Demo: Real-time Tracking of Round Object

by: Brianna Bikker and David Price, TAMU
Course Instructor: Professor Deepa Kundur

Introduction

Our project is intended to track the motion of a round object in real-time. To accomplish this goal, we need to first single out the moving object from its background. Once it has been singled-out and identified, we need to track the object. Since the object is in motion, this would be difficult to implement in real time without some sort of predicting mechanism. Finally, there must be a way to show that the object is being tracked. Let's look at an algorithm that implements these techniques using the Kalman Filter.

Autothreshold

The Autothreshold block takes an intensity image and converts it to a binary image using Otsu's method. This process takes in an image matrix (an intensity image, in the case of Figure 1 below) and computes a threshold value by splitting the histogram of the input such that the variance of each pixel group is minimized.

Figure 1- Original Image ("Otsu's Method")

The thresholding operator (user-defined) then determines which pixels become 0 or 1. For instance, if you select > and the input value is greater than the threshold value, the block outputs 1 at the BW port; otherwise, it outputs 0. This produces the overall binary image (shown in Figure 2 below).

Figure 2- Threshold of image using Otsu's method ("Otsu's Method")

Blob Analysis

The Blob Analysis block is used to calculate statistics for labeled regions in a binary image. First the Blob Analysis block marks regions of identical values (a group of 0s or a group of 1s) from within the binary image as "blobs". In this demonstration, we will calculate the centroid and area of the white region in the binary image. The centroid statistic outputs the coordinates of the centroid of each blob. The Area statistic outputs the total number of pixels contained within the blob. Once the Blob Analysis Block outputs the blob statistics (area and centroid) they must be contained in a lookup table that can be accessed by the motion predicting filter (here we used a Direct Lookup Table block).

Kalman Filter

The Kalman Filter uses Bayesian estimation to predict motion of an object. Given a state-space model of the plant (u) and the process and measurement noise covariance data (v), the Kalman filter constructs a state estimate with minimum steady state error (ŷ is the true plant output, whereas x is the ideal output). The state diagram of a Kalman filter is depicted in Figure 3 below.

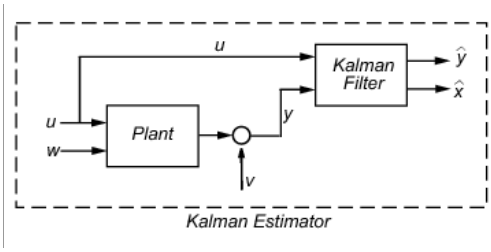

Figure 3- Kalman Estimator ("MATLAB Help")

Draw Markers

In order for a human to verify that the algorithm is actually tracking the moving ball, we must implement a visual indicator. In this demonstration, we will use a Draw Markers block in Simulink to draw a box around the object during the video monitoring of the object.

Design and Implementation

We start by capturing imaging by importing an .mpeg file into Simulink which outputs the video in the form of an intensity video. We then threshold the video to remove the background and only output the moving object as a binary image. To detect the contents of the binary image, we apply "Blob Analysis", which analyzes the contents into 2 parameters: Area and Centroid. These parameters are then sorted into a Direct Lookup Table. The Kalman Filter then takes the values in the Direct Lookup Table, along with the assigned initial state space and transition matrix, to predict the movement of the blobs labeled by the Blob Analysis block (Prerak).

The design of the Kalman Filter is important to guarantee effective tracking. Since the Kalman filter "tracks" the object by predicting the motion of that object, it is important that the filter is set up correctly. As mentioned above, the Kalman filter consists of several parameters to be manipulated: the state-space model, the process and measurement noise covariance data. In the Simulink block for the Kalman Filter, these parameters are indicated as: 1) the initial condition for estimated state; 2) the initial condition for estimated error covariance; 3) the state transition matrix; 4) the process noise covariance; 5) the measurement matrix source; 6) the measurement matrix; and 7) the measurement noise covariance.

The default settings of the Kalman Filter block are appropriate for a simple state prediction. For our project, however, we modified the values in the Kalman filter to be better suited for motion detection of an object in a video feed (see Prerak). For instance, the state transition matrix should account for the video frame rate in order to better predict movement; the frame rate limits the speed at which the Kalman Filter needs to update the state. Furthermore, the noise covariance is affected by the frame rate and must be scaled to account for it.

