Impact Alert System

Team:
David Goodman
Sharon Rabinovich

Description

- Goal: Detect a ball moving towards the camera, and predict if the ball will hit the camera.

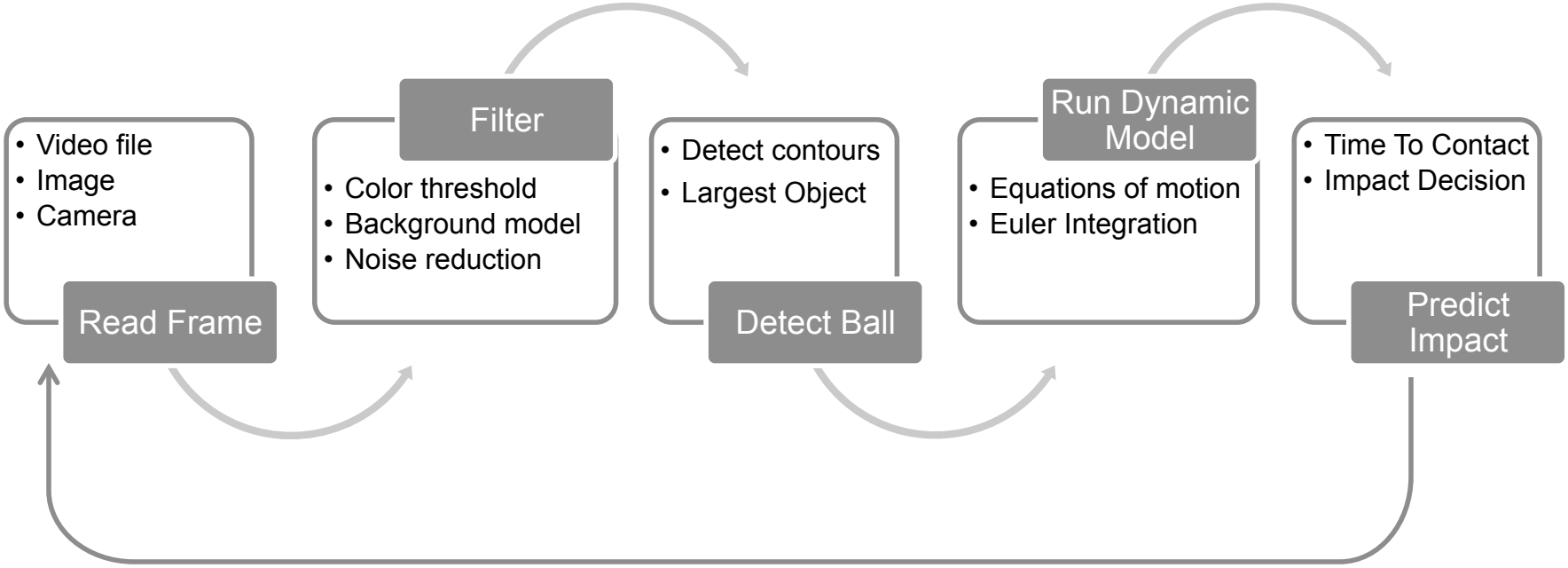
- References:
  - FastTrack by Lior Chen. Tracking an orange ball. lirtex.com 2010
  - OpenCV (Open Source Computer Vision). Opencv.org
  - Physics of Baseball & Softball, Ch. 3 Ball Trajectories by Rod Cross. Springer 2011
  - Small Unmanned Aircraft – Theory and Practice. Randal W. Beard, Princeton 2012
  - Topological Structural Analysis of Digitized Binary Images by Border Following. Satoshi Suzuki 1985
  - Improved Adaptive Gaussian Mixture Model for Background Subtraction. Zoran Zivkovic 2004

