We have currently used MATLAB to implement our tracking code as there is excellent support for hardware. We are planning to use MATLAB wrapper for BeagleBone Black for rapid deployment.

Following is the main code for video processing:

```
clear all; close all; clc;

matlab_files_path = 'C:\Users\Prasad N R\Documents\MATLAB\TraQuad';
video_path = 'E:\TraQuad\TraQuad Indiegogo campaign\video\ScreenCast\original';

filename = 'NFS1.mp4';

cd(video_path);

video_object = VideoReader(filename);
%video = read(video_object);

videoReader = vision.VideoFileReader(filename);
videoWriter = vision.VideoFileWriter('mask.avi','FrameRate',videoReader.info.VideoFrameRate);

%number_of_frames = size(video,4);

%clear video;

multiplied = zeros(video_object.Height,video_object.Width,3);
count = 1;
while ~isDone(videoReader)
   fprintf('%d',count);
   videoFrame = step(videoReader);
   videoFrame = uint8(255*videoFrame);
   maskRGB = createMaskHSV(videoFrame);
```

```
    mask = createMaskNFS(maskRGB);


    mask = uint8(mask);
    multiplied(:,:,1) = mask.*videoFrame(:,:,1);
    multiplied(:,:,2) = mask.*videoFrame(:,:,2);
    multiplied(:,:,3) = mask.*videoFrame(:,:,3);


    multiplied = uint8(multiplied);


    step(videoWriter, multiplied);
    count = count + 1;
    clc;
end


release(videoReader);
release(videoWriter);
```

We have used two color thresholding functions: One in RGB and one in HSV color spaces and chosen in series to eliminate most of the noise (color-thresholding app has been employed for the process):

The HSV thresholding function is as follows:

```
% Auto-generated by colorThresholder app on 07-Aug-2015
%-------------------------------------------------------


function RGB = createMaskHSV(RGB)


% Convert RGB image to chosen color space
I = rgb2hsv(RGB);


% Define thresholds for channel 1 based on histogram settings
```

```
channel1Min = 0.938;
channel1Max = 0.079;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.151;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.019;
channel3Max = 0.503;

% Create mask based on chosen histogram thresholds
BW = (I(:,:,1) >= channel1Min ) | (I(:,:,1) <= channel1Max ) & ...
   (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
   (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

RGB thresholding function is as follows:

```
% Auto-generated by colorThresholder app on 06-Aug-2015
%-------------------------------------------------------

function BW = createMaskNFS(RGB)

% Convert RGB image to chosen color space
I = RGB;
```

```
% Define thresholds for channel 1 based on histogram settings
channel1Min = 87.000;
channel1Max = 181.000;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 8.000;
channel2Max = 64.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 4.000;
channel3Max = 60.000;

% Create mask based on chosen histogram thresholds
BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
```

The final motion tracking code is as follows:

```
%% Motion-Based Multiple Object Tracking
% This example shows how to perform automatic detection and motion-based
% tracking of moving objects in a video from a stationary camera.
%
% Copyright 2012 The MathWorks, Inc.

%%
% Detection of moving objects and motion-based tracking are important
% components of many computer vision applications, including activity
% recognition, traffic monitoring, and automotive safety.  The problem of
% motion-based object tracking can be divided into two parts:
```

```
%
% # detecting moving objects in each frame
% # associating the detections corresponding to the same object over time
%
% The detection of moving objects uses a background subtraction algorithm
% based on Gaussian mixture models. Morphological operations are applied to
% the resulting foreground mask to eliminate noise. Finally, blob analysis
% detects groups of connected pixels, which are likely to correspond to
% moving objects.
%
% The association of detections to the same object is based solely on
% motion. The motion of each track is estimated by a Kalman filter. The
% filter is used to predict the track's location in each frame, and
% determine the likelihood of each detection being assigned to each
% track.
%
% Track maintenance becomes an important aspect of this example. In any
% given frame, some detections may be assigned to tracks, while other
% detections and tracks may remain unassigned.The assigned tracks are
% updated using the corresponding detections. The unassigned tracks are
% marked invisible. An unassigned detection begins a new track.
%
% Each track keeps count of the number of consecutive frames, where it
% remained unassigned. If the count exceeds a specified threshold, the
% example assumes that the object left the field of view and it deletes the
% track.
%
% This example is a function with the main body at the top and helper
% routines in the form of
%
<matlab:helpview(fullfile(docroot,'toolbox','matlab','matlab_prog','matlab_prog.map'),'nested_functions') nested functions>
```

```
% below.

function multiObjectTracking()

% Create system objects used for reading video, detecting moving objects,
% and displaying the results.
obj = setupSystemObjects();

tracks = initializeTracks(); % Create an empty array of tracks.