Building the Simulink Model

1. Insert From Multimedia File upload BikkerPrice.MPG video, set image color space output to intensity.
2. Insert Autothreshold block, Connect output From Multimedia File block to input of Autothreshold.
3. Insert a Blob Analysis block from Video and Image Processing Blockset -> Statistics. Set block to have Area and Centroid statistics, with "Statistics output data type" set to Specify via fixed-point tab. Connect input of Blob Analysis Block to output of Autothreshold block.
4. Insert Maximum Block from Video and Image Processing Blockset -> Statistics. On Main tab, set Mode to index, Index base to Zero and Find the maximum value over to Each Column. Connect input of Maximum Block to the Area output of the Blob Analysis Block.
5. Insert Data Type Conversion Block from Simulink->Signal Attributes. Set Output Data Type to uint32. Connect input of Data Type Conversion to Centroid output of Blob Analysis Block.
6. Insert Direct Lookup Table (n-D) Block from Simulink->Lookup Tables. Set Number of table dimensions to 2, Inputs select this object from table to Column, and check the Make table an input box. Connect the output of Maximum Block to the first input of the Direct Lookup Table (n-D) Block and connect the output of the Data Type Conversion Block to the second input of the Direct Lookup Table (n-D) Block.
7. Insert Data Type Conversion Block, set Output Data Type to fixdt(1,32,15). Connect input of Data Type Conversion Block to output of Direct Lookup Table (n-D) Block.
8. Insert Kalman Filter from Video and Image Processing Blockset -> Filtering. Set Number of filters to 1 and Enable filters to Always. Set other parameter according to Figure below. **Note:** 1/30 is the sampling period (Frame Rate) of video.