What We Did

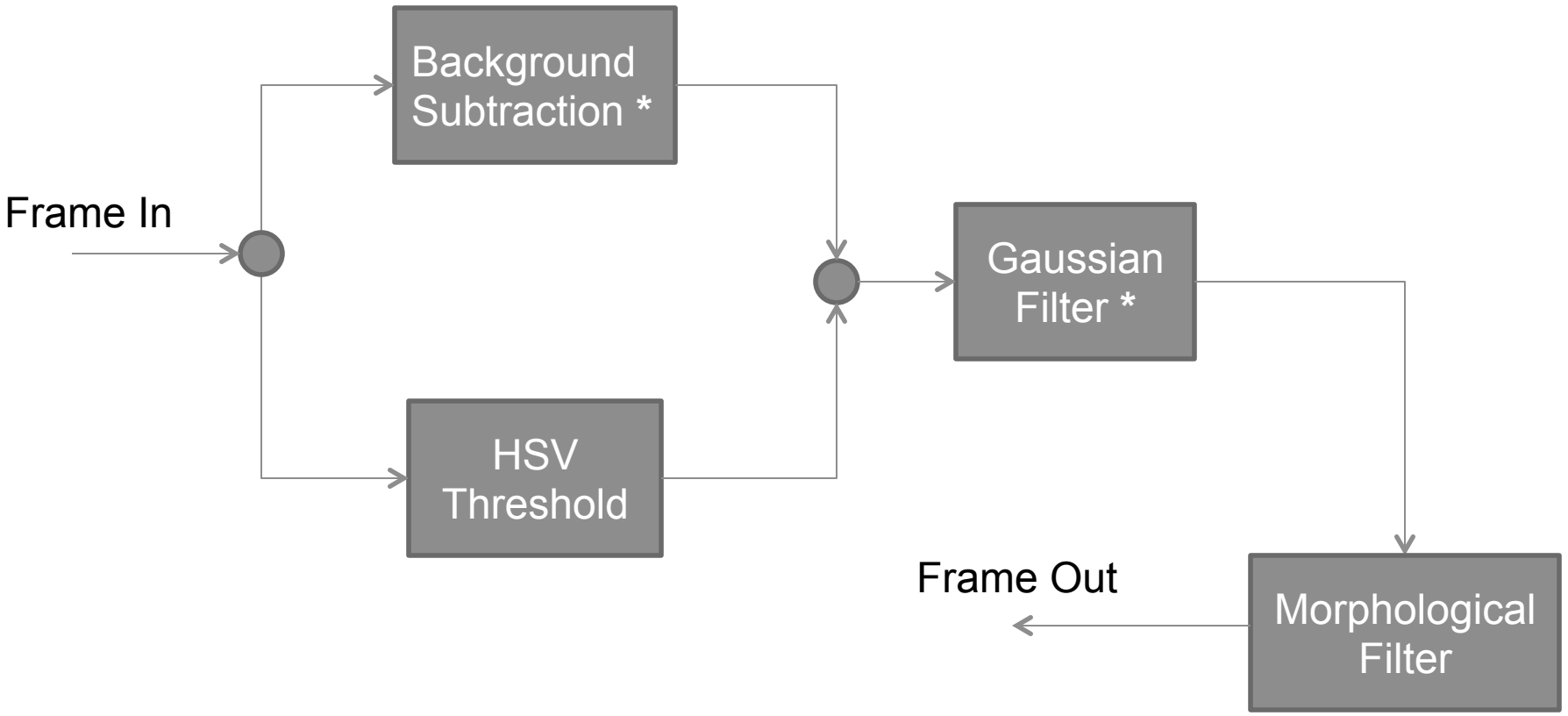
- Started with Lior Chen's FastTrack project. Re-wrote it as C++ application in Visual Studio 2010
  - Pause playback, step through frames (camera, video, images)
  - Parameter sliders and logging
- Very accurate tracking with filtering and contour detection
- Distance estimation function and camera calibration
- Dynamic model and prediction in MATLAB
- Implemented dynamic model and prediction in C++.

Algorithm

- Implemented in OpenCV 3.0.0-beta, C++
- Runs in real-time
- Tuned for each scenario



Filter Algorithm



* Optional (can be turned on and off during run-time).

Background Subtraction

- Convert image to grayscale
- Creates model of background
  - Learning disabled: scene is static
- Subtract model, threshold with GMM to generate mask

$$P(\vec{x}^{(t)}|BG) > C_{thr}$$

HSV Threshold

- Convert image to HSV
- Generate a binary image by thresholding

$$x_{out,i} = \begin{cases} 255, & \text{if } h_{min} \leq x_{hue,i} \leq h_{max} \\ & \quad \text{and } s_{min} \leq x_{sat,i} \leq s_{max} \\ & \quad \text{and } v_{min} \leq x_{val,i} \leq v_{max} \\ 0, & \text{otherwise} \end{cases}$$

Gaussian Filter

- Convolve grayscale image with Gaussian kernel to blur
- Removes noise, recovers shape

