Introduction to Vision and Robotics
Coursework 1

Report

Lubomir Vikev (1117764)
Lukas Dirzys (1119520)

Organisation

We have organised well. We both met probably more than 5 times to write the code for this assignment. At the beginning, we have started sharing the ideas about how we might accomplish every separate task, what techniques we might use and how we are going to separate the task. We have started writing some examples together and once we got the feeling that things can be separated, we created GIT repository and started working separately (note, however, we still were sitting near each other, discussing different approaches, helping each other). More specifically, one of us (Lubomir) was developing an algorithm to detect the objects more accurately and algorithm to track separate objects at the same time, while the other one (Lukas) was trying to detect the ball and the highest point of the object as well as drawing objects paths, bounding boxes, highest points.
Since, we both worked a lot and separated the work equally we are spreading the mark equally as well (50:50).

Introduction

The main method we used for extracting foreground is background subtraction. Once we have foreground we smooth and filter the image to reduce the noise, separate the different objects by detecting the regions of the objects in the foreground and track them across the frames. We then (in every frame) check whether the object is a ball or not by using compactness, eccentricity and area measures of each of the objects, draw the path of the object by keeping the list of object centres at every frame they appear in and, finally, find and draw the highest point of the ball by using the velocity (distance moved per frame) and sine of the angle between ball moving direction and horizontal line.

Methods

Background removal:

We began our program by calculating a simple mean value of the first 25 (learning size) frames. Once we got the mean we subtracted it from each of the following frames and compared the absolute difference to a pre-set threshold which after a few trials was fixed to 8. From this difference we formed a black and white (binary) image which is black for background and white for foreground. However there was too much noise so we applied some changes to the binary image – erosion, closure and median filtering. This removed small isolated groups of pixels causing noise and misdetection. By applying closure we filled potential gaps in the objects misidentified as background. This turned out to work quite well until we noticed that an object which hits the background drastically changes the RGB values of the background. We fixed that by adding an algorithm which updates the background value not only during the first frames while the learning is done but it updates it whenever there is no moving object present as well. The check for no moving objects/no foreground is done by comparing the current frame with the value of a frame observed previously similarly to the background subtraction (there is an input parameter when calling the algorithm which sets how far back should the algorithm look – 0 stands for no look back/no background value update and any natural number represents the number of frames should be looked back). This however introduced potential problem discussed later.

Object detection:

For every binary image created by the background subtraction the algorithm labels all groups of pixels as different objects and extracts their properties - the object’s centroid, area, perimeter and eccentricity. Once it has all objects it goes through them and remove the ones which have area less than the specified in the calling parameter as they are considered to be noise. Moreover, it is not detecting new objects when they are found anywhere further than 20 pixels from the frame edges. That helps to avoid noise and at the same time does not miss the objects thrown, because they are always coming from one of the sides and are passing this 20 pixel distance in which they are assigned to an object.

Updating objects:

To keep track of the objects we have created the structure that contains information about every object occurred now and in the previous frames. On each frame when the objects are detected this structure is scanned if it already contains the current object by checking the difference between the current object’s centre and the last centre from the path of the object in the structure. If those are close enough (found the best value for this being less than 20) one more check is being done – if the normalized RGB colour of the centre is close (threshold = 5) to the colour stored in the structure then the object is one of the objects already recognized. In that case, the path for this object is extended by adding the current centre of the object and updating the number of the frame at which the object was seen last time (current frame). Then we are checking whether the object was already recognized as a ball previously (keeping that entry in our structure) and if not we then check if this object is now recognized as a ball and depending on the answer we update the object (whether it is a ball or not). If, however, no match in the current set of objects was found - a new object entry is created and stored in the structure.