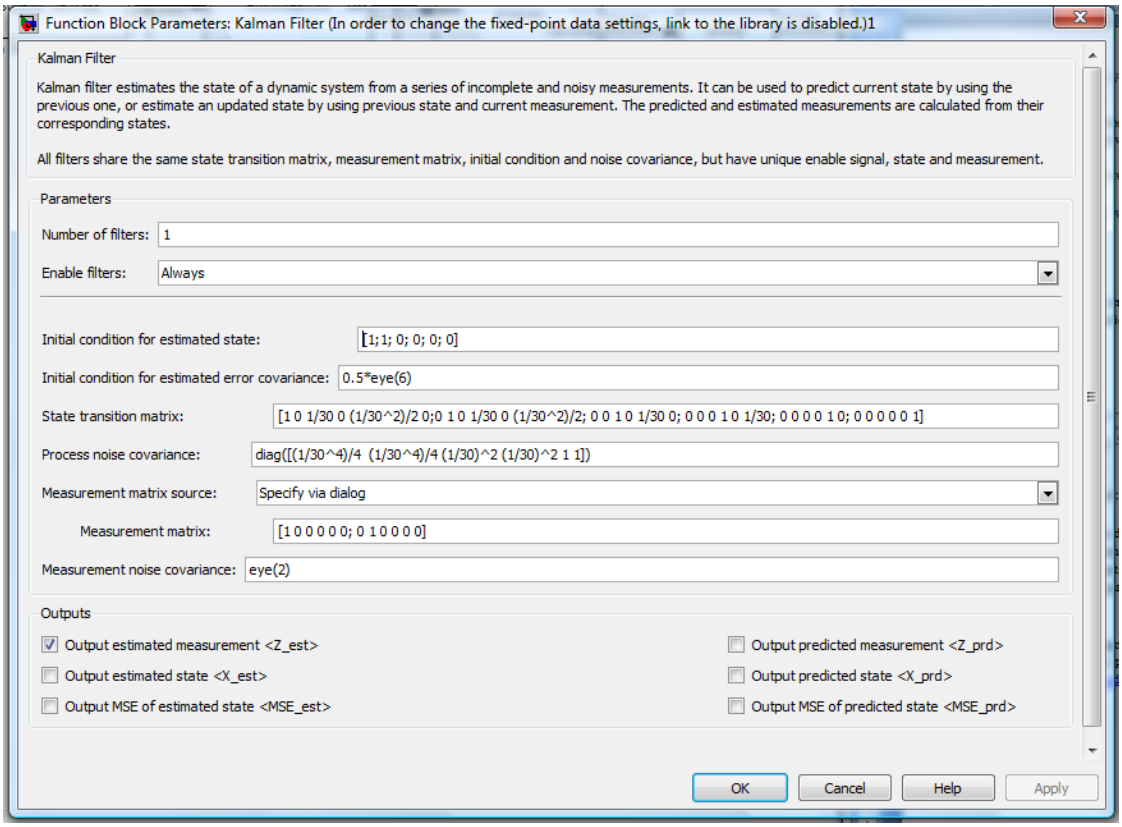

Figure 4- Kalman Filter Parameters

9. Insert Data Type Conversion Block and set Output data type to uint32. Connect input of Data Type Conversion Block to output of Kalman Filter.
10. Insert Goto from Simulink->Signal routing. Connect input of Goto[A] to output of Data Type Conversion.
11. Insert From from Simulink->Signal routing.
12. Insert Data Type Converstion Block. Set Output data type to uint32. Connect input of Data Type Conversion Block to output of From[A].
13. Insert Draw Markers from Video and Image Processing->Text & Graphics. Set Marker shape to Square, Marker size to 70, Border color to white. Connect Image port to output of From Multimedia File, Pts port to output of Data Type Conversion Block after From[A].
14. Insert To Video Display from Video and Image Processing->Sinks. Connect input of To Video Display to output of Draw Markers Block.
15. Press the Play button in Simulink to run simulation.

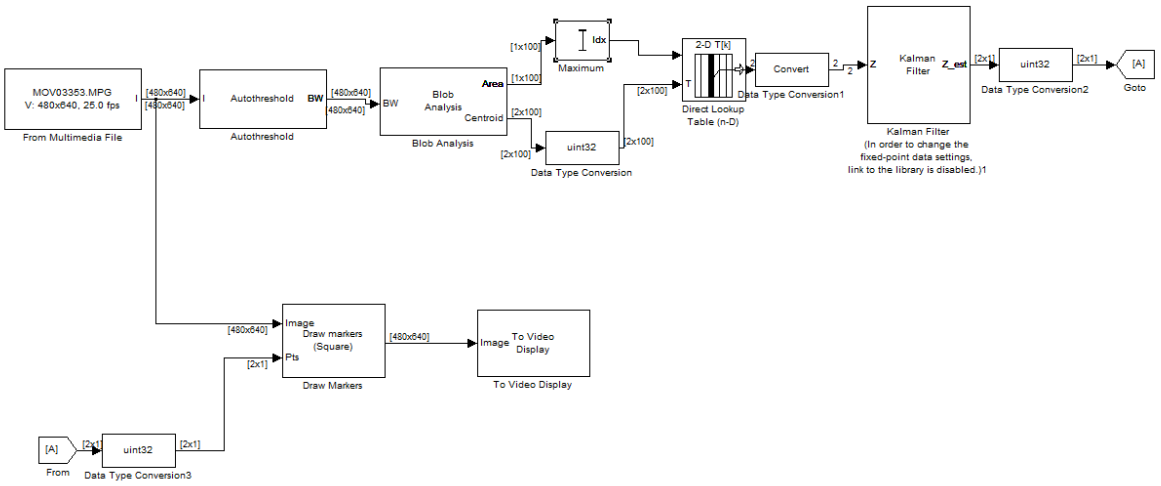

Figure 5- Simulink Model of Real-Time Ball Tracking

Digital Video Hardware Implementation

1. Take the Simulation Model created in the Building the Simulink Model section.
2. Delete the From Multimedia File and the To Video Display
3. Insert the DM6437EVM from Target Support Package→Supported Processors→TI C6000→Target Preferences.
4. Insert the Video Capture Block from Target Support Package→Supported Processors→TI C6000→Board Support→DM6437EVM.
5. Insert the Deinterleave Block from Target Support Package→Supported Processors→TI C6000→Board Support→DM6437EVM. Connect the Input of the Deinterleave block to the output of the Video Caputure Block.
6. Insert Goto from Simulink->Signal routing. Label it as [Cb]. Connect input of Goto[Cb] to the Cb output of the Deinterleave Block.
7. Insert another Goto from Simulink->Signal routing. Label it as [Cr]. Connect input of Goto[Cr] to the Cr output of the Deinterleave Block.
8. Connect the Y output of the Deinterleave block to the I input of the Authreshold Block.
9. Insert an Interleave block from Target Support Package→Supported Processors→TI C6000→Board Support→DM6437EVM. Connect the Y input of the Interleave block to the output of Draw Markers block.
10. Insert From block from Simulink->Signal routing. Label it as [Cb]. Connect the output of the From[Cb] to the Cb input of the Interleave block.
11. Insert From block from Simulink->Signal routing. Label it as [Cr]. Connect the output of the From[Cr] to the Cr input of the Interleave block.
12. Insert a Video Display block from Target Support Package→Supported Processors→TI C6000→Board Support→DM6437EVM. For maximum video visibility, set the Video window position to [0, 0, 480, 480]. Connect the input of the Video Display block to the output of the Interleave block.
13. Turn on the CM208 camera and connect to the Video In line on the DAVINCI DM6437EVM. Connect the J2 output of the DM6437EVM to Input A on the JVC screen.
14. Power on the DM6437EVM and press Ctrl+B to start the hardware implementation of the program through Simulink.

