proven that the *RMS error* depends on the square of the distance of the desired point.

3.2 Pose Estimation

As mentioned at the beginning of this chapter, the camera\cue pose estimation was done through the aid of an ArUco marker (Figure 3.1). In many robotics application, vision systems are exploited for pose estimation. This process consists of finding the correspondence between a real-world point and its 2D image plane projection. Given that this task is usually hard, it is common to use a fiducial marker, since this lower the complexity of the problem. One of the most popular of these markers comes with the ArUco library [11] [1].

3.2.1 ArUco: Detection & Pose

The marker is a synthetic square composed by a black border and an inner binary matrix, which determines the identifier. The black border allows for fast detection, while the binary codification allows maker identification as well as error detection and correction. The marker can be found in any possible rotation, but the detection process is still able to find the corners unequivocally, this is possible thanks to the binary codification [1].
In order to detect the marker, the original image has to be converted into a binary one through thresholding, which is shown in Figure 3.7. The algorithm then extracts the

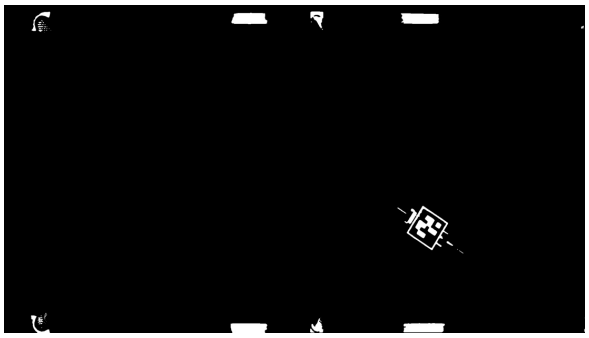

Figure 3.7: ArUco threshold

Figure 3.8: ArUco axis

contours of the thresholded image. Once this step is completed, by analysing the inner codification it possible to determine if the detected object is indeed a marker. The candidate tags are divided into four cells, the bits of these are compared to the user-specified dictionary. In Figure 3.8 the tag was correctly identified and its corner points were drawn. Once the marker was correctly identified it is possible to proceed with the actual pose estimation. To do the latter it is necessary to know the calibration parameters of the camera (i.e. intrinsic camera matrix and distortion coefficients). The algorithm returns

the rotation and translation vectors needed for implementing the homogeneous transformation from tag to camera frame. This data was exploited to plot the tag reference frame in Figure 3.8. It can be seen that the origin of the frame corresponds with the centre of the marker.

3.2.2 Human in the Loop

Given that our system right now is solely based on a linear motor, rather than a complete robot. We have to manually move the motor in order to obtain the desired orientation for the shot. Hence the marker detection algorithm was added to the ROS system, in particular to the *Show* node, to provide the user with some visual feedback of how he or she has to move. In Figure 3.9 one can see that the tag was placed on the linear motor

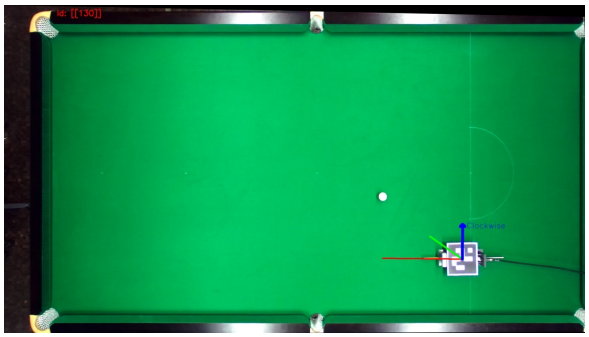

Figure 3.9: Not aligned marker

Figure 3.10: Aligned marker

in such a way that one of the axes would be aligned with the cue. Thanks to the pose estimation, the coordinates of this axis are known in the image plane. This information is enough to extrapolate the tag orientation. As mentioned in one of the previous sections the image has a left hand sided coordinate system, centred in the top left corner. The same kind of coordinate system was applied to the table. Hence, the positive x-axis goes from the upper left to right table corner. While the positive y-axis goes from the upper to lower left corner. The angle was defined in a counterclockwise fashion, starting at zero on the x-axis and increasing as we move towards the y-axis. The green line in Figure 3.9 corresponds to the desired angle for the shot, this information is provided by the AI model. Moreover, the blue arrow provides the user with the direction the tag has to be rotated to in order to achieve the desired orientation. Once this is achieved, both the green line and the blue arrow disappear, as can be seen, in Figure 3.10. Due to the fact that it is rather difficult for a human to perfectly align the tag with the desired position. The matching condition was considered as the one in which the absolute value of the difference between desired and the measured value was lower than 0.5°.
In order to verify the performance of the algorithm, we decided to place linear motor at different angles through the procedure described above. Then a shot was taken with the motor and the ball path was extracted through the camera system, by analysing the images that were saved frame by frame. A line was fit to the path in order to be able to find the angle at which the ball moved with respect to the table reference frame (Figure 3.11). Unfortunately, when taking these measurements, the linear motor would sometimes move before hitting the ball, since its wasn't heavy enough. The just described scenario led to the highest absolute error measurements, with a value of about 4°. Given that the motor, rather than the pose estimation, is the cause of this problem, we ignored all the image sequences in the which the motor moved. In this way, the average absolute angular

Figure 3.11: Ball path Linear Motor

error is of 1.27°. Which we consider good enough, considering the human involvement. In Figure 3.12 the final output of the *Show* node can be seen.