Recognising spherical balls:

Once we got the current and potentially some past information about the object, the algorithm then tries to detect whether the object is a ball or not. Note, however, that in order for an object to be surely recognized as a ball it must be recognized as a ball in at least 5 frames sometime in the past. Actually, it is going to be exactly 5, because after 5 successful detections we are no longer checking if that object is still a ball. That makes our algorithm to be more certain about the objects’ nature and it avoids outliers (without this check, the algorithm still worked pretty well, however, misclassified the rugby ball and cube as a ball once, because of the view point) and it also does not reduce the ability to recognize balls, because obviously, balls are, in fact, balls in the majority of the frames. Also, we are assuming that the object cannot be a ball if it already is moving down. We have made this assumption, because our main task is to recognize the ball at its highest point, therefore, if the object is no longer moving up and it still has not been recognized then there is no point to find whether that is a ball or not if we have already missed the highest point anyway.
Finally, when we are checking if the object is a ball, we are using 3 main measures of the object: area, perimeter and eccentricity (which is actually the ratio of the distance between the centre of ellipse with the same second-moments as the object and its major axis length). We are mainly using area and perimeter to only calculate the compactness of the object. When detecting a ball we are looking for compactness to be as close to one as possible since that is the similarity to circle measure. Based on trials we found that every ball sooner or later has the value of more than 0.94. We are then checking for the eccentricity, since that is the measure of how much the object deviates from being circular. Perfectly, it should be 0, however, trials showed that it might go up to 0.75 for a ball, therefore, we are checking for the eccentricity to be less than this value. That was still not enough, so we have also spotted that the summation of compactness and eccentricity

measures for the ball mostly lies between 1.6 and 1.7 (~0.94 + 0.75) while for other object this measure is never like that (because of usually one of the measures being much higher/lower than the threshold.
This task was one of the most challenging one. We have tried a lot of other different techniques (corners in the object, convexity, solidity and others), but that, however, didn't work well. This is, as I said, really complicated task, especially recognizing between spherical and rugby ball or always spinning cube. Convexity was absolutely useless, however, we were hoping that at least corner properties are going to separate the objects better, however, that didn't work well either, probably because of the balls still having some corners after background subtraction and image cleaning. At the end, we have had proved that simple things usually work best.

Removing objects:

After updating objects, the structure is scanned again - this time for entries in the structure which have not been updated for a few frames (chose 10 frames). If such entries have been found they are removed as they (most probably) have disappeared (and maybe left some noise after that needs to be removed).

Drawing information and showing the highest point:

Finally, before we move to the next frame, we present the information we have about the objects on the frame. For testing purposes, we have showed 3 images (original frame with information drawn, foreground and the difference between the current frame and the chosen past frame) rather than only 1 original image. That helped us to see better the actual effect of the foreground to the decisions made.
For each object we are drawing red bounding box and the path so far if the object is not a ball and blue otherwise (That helps to easily see when the object is recognised as a ball or when the track breaks and changes the decision). We are drawing the path simply by drawing separate lines between every two consecutive object's centres in the path array. Finally, if the object is a ball and the highest point for this ball has not been found yet, we are checking if the ball in this current frame is at its highest point and if yes – we draw a red cross and pause the video for 3 seconds after drawing information about all objects first (so that if two objects are in the video at the same frame and the highest point is found – no paths will disappear for these 3 seconds).
For checking if the ball is at its highest point we are only using previous path (with the current centre of the object). We know that ball's velocity at its highest point is nearly 0 (it hits 0 at some point anyway). First approach to this was by checking when the velocity in the last and current frames is less than some threshold value (picked 1). It did not work well,

because sometimes even though the ball is at its highest point the velocity is still much bigger than set threshold, just because of huge horizontal velocity (imagine if the ball is thrown in angle less than 45 degrees). Therefore, we have tried to set the thresholds for both horizontal and vertical velocities separately. Unfortunately, that did not work fine either. At the end, we have come up with an idea to check only vertical velocity, but also include the sine value between the direction of the ball and horizontal line. Now, it was much easier to find the right thresholds that works – the sine and the vertical velocity at the highest point will both be very close to 0. That worked much better, however, we were still recognising as the highest point the point occurred just after the real highest point. We increased the maximum sine threshold to be 0.1 and to make sure that the highest point is not predicted too early, we are now checking vertical velocity and sine between direction and horizontal line for the current and the last frame. Now it worked even better for the balls that are thrown with the lower angle. However, we still were not able to found the thresholds that might recognize the ball’s highest point 100% accurately. However, without using any future frames, we were able to predict this point accurately for some balls and had one point difference (less than a pixel) otherwise.