$$h(x,y) = Ke^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

Morphological Filter

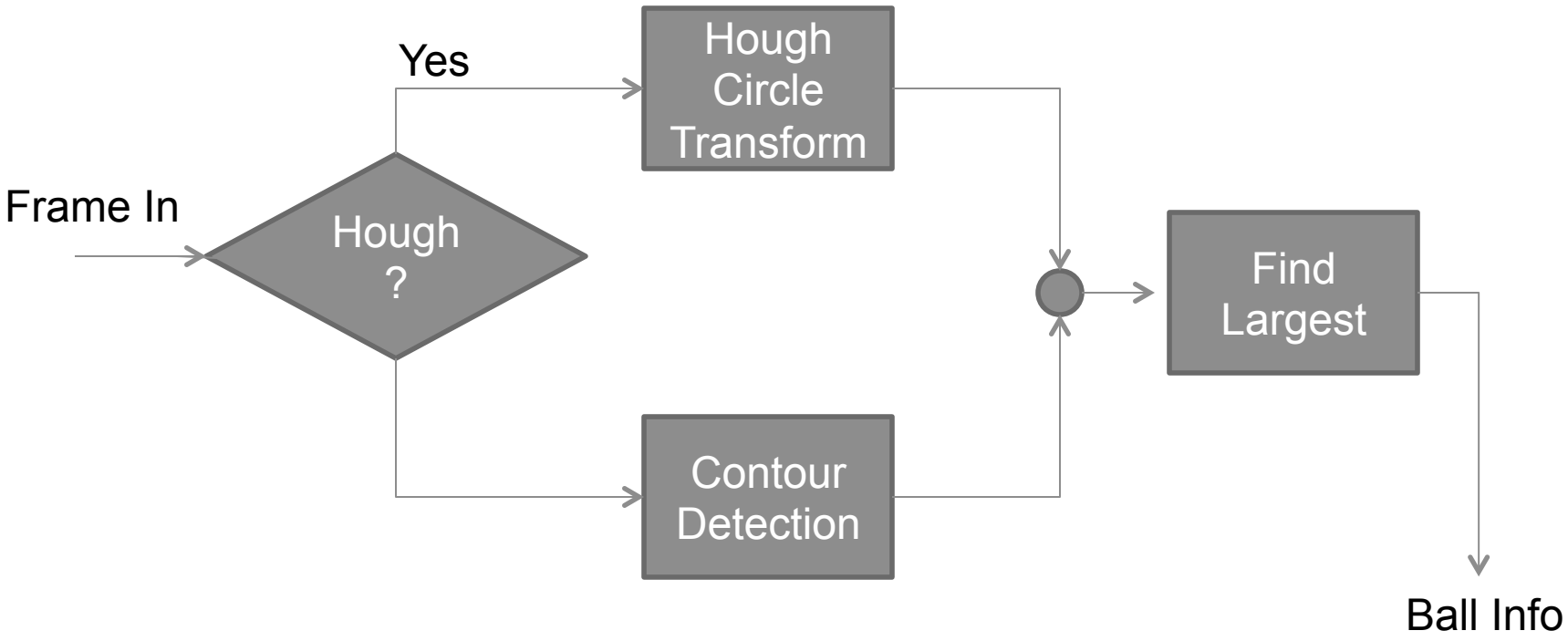
- Brightness of each pixel from comparison with neighbors
- Elliptic structuring element sensitive to shapes (ball)
- Erode: Removes pixels from boundaries

$$\vec{x}_{out,i} = \min_{\vec{x}':\ \text{element}\ \vec{x}' \neq 0} \left(\vec{x}_{in,i} + \vec{x}'\right)$$

- Dilate: Adds pixels to boundaries (preserve radius)

$$\vec{x}_{out,i} = \max_{\vec{x}':\ \text{element}\ \vec{x}' \neq 0} \left(\vec{x}_{in,i} + \vec{x}'\right)$$

Detection Algorithm



Hough Circle Transform

- Uses Canny edge detector
- Generalized Hough transform on gradient for circle eq.
- Robust, but slow, and detected radius is noisy.

Contour Detection

- Detect border pixels by looking at neighbors
- Finds rasterized contours and hierarcy from scan
- Compute moment (Green's theorem) for centroid & radius

Find Largest

- Find circle/contour with largest radius where:

$$r_{min} < r_{ball} < r_{max}$$

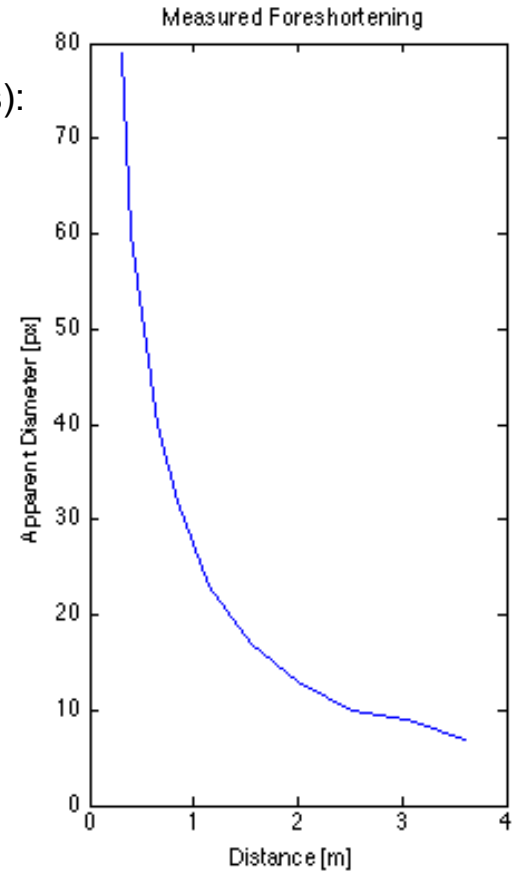
Distance Estimation

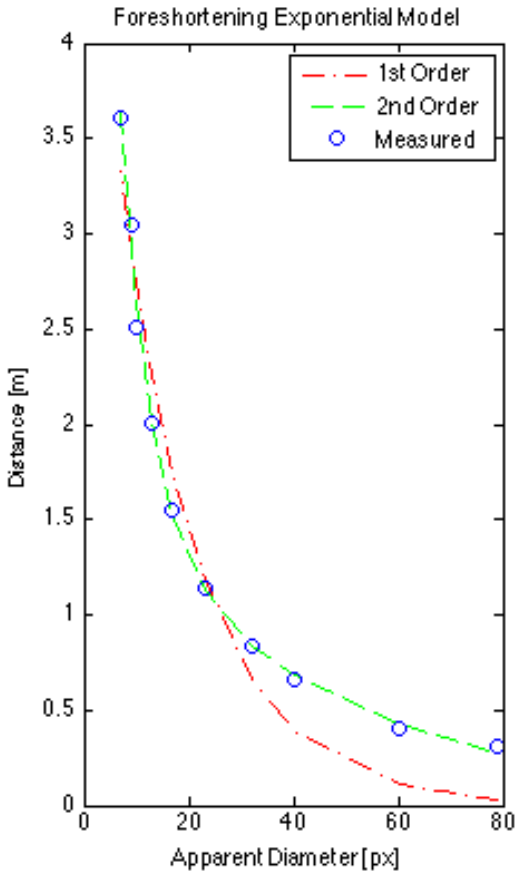
- Fit exponential model to measured foreshortening

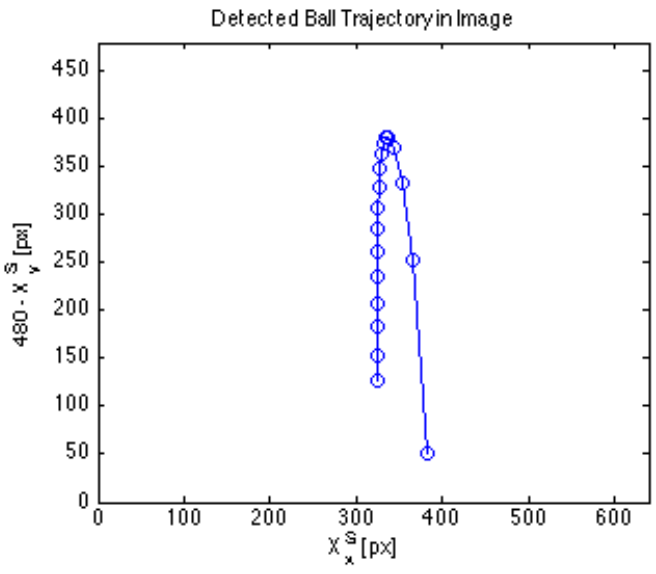
f(r) = a*exp(b*r) + c*exp(d*r)
Coefficients (with 95% confidence bounds):
a = 7.479 (5.066, 9.892)
b = -0.1759 (-0.245, -0.1069)
c = 1.7 (0.8402, 2.559)
d = -0.0231 (-0.03496, -0.01123)

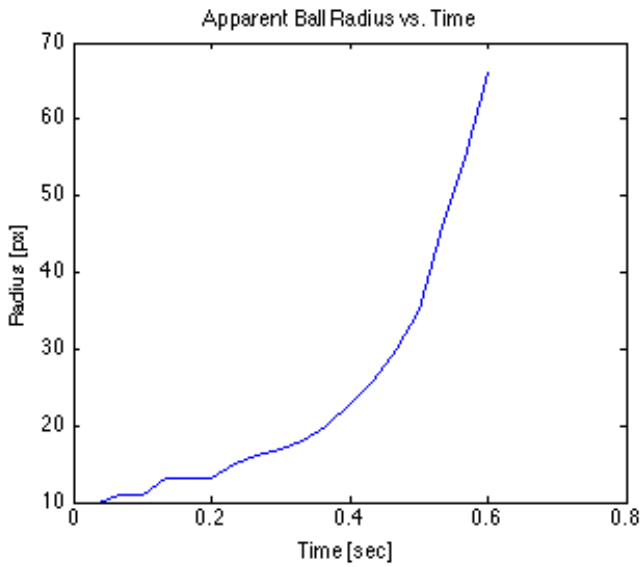
Estimate distance each frame ball is detected:

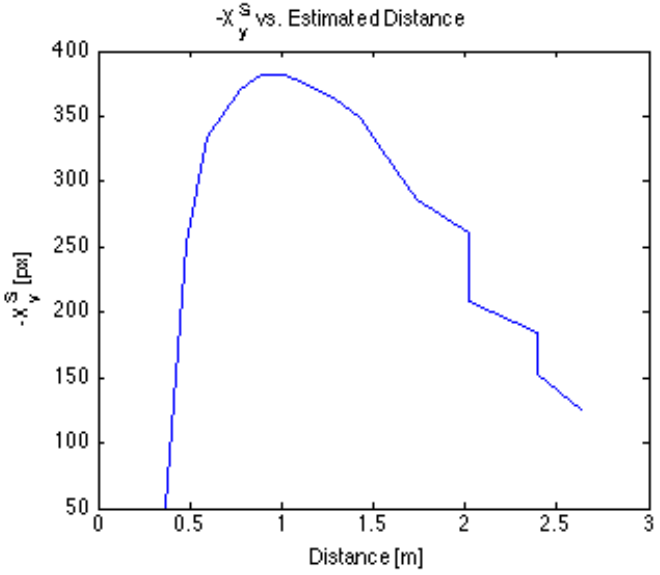
$$|\vec{P}^C| = f(\hat{r}_{ball})$$

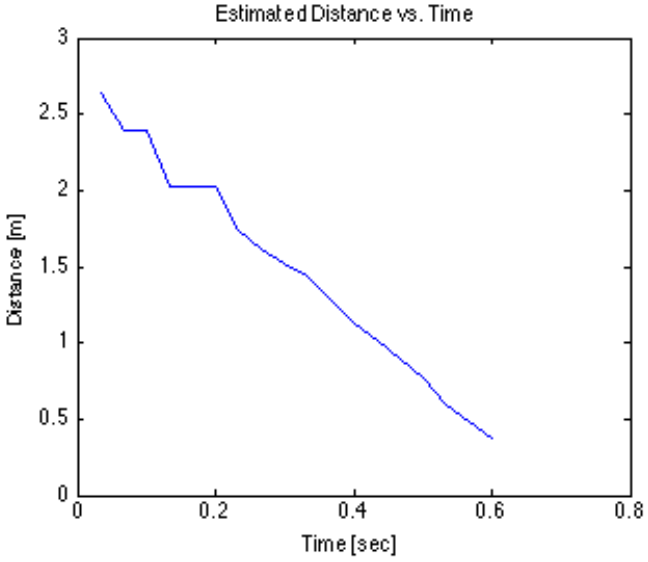





Detected Ball Trajectory in Image
480 - X^S_y [px]
X^S_x [px]
0
50
100
150
200
250
300
350
400
450
0
100
200
300
400
500
600


Apparent Ball Radius vs. Time
Radius [px]
Time [sec]
10
20
30
40
50
60
70
0
0.2
0.4
0.6
0.8


-X^S_y vs. Estimated Distance
-X^S_y [px]
Distance [m]
50
100
150
200
250
300
350
400
0
0.5
1
1.5
2
2.5
3


Estimated Distance vs. Time
Distance [m]
Time [sec]
0
0.5
1
1.5
2
2.5
3
0
0.2
0.4
0.6
0.8

Distance Estimation continued…

- Use distance estimation to find position of ball, $\vec{P}^C$

We also know $\vec{x}^C$ from:

$$\vec{x}^C = M\vec{x}^S$$

So we can solve for α to find $\vec{P}^C$:

$$\vec{x}^C = \alpha\vec{P}^C$$

$$|\vec{x}^C| = \alpha|\vec{P}^C| \quad \rightarrow \quad \alpha = \frac{|\vec{x}^C|}{|\vec{P}^C|}$$

Update dynamic model each frame, since:

$$\vec{P}^C = \vec{x}^C \cdot \frac{1}{\alpha}$$

Dynamic Model

- Assuming no rotation, ignore lift force. Use Newton's second law:

$$\vec{f} = m\vec{a} = m\frac{d\vec{V}}{dt}$$

- From our FBD, we see that:

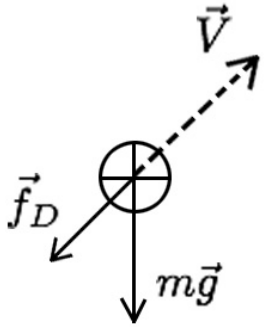
$$\vec{f} = m\vec{g} - \vec{f}_D$$

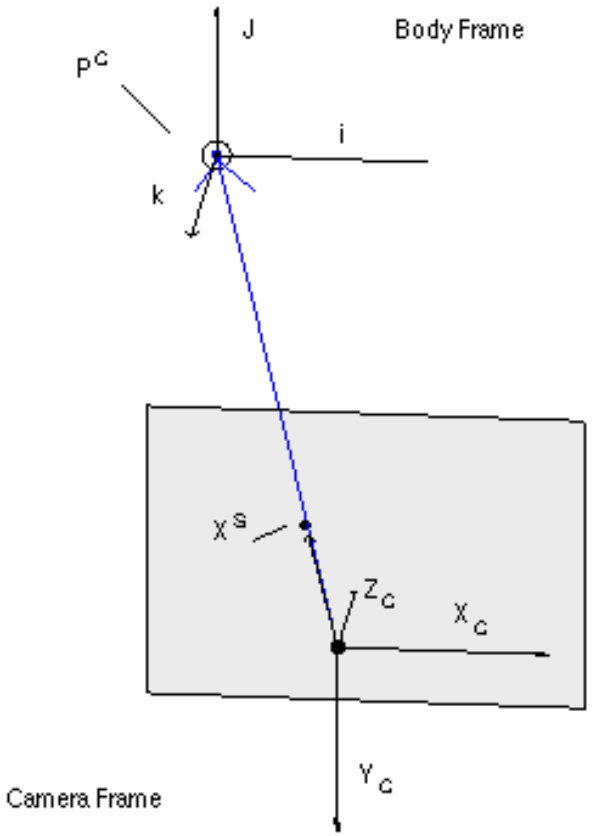
- For state space, let

$$\dot{\vec{x}} = \frac{d\vec{V}}{dt} = \vec{a}$$

$$\vec{x} = \frac{d\vec{x}}{dt} = \vec{V}$$

Free Body Diagram:





Dynamic Model continued…

- Thus we end up with

$$\ddot{\vec{x}} = \frac{1}{m}(m\vec{g} - \vec{f}_D)$$
$$= \vec{g} - \frac{1}{2m}\rho S|\dot{\vec{x}}|^2 C_D \hat{\dot{\vec{x}}}$$

Where ρ is air density, S is cross-sectional area of the ball, and C_d is the drag coefficient. Gravity vector $\vec{g}$ is positive in the camera frame.

