PROJECT
SMART PINBALL

2004

FINAL REPORT

Final report

SMART PINBALL PROJECT

By

Simon Gustafsson

Juan Emilio Rubio Munoz

Susanna Norell

Denis Real

Yunyi Xiao

Abstract

The task is to make a pinball play on its own, or put in another way, "without human intervention". For that we'll need some external help, in the form of computational power and communication between resources as a digital camera, which records a top view of the machine, and a bally controller that activates the flippers. The rest is learning how to transform that raw processing power into a series of specific algorithms that suit the task. The result must be an automatic system that keeps track of the ball, and flips it when necessary.

We propose a solution where a DSP is fed with a continuously updated sequence of grayscale images taken by a webcamera. In the DSP, calculations are made to predict the location of the ball and its future position. The algorithm for detection uses only a smaller part of the grabbed frame from the webcamera. This part of the image is subjected to smoothing, to minimize the noise level, before the brightest pixel in the image is defined as the balls current position.

For prediction we use two methods; linear prediction and Kalman filtering. Both methods work satisfactory and they both manage to beat the high score achieved in the pinball project in 2003. Also, the number of unnecessary flipper activations is virtually zero.

The prototype comes with a GUI which allows the user to edit decision regions around the flippers, view the speed and direction of the ball and chose whether to use linear prediction or Kalman filtering to predict the balls position.

Possible applications to use our system solution in could be safe car driving, surveillance systems, visual tracking for aircrafts and computer vision.

1. INTRODUCTION 1

1.1 Background 1

1.2 Problem 1

1.3 Objective 1

1.4 System model 1

1.5 Equipment 2

1.6 Outline 3

2. THEORY 3

2.1 Ball detection 3
2.1.1 Image capture – size and colour mode 3
2.1.2 Creation of a difference image 4
2.1.3 Smoothing of the difference image 5
2.1.4 Handling lights 5
2.1.5 Processing of the whole or just the part of the image 6
2.1.6 Finding the ball in the image 7

2.2 Ball movement prediction 7
2.2.1 Why use prediction – system delay 7
2.2.2 Kalman prediction 8
2.2.3 Design of Kalman filter 9

3. MATLAB IMPLEMENTATION OF THE ALGORITHMS 13

3.1 Ball detection algorithms 13
3.1.1 Matlab implementation and evaluation 13

3.2 Motion prediction algorithm 16
3.2.1 Design of linear prediction 16
3.2.2 Matlab implementation and evaluation 16

4. IMPLEMENTATION FOR DSP 18

4.1 Communication PC ⇔ DSP 19

4.2 Flipper activation 21
4.2.1 Decision algorithm 21
4.2.1.1 Decision regions 21
4.2.1.2 Interpolating the balls position between frames 21
4.2.1.3 Combining several predicted positions 22
4.2.2 Flipper activation cycle 22
4.2.2.1 Not breaking ball localization 23
4.2.2.2 Get the flipper to activate and deactivate for specific durations 23

4.2.2.3 The final non-blocking flipper state machine 24
4.2.3 Flipper activation model 25

5. THE GRAPHICAL USER INTERFACE (GUI) 26

5.1 Image capture 26

5.2 Displaying results 27

5.3 Getting user feedback 27

6. EVALUATION 28

7. RESULTS 28

8. FURTHER IMPROVEMENTS 29

8.1 Timer based interrupts 29

8.2 Modified flipper decision 29

8.3 Improved detection algorithm 30

APPENDIX A 31

SUMMARY OF LITERATURE STUDY 31

A.1 Latencies of systems 31

A.2 Object detection 31

A.3 State of the ball (position, velocity) 34

A.4 Prediction 35

A.5 Flipper activation 35

A.6 Limiting regions to save CPU 35

A.7 Other observations 36

APPENDIX B 36

USING THE GRAPHICAL USER INTERFACE 36

B.1 PROCESSING FRAME 37

B.2 PREDICTION FRAME 37

B.3 DECISION FRAME **37**

B.4 EDIT REGIONS WINDOW **37**

REFERENCES **39**

1. INTRODUCTION

1.1 BACKGROUND

Good solutions for recognition, localization and tracking objects in real life environments are desirable in many applications. At present however, the field of computer vision offers no efficient, generally applicable algorithm. Experiments in simplified environments, with a limited number of objects, have been proved to be important in helping to design better algorithms for computer vision.

1.2 PROBLEM

The aim of this project is to specify, develop and implement a system enabling a real Pinball machine to play on its own (without human intervention). A web-cam is placed over the pinball and enables detection of the ball. The task of software executing on a DSP board is to detect the ball, track it, estimate its future position and when necessary, be able to activate the flippers of the pinball machine to keep the ball in play. The activation of the flippers should be determined in such a way that a good score is achieved. The success of the project is measured by the high score achieved while minimizing the number of activations of the flippers.

1.3 OBJECTIVE

This project has been tested during the project course of 2003 and it turned out a working system. The ambition of our group this year is to achieve a better high score than what was possible during last year's project, while minimizing the number of activations of the flippers.

1.4 SYSTEM MODEL

The overview of the system is described in the figure below:

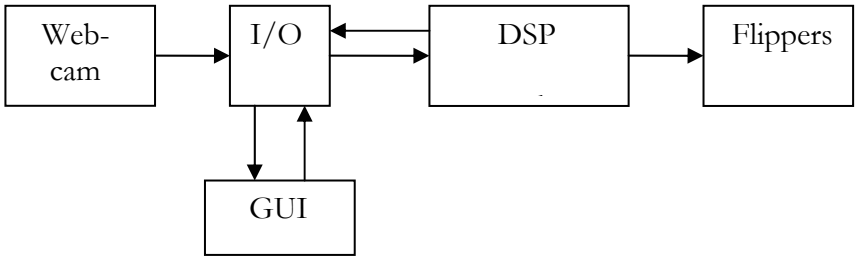


Figure 1 Overview of the system

1.5 EQUIPMENT

To implement the system and fulfil the requirement of documentations, webpage and poster, the following resources have been allocated.

Hardware

- One pinball machine (Silverball Mania)
- One web-cam (Philips PCVC 720K: 30 fps at resolution 352x288)
- One DSP card (Texas Instruments C6701)

Software

- Code Composer Studio 2.0
- Matlab 6.5
- Microsoft Visual C++ 6.0
- Microsoft Windows XP Professional
- Adobe Illustrator 10

1.6 OUTLINE

This section briefly describes the contents of the following chapters presented in the report. Chapter 2 addresses the fundamental theory behind this project, such as ball detection algorithms and ball movement prediction algorithms. Chapter 3 deals with issues concerning the actual implementation of the algorithms in Matlab. Also, the different performances for every algorithm will be highlighted. Implementations for DSP will be presented in Chapter 4. It covers communications between PC and DSP, and the flipper activation model. Chapter 5 mainly describes the software running on the host computer. Finally, based on the evaluations done in Chapter 6, we illustrate the conclusion of the project in Chapter 7. Some further improvements are described in Chapter 8. The appendix includes a summary of a literature study conducted during the beginning of the project as well as a user manual for the GUI of our final prototype.

2. THEORY

The problem stipulated in the project description requires solutions of two main theoretical tasks. These are algorithms for (1) ball detection and (2) ball movement prediction. The ball detection task can be solved in different ways depending on which parameter we decide should be regarded in the final prototype. For instance, if we would decide to use an image of the whole pinball as input to the DSP, we would have to be able to handle the lights turning on and off on the pinball game. On the other hand, if we decide to use only a portion of the image as input to the DSP, we may disregard the lights in our detection algorithm. To predict the movement of the ball over the pinball game, we consider the use of linear prediction or Kalman filtering.

2.1 BALL DETECTION

2.1.1 IMAGE CAPTURE – SIZE AND COLOUR MODE

According to our literature study, the choice of image capturing technique has great impact on how easily one can detect a specific object in a frame [2]. Another factor that might effect object detection is the resolution of the captured image [8]. For instance, the project group of year 2003 talked about using a very small resolution as a direct method of detecting the ball in a captured frame.

The program used for the webcamera offers capturing of frames with four different resolutions, the lowest being 160×120 and the highest being 352×288. A lower resolution would minimize the number of pixel computations but it could also be too poor to enable ball detection. A higher resolution would probably facilitate detection but requires too much of the DSPs computational power. A test run on the DMA showed that using the highest resolution would make it impossible to reach the frame rate needed for real time processing. Therefore, a compromise of taking the intermediate resolution (i.e. the second largest 176×144) was chosen.

From our literature study, we know that it is easier to recognize an object by its contrast to the background in RGB-mode than it is in grayscale-mode [2]. This is because, a frame in RGB-mode handles three times more information than a frame captured in grayscale. This also means that RGB-mode requires much more computational power than grayscale mode. If it proves to be equally efficient to use grayscale mode in our detection algorithm this would be the wiser choice.

2.1.2 CREATION OF A DIFFERENCE IMAGE

The easiest way to spot the existence of a moving object in a series of images is to subtract the current image from a previous image [1]. The result is called a difference image (see figure 2), which shows the absolute change between the two images;

Difference Image = abs (Current Image - Previous Image)

Figure 2 Difference image, Current Image, and Previous Image

However, if the moving object moves slowly, the difference image above will be almost equal to zero for all pixels. This means it's going to be harder to detect a slowly moving object. To solve this problem one can introduce a reference image [1, 5, 6, 8], which is updated at a slower rate. Here, we will use the formula of low pass filtering to update the reference image [1];

Reference image(t) =α × Current Image(t) + (1-α) × Reference Image(t-1)

where Reference image(t) means the Reference Image at time t, and $0 < \alpha \leq 1$

Figure 3 Reference images for different α; from left to right α=0.9, α=0.5 and α=0.1

In our prototype we choose α=0.125 which makes the reference image approach the input image. In this process, small moving objects will be averaged out of the reference image. The reference image can be said to be equal to the background image, without any moving objects. Using the method with a reference image, the difference image can now be written as:

Difference Image = abs (Current Image – Reference Image)

Figure 4 Difference image, Current Image, and Reference Image

2.1.3 SMOOTHING OF THE DIFFERENCE IMAGE

Several factors, one being the image quality from the webcamera, can cause noise in the difference image. In a noisy image, it will be harder to differentiate between noise and a moving object. One way to handle this is to convolve the noisy image with a kernel in two dimensions. Small, high-contrast pixel regions in the image will then be leveled out, leaving only one large, high-contrast region, which is the moving object. How large the kernel should be depends on both the noise level and the amount of computational power it requires. We tried several different sizes on the kernel and came to the conclusion that a very small one was enough to even out most of the noise. (see section about evaluating algorithms 3.1.1)

2.1.4 HANDLING LIGHTS

The approach of smoothing using a kernel [11] can not be used on large noise levels when lights on the pinball machine turn on and off. One way to solve this problem is to use two thresholds for each pixel [6]. These two thresholds are the maximum and minimum intensities for each pixel respectively. The pixel is considered a part of the background image when its intensity is within these two thresholds; otherwise the pixel is considered as part of a moving object.