Results

- Results on training videos
  - Video 1 with 150 pixels minimum area and 6 frames look back:

Instance.	Is ball?	Found highest point	Real highest point	Recognized correctly?	Drawn path correctly?
1.	yes	211.5	210.2	yes	yes
2.	yes	167.8	167.2	yes	yes
3.	yes	208.9	208.1	yes	yes
4.	no	-	-	yes	yes
5.	no	-	-	yes	yes
6.	no	-	-	yes	yes
7.	yes	280.7	280.4	yes	yes
8.	yes	51.7	51.7	yes	yes
9.	yes	196.0	195.8	yes	yes

  - Video 2 with 150 pixels minimum area and 6 frames look back:

Instance.	Is ball?	Found highest point	Real highest point	Recognized correctly?	Drawn path correctly?
1.	yes	279.8	278.9	yes	yes
2.	yes	204.2	203.7	yes	yes
3.	no	-	-	yes	yes
4.	no	-	-	yes	yes
5.	no	-	-	yes	yes
6.	no	-	-	yes	yes
7.	yes	271.1	271.1	yes	yes

8.	yes	198.6	198.6	yes	yes

- Results on test videos
  - Video 3 with 150 pixels minimum area and 6 frames look back:

Instance.	Is ball?	Found highest point	Real highest point	Recognized correctly?	Drawn path correctly?
1.	yes	220.1	219.9	yes	yes
2.	yes	259.5	259.4	yes	yes
3.	yes	183.1	182.9	yes	yes
4.	yes	223.9	223.5	yes	yes
5.	no	-	-	yes	yes
6.	no	-	-	yes	yes
7.	no	-	-	yes	yes
8.	no	-	-	yes	yes
9.	yes	223.1	223.1	yes	yes
10.	yes	214.3	214.1	yes	yes

  - Video 4 with 150 pixels minimum area and 6 frames look back:

Instance.	Is ball?	Found highest point	Real highest point	Recognized correctly?	Drawn path correctly?
1.	no	-	-	yes	yes
2.	yes	249.9	249.8	yes	yes
3.	yes	177.8	177.8	yes	yes
4.	yes	123.7	123.6	yes	yes
5.	no	-	-	yes	yes
6.	no	-	-	yes	yes

7.	yes	228.2	228.2	yes	yes
8.	yes	148.2	147.9	yes	yes
9.	yes	257.1	256.8	yes	yes
10.	yes	158.6	158.6	yes	yes
11.	yes	140.3	140.1	yes	yes

- <u>Success / Fail ratio</u>
  - Correct/missed/invalid object detections:

    We have managed to detect 100% of the moving objects. However, there were some problems if the background is hit by the moving object (for example, kangaroo in the first video). This noise is then considered as a new object (or even few smaller ones). For obtaining the results shown above, we have used couple of techniques to avoid such noise - not considering object with area less than some threshold(150 in the current test), removing object if it has not moved during the past few frames and not detecting new object anywhere further than 20 pixels from the frame edges.
  - Correct/broken/incorrect tracks:

    We managed to draw all paths 100% accurately.
  - Correct/incorrect highest point:

    As we can see from the results in the tables the difference between our drawn and the real highest points is never bigger than 1 pixel (that is only one frame too late or too early), for 3 out of 10 training ball instances and for 4 out of 14 test ball instances we are able to identify the highest point correctly. Therefore, in ~30% of cases we are able to find the highest point correctly and in other 70% we finding the second highest point.
  - Correct/incorrect ball detections:

    We were able to separate balls from other objects 100% of the time while we had the same accuracy for not recognising other object as a ball.
- <u>Possible failures due to different input parameters when calling the function</u>

  The algorithm is highly sensitive to the input parameters. Different parameters lead to different results:

- Minimum area - if the minimum area is too small noise appears and it is possible to break the track of the object. However a value of 150 works fine as long as there are no too small objects in the video.
- Look back - potential problem with this is if the value is too low and the object is thrown vertically it starts to consider the object as part of the background when it reaches its highest point as it stays still in a few frames. However if the value is chosen carefully it updates the background correctly and removes the noise cause by collisions of objects into the background. If that option is disabled (input 0) the parts of the background that were changed would be tracked as an object until the rest of the video.