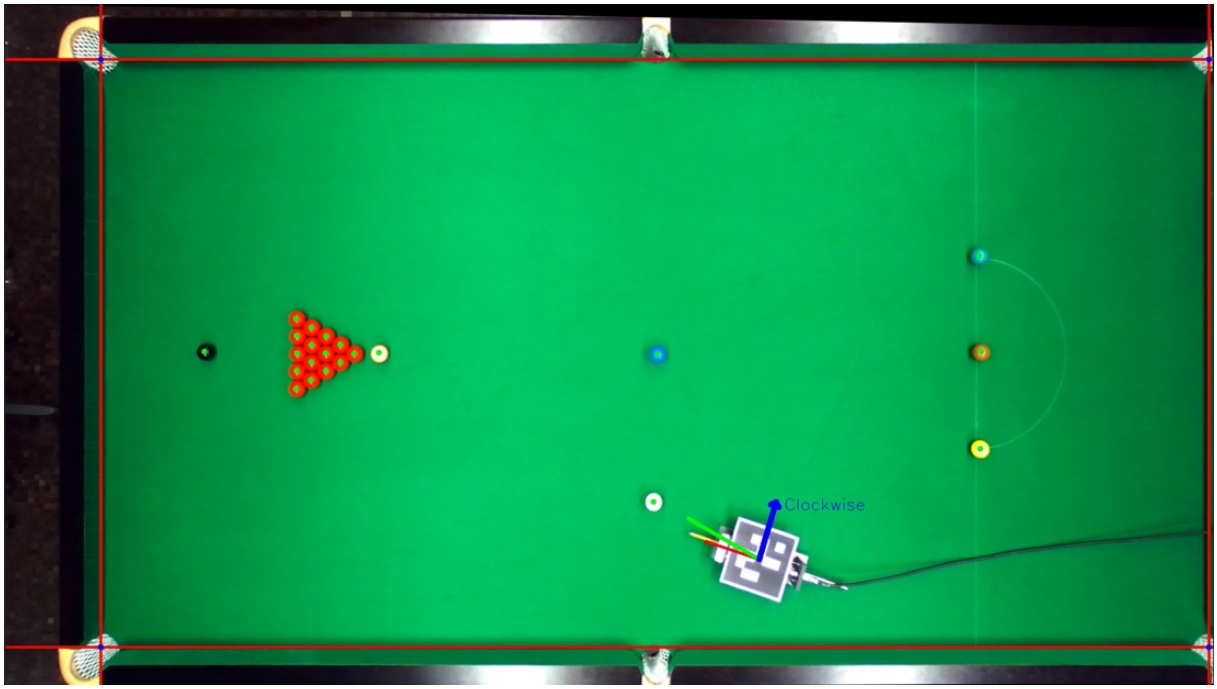

Figure 3.12: ROS final Output

Chapter 4

Conclusion

In this project, we developed a ceiling mounted ball detection system. Which provides position estimates with relatively good accuracy, even if there are some image distortion problems left. In the future, this could be solved by using more than one camera, each in charge of smaller table portions. The filtering algorithm that was implemented, would be able to handle such a change without issue. Obviously, if such a change was to be made, one had to be sure to exactly calibrate the system. So that the exact coordinates of all the table portions are known. If this is not done, one might actually worsen the estimate. The vision system was also exploited to find the dynamical parameters of the table. In particular both the slipping and rolling friction coefficients, the coefficient of restitution between the cue ball and a target ball, and between the cue ball and the sides of the table. Even though the available measurements seemed rather perturbed, the obtained values were reasonable and comparable with the ones that can be found in the literature.
The last part was concerned with setting up a cue camera for ball detection. The latter will be used to obtain even better position estimates in the future, by fusing the data from both cameras together. Moreover, some features were added that allow a human to interact with the system, in a *Human in the loop* fashion. This is only a temporary feature, as in the feature a robot should take over the positioning of the cue.

Bibliography

[1] Detectio of aruco markers. `https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html`. Accessed: 2018-12-21.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 3464–3468. IEEE, 2016.

[3] A Domenech and E Casasús. Frontal impact of rolling spheres. *Physics Education*, 26(3):186, 1991.

[4] Elan Dubrofsky. Homography estimation. *Diplomová práce. Vancouver: Univerzita Britské Kolumbie*, 2009.

[5] Hao Guo and Brian Mac Namee. Using computer vision to create a 3d representation of a snooker table for televised competition broadcasting. 2007.

[6] J Hierrezuelo and C Carnero. Sliding and rolling: the physics of a rolling ball. *Physics Education*, 30(3):177, 1995.

[7] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[8] David Marr and Tomaso Poggio. A computational theory of human stereo vision. *Proc. R. Soc. Lond. B*, 204(1156):301–328, 1979.

[9] S Mathavan, MR Jackson, and RM Parkin. Application of high-speed imaging to determine the dynamics of billiards. *American Journal of Physics*, 77(9):788–794, 2009.

[10] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 473–480. IEEE, 2013.

[11] Rafael Munoz-Salinas. Aruco: a minimal library for augmented reality applications based on opencv. *Universidad de Córdoba*, 2012.

[12] R. Evan Wallace and Michael C. Schroeder. Analysis of billiard ball collisions in two dimensions. *American Journal of Physics*, 56(9):815–819, 1988.

[13] J. Witters and D. Duymelinck. Rolling and sliding resistive forces on balls moving on a flat surface. *American Journal of Physics*, 54(1):80–83, 1986.

[14] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *Image Processing (ICIP), 2017 IEEE International Conference on*, pages 3645–3649. IEEE, 2017.

[15] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.