The thresholds for each pixel are continuously updated according to the algorithm below. It also checks whether the pixel is inside its normal regions. If diff is set to 1, the pixel is outside of its normal region. Delta is a variable which determines the adaptation rate, and pixel is the value of the pixel processed. This piece of code should be invoked for each pixel and its corresponding thresholds.

```
lowThreshold++;
highThreshold++;
diff = 0;
if pixel < (lowThreshold – delta),
    diff = 1;
        lowThreshold = lowThreshold – delta;
else
    if pixel < lowThreshold,
```

```
        lowThreshold = pixel;
    end
end
if pixel > (highThreshold + delta),
    diff = 1;
        highThreshold = highThreshold + delta;
else
    if pixel > highThreshold ,
        highThreshold = pixel;
        end
end
```

In this way, when the thresholds have adjusted, any flashing lights will be ignored. The drawback of this algorithm is that it takes some time to adjust the thresholds before flashing lights are handled.

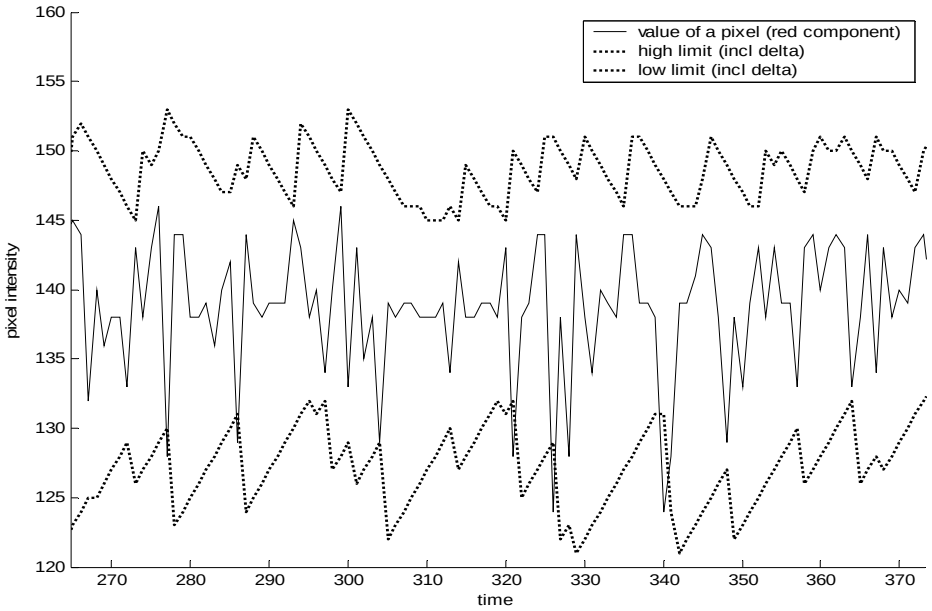


Figure 5 Continuous updating of pixel threshold values

2.1.5 PROCESSING OF THE WHOLE OR JUST THE PART OF THE IMAGE

To reduce computations, one can choose to process just one part of the image. The nature of the pinball game allows us to make computations only in the region near the flippers since there is no need to decide whether to activate the flippers when the ball is at the top of the pinball machine. This will also bypass the problem of handling lights on the pinball, since only two lights are localized in this region and they are very seldom turned on.

2.1.6 FINDING THE BALL IN THE IMAGE

Once the image has been freed from noise by smoothing, the brightest pixel in the image can be identified as the position of the ball. To localize the brightest pixel, the algorithm loops through the entire image examining the intensity of each pixel.

2.2 BALL MOVEMENT PREDICTION

2.2.1 WHY USE PREDICTION – SYSTEM DELAY

The Smart Pinball system has some drawbacks, which make it hard to implement a working pinball playing prototype. The most noticeable problem is the system delay. The total system delay is the delay from when a ball physically moves into a position that prompts for a flipper activation, until the flipper physically starts moving.

Experiments showed that it would be impossible to have a good implementation of a pinball playing software without taking the total system delay into account. The total system delay is the sum of the following delays:

Name of delay source	Source type
dt1	Camera exposure time
dt2	Time for serial transfer of image to the PC
dt3	Windows camera driver processing time
dt4	Transfer time for transfer of image to DSP board
dt5	Image processing time on DSP board
dt6	Prediction and flipper decision delay on DSP board
dt7	Audio codec delay
dt8	Bally controller delay
dt9	Delay until the activated flipper moves physically

Table 1 Different sources of delay in the system

When measuring the total system delay, a measurement of the time from one physical event at time t1 to another physical event at time t2 is needed:

t1: time instant when the ball moves into position prompting a flipper activation
t2: time instant when the flipper starts to move physically

Total system delay = t2-t1
= dt1 + dt2 + … + dt9

How to measure the total system delay is not obvious, since we have no clue of when events occur in the physical world. However, it can be shown that the system delay is equivalent to the delay between two other events occurring at times t3 and t4:

t3: time instant when the DSP finished decision to activate flipper
t4: time instant when DSP board makes a new flipper decision directly after seeing that the flipper has moved

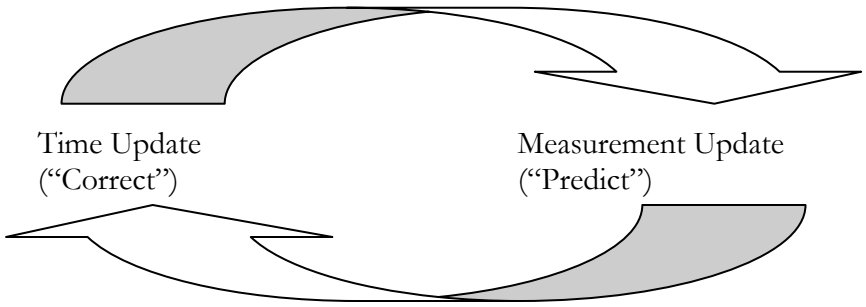
Total system delay = t4-t3
= dt7 + dt8 + dt9 + dt1 + dt2 + … + dt6
= Total system delay

By using the relation above, measurement of the total system delay can be performed. The prototype measured the system delay in the quantity of frames, and at the highest frame rate from the camera (25 fps). The result of the total system delay is three frames. This means that an activated flipper will be noticed by the DSP three frames later.

2.2.2 KALMAN PREDICTION

The Kalman filter is a set of mathematical equations that permits an efficient computational (recursive) means to estimate the state of a process. It minimizes the mean of the squared error. The filter is very powerful in several aspects: it can be used with estimations of past, present, and even future states, even when the precise nature of the modeled system is not really known.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: *time update* equations and *measurement update* equations. The time update equations permit to project forward (in time) the current state and error covariance estimates to obtain the *a priori* estimates for the next time step. The measurement update equations permit the feedback—i.e. for incorporating a new measurement into the *a priori* estimate to obtain an improved *a posteriori* estimate. The time update equations can also be thought of as *predictor* equations, while the measurement update equations can be thought of as *corrector* equations. Indeed the final estimation algorithm resembles that of a *predictor-corrector* algorithm for solving numerical problems.



Time update equations:

$$\hat{x}(n+1 \mid n) = F\hat{x}(n \mid n)$$
$$\hat{y}(n+1 \mid n) = H\hat{x}(n+1 \mid n)$$
$$P(n+1) = FQ(n)F^T + GR_1G^T$$

Measurerment update equations:

$$L(n+1) = P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}$$

$$\hat{x}(n+1\,|\,n+1) = \hat{x}(n+1\,|\,n) + L(n+1)\left(y(n+1) - H\hat{x}(n+1\,|\,n)\right)$$

$$Q(n+1) = P(n+1) - P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}HP(n+1)$$

where

y: measured signal, y=[y(1)'; y(2)'; ...; y(K)']

F: State space model: function of the system

G: State space model: noise of the system

H: State space model: observation function

R_1: covariance of the process noise

R_2: covariance of the measurement noise

x_0: Initial state value

Q_0: Initial error covariance

$\hat{y}(n+1\,|\,n)$: estimate of y(n+1)

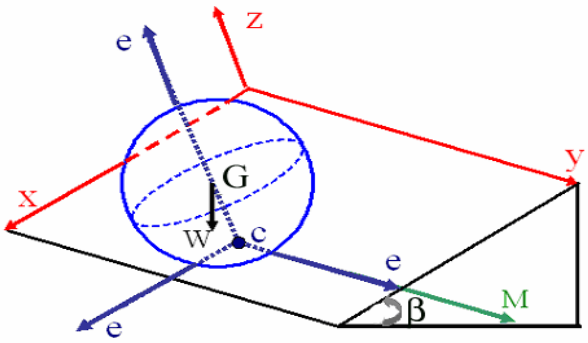
$\hat{x}(n\,|\,n)$: filtered x estimate x(n|n)

$\hat{x}(n+1\,|\,n)$: predicted x

P: Final posterior error covariance

Q: Final error covariance

2.2.3 DESIGN OF KALMAN FILTER



m = masse of the ball

R = the radius of the ball

ß = the angle of inclination

G = the constant of gravity

J = the moment of inertia of the sphere and $J = \frac{2}{5}mR^2$

A more physical demonstration can be founded in [10]. We can assume the sphere rolls without slipping. Thus the point C on the sphere in contact with the plan is instantaneous at rest: its instantaneous speed in the pinball referential is 0. The motion of the sphere is thus

pure rotation around C. The moment of inertia of the sphere with respect to any axis in the plane through C is, according to the parallel axes theorem:

$$J_C = J + mR^2$$

We will use the equation of the motion $\dot{L}_C = M_C$

$L_C = \overline{C}\overline{G} \times p + L_G^{'} = \mathrm{Re}_Z \times mv_G + J\varpi$ with ϖ is the instantaneous angular speed.