For further testing we created our own video where we rolled different objects on the ground (we couldn't get clear images if we were throwing them in the air) and our program was able to identify correctly all balls. However due to shadows and other rapid changes in the illumination it was detecting parts of the background as well.

Discussion

Overall our program manages to detect the objects and keep track of them across the different frames. However there are some limitations and drawbacks. The video speed is significantly decreased by the amount of calculations done so the fps has dropped. Another possible problem is that if one object disappears and a new one appears immediately in the frame close to the centre of the previous one it might get assigned to the old track instead of a new track being created for the object. Another possible problem is if a multicolour object is tracked its track might be lost due to different centre colour as the object rotates. The last two problems can be solved by improving the assignment algorithm so that it not just checks the colour of the centre but the mean colour of the all object's pixels for example. Shape and other properties also might be included along with the colour. However that will introduce more calculations and slow down the speed even further. The highest point detection algorithm might as well be improved, however, we have tried a lot of different approaches for this and nothing worked perfectly for all cases. Moreover, it would probably overfit the data if we find the exact values and thresholds.

Code

```
function track(FILE_DIR,LEARN_SIZE,MIN_AREA,LOOK_BACK,VIEW)

BALL_FOR_SURE = 5;

if MIN_AREA < 50
    MIN_AREA = 50;
end

filenames = dir([FILE_DIR '*.jpg']);

frame = imread([FILE_DIR filenames(1).name]);
figure(1);

% dimensions of the frame
y = size(frame, 1);
x = size(frame, 2);

objects = struct('path', {}, 'lastSeen', {}, 'colour', {}, 'highest', 0, 'box', {}, 'isBall', {}, 'ballCount', 0);

backgroundSum = double(zeros(y, x, 3));
foreground = zeros(y, x);
diff = zeros(y, x);
frameD = double(frame);

frameBuff(:, :, :, 1) = frameD;
if LOOK_BACK > 0
    frameBuff(:, :, :, LOOK_BACK) = frameD;
end

% play 'video'
for k = 1 : size(filenames, 1)
    frame = imread([FILE_DIR filenames(k).name]);
    frameD = double(frame);
    % get the sum of the rgb values of the first 25 (LEARN_SIZE) frames
    if k <= LEARN_SIZE
        backgroundSum = frameD + backgroundSum;
    end
    if k == LEARN_SIZE
        background = backgroundSum/LEARN_SIZE;
    end
    % show frame
    if VIEW == 0
        imshow(frame);
    else
        d=diff*255;

        f=foreground*255;
        bwims = cat(2, cat(3,f,f,f), cat(3,d,d,d));
        imshow(cat(2,frame,bwims));
    end
    if k>LEARN_SIZE
        props = extractForegroundObjects();
        centres = cat(1, props.Centroid);
        updateObjectsStruct();
        removeLostObjects();
        drawInfo();

    end

    if k > LEARN_SIZE && LOOK_BACK > 0
        diff=zeros(y, x);
        diff(abs(frameD(:, :, 1) - frameBuff(:, :, 1, LOOK_BACK)) > 5) = 1;
        diff(abs(frameD(:, :, 2) - frameBuff(:, :, 2, LOOK_BACK)) > 5) = 1;
        diff(abs(frameD(:, :, 3) - frameBuff(:, :, 3, LOOK_BACK)) > 5) = 1;
        diff = bwmorph(diff, 'erode', 2);

        if diff == zeros(y,x)
            background = (background + 2*frameD)/3;
        end
    end

    if LOOK_BACK > 0
        updateFrameBuffer();
    end

    drawnow('expose');
end

% this function compares the current frame to the estimated backgroung value and returns the
foreground objects
    function props = extractForegroundObjects()
        % create a binary image with the foreground using background subtraction.
        % 0 - background; 1 - foreground.
        foreground = zeros(y,x);
        foreground(abs(frameD(:, :, 1) - background(:, :, 1)) > 8) = 1;
        foreground(abs(frameD(:, :, 2) - background(:, :, 2)) > 8) = 1;