Trajectory Prediction

Find ball velocity using previous update (camera frame):

$$\vec{V}_n \simeq \frac{\vec{P}^C_n - \vec{P}^C_{n-1}}{\Delta t}$$

Use forward Euler method to predict velocity in Δt_s seconds:

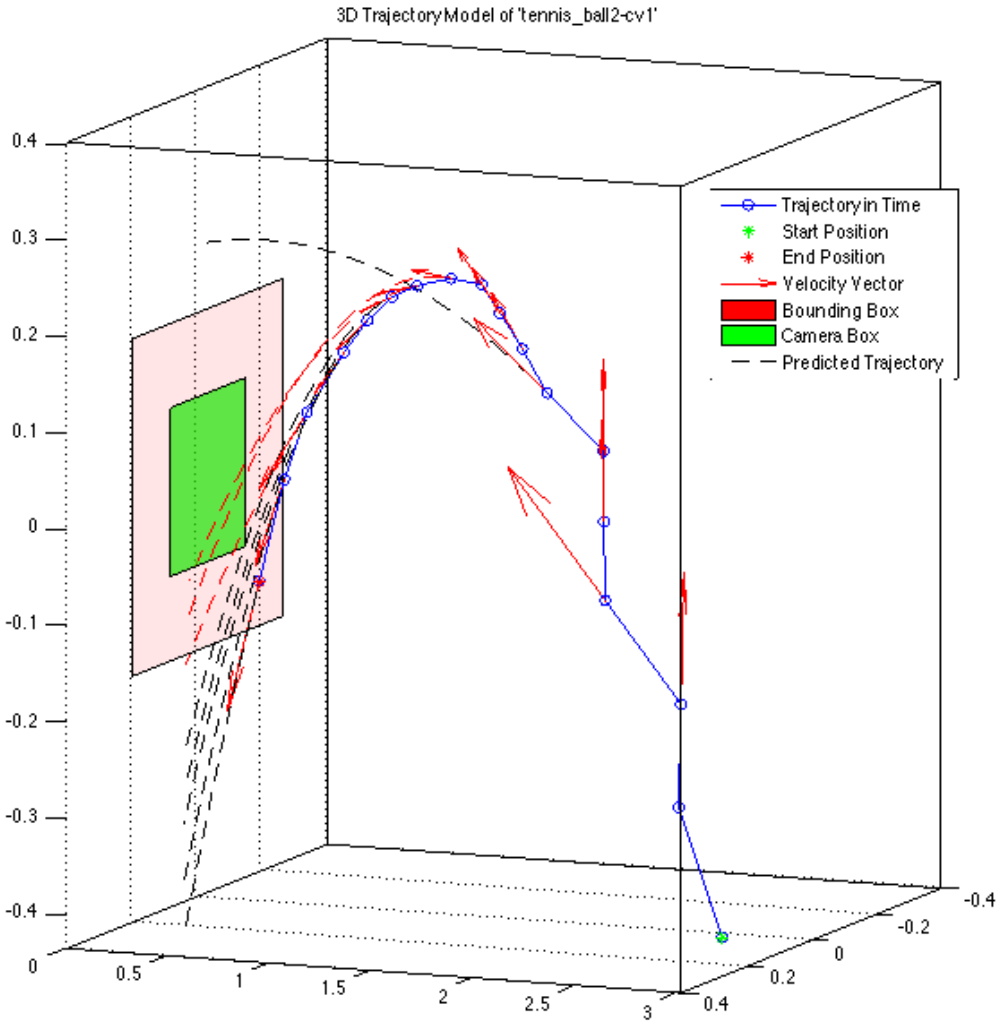
$$\hat{\vec{V}}_{n+1} \simeq \vec{V}_n + \Delta t_s \cdot \vec{\ddot{x}}_n$$