$M_C = M = \overline{C}\overline{G} \times W = R \cdot m \cdot g \cdot \sin\beta \cdot e_y$

where W = -mg (cos ß e_z – sin ß e_x) is the weight of the sphere. Since $V_C = 0$, the connection formula for velocities in a rigid sphere gives us

$V_G = V = \overline{C}\overline{G} \times \varpi = (\varpi_X e_X + \varpi_Y e_Y + \varpi_Z e_Z) \times \mathrm{Re}_Z$ for the centre of mass velocity, V, of the sphere. The three components of this equation can be written:

$$\dot{x} = R\varpi_Y$$

$$\dot{y} = -R\varpi_x$$

$$\dot{z} = 0$$

so that we can easily have:

$L_C = m\,\mathrm{Re}_Z \times (R\varpi_y e_x - R\varpi_x e_y) + J\varpi = (J + mR^2)\varpi_x e_x + (J + mR^2)\varpi_y e_y + J\varpi_z e_z$

with $\dot{L}_C = M_C$ we have:

$$(J + mR^2)\dot{\varpi}_x e_x + (J + mR^2)\dot{\varpi}_y e_y + J\dot{\varpi}_z e_z = R \cdot m \cdot g \cdot \sin\beta \cdot e_y$$

The components of this vector equation are

$$(J + mR^2)\dot{\varpi}_x = 0$$

$$(J + mR^2)\dot{\varpi}_y = R \cdot m \cdot g \cdot \sin\beta$$

$$J\dot{\varpi}_z = 0$$

Let's introduce this angle $\alpha \equiv \dot{\varpi}_y = R\dfrac{mg\sin\beta}{J + mR^2}$

For the angular acceleration around the Y-axis, we can solve these equations for the angular velocity components as follows

$$\varpi_x(t) = \varpi_x(0) = const$$

$$\varpi_y(t) = \varpi_y(0) + \alpha t \;\Rightarrow \varpi_y(t + T) = \varpi_y(t) + \alpha T$$

$$\varpi_z(t) = \varpi_z(0) = const$$

this can be inserted into the above equation and integrated to give

$$\dot{x}(t) = R\varpi_y(t) = R(\varpi_y(0) + \alpha t) \Rightarrow \int_0^t R(\varpi_y(0) + \alpha t'))dt' = x(t) - x(0)$$

$$\Rightarrow R\varpi_y(0)t + R\frac{1}{2}\alpha t^2 = x(t) - x(0)$$

$$\Rightarrow x(t) = x(0) + R\varpi_y(0)t + R\frac{1}{2}\alpha t^2$$

$$\Rightarrow x(t+T)-x(t)=(x(0)+R\varpi_y(0)(t+T)+R\frac{1}{2}\alpha(t^2+2tT+T^2))-(x(0)+R\varpi_y(0)t+R\frac{1}{2}\alpha t^2)$$

$$\Rightarrow x(t+T)=x(t)+R\varpi_y(0)T+R\frac{1}{2}\alpha(2tT+T^2)$$

To find the time dependence of the y-coordinate, we can do basically the same thing.

$$y(t)=y(0)-R\varpi_x(0)t \Rightarrow y(t+T)=y(t)-R\varpi_x(0)T$$

But there is still a problem because the expression of x depends on "t". So we can use this kind of trick:

$$z(t)=x(t)-R\alpha\frac{t^2}{2}$$

$$z(t+T)=x(t+T)-R\alpha\frac{(t+T)^2}{2}$$

$$z(t+T)=\left[x(t)+R\varpi_y(0)T+R\frac{1}{2}\alpha(2tT+T^2)\right]-\left[R\alpha\frac{(t^2+T^2+2tT)}{2}\right]$$

$$z(t+T)=x(t)-R\alpha\frac{t^2}{2}+R\varpi_y(0)T$$

$$z(t+T)=z(t)+R\varpi_y(0)T$$ Which doesn't depend on "t"

Thus, we can consider this simple model for our Kalman filter:

$z(t+T)=z(t)+R\varpi_y(0)T$	$\Leftrightarrow$	$z(i+1)=z(i)+R\varpi_y(0)T$
$y(t+T)=y(t)-R\varpi_x(0)T$	$\Leftrightarrow$	$y(i+1)=y(i)-R\varpi_x(0)T$
Mechanical notation		Indicial notation

And then deduce x from z thanks to $x(t)=z(t)+R\alpha\frac{t^2}{2} \Leftrightarrow x(i)=z(i)+R\alpha\frac{iT^2}{2}$.

The Kalman filter equations for the ball are as follows:

For z: is obtained thanks to x, the position of the ball on the X axis

Time update equations:

$$\begin{bmatrix}\hat{z}(n+1\mid n)\\1\end{bmatrix}=F\times\begin{bmatrix}\hat{z}(n\mid n)\\1\end{bmatrix}$$

$$\begin{bmatrix}yhat\,(n+1\mid n)\\1\end{bmatrix}=H\begin{bmatrix}\hat{z}(n+1\mid n)\\1\end{bmatrix}$$

$$P(n+1)=FQ(n)F^T+GR_1G^T$$

Measurerment update equations:

$$L(n+1) = P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}$$

$$\begin{bmatrix}\hat{z}(n+1\mid n+1)\\ 1\end{bmatrix} = \begin{bmatrix}\hat{z}(n+1\mid n)\\ 1\end{bmatrix} + L(n+1)\left(\begin{bmatrix}z(n+1)\\ 1\end{bmatrix} - H\begin{bmatrix}\hat{z}(n+1\mid n)\\ 1\end{bmatrix}\right)$$

$$Q(n+1) = P(n+1) - P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}HP(n+1)$$

Where:

y $y = \begin{bmatrix}z\\ 1\end{bmatrix}$ **F** $F = \begin{bmatrix}1 & +R\varpi_y(0)T\\ 0 & 1\end{bmatrix}$

G $G = \begin{bmatrix}1\\ 0\end{bmatrix}$ **H** $H = \begin{bmatrix}1 & 0\end{bmatrix}$

R_1 covariance of the process noise w(n) will be assume to be 150

R_2 covariance of the process noise v(n) will be assume to be 0.01

Q_0 Initial error covariance is assumed to be $Q_0 = \begin{bmatrix}0.1 & 0\\ 0 & 0\end{bmatrix}$

The Kalman filter is initialized at the 2nd frame. Thus the initial state value is given by the position founded thanks to the 1st frame.

For y: is the position of the ball on the Y axis

Time update equations:

$$\begin{bmatrix}\hat{y}(n+1\mid n)\\ 1\end{bmatrix} = F \times \begin{bmatrix}\hat{y}(n\mid n)\\ 1\end{bmatrix}$$

$$\begin{bmatrix}yhat(n+1\mid n)\\ 1\end{bmatrix} = H\begin{bmatrix}\hat{y}(n+1\mid n)\\ 1\end{bmatrix}$$

$$P(n+1) = FQ(n)F^T + GR_1G^T$$

Measurerment update equations for y:

$$L(n+1) = P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}$$

$$\begin{bmatrix}\hat{y}(n+1\mid n+1)\\ 1\end{bmatrix} = \begin{bmatrix}\hat{y}(n+1\mid n)\\ 1\end{bmatrix} + L(n+1)\left(\begin{bmatrix}y(n+1)\\ 1\end{bmatrix} - H\begin{bmatrix}\hat{y}(n+1\mid n)\\ 1\end{bmatrix}\right)$$

$$Q(n+1) = P(n+1) - P(n+1)H^T\left[HP(n+1)H^T + R_2\right]^{-1}HP(n+1)$$

Where:

y $y = \begin{bmatrix} y \\ 1 \end{bmatrix}$ **F** $F = \begin{bmatrix} 1 & -R\varpi_x(0)T \\ 0 & 1 \end{bmatrix}$

G $G = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ **H** $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$

$\mathbf{R_1}$ covariance of the process noise w(n) will be assume to be 150
$\mathbf{R_2}$ covariance of the process noise v(n) will be assume to be 0.01
$\mathbf{Q_0}$ Initial error covariance is assumed to be $Q_0 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0 \end{bmatrix}$

The Kalman filter is initialized at the 2nd frame. Thus the initial state value is given by the position founded thanks to the 1st frame.

3. MATLAB IMPLEMENTATION OF THE ALGORITHMS

3.1 BALL DETECTION ALGORITHMS

3.1.1 MATLAB IMPLEMENTATION AND EVALUATION

Having the sections in 2.1 in mind, we have developed a number of different algorithms in Matlab. They are evaluated and the best in terms of speed and accuracy is implemented on the DSP.

Algorithm	Colour mode	Kernel size	Image size
A	RGB	16×16	Whole image
B	RGB	3×3	Whole image
C[1]	RGB	3×3	Whole image
D	Grayscale	7×7	Smaller part
E	Grayscale	3×3	Smaller part

Table 2 Features of algorithms implemented in Matlab

- **A** – Uses a whole image in RGB as input. The difference image is smoothed using a 16×16 kernel. It uses the method described in 2.1.5 for handling the lights on the pinball.

[1] See explanation below the table for differences between B and C

- **B** – Same structure as for A, but a smaller kernel is used; 3×3.
- **C** – Same structure as for B, but now the ball's position is improved in a second step.
- **D** – Uses only a small part of the grayscale image as input. The difference image is smoothed using a 7×7 kernel. No threshold is used.
- **E** – Same input as for D, but a smaller kernel is used; 3×3.

The results of the detection algorithms are shown in figure 6 and 7. Here, the absolute difference between the estimated position and the real position is plotted.

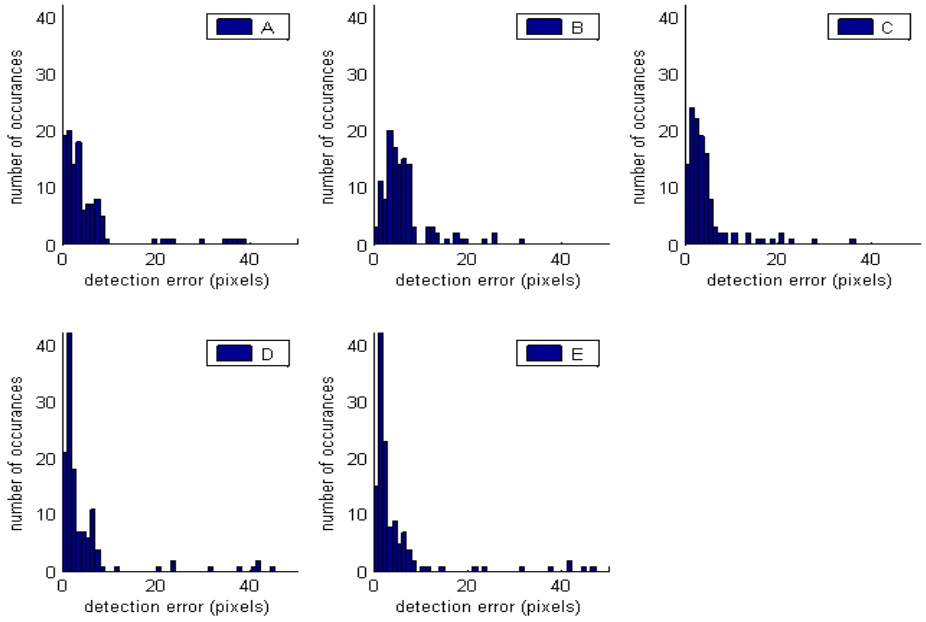


Figure 6 Histograms of detection errors for some of the algorithms

As it turns out, all algorithms show satisfactory results regarding accuracy.

Since the prototype is a real time system, we also have to take under consideration what the computational demands are for each algorithm. When evaluating we make a crude approximation of the number of needed computations for each algorithm. For the approximations we have used our implementations in Matlab were the code have been modified to have a more C-like structure.

Algorithm	Number of computations (width×height×constant)[2]
A	w×h×**144**
B	w×h×**32**
C[3]	w×h×**32+20000**
D	w×h×**36**
E	w×h×**19**

Table 3 Approximated needed computations for algorithms A-E

The "most costly" part of all algorithms is the smoothing which minimizes the noise in the grabbed frames. Since the detection seems to work satisfactory even with such a small kernel as 3×3 we should use that. When comparing algorithms B, C and E it is also apparent that it is not necessary to process the whole image to get a good prediction. For these reasons we choose to implement algorithm E on the DSP.