        foreground(abs(frameD(:, :, 3) - background(:, :, 3)) > 8) = 1;

        % Filters
        foreground = bwmorph(foreground, 'erode', 1);
        foreground = bwmorph(foreground, 'close', Inf);
        foreground = medfilt2(foreground);

        % Label the object and get their properties
        labels = bwlabel(foreground, 4);
        props = regionprops(labels, 'centroid', 'perimeter', 'area', 'boundingbox', 'eccentricity');

        % Remove small objects (noise)
        rm = [];
        for i = 1 : length(props)
            if props(i).Area < MIN_AREA
                rm = [rm i];
            end
        end
        props(rm) = [];
    end

% dispaly paths, centroids and centers for every object in objects struct
    function drawInfo()
        needPause = 0;
        for i = 1 : size(objects, 2)
            % Draw blue bounding box if the object is a ball
            % red box - otherwise and path
            path = objects(i).path;
            if objects(i).isBall
                if objects(i).lastSeen == k
                    drawBox(objects(i).box, 'b');
                end
                drawPath(path, 'b');
            else
                if objects(i).lastSeen == k
                    drawBox(objects(i).box, 'r');
                end
                drawPath(path, 'r');
            end
            % Draw highest point if present
            [p1, p2] = size(path);
            if p1 > 2 && objects(i).highest == 0 && objects(i).isBall
                if drawHighest(path)
                    objects(i).highest = path(end, 2);
                    needPause = 1;
                end
            end
        end
        if needPause == 1
            pause(3);
        end
    end

% update the buffer for past frames
    function updateFrameBuffer()
        if LOOK_BACK > 1
            for i = LOOK_BACK : -1 : 2
                frameBuff(:, :, :, i) = frameBuff(:, :, :, i-1);
            end
        end
        frameBuff(:, :, :, 1) = frameD;
    end

% update the objects structure by assignining/adding the objects
% detected in the current frame
    function updateObjectsStruct()
        centres = cat(1, props.Centroid);

        % go through all objects currently detected in the frame
        for i = 1 : size(centres,1)
            assigned = false;
            c1 = min(centres(i, 1),x-1);
            c2 = min(centres(i, 2),y-1);
            centrePixel=im2double(frame(uint8(c1),uint8(c2),:));
            currCentreColour = bsxfun(@rdivide, centrePixel, sum(centrePixel,3,'native'));
            % scan the objects tracked in the previous frames and update
            % them
            for j = 1 : size(objects, 2)
                if objects(j).lastSeen < k
                    x1 = objects(j).path(end,1);
                    y1 = objects(j).path(end,2);

                    dist = distance(x1,c1,y1,c2);
                % if the distance between the centres is small and the
                % colour is close enough those are the same object.
                % Update.
                if dist < 30 && isCloseRGBVal(objects(j).colour, currCentreColour,5)
                    objects(j).path = [objects(j).path; [c1 c2]];
                    objects(j).lastSeen = k;
                    objects(j).colour = currCentreColour;
                    objects(j).box = props(i).BoundingBox;
                    vel = objects(j).path(end-1, 2) - objects(j).path(end, 2);
                    if ~objects(j).isBall && vel > -1 && isBall(props(i).Perimeter, props(i).Area,
props(i).Eccentricity, MIN_AREA)
                        if objects(j).ballCount == BALL_FOR_SURE
                            objects(j).isBall = true;
                        else
                            objects(j).ballCount = objects(j).ballCount + 1;
                        end
                    end
                    assigned = true;
                    break;
                end
            end
        end