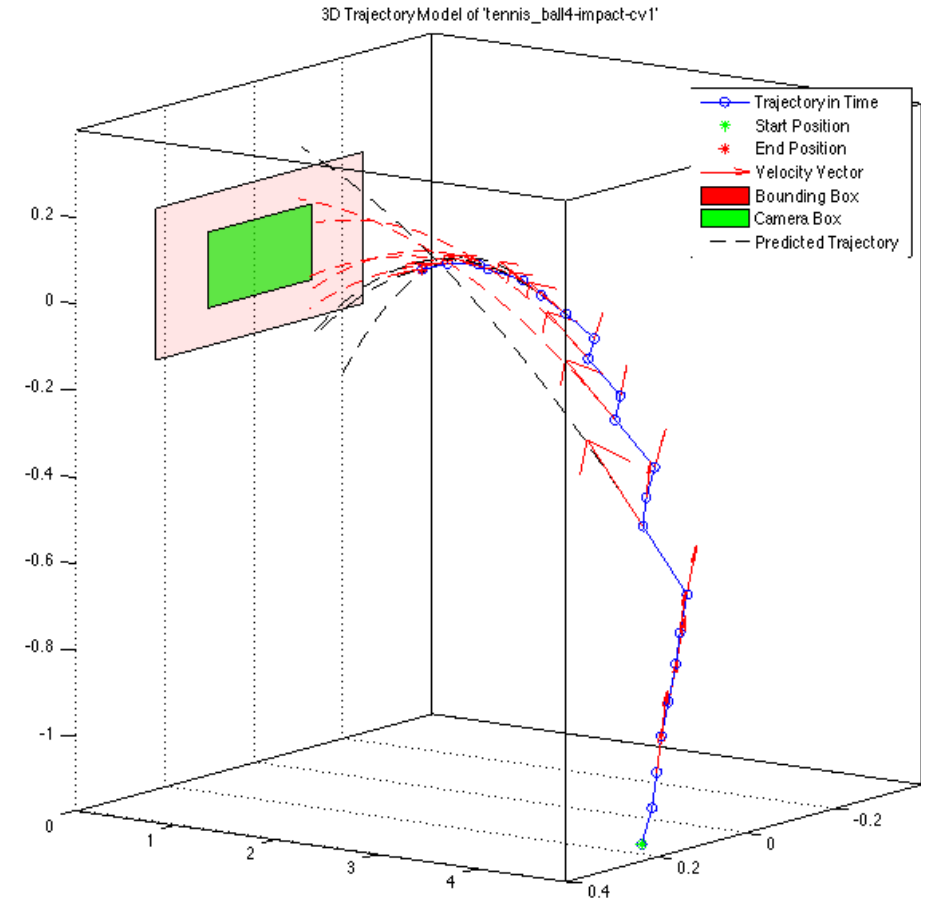
Finally, predicted position is:

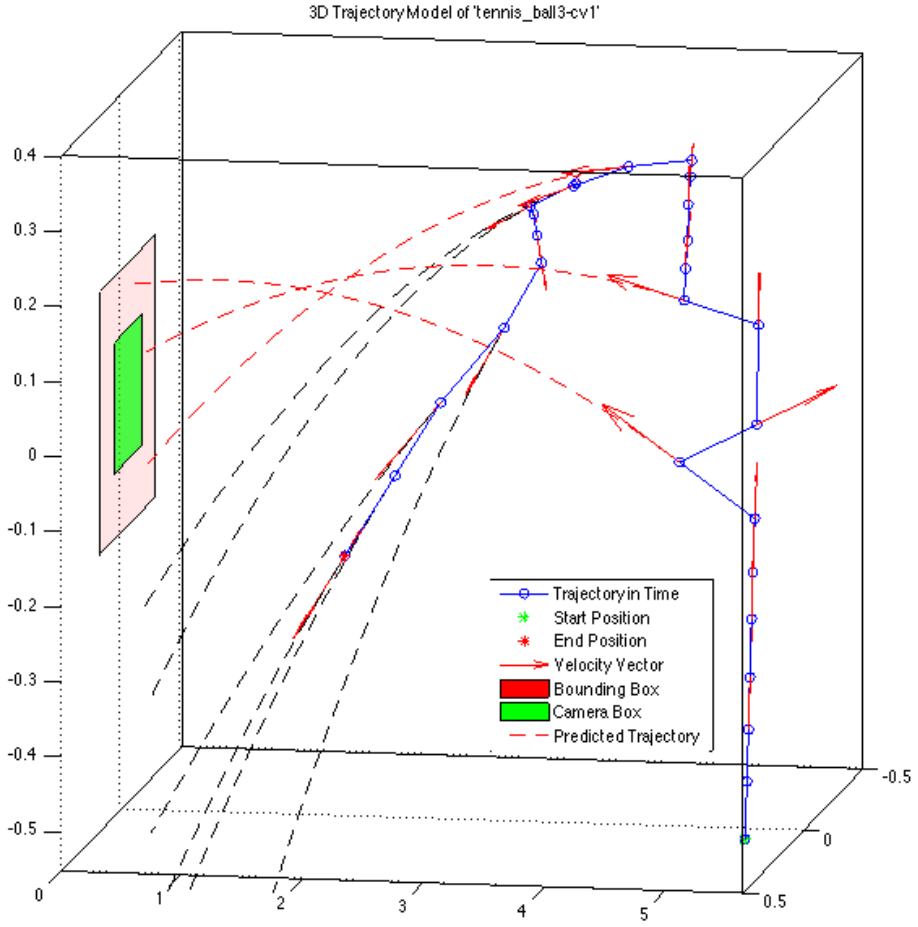
$$\hat{\vec{x}}_{n+1} \simeq \vec{x_n} + \Delta t \cdot \vec{V}_{n+1}$$

Since

$$\hat{\vec{V}}_{n+1} \simeq \frac{\hat{\vec{x}}_{n+1} - \vec{x}_n}{\Delta t}$$




3D Trajectory Model of 'tennis_ball4-impact-cv1'
Trajectory in Time
Start Position
End Position
Velocity Vector
Bounding Box
Camera Box
Predicted Trajectory