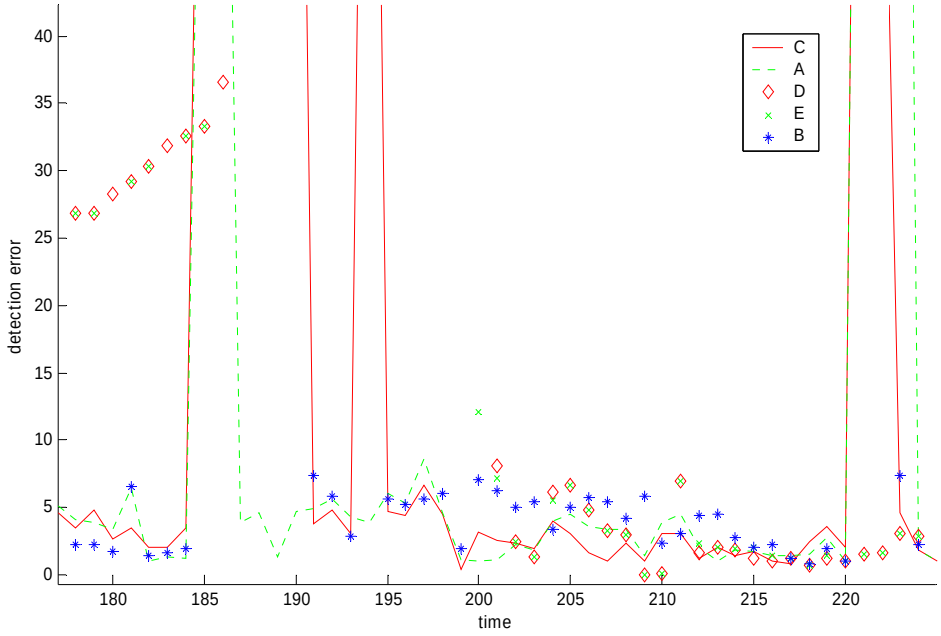


Figure 7 Detection error as a function of time for the algorithms. The largest detection error arises either when the flippers are moving, or when the ball is outside the processed region

[2] The width and the height are dependent on the chosen resolution of the webcamera.

[3] See explanation below table 1 for differences between B and C

3.2 MOTION PREDICTION ALGORITHM

3.2.1 DESIGN OF LINEAR PREDICTION

Linear prediction is a tool used to combat problems caused by the large system delay. To predict whether to activate a flipper or not, we use the ball's two most recent positions. This makes it possible to determine the distance the ball traveled between two frames.

Traveled Distance = Current Position – Previous Position

According to 2.2.1, the total system delay is three frames. Thus, to get the ball's position three frames ahead, we just add three times the distance traveled in one frame to the current position of the ball.

Predicted Position = Current Position + 3 ×Traveled Distance

It is the ball's predicted position which is used to determine whether to activate a flipper or not.

In real conditions, however, two modifications can improve the performance significantly, which are described in 4.2.1.2 and 4.2.1.3.

3.2.2 MATLAB IMPLEMENTATION AND EVALUATION

We have designed a whole Kalman filter procedure and the linear prediction in Matlab. For Kalman filter, quite the same procedure has been used for filtering y and z, the two variables that have been defined in the theoretical part about Kalman filtering. All data is stored as a column of N rows where each row is the transposed state/data vector at the corresponding time instance.

The following figures show the different performances between Kalman filter and linear prediction in Matlab.

- for a pinball without lights

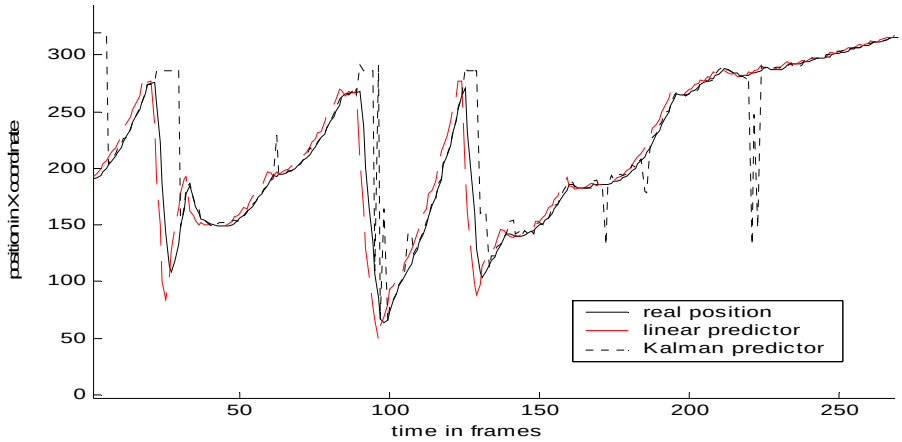


Figure 8 Prediction in hor-coordinate X without lights

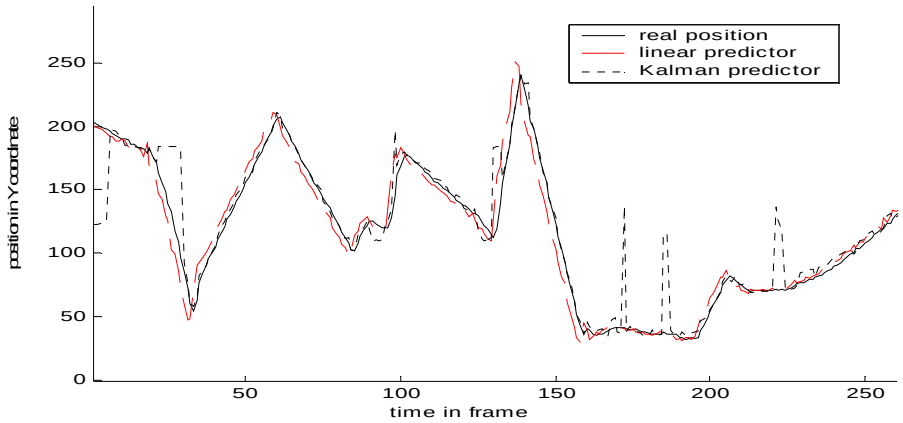


Figure 9 Prediction in ver-coordinate Y without lights

- for a pinball with lights

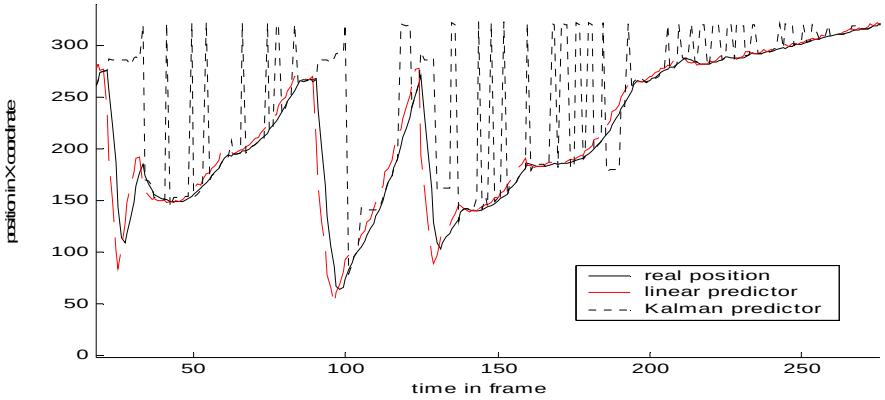


Figure 10 Prediction in hor-coordinate X with lights

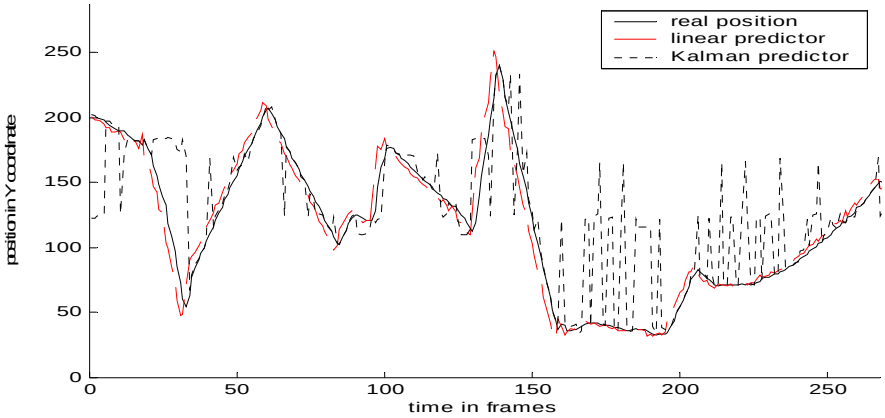


Figure 11 Prediction in ver-coordinate Y with lights

From the figures, we see that when the Kalman filter arrives to follow the ball, it is better than the linear prediction. But when it looses the ball, it completely looses the ball. Thus, linear prediction is generally better than Kalman filter in Matlab.

4. IMPLEMENTATION FOR DSP

This section is devoted to the description of the practical matters concerning the implementation and interoperation of our algorithms in the Digital Signal Processor. This hardware resource is the most important for our project, since it is the platform where the actual signal processing occurs. Texas Instruments® is the manufacturer of the product, which consists on the DSP processor, a TMS320C6701 with floating point operation, mounted on a EVM (Evaluation Module) board that supplies increased functionality, such as external RAM memory, a stereo audio codec for data acquisition and reproduction, a standard PCI interface, power and clocking hardware, and some other features.

In order to simplify the programming activity, we make use of some support software components packaged with the product, mainly the EVM Windows® drivers and DLL API, and C libraries for the configuration and control of the hardware components. Code Composer's Studio 2 © has been the tool used to implement, debug and test our code on the DSP until the final stage of the development.

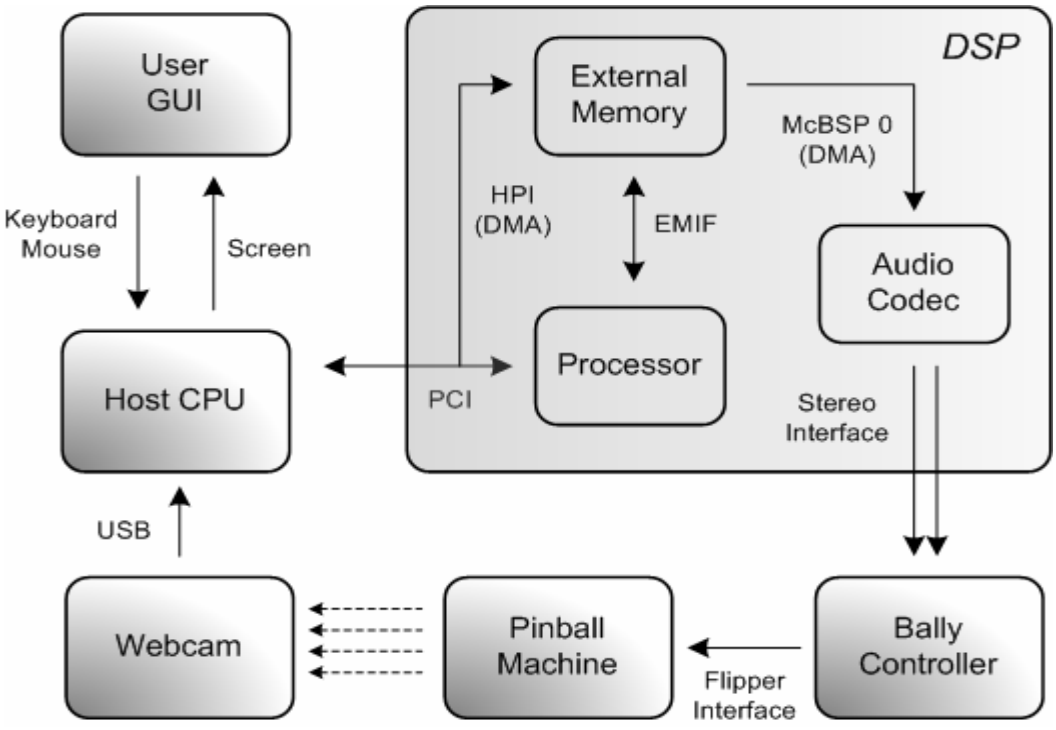


Figure 12 System interconnection

4.1 COMMUNICATION PC ⇔ DSP

One of the tasks of the host program, running on the PC, is to initialize the board and DSP for proper working, and be able to communicate with it and control its operation later. This is done in both high and low level fashions: sometimes standard libraries are used, and sometimes we access the DSP memory and registers directly. Every single command or result exchanged between the PC and the EVM board is sent over the PCI interface, which is accessed by the DSP through its HPI interface. This interface has a dedicated DMA channel, in order to free the DSP from the considerable load imposed by these kinds of transfers.