nextId = 1; % ID of the next track
global file_path;
global real_file_name;

file_path = 'E:\TraQuad\TraQuad Indiegogo campaign\video\ScreenCast\640_360';
real_file_name = 'NFS1motion.avi';

real_video_object = VideoReader(real_file_name);

videoReader = vision.VideoFileReader('NFS1.avi');
videoWriter = vision.VideoFileWriter('NFS1final.avi','FrameRate',videoReader.info.VideoFrameRate);
% Detect moving objects, and track them across video frames.
while ~isDone(obj.reader)
  frame = readFrame();
  realFrame = step(videoReader);
  [centroids, bboxes, mask] = detectObjects(frame);
  predictNewLocationsOfTracks();
  [assignments, unassignedTracks, unassignedDetections] = ...
    detectionToTrackAssignment();

  updateAssignedTracks();
```

```
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();

    displayTrackingResults();
end


%% Create System Objects
% Create System objects used for reading the video frames, detecting
% foreground objects, and displaying results.

    function obj = setupSystemObjects()
        %global file_path;
        %global real_file_name;
        % Initialize Video I/O
        % Create objects for reading a video from a file, drawing the tracked
        % objects in each frame, and playing the video.
        %cd(file_path);
        cd('E:\TraQuad\TraQuad Indiegogo campaign\video\ScreenCast\640_360');
        % Create a video file reader.
        %obj.reader = vision.VideoFileReader(real_file_name);
        obj.reader = vision.VideoFileReader('NFS1motion.avi');

        % Create two video players, one to display the video,
        % and one to display the foreground mask.
        obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700, 400]);
        obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700, 400]);

        % Create system objects for foreground detection and blob analysis
```

```
        % The foreground detector is used to segment moving objects from
        % the background. It outputs a binary mask, where the pixel value
        % of 1 corresponds to the foreground and the value of 0 corresponds
        % to the background.

        obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
            'NumTrainingFrames', 40, 'MinimumBackgroundRatio', 0.7);

        % Connected groups of foreground pixels are likely to correspond to moving
        % objects.  The blob analysis system object is used to find such groups
        % (called 'blobs' or 'connected components'), and compute their
        % characteristics, such as area, centroid, and the bounding box.

        obj.blobAnalyser = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
            'AreaOutputPort', true, 'CentroidOutputPort', true, ...
            'MinimumBlobArea', 50);
    end

%% Initialize Tracks
% The |initializeTracks| function creates an array of tracks, where each
% track is a structure representing a moving object in the video. The
% purpose of the structure is to maintain the state of a tracked object.
% The state consists of information used for detection to track assignment,
% track termination, and display.
%
% The structure contains the following fields:
%
% * |id| :                  the integer ID of the track
% * |bbox| :                the current bounding box of the object; used
%                           for display
% * |kalmanFilter| :        a Kalman filter object used for motion-based
```

```
%                    tracking
% * |age| :               the number of frames since the track was first
%                    detected
% * |totalVisibleCount| :   the total number of frames in which the track
%                    was detected (visible)
% * |consecutiveInvisibleCount| : the number of consecutive frames for
%                         which the track was not detected (invisible).
%
% Noisy detections tend to result in short-lived tracks. For this reason,
% the example only displays an object after it was tracked for some number
% of frames. This happens when |totalVisibleCount| exceeds a specified
% threshold.
%
% When no detections are associated with a track for several consecutive
% frames, the example assumes that the object has left the field of view
% and deletes the track. This happens when |consecutiveInvisibleCount|
% exceeds a specified threshold. A track may also get deleted as noise if
% it was tracked for a short time, and marked invisible for most of the of
% the frames.

  function tracks = initializeTracks()
    % create an empty array of tracks
    tracks = struct(...
      'id', {}, ...
      'bbox', {}, ...
      'kalmanFilter', {}, ...
      'age', {}, ...
      'totalVisibleCount', {}, ...
      'consecutiveInvisibleCount', {});
  end
```

```
%% Read a Video Frame
% Read the next video frame from the video file.
  function frame = readFrame()
    frame = obj.reader.step();
  end

%% Detect Objects
% The |detectObjects| function returns the centroids and the bounding boxes
% of the detected objects. It also returns the binary mask, which has the
% same size as the input frame. Pixels with a value of 1 correspond to the
% foreground, and pixels with a value of 0 correspond to the background.
%
% The function performs motion segmentation using the foreground detector.
% It then performs morphological operations on the resulting binary mask to
% remove noisy pixels and to fill the holes in the remaining blobs.

  function [centroids, bboxes, mask] = detectObjects(frame)

    % Detect foreground.
    mask = obj.detector.step(frame);