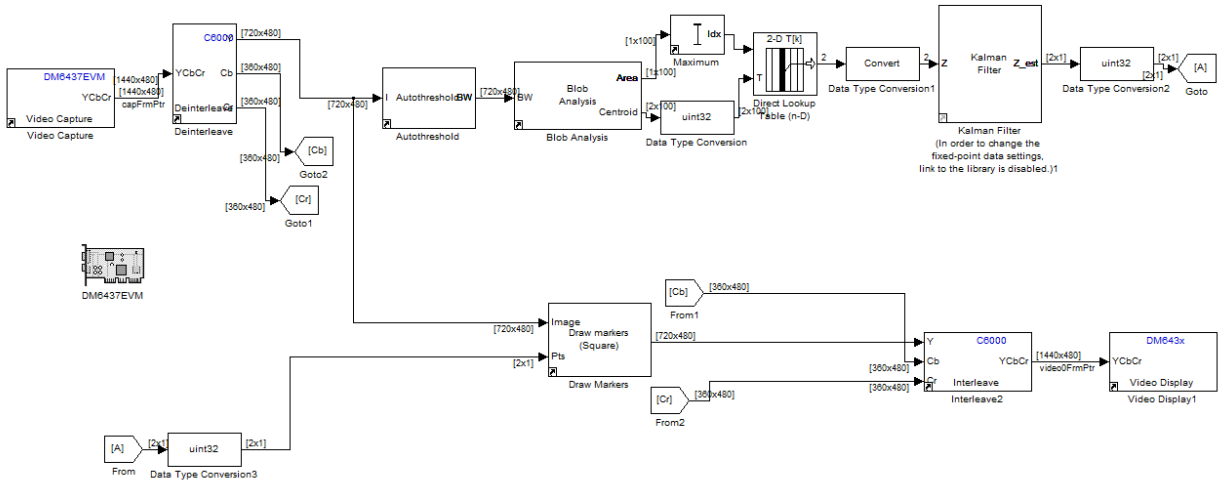

Figure 6- Video Hardware Implementation of Real-Time Ball Tracking

Results

Simulink Model

We implemented the Simulink Model in file BikkerPrice1.mdl. As you can see from Figure 7 below, we started with a video of a ball in motion. Through the auto-threshold, we dropped the background and captured the moving object in a binary image. This is shown in Figure 8 below. Applying the Kalman Filter, we estimate the location of the moving ball. We then draw a box around this location and update the video with this marker. The resulting video and combined marker are then output as a video display; as seen in Figure 9, the algorithm we implemented is able to detect the location of the ball and track it as it moves across the screen.

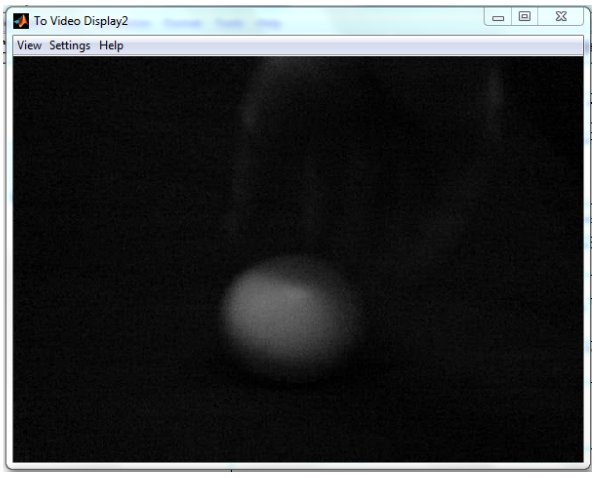

Figure 7- MPEG video screenshot of ball in motion

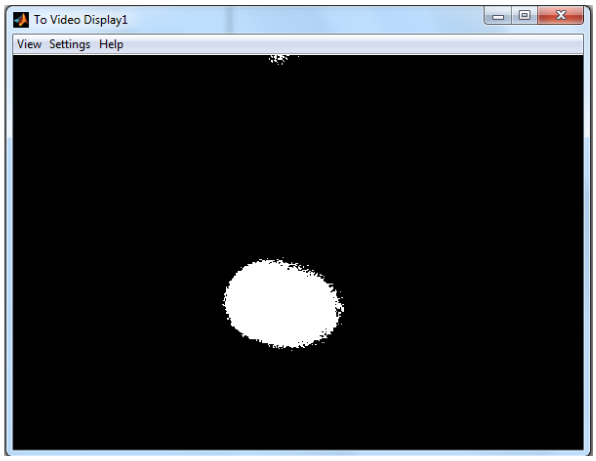

Figure 8- Threshold of video calculated with Autothreshold Block

Figure 9- Screenshot of video tracking of ball in motion

One of the problems we came across during the Simulink simulation was that while the ball was out of frame, the tracking marker would shake sporadically because of the noise produced by the Autothreshold Block. This noise can be seen below in Figure 10. However, this characteristic did not affect our ability to detect the ball's motion while in frame; the tracking marker immediately detected the ball (due to the sharp image produced by the autothreshold) and following it entirely while on screen.