In order to make both sides of the application understand each other, some conventions must be adopted. As we can define the desired memory mapping of the DSP (linker **.cmd* file and proper use of the *#pragma* directive), it is possible for the host program to know the addresses of certain variables. For this purpose, we define C data structures that are equivalent in both PC and DSP programs, so that we only need to treat them as single data blocks. The key idea is the definition, in the host program, of the memory address where the DSPs compiler has allocated a struct which contains all the relevant information. This information is either directly stored, as atomic variables, or indirectly referenced, in the case of pointers to the first addresses of the memory buffers.

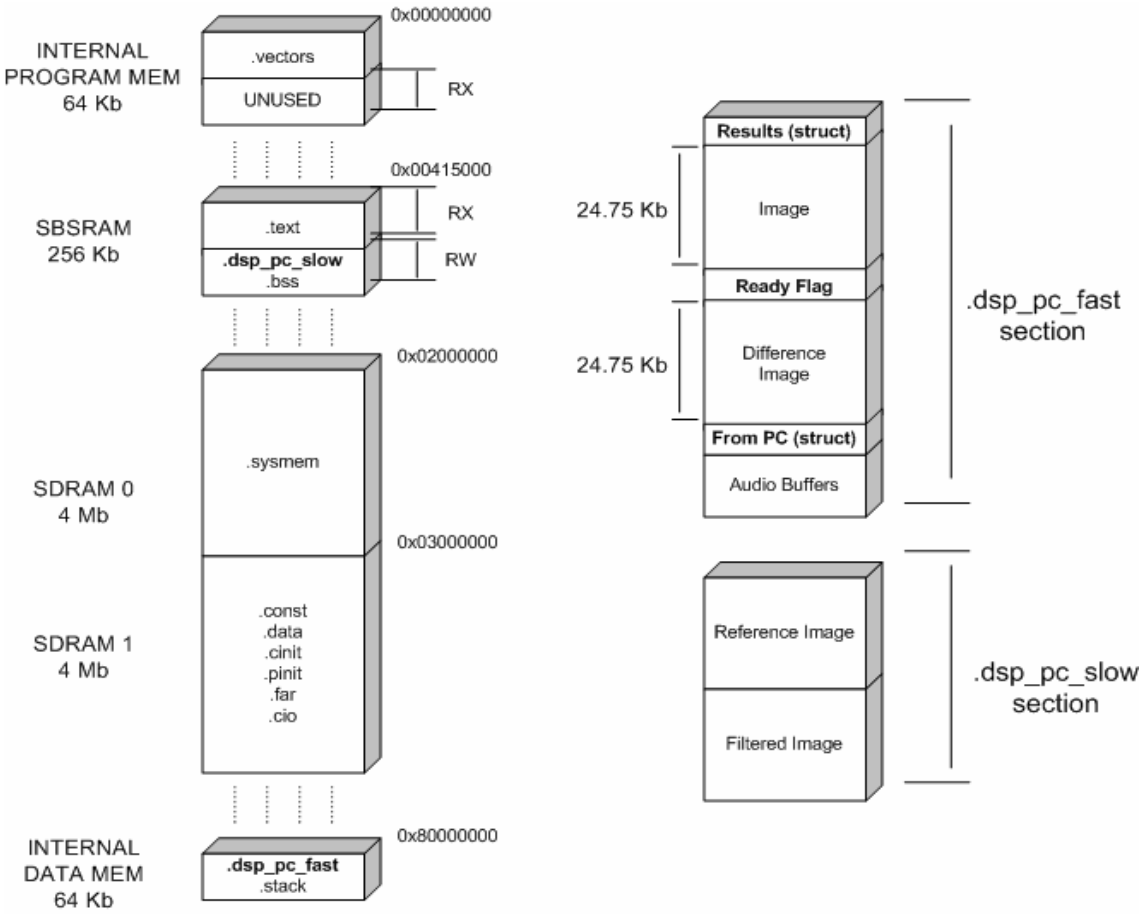


Figure 13 DSP Memory Map

The DSPs main program works as a state machine, whose states change due to internal operation or external access. It is actually implemented using an infinite loop that continuously checks the value of a communication flag with a "switch" statement. We can regard this operation mode as a master-slave, where the PC is the side that always triggers events on the DSP, changing this flag's value as required during normal processing and at user's prompt (it is remarkable that although the communication is bidirectional, it is never started by the DSP, but it is the host that has to begin any copy or write event).

We reached the conclusion that this implementation is quite robust and flexible, in the sense that we can always monitor and change the state of the DSP, and that we would only need to include more states for increasing complexity. The explanation of the states follows:

- **State = -2** → Delay test, used to estimate the total system delay (in number of frames) from the image grabbing at the web-cam to the flipper activation.
- **State = -1** → Initialization of variables and configuration of hardware resources.
- **State = 0** → Signaled by the PC, a frame has just been copied and it is ready to be processed.
- **State = 1** → Previous task finished, waiting state.

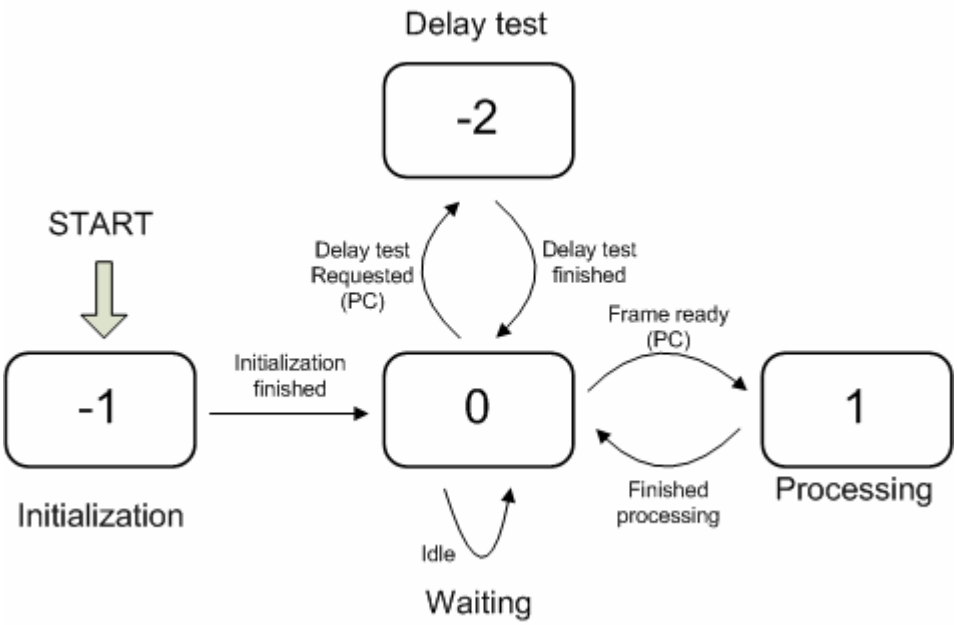


Figure 14 Processing State Machine

4.2 FLIPPER ACTIVATION

4.2.1 DECISION ALGORITHM

4.2.1.1 Decision regions

There are two flipper decision regions. One for the left flipper and one for the right. Both regions are shaped as rectangles. When the predicted ball's position is inside one of these regions, the corresponding flipper is going to activate. If the predicted ball's position is inside both regions, both flippers will activate. The coordinates of these regions are sent to the DSP by the user interface.

Figure 15 Left and right decision regions.

4.2.1.2 Interpolating the balls position between frames

Predicting the ball's position in between consecutive frames should be used, since the ball sometimes moves with a very large velocity. At large velocities, the ball could pass through a decision region without activating the flipper. Consider the case when the ball comes to the left of a region in one frame, and has moved to the right of the region in the next frame (see figure 16). Then the corresponding flipper wouldn't be activated unless we check the ball's positions on a line between the two consecutive frames.

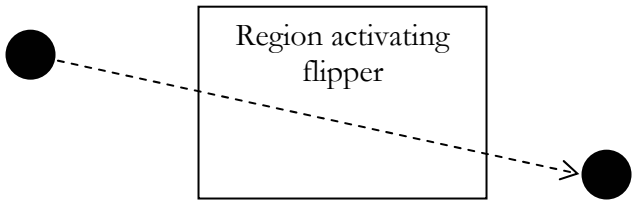


Figure 16 Interpolation of the balls position activates the flippers for fast moving balls

4.2.1.3 Combining several predicted positions

Unfortunately, the position of the ball which the DSP estimates was the physical position of the ball around 0.12 seconds (three frames) ago. Therefore, the balls predicted position three frames ahead would be a better estimate.

However, since the prediction isn't perfect, the prediction three frames ahead won't always be accurate. Therefore, besides three frames ahead prediction, we need other options of prediction. Experiments showed that predictions four or five frames ahead would activate the flippers too early; while predictions one or two frames ahead would activate the flippers after the ball has bunched on the flipper, but still managed to hit the ball.

Therefore a flipper is activated either when the detected position of the ball or any of the predicted positions (at most three frames ahead) of the ball ends up inside a decision region. It turns out that this strategy greatly improves the performance.

Figure 17 Situation for using more than three frames ahead prediction. (Ball not hit by flipper)

Figure 18 Situation for using less than three frames ahead prediction.(Can hit the ball a moment later)

4.2.2 FLIPPER ACTIVATION CYCLE

The ball localization routine has one big problem: a moving flipper is usually mistaken for a moving ball. This in turn can easily get the flippers to activate and deactivate in an oscillating manner, preventing any further localizations of the ball. This is hard to avoid when changes in the image are used to locate the ball.

The solution used by the prototype, was to implement two separate state machines: one state machine for the left flipper and the other one for the right. These state machines have three goals: (1) Not break ball localization during flipper movement. (2) Get the flippers to activate and deactivate for specified durations. (3) Not block the DSP while moving flippers.

4.2.2.1 Not breaking ball localization

When a flipper is moving, the biggest change in the input image would be the flipper instead of a moving ball as assumed. The algorithm used for ball localization would then detect the moving flipper as the ball. One way to handle this is to stop tracking the ball completely while a flipper is activated (as last year's project group did), but that is far from ideal.