    % Apply morphological operations to remove noise and fill in holes.
    mask = imopen(mask, strel('rectangle', [3,3]));
    mask = imclose(mask, strel('rectangle', [15, 15]));
    mask = imfill(mask, 'holes');

    % Perform blob analysis to find connected components.
    [~, centroids, bboxes] = obj.blobAnalyser.step(mask);
  end

%% Predict New Locations of Existing Tracks
```

```
% Use the Kalman filter to predict the centroid of each track in the
% current frame, and update its bounding box accordingly.

    function predictNewLocationsOfTracks()
        for i = 1:length(tracks)
            bbox = tracks(i).bbox;

            % Predict the current location of the track.
            predictedCentroid = predict(tracks(i).kalmanFilter);

            % Shift the bounding box so that its center is at
            % the predicted location.
            predictedCentroid = int32(predictedCentroid) - bbox(3:4) / 2;
            tracks(i).bbox = [predictedCentroid, bbox(3:4)];
        end
    end

%% Assign Detections to Tracks
% Assigning object detections in the current frame to existing tracks is
% done by minimizing cost. The cost is defined as the negative
% log-likelihood of a detection corresponding to a track.
%
% The algorithm involves two steps:
%
% Step 1: Compute the cost of assigning every detection to each track using
% the |distance| method of the |vision.KalmanFilter| System object. The
% cost takes into account the Euclidean distance between the predicted
% centroid of the track and the centroid of the detection. It also includes
% the confidence of the prediction, which is maintained by the Kalman
% filter. The results are stored in an MxN matrix, where M is the number of
% tracks, and N is the number of detections.
```

```
%
% Step 2: Solve the assignment problem represented by the cost matrix using
% the |assignDetectionsToTracks| function. The function takes the cost
% matrix and the cost of not assigning any detections to a track.
%
% The value for the cost of not assigning a detection to a track depends on
% the range of values returned by the |distance| method of the
% |vision.KalmanFilter|. This value must be tuned experimentally. Setting
% it too low increases the likelihood of creating a new track, and may
% result in track fragmentation. Setting it too high may result in a single
% track corresponding to a series of separate moving objects.
%
% The |assignDetectionsToTracks| function uses the Munkres' version of the
% Hungarian algorithm to compute an assignment which minimizes the total
% cost. It returns an M x 2 matrix containing the corresponding indices of
% assigned tracks and detections in its two columns. It also returns the
% indices of tracks and detections that remained unassigned.

 function [assignments, unassignedTracks, unassignedDetections] = ...
     detectionToTrackAssignment()

   nTracks = length(tracks);
   nDetections = size(centroids, 1);

   % Compute the cost of assigning each detection to each track.
   cost = zeros(nTracks, nDetections);
   for i = 1:nTracks
     cost(i, :) = distance(tracks(i).kalmanFilter, centroids);
   end

   % Solve the assignment problem.
```

```
        costOfNonAssignment = 20;
        [assignments, unassignedTracks, unassignedDetections] = ...
            assignDetectionsToTracks(cost, costOfNonAssignment);
    end

%% Update Assigned Tracks
% The |updateAssignedTracks| function updates each assigned track with the
% corresponding detection. It calls the |correct| method of
% |vision.KalmanFilter| to correct the location estimate. Next, it stores
% the new bounding box, and increases the age of the track and the total
% visible count by 1. Finally, the function sets the invisible count to 0.

    function updateAssignedTracks()
        numAssignedTracks = size(assignments, 1);
        for i = 1:numAssignedTracks
            trackIdx = assignments(i, 1);
            detectionIdx = assignments(i, 2);
            centroid = centroids(detectionIdx, :);
            bbox = bboxes(detectionIdx, :);

            % Correct the estimate of the object's location
            % using the new detection.
            correct(tracks(trackIdx).kalmanFilter, centroid);

            % Replace predicted bounding box with detected
            % bounding box.
            tracks(trackIdx).bbox = bbox;

            % Update track's age.
            tracks(trackIdx).age = tracks(trackIdx).age + 1;
```

```
        % Update visibility.
        tracks(trackIdx).totalVisibleCount = ...
            tracks(trackIdx).totalVisibleCount + 1;
        tracks(trackIdx).consecutiveInvisibleCount = 0;
    end
end

%% Update Unassigned Tracks
% Mark each unassigned track as invisible, and increase its age by 1.

function updateUnassignedTracks()
    for i = 1:length(unassignedTracks)
        ind = unassignedTracks(i);
        tracks(ind).age = tracks(ind).age + 1;
        tracks(ind).consecutiveInvisibleCount = ...
            tracks(ind).consecutiveInvisibleCount + 1;
    end
end