        % if this is a new object near the edges of the frame (prevents
        % adding detections caused by noise in the middle of the frame) create
        % new instance for it
        if ~assigned && (c2>y-20||c1<20||c1>x-20)
            objects(end + 1) = struct('path', [[c1 c2]], 'lastSeen', k, 'colour', currCentreColour,
'highest', 0, 'box', props(i).BoundingBox, 'isBall', false, 'ballCount', 0);
        end
    end
end

% remove lost objects from objects struct
function removeLostObjects()
    rm = [];
    for i = 1 : size(objects, 2)
        if(objects(i).lastSeen < k-10)
            rm = [rm i];
        end
    end
    objects(rm) = [];
end
```

```
% check two rgb pixels if their values are close enough.
    function is = isCloseRGBVal(pixel1, pixel2,thresh)
        is = true;
        if abs(pixel1(1, 1, 1) - pixel2(1, 1, 1)) > thresh
            is = false;
        end

        if abs(pixel1(1, 1, 2) - pixel2(1, 1, 2)) > thresh
            is = false;
        end

        if abs(pixel1(1, 1, 3) - pixel2(1, 1, 3)) > thresh
            is = false;
        end
    end
end

function d = distance(x1, x2, y1, y2)
 % Find the euclidean distance between two points
 % (x1, y1) and (x2, y2)
 d = sqrt((x1 - x2)^2 + (y1 - y2)^2);

function drawBox(box, color)
    % Draw the bounding box
    hold on;
    rectangle('Position', [box(1),box(2),box(3),box(4)], 'EdgeColor', color,'LineWidth', 1);
    hold off;

function highest = drawHighest(path)
    % Recognizes whether the ball is at his highest point,
    % draws the cross

    highest = 0;

    hold on;
    if isHighest(path)
        plot(int32(path(end, 1)),int32(path(end, 2)), 'xr', 'MarkerSize',20);
        highest = 1;
    end
    hold off;

function drawPath(path, color)
```

```
    % Draw the path by drawing lines between two lines in path array

    n = length(path);
    if n > 2
      hold on;
      for i = 1 : n-1
         line([path(i, 1), path(i+1, 1)],[path(i, 2), path(i+1, 2)],'Color',color,'LineWidth',1);
      end
      hold off;
    end

function is = isBall(P, A, E, MIN_AREA)
   % Determine if the object is ball or not
   % given the perimeter P,
   % area A,
   % eccentricity E of the object
   % and minimum area MIN_AREA

   MIN_COMP = 0.94;
   MAX_ECC = 0.75;
   MIN_SUM = 1.6;
   MAX_SUM = 1.7;
   is = false;

   % Calculate the compactness of the object
   comp = 2 * sqrt(A*pi)/P;

   % Calculate the sum of compactness and eccentricity
   sum = comp + E;

   % The object is ball if all of the following are true:
   %  - Object's area is larger than MIN_AREA (avoiding noise)
   %  - Compactness is nearly 1 since we are looking for
   %    a ball (ball is indeed similar to circle
   %  - Eccentricity is less than 0.75
   %  - The sum of compactness and eccentricity is between
   %    1.6 and 1.7 (found by performing a lot of tests)
   % 0.948 no more
   if comp > MIN_COMP && A > MIN_AREA && E < MAX_ECC && MIN_SUM <= sum && sum
<= MAX_SUM
      is = true;
   end
```

```
function is = isHighest(path)
    % Knowing a path so far check if that's the highest point of the path

    MAX_SIN_ALPHA = 0.1;
    MIN_SPEED = -1;
    is = false;

    % Find the point where the ball stops moving up
    % by finding sin(angle) between ball moving
    % direction and y = 0 in the current and last frames

    v1 = path(end-1, 2) - path(end, 2);
    sin_alpha = v1/distance(path(end-1, 1), path(end, 1), path(end-1, 2), path(end, 2));
    v2 = path(end-2, 2) - path(end-1, 2);
    sin_alpha2 = v2/distance(path(end-2, 1), path(end-1, 1), path(end-2, 2), path(end-1, 2));

    % If sine is less than or equal to MAX_SINE_ALPHA -> ball no longer
    % moves up
    % Check it twice, since the ball is not moving in the highest point for
    % a few frames.
    % Also check if the ball is not moving down to fast to be at
    % the highest point
    if sin_alpha < MAX_SIN_ALPHA && sin_alpha2 < MAX_SIN_ALPHA && v1 >= MIN_SPEED
        is = true;
    end
end
```