When a flipper is moving, our solution tracks the ball in most parts of the image. Only a small region around the moving flipper is excluded from the area of ball detection. One of these regions can be seen in figure 19. Experiments showed a low probability of the ball being inside this region while the flipper is moving. We therefore believe that our solution almost never breaks ball localization at all.

Figure 19 Region excluded when flipper is moving

4.2.2.2 Get the flipper to activate and deactivate for specific durations

The flippers need to be activated for at least a certain amount of time, before it physically moves into its upward position (see figure 20). If it's not activated long enough, it just won't reach so far. Experimentally, it turns out that 200 ms is a good time for activation. Deactivation is set a little bit longer, 400 ms, to get the flipper fully back to its downward position before we allow the flipper to be activated again.

Figure 20 Left: upward (activated), right: downward (deactivated)

When the flipper has been deactivated for a specific time, the small region around the flipper is again used for ball localization. Therefore, the flipper has to move fully back during deactivation. Otherwise the moving flipper will be detected as a ball, causing the flipper to activate again even if the real ball is far away from it.

4.2.2.3 The final non-blocking flipper state machine

The figure below shows how the state machine governing flipper activation works. There are two separate state machines, one for each flipper. This makes the two flippers able to activate independently of each other as needed. All states are "non-blocking", which means that the DSP always is working.

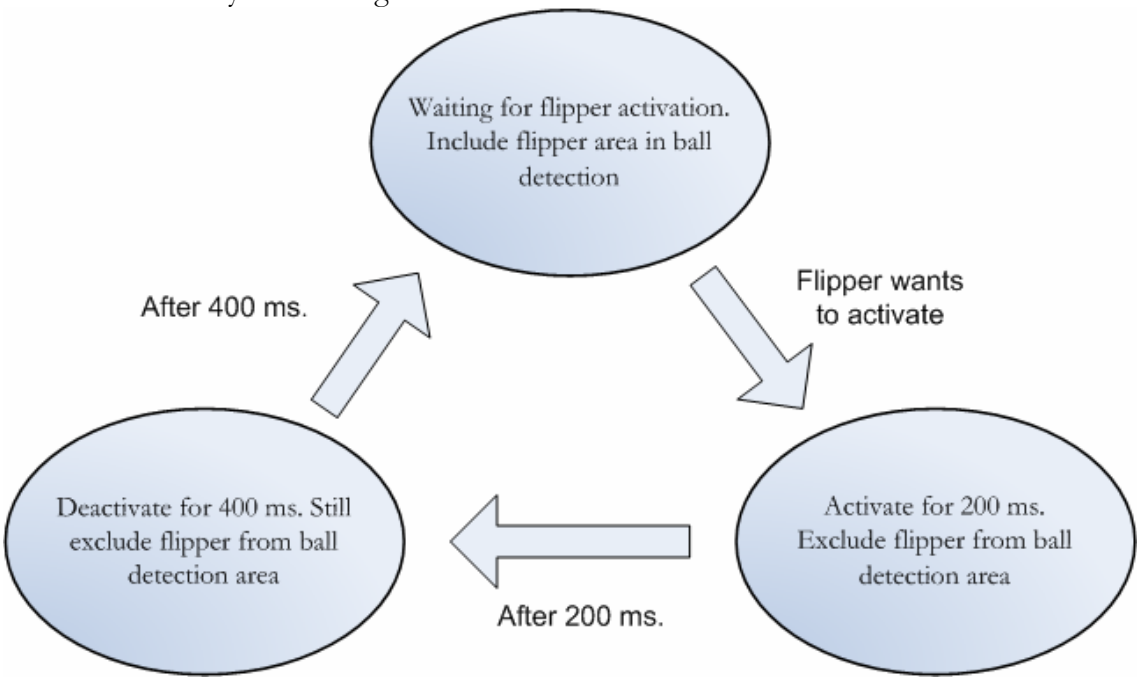


Figure 21 The state machine of each flipper

4.2.3 FLIPPER ACTIVATION MODEL

For the flipper activation, we need a 10 KHz. sinusoidal wave as excitation for the flipper inputs. The EVM supplies a stereo audio codec, which approximately covers the frequency range of human hearing (0 – 20 KHz) for both channels, and happens to be very suitable for this purpose.

Before being able to send any data at all to the audio codec, we must initialize it and configure several parameters for both channels:

- Format of the samples, set as 16-bit signed integers.
- Sampling frequency, closest possible value to 20 KHz, actually 22050 Hz
- Output gain, 0 dB

Apart from that, we need to prepare channel 0 of the McBSP (Multi-Channel Buffered Serial Port) for sending data to the codec. Using it, we take advantage of continuous and concurrent DMA transfers, which make the flipper activation routines almost transparent to the main program. We use the function *mcbsp_async_send( ),* available in the *mcbspdrv.h* support library, to start the DMA processes. Once a full buffer of data is played, a callback function interrupts the main program, and we use it to restart the transfer with another equivalent call.

These buffers are initialized at the beginning of the program, and they consist on the four possible combinations of the signals sent to the pinball: no flipping, left flipping, right flipping and both flipping. For each of them, if the corresponding channel is active, the signal varies from the minimum possible value in the current data format to the maximum one; if it shouldn't be active then the signal is a constant zero. As the frequency response of the audio codec is not ideal, it low-pass filters the high harmonic components of the square wave and outputs a rather acceptable sine wave.

The frequency of the interrupts depends on the buffer length: for our prototype, we use an audio buffer length of 16 stereo samples, which results in a rate of 1250 interrupts per second. We have to maintain a trade-off between this interrupt rate and the memory space needed for allocating the buffers, as the DSP is quite limited in terms of memory. We must also keep in mind that too large buffers would lead to additive delay in the flipper activation (with this length, the delay is always less than 1 ms). The workload imposed to the processor should be minimized for two reasons:

- We don't want the flipper activation routines to lower the performance of the processing algorithms, consuming more time than the strictly necessary.
- If the interrupt service routine is too complex, there would be some discontinuities in the output, as there is no transfer active from the moment when a buffer has been finished to the moment when the next DMA process is started over.

In our tests, the system performance has not suffered from these problems, and the flipper activation has been made almost transparent to the main program.

5. THE GRAPHICAL USER INTERFACE (GUI)

The software running on the host computer has the purpose of doing the tasks that the DSP can't do by itself. These tasks include capturing images from the web cam; displaying results on the computer screen; and getting user feedback.

5.1 IMAGE CAPTURE

The capture of images is performed through the help of DirectShow, one part of DirectX. DirectShow handles all its playback and capture of media with the help of filters and filter graphs.

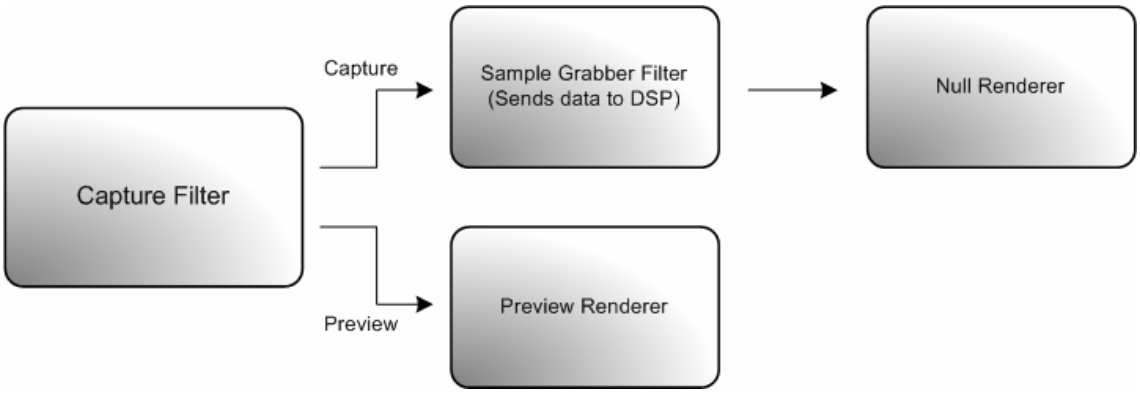


Figure 22 A simplified view of the filter graph used for video capture

The filter graph above is a description of the video capture system, showing how individual filters are connected together. The capture filter sends images from the web cam to other filters further down in the graph. The capture pin has higher priority than the preview pin, so the capture pin of the capture filter is used for sending data to the DSP.

The filter graph above also shows how a Sample Grabber filter receives the captured images from the web cam. This Sample Grabber has a callback function which is called whenever a new image has been captured. The callback's most important task is to forward the captured image to the DSP. It might seem odd to connect a null renderer to the output of the sample grabber filter, but this is a requirement of DirectShow.

All the individual filters in the figure above are connected with the help of two objects, the capture graph builder and the filter graph. The filter graph keeps track of how all pins of all filters are connected together. The capture graph builder is a tool for adding filters to the filter graph object without considering much about which filters are compatible, or exactly

where to put the filters in the graph. The capture graph builder figures out how to connect the filters, and if necessary, it adds extra filters to the graph.

The images from the web-cam are captured in black and white, and in a low resolution. This reduces the computational burden on the DSP, so the frame rate won't have to be lowered.

5.2 DISPLAYING RESULTS

Another important task of the host software is to display results of calculations in the DSP. The user can select which internal image buffer to retrieve from the DSP board, for instance, the input image, the reference image, or the difference image. The user can also choose to overlay the position of the ball and of different regions on top of the retrieved image. Since different palettes are more suitable for displaying different results, the palette can be changed as needed. Besides this, the host software also reads a string stored on the DSP board, which is used to show what the DSP is doing currently.

5.3 GETTING USER FEEDBACK

This is the last task of the host software. User feedback is very important for the prototype. There exists buttons in the GUI for running specialized code on the DSP, such as a delay estimator, which determines the total system delay. Another specialized routine is the "DSP speed test", which determines the mean frame rate of the DSP implementation when the DSP is fed images as frequent as possible.

However, the most important user feedback is supplied in the form of regions. The user can easily move and resize regions, and the software is able to store all regions, so they automatically reload whenever the application is started.

Besides this, the user can also supply some important parameters to the DSP. Currently, these parameters include the number of frames ahead of prediction, which prediction algorithm to use, and a threshold for ball detection.

6. EVALUATION

	2003	2004	
	Linear Prediction	Linear Prediction	Kalman Filtering
High score	254 220	379 379	330 020
Average score	Unknown	178 902	177 346
Median score	Unknown	162 500	163 870

Table 4 Comparison between pinball projects for 2003 and 2004

7. RESULTS

The aim of this project was to develop a system enabling a real pinball machine to play without human intervention. The prototype developed was a complete success. It works in real time, does all signal processing in the DSP, and has an intuitive user interface. Besides developing the prototype, all administrative tasks and deadlines for delivering materials to our project assistant has been met.

The performance of the project was initially going to be measured by the high score achieved while minimizing the number of flipper activations. The choice of methods for ball detection, prediction and flipper activation has however resulted in virtually no unnecessary flipper activations at all, so flipper activation is always minimized. At the same time the scores achieved by the prototype is quite high.