3D Trajectory Model of 'tennis_ball3-cv1'
Trajectory in Time
Start Position
End Position
Velocity Vector
Bounding Box
Camera Box
Predicted Trajectory

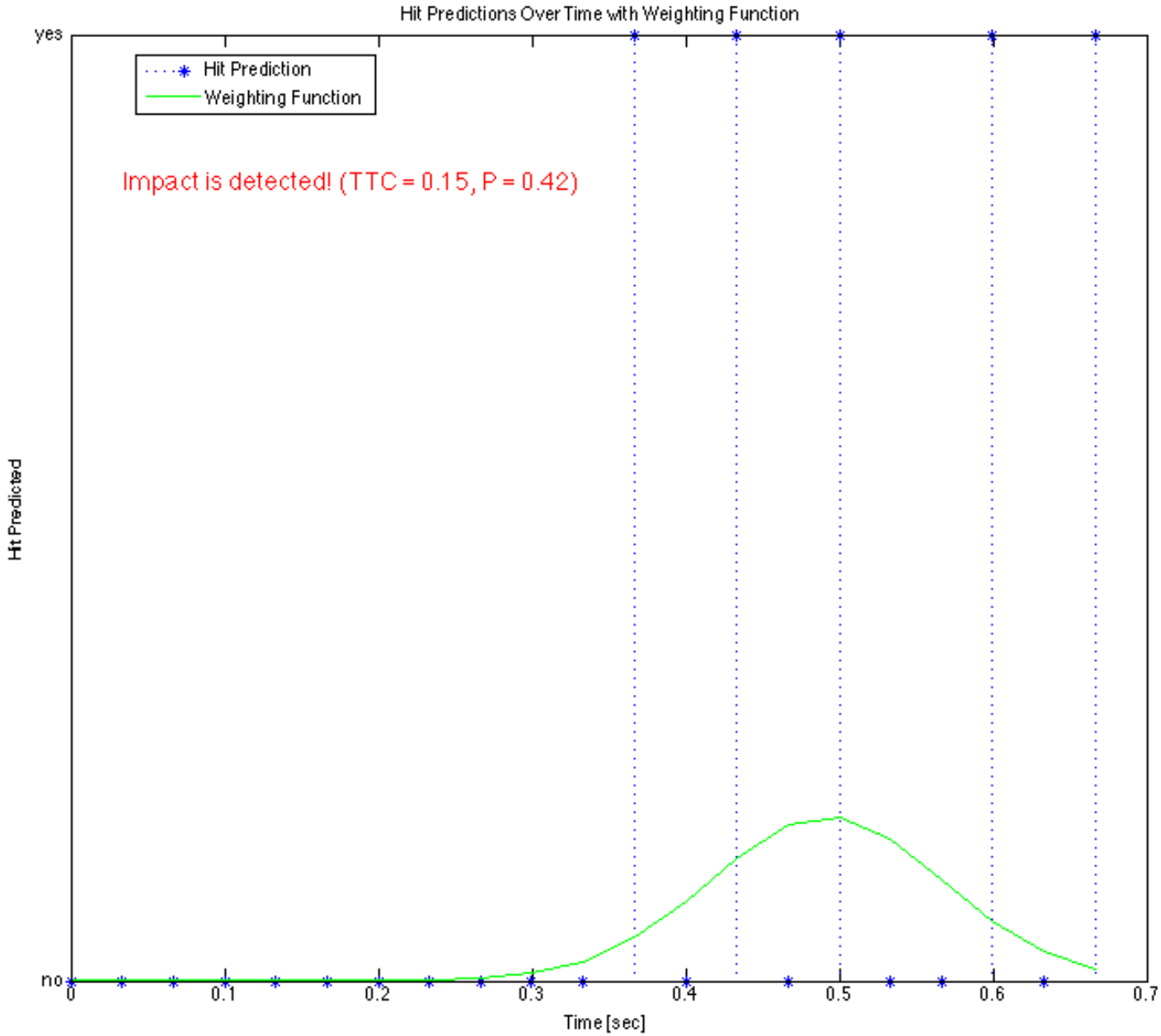
Impact Predictor

- Run trajectory prediction max N steps, or until $\hat{z}_{n+1} \leq 0$
- Declare impact if:

$$
\begin{aligned}
x_{min} \leq \hat{x}_{n+1} \leq x_{max} \\
\text{and } y_{min} \leq \hat{y}_{n+1} \leq y_{max} \\
\text{and } \hat{z}_{n+1} \leq 0 + \epsilon
\end{aligned}
$$

Where min and max x and y values define bounding box around focal plane.

- Works, but distance estimates are noisy. Need to weight predictions based on time.


Hit Predictions Over Time with Weighting Function
yes
no
Hit Predicted
Hit Prediction
Weighting Function
Impact is detected! (TTC = 0.15, P = 0.42)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Time [sec]

Future Work

- Use machine learning to train hit predictor.
- Implement Kalman filter for noisy measurements.
- Automate filter tuning/calibration process.
- Solve inverse problem (static scene, moving camera).

Dynamic Model

- .