%% Delete Lost Tracks
% The |deleteLostTracks| function deletes tracks that have been invisible
% for too many consecutive frames. It also deletes recently created tracks
% that have been invisible for too many frames overall.

function deleteLostTracks()
    if isempty(tracks)
        return;
    end

    invisibleForTooLong = 3;
    ageThreshold = 10;
```

```
        % Compute the fraction of the track's age for which it was visible.
        ages = [tracks(:).age];
        totalVisibleCounts = [tracks(:).totalVisibleCount];
        visibility = totalVisibleCounts ./ ages;

        % Find the indices of 'lost' tracks.
        lostInds = (ages < ageThreshold & visibility < 0.2) | ...
            [tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;

        % Delete lost tracks.
        tracks = tracks(~lostInds);
    end

%% Create New Tracks
% Create new tracks from unassigned detections. Assume that any unassigned
% detection is a start of a new track. In practice, you can use other cues
% to eliminate noisy detections, such as size, location, or appearance.

    function createNewTracks()
        centroids = centroids(unassignedDetections, :);
        bboxes = bboxes(unassignedDetections, :);

        for i = 1:size(centroids, 1)

            centroid = centroids(i,:);
            bbox = bboxes(i, :);

            % Create a Kalman filter object.
            kalmanFilter = configureKalmanFilter('ConstantVelocity', ...
                centroid, [75, 150], [50, 100], 100);
```

```
            % Create a new track.
            newTrack = struct(...
                'id', nextId, ...
                'bbox', bbox, ...
                'kalmanFilter', kalmanFilter, ...
                'age', 1, ...
                'totalVisibleCount', 1, ...
                'consecutiveInvisibleCount', 0);

            % Add it to the array of tracks.
            tracks(end + 1) = newTrack;

            % Increment the next id.
            nextId = nextId + 1;
        end
    end

%% Display Tracking Results
% The |displayTrackingResults| function draws a bounding box and label ID
% for each track on the video frame and the foreground mask. It then
% displays the frame and the mask in their respective video players.

    function displayTrackingResults()
        % Convert the frame and the mask to uint8 RGB.
        frame = im2uint8(frame);
        mask = uint8(repmat(mask, [1, 1, 3])) .* 255;

        minVisibleCount = 30;
        if ~isempty(tracks)
```

```
% Noisy detections tend to result in short-lived tracks.
% Only display tracks that have been visible for more than
% a minimum number of frames.
reliableTrackInds = ...
    [tracks(:).totalVisibleCount] > minVisibleCount;
reliableTracks = tracks(reliableTrackInds);

% Display the objects. If an object has not been detected
% in this frame, display its predicted bounding box.
if ~isempty(reliableTracks)
    % Get bounding boxes.
    bboxes = cat(1, reliableTracks.bbox);

    % Get ids.
    ids = int32([reliableTracks(:).id]);

    % Create labels for objects indicating the ones for
    % which we display the predicted rather than the actual
    % location.
    labels = cellstr(int2str(ids'));
    predictedTrackInds = ...
        [reliableTracks(:).consecutiveInvisibleCount] > 0;
    isPredicted = cell(size(labels));
    isPredicted(predictedTrackInds) = {' predicted'};
    labels = strcat(labels, isPredicted);

    realFrame = insertObjectAnnotation(realFrame, 'rectangle', ...
        bboxes, labels);

    % Draw the objects on the frame.
    %frame = insertObjectAnnotation(frame, 'rectangle', ...
```

```
            %    bboxes, labels);

            % Draw the objects on the mask.
            %mask = insertObjectAnnotation(mask, 'rectangle', ...
            %    bboxes, labels);
        end
    end

    % Display the mask and the frame.
    %obj.maskPlayer.step(mask);
    %obj.videoPlayer.step(frame);
    %obj.videoPlayer.step(realFrame);
    step(videoWriter, realFrame);
  end
release(videoReader);
release(videoWriter);
%% Summary
% This example created a motion-based system for detecting and
% tracking multiple moving objects. Try using a different video to see if
% you are able to detect and track objects. Try modifying the parameters
% for the detection, assignment, and deletion steps.
%
% The tracking in this example was solely based on motion with the
% assumption that all objects move in a straight line with constant speed.
% When the motion of an object significantly deviates from this model, the
% example may produce tracking errors. Notice the mistake in tracking the
% person labeled #12, when he is occluded by the tree.
%
% The likelihood of tracking errors can be reduced by using a more complex
% motion model, such as constant acceleration, or by using multiple Kalman
% filters for every object. Also, you can incorporate other cues for
```

```
% associating detections over time, such as size, shape, and color.

displayEndOfDemoMessage(mfilename)
end
```

There are many improvements still needed which we are planning to develop over time.