Two different prediction strategies was used, namely Kalman filtering and linear prediction. These two predictions show almost equal performance, with average scores of approximately 180 000 points each. Out of 25 test runs, linear prediction beat last years best high score of 254 220 in four games, while Kalman prediction beat it during three games. The maximum high score of 379 790 was achieved with linear prediction, and proves that wise choices of algorithms was made.

The project was also a success in a completely different aspect. During the course of the project, the members of our group developed large insights into the practical world of developing software and algorithms.

8. FURTHER IMPROVEMENTS

Due to unexpected problems, lack of time and several other reasons, we have not been able to fully implement some features that we had kept in mind from the beginning of the project. Below we explain what our intentions were, how far we got into the development and the main reasons that prevented us from reaching those goals.

8.1 TIMER BASED INTERRUPTS

The DSP we are working with, or more accurately the EVM board, has some hardware capable of timing. Apart from that, the only synchronizing source we have access to is the web camera itself, which produces an image 25 times every second and notifies the host program about the grabbing. It could be quite useful to be able to trigger certain events between frames, for example we would be able to flip the ball at a higher frame rate, what would be accomplished interpolating the ball's position with some modified prediction algorithms.

In the case of linear prediction, this predict-and-interpolate mode would be very easy to implement, and for Kalman filtering we could use a modified version which uses a Kalman predictor (future samples) and a linear interpolator (in-between samples). In this way, our idea was setting up one of the timers to interrupt the main program when necessary, so that the service routine could change the flipping variables more often than the frame rate.

There are two hardware pieces capable of timing in the DSP, which could serve for our purposes:

1. Dual timers, independent from each other and from the rest of the hardware: we managed to set the routines to work in a separate project, but we encountered problems when integrating that code into the main project, there seemed to be conflicts between the different interrupts attended by the processor. This path was abandoned after some days.
2. Internal audio codec timer: it has quite fewer options than the independent timers, but it was easier to initialize. Anyway, we couldn't set any optimized way of continuously interrupting the processor, and as we were running out of time we left this track as well.

8.2 MODIFIED FLIPPER DECISION

Our current program uses two decision regions, one for each flipper, to decide when it is better to flip the ball. There are two different improvements that we are working at this time, and that might be included in the final prototype:

1. Definition of multiple decision regions for each flipper, in order to generate more complexly shaped regions. Each region could also have their own activation rules, based on the speed and the direction of ball movement. All of these regions would be modifiable through the GUI, we would only need to exchange more data with the

DSP card; the most difficult task is mapping "intelligent" decisions based on these regions.

2. Trajectory estimation and decision based on minimum distance to the flippers, considered as single points in the 2-dimensional space. We think we could benefit from this approach mainly because it makes the flipping process independent from the ball's speed. The computational cost of these algorithms is little higher than that of the decision regions, but we believe it would be insignificant compared to that of the processing algorithms.

8.3 IMPROVED DETECTION ALGORITHM

The main factor that constrains us from improving the detection algorithm is the computational power of the DSP: we have estimated that our current detection routines take more that half of the time dedicated to each frame. Some possible improvements in such an important part of the design, and their limited resources, are:

1. Increasing the kernel size of the smoothing function, that is directly proportional to the memory accesses per smoothed frame. However, this size must not be too high since we would blur the image and destroy the information contained in it.
2. Using a different windowing in the kernel (Hamming, Blackman…), instead of a constant rectangular window. In those cases, each pixel has to be multiplied by a different value, what can lower the overall speed.
3. After having obtained the estimated position of the ball with any of the explained algorithms, we could achieve a more exact tracking of the ball with a post-processing algorithm; we have an image of the ball stored in memory, and we convolute it with the boundaries of the noisy position already obtained, to refine our estimate.

APPENDIX A

SUMMARY OF LITERATURE STUDY

During the first week of the project, we performed a literature study searching for artifacts that could prove useful when deciding on how to develop an autonomous pinball playing software. This summary is an attempt to let the reader know of the most interesting features we found. Even some findings in the study which have not been used in our final prototype are included.

A.1 LATENCIES OF SYSTEMS

Latencies of the pinball player system are expected to be similar to the latencies listed in the setup of the system in [6]. Their latencies comprise:

Task	**Time** (ms)
Exposure time	*5*
Image transfer time	*20*
Frame grabber driver	*20*
Vision algorithm	*<14*
Prediction	*2*
Robot control (i.e. flipper activation in this project)	*10*

These are more or less exactly the same latencies which are expected for our system, but the only two latencies of use to us are the exposure time and total delay of our system. The exposure time determines the elongation of the ball which is quite large when the ball is moving fast. Simpler ball detection algorithms will find the center of the elongated ball. This will constitute an error, since the ball's true position at a specific time instance will be half of the exposure time later than that calculated by the algorithm. The total delay of the system also has to be known to enable accurate predictions of the position of the ball and result in activating flippers at the correct time instant. Unfortunately, the problem of latencies wasn't addressed neither in last year's pinball player project [8], nor in the other pinball player project examined [5].

A.2 OBJECT DETECTION

In the literature, several different schemes for ball detection are described. Detection of objects by their shapes is rejected in [6], since object recognition is considered difficult even without real time constraints. The same source also rejects object detection using color due to the need of a high contrast between the object and the background. Also, different lighting conditions imposed on the object as it moves would make the color scheme less useful.

These problems are addressed in [2] where four different kinds of image acquisition techniques are described. These are grayscale-imaging, RGB-imaging, narrow bandpass filtering and spectral distribution. All of them are based on the concept of edge detection, which describes how one can detect the edges of an object when it is projected on a background. The idea is that object contours in a "pixelated" image can be detected by checking the correlation coefficient between neighbouring pixels. This can be used to detect objects that have the same colour as the background since with a matte object with a uniformly coloured surface; a shading effect does not change the colour but the brightness. That is, the spectral distribution may be changed by shading.

The results in [2] are presented in the table below:

Kind of image **Type of materials (colour)**	Gray-scale	RGB-colour	Narrow band-pass filter	Spectral distribution
Paper (green) Feltcloth (green)				
Paper (blue) Plastic (green)				
Feltcloth (green) Plastic (green)				

Table 5 Image acquisition techniques

Green square means that the method can recognize between the two objects. Red square means the opposite. Yellow means "almost". Although the method using spectral distribution shows the best results using a spectrometer is not an option in this project. The interesting results for us are for grayscale and RGB-colour. RGB-colour is able to distinguish between a blue paper and a green feltcloth due to difference in colour. However, it is not able to distinguish between the green feltcloth and the green plastic, because both surfaces have the same colour and the same brightness. Gray-scale was able to detect the edges of an object only in one case.

The simplest way to detect changes in an image (an object moving in it) is by extracting the difference between to images [3, 6]. The changes result in large values in a difference map which is computed as the absolute values of the difference between the pixels in the two images. The difference map is then usually binarized by setting a threshold [3, 5, 6, 8].

Computing the difference image between the actual image and some reference image (see section on reference images below) and detecting its shape in the result using some kind of threshold was considered the best alternative in [6]. They refer to it as very robust, only requiring the background to be static and an object appearing different from the background.

In [5], motion was detected by subtracting each image from its predecessor and using a threshold. Two consecutive difference images could then be used to get a binary image only containing the current position of the ball. This method was found faulty when the flippers moved, as they were mistaken for the ball. To bypass this in [5] they ignored any activity in the regions of the flippers. They also had problems with lights flashing on the playfield,

which forced them to turn off the lights on their pinball machine. Finally, they had troubles when people moved close to the pinball machine, so they made sure that nobody was close to it when it was playing.

In [8], a combination of schemes was used. First, a difference image was formed between two successive frames. The result was thresholded with alpha*max_intensity_in_result (0<alpha<1). The position of the ball was assumed to be the mean of the coordinates of all points in the thresholded image. In [8] they state in one place that the difference image was convoluted with an image of the ball before thresholding, but don't mention it at other places, so whether this was done or not is a mystery. There were also speculations on scaling down the difference image to less than 50x50 pixels. Then normally the darkest pixel (but sometimes the brightest) would give the position of the ball. They saw an advantage of this, since they then didn't have to do any convolution with an image of the ball in the difference image. But frame rate issues prevented them from using this concept. They also felt that it was fully satisfactory if the coordinates for the ball was on the ball.

The illumination of the environment where the object to be detected is often not controllable and can cause false detection. In [3] the attempt is to make change detection technique robust to illumination variations.

The proposed solution is to use of a colour edge detection scheme applied to the difference between a current and a reference image. The value of the threshold depends on, according to [3], among other factors (such as camera noise) on illumination conditions.

In the text, they mention several features that are invariant to illumination conditions: pixel ratios, level lines, edge maps and vector differences.

- **Pixel ratios**. Here structural changes are detected by using the ratio of intensities of each pixel in two corresponding regions in two images. A structural change in the region is found when the intensity ratios vary inconsistently. This makes the detection less sensitive to illumination variations but requires much computational power.

- **Level lines**. The idea is that global illumination does not alter the geometry of the level lines, but will just remove or make new ones. The drawback is it also requires more computational power. To overcome this, a simplification of the computations is proposed but this draws done the system performance considerably.

- **Edge maps**. This method is based on moving edges and has low computational cost. The drawback is that it can only deal with one moving object at a time.

- **Vector differences**. A method based on local characterization of structures invariant to changes in illumination in the image. This technique gives good computational complexity but not so good accuracy in detecting object contours.

According to [3] early edge based change detection algorithm were divided in two steps. First, the edges in the background reference image and the edges in the current image were extracted. Then the obtained edge maps were compared by a xor-operator. This provides good accuracy and illumination independency by the use of edges. On the other hand, if the reference image is very complex with many edges the detection becomes more difficult. To

avoid this [3] propose to compute edges after differencing. The edges are detected after pixel differentiation in the two frames.

They propose the use of colour edges to improve contour detection. [These techniques are applied to three colour channels independently and the results (an edge map for each colour channel) are combined by a logic or operator. To extract edge information a Sobel edge filter is used. The Sobel edge extractor provides thick edges which are useful in the application of change detection.

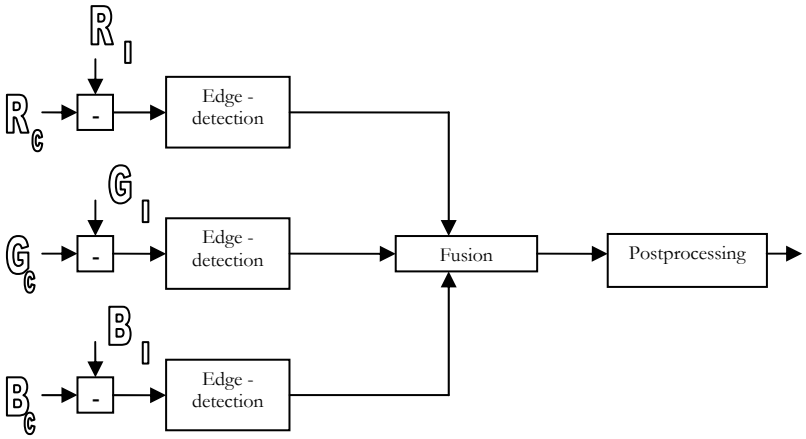


Figure 23 Edge detection on colour channels

In the literature, different schemes are used to obtain a reference image. Slowly adapting reference images are employed in [1] and [6]. Use of one [8] or a few [5] previous frames and combining them to get the current position of a ball is also described in the literature.