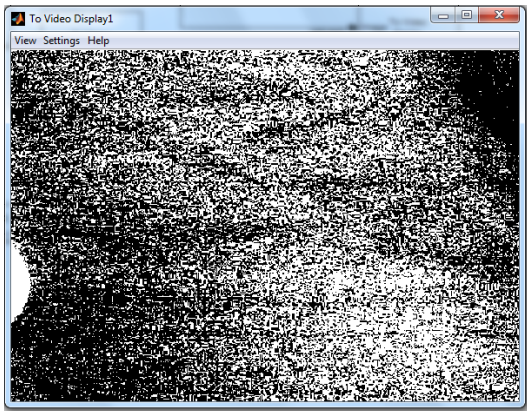

Figure 10- Noise in thresholded image when ball is outside of frame

Hardware Implementation

We implemented hardware in file BikkerPrice2.mdl. The hardware implementation of the motion tracking algorithm was very similar to the Simulink model. However, the major difficulty in implementing this algorithm in hardware was due to the video input type that is produced at the input. The given CM208 camera produces an YCbCr output, whereas our simulation .MPEG file produces an Intensity output. Because of this, we had to first, deinterleave our video signal into separate Y, Cb, and Cr feeds. Then, we considered the Y feed as the "Intensity" image to do our blob analysis and Kalman filter on. The Cb and Cr aspects of the video were fed along with the filtered Y signal directly back into the output video stream (see Figure ## for visual).

During real-time tracking, our algorithm is able to track movement of a round object, as designed. However, there are some limitations. First, the motion of the object must be relatively slow compared the motion that the simulation was capable of tracking. Second, the object being tracked must have a relatively bright color. And third, the portion of the screen in which the object is can affect the quality of the tracking.

Let's address why these limitation exist. The first limitation, related to the speed of the object, exists simply because of the lag in processing on the board. Unfortunately, this cannot be corrected. However, it does not *significantly* affect our results in a negative manner.

The second limitation, related to the color intensity of the object, is due to the signal produced by the camera. Since the CM208 produces a YCbCr image, we had to create an intensity image from these feeds. From experimenting with color space conversions, we found that simply taking the Y signal of the camera feed as our intensity image works the best. Although the Y signal gives us a decent "Intensity" image, the auto-threshold of this video signal is fairly poor at detecting variation in color unless it is a very bright color (such as a bright red); this causes a poorer tracking functionality than is available with the simulation. There may be a solution to this limitation by further manipulating the YCbCr output of the signal to better detect intensity contrasts.

The third limitation, related to the location of the object in the frame, is most likely to the way the camera is actually designed. It can be reasonable to assume that the camera is less accurate along the perimeter due to lens limitations. This normally shouldn't be a problem, as one could expect to conduct motion tracking in the center of the camera frame to assure proper video capture. However, the CM208 is designed to output a 1440x480 signal, while the display screen we are given to work with is only 480x480. So the camera capture space that is actually displayed on the video screen is relatively small. We observed that the motion tracking only works well when the ball is in the half of the display screen that is closest to the camera. This makes sense, as this is the region that the ball will be properly captured in the video intake. To correct this limitation, it would be best to view the video output results on a viewing monitor that is at least 1440x480.

Because of these limitations, the real-time tracking of a round object isn't as reliable as desired; however, we still observe a very distinct motion tracking when a round object is placed in the camera frame. The tracking marker appears on the screen as the ball or round object enters the frame, surrounds it to mark its location, and follows the location of that object as it moves throughout the frame.

Furthermore, the algorithm is successful in that it *only* detects a ball (or round object) in the frame while ignoring an edgy object (such as a square). Improvements in the algorithm can be made, but this demonstration shows that the Kalman filter can be successfully implemented to track a moving round object.

References

Matlab Help Manual. MATLAB 7.9.0 (R2009b).

"Ostu's Method." http://en.wikipedia.org/wiki/Otsu%27s_method

Prerak. "Kalman filter(fixed point version)." Matlab Central. 1 December 2011 http://www.mathworks.com/matlabcentral/fileexchange/25877-kalman-filterfixed-point-version/all_files