In [6], not one reference image, but two is employed. These are used to indicate the highest and lowest value which should be considered as normal variations for each pixel. This is their solution to fast changes occurring at image edges due to pixel jitter, small camera vibrations and interlacing. Their two reference images are completely updated within five frames to cope with slow changes in the image.

In [1], the reference image is updated in an elaborate way. If a pixel is considered to be static, and there is no previous reference value for it, the reference value is set to the current intensity of the pixel. If the pixel is moving, the reference is not updated at all. If the pixel is static, and there exists a previous reference value, the new value is I_ref=0,5*I_ref + (1-0.5)*I_current.

In [8], the previous frame (including the ball) is simply used as the reference image.

A.3 STATE OF THE BALL (POSITION, VELOCITY)

In [6], the current state of the ball is determined using an Extended Kalman Filter. But in their system, the ball is thrown in the air and the balls position is determined at about 50 points of the balls trajectory, giving the Kalman filter a lot of useful input. In our case, we might only have a few points. In [5], there seems to be no use of the balls velocity, and hence no prediction, which is odd considering their low frame rate which would make this very

useful. In [8], they use Kalman filtering (as well as linear prediction) to get the state of the ball.

A.4 PREDICTION

Prediction is used in [6], to determine how to position a robotic arm receiving a ball thrown from five meters distance. They use a model of the trajectory of a ballistic throw to predict future positions of the ball. They use numerical integration to predict where the ball trajectory will pass a surface close to the robot, and they then position the robot to be able to receive the ball.

In [5], they don't disclose whether they use prediction or not. There seems to be no use of the balls velocity, and hence no prediction, which is odd considering their low frame rate which would make this very useful

In [8], they try linear prediction. They also try Kalman filtering for both prediction and for smoothing the path of the ball.

A.5 FLIPPER ACTIVATION

In [5], they had three strategies for flipper activation; the simplest is to activate and deactivate the flippers all the time. The next strategy they have is to divide the board into rectangular regions, and activating one or both flippers if the ball is detected in a certain region. Their third strategy is to use a finite state machine (FSM), where the board is divided into rectangles, and the FSM enters a new state whenever the ball moves into another region. Some states are associated with flipper activation, some with deactivation, and the rest does nothing. The state entered when entering a new region depends on the previous state as well as the new region.

In [8], they try using rectangular decision regions, as well as an Artificial Neural Network (ANN). They change the size of their rectangular decision region depending on the speed of the ball. That indicates an error in the estimation of the future position of the ball or an ability for fast moving balls to "jump" across the decision region, as their current position is on one size of the region, and their predicted position is on the other side. It is enough for the ball to be in a region, or the predicted position to be in a region to cause activation. The ANN network does not give satisfactory results.

They sometimes detect the flipper as a ball when a flipper was activated, resulting in an infinite loop of continuous flippering. Their solution is to stop all ball detection while the flipper is activated (for about one second).

A.6 LIMITING REGIONS TO SAVE CPU

In [6], the computation of a difference image is only applied to a limited region of interest (ROI) to save computation time. This ROI varied from 2% to 40% depending on how well they believe their estimate of the balls next position is. They also only update one fifth of their two reference images in an interleaved pattern each frame.

A.7 OTHER OBSERVATIONS

In [5], a list of measurable quantities is presented, which could be useful when trying to determine the performance of a pinball playing system.

In [8], they describe a scheme they call "stereo-flippering", which is able to catch some more balls in the case that the ball is almost in the middle of the two flippers. They first activate the flipper closest to the ball, and after some small delay the other flipper is activated. This has to be implemented as a separate algorithm in their case, since they turn of their ball detection when the first flipper is activated, giving the system no chance to realize when to activate the second flipper.

APPENDIX B

USING THE GRAPHICAL USER INTERFACE

The graphical user interface has been designed to be as simple as possible to use. To start the application, double-click on Smartpinball.exe. The software then starts to capture video from the first video capture device, and a preview of this video is shown in the web-cam frame.

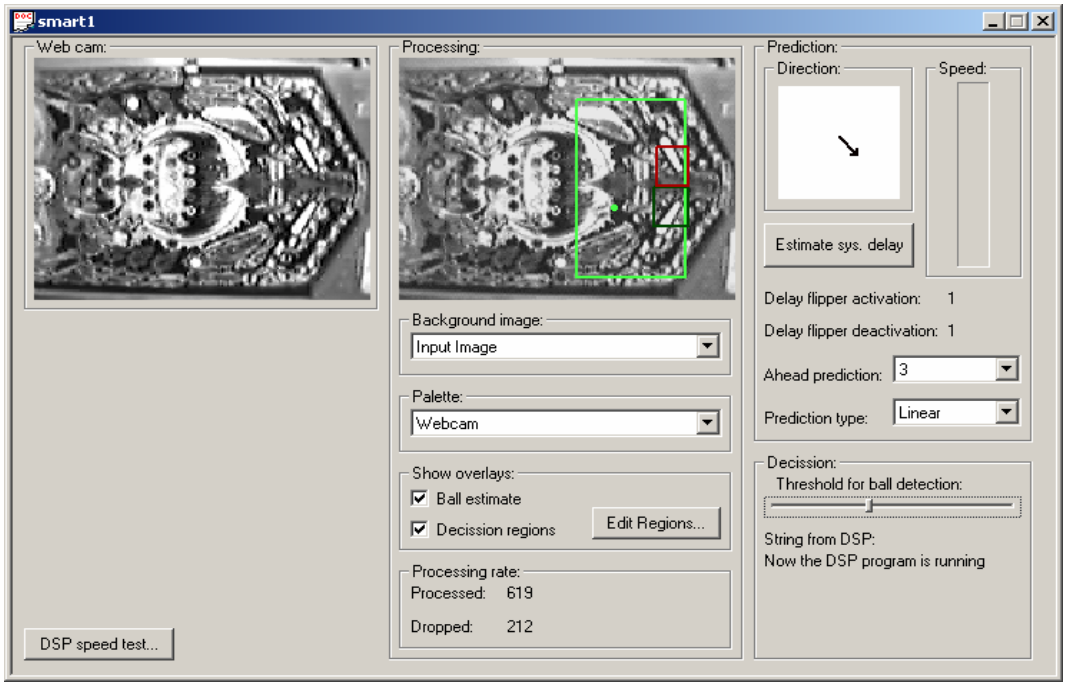


Figure 24 the user interface

B.1 PROCESSING FRAME

In the processing frame, the user can select a second image to view. It could be the input image, the reference image, the difference image, or the filtered reference image. The user can also choose the palette of these images, to facilitate interpretation of the viewed image. Besides that image, the user could also choose to display the ball's position and decision regions on top of the second image. Finally, some information on the number of processed and dropped frames can be seen.

The button "Edit regions…" opens a new window, where the user can modify all regions of the application. This window is described further down below.

B.2 PREDICTION FRAME

In the Prediction frame, the direction and speed of the ball can be seen. This frame also has a button, which upon activation lets the DSP automatically determine the total system delay in number of frames. The delay determined is shown, but the user has to manually select how far ahead to predict in a drop down list. The user can also select which prediction method to use. The choices are between none, linear, and Kalman prediction.

B.3 DECISION FRAME

The final frame, labeled decision, contains a slider for selecting a threshold for ball detection. This is necessary since noise otherwise would be detected as the ball when the ball is out of view. The decision frame also can display a string from the DSP, whose content may vary.

B.4 EDIT REGIONS WINDOW

This is a window of the main application used for drawing different regions. The view can be zoomed to facilitate placement of the regions, since the original size of 176x144 is very small.

There exist three different kinds of regions in this application.

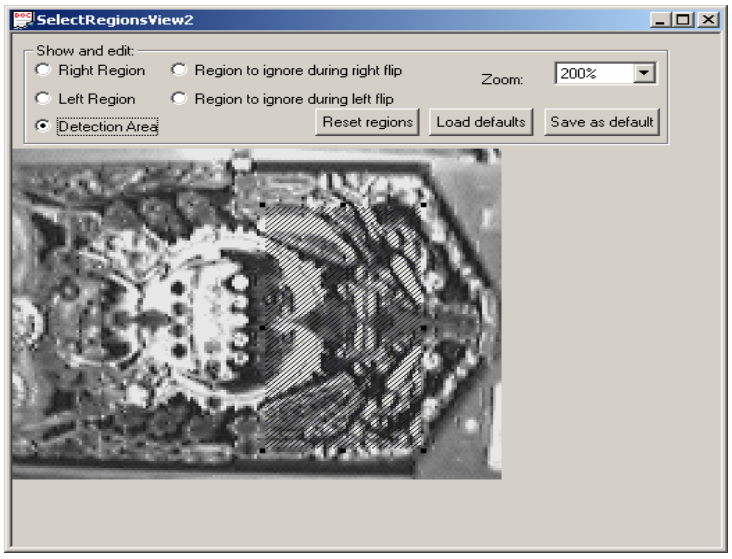


Figure 25 The edit regions view

First, there are the left and right flipper activation regions. These regions should be placed in front of the flippers. Try moving these regions closer and further away from the flipper until you discover for which placement the ball is properly hit by the flippers.

Secondly, there is a large detection region which determines what part of the image to process. This region should be sized to include a large area in front of the flippers while still excluding as many flashing lamps as possible of the pinball machine. The size of this region also affects the processing time for the DSP. Decreasing the size of this region helps if the DSP don't have time to process enough frames.

The last region types are the left and right flipper ignored regions. These regions should be placed so the corresponding flipper is always inside its ignored region regardless of whether the flipper is activated or not.

REFERENCES

[1] Alice Caplier et al., **"Robust fast extraction of video objects combining frame differences and adaptive reference image"**, Laboratory of Image and Signal, Grenoble Cedex, France

[2] Yoshitsugu Manabe et al., **"An Object Recognition through Continuous Spectral Images"**, Osaka University, Japan

[3] Andrea Cavallaro and Touradj Ebrahimi, **"Change detection based on colour edges"**, Swiss Federal Institute of Technology, Lausanne, Switzerland

[4] David Salmond, **"Target trackning: Introduction and Kalman tracking filters"**, Code Technology Park, Farmbourugh, Hants, GU14 0LX

[5] Dayton Clark, **"Progress Toward an Inexpensive Real-Time Testbed The Pinball Player Project"**, Brooklyn College/CUNY, New York, USA

[6] U. Frese et al., **"Off-the-Shelf Vision for a Robotic Ball Catcher"**, German Aerospace Center, Wessling, Germany

[7] Jun Zhang and Gerald G. Hanauer, **"The Application of Mean Field Theory to Image Motion Estimation"**, Member, IEEE

[8] Group Blue, **Smart Pinball Final Report**, Spring 2003, KTH.

[9] Håkan Hjalmarsson and Björn Ottersten, **Lecture Notes in Adaptive Signal Processing**, 2002-12-19, KTH.

[10] Feymand **Lecture on Physics**, 1977

[11] Gonzales and Woods, **"Digital Image Processing",** 2:nd ed., , Prentice Halls, ISBN 0-201-18075-8.