Comp Photography Final Project

Syed Nasar

Summer 2016

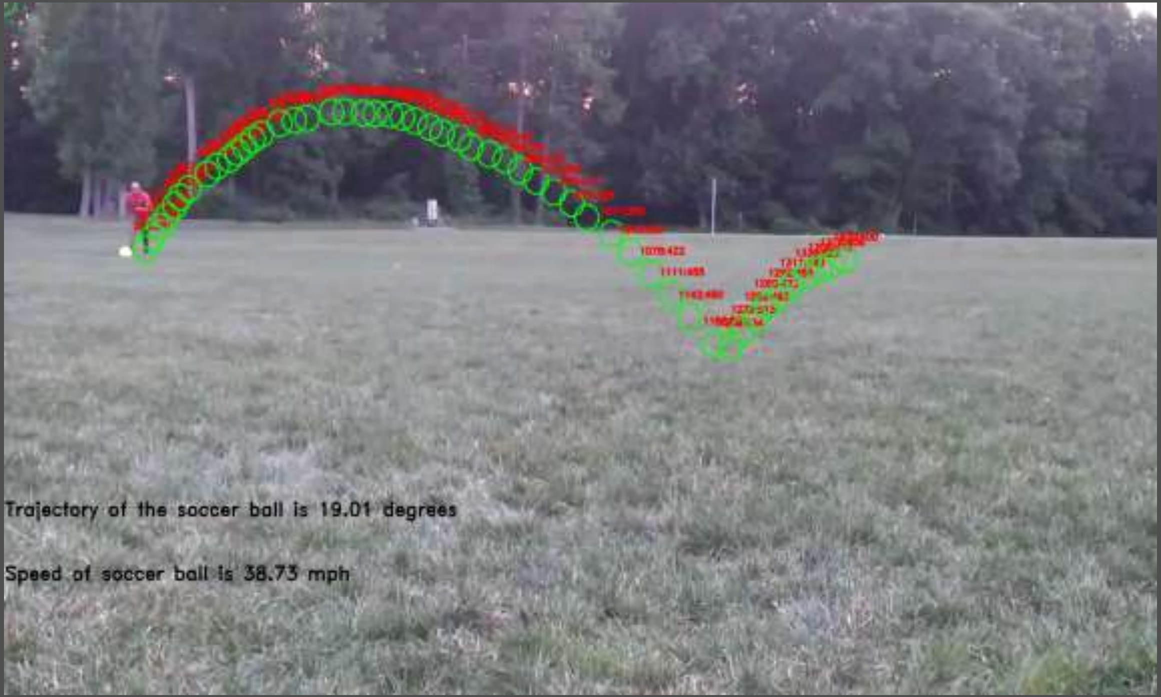
snasar3@gatech.edu

●●●

Teamed up with: Balaji Sundaresan

Soccer Ball Assistant

Tracking the trajectory of the ball in a soccer video.



Trajectory of the soccer ball is 19.01 degrees
Speed of soccer ball is 38.73 mph

The Goal of the Project

A video of a player taking a shot at a soccer ball is analyzed. The video analysis detects the trajectory of the soccer ball, and creates an image tracing the path of the soccer ball, showcasing the curvature and trajectory of the ball. This analysis would help the player and the coach analyze the quality & accuracy of the shots.

Additional Effort: This is a stretch goal in which the various geometric and statistical values would be extracted and presented in textual format. One of the values that was calculated was the speed of soccer ball and the angle of the trajectory of the soccer ball.

Motivation for this project: Myself and my team mate both are huge soccer fans. We both are also invoved in the local community where lot of kids get soccer training. This project would help the local coaches to explain to kids, the strengths and flaws in their techniques more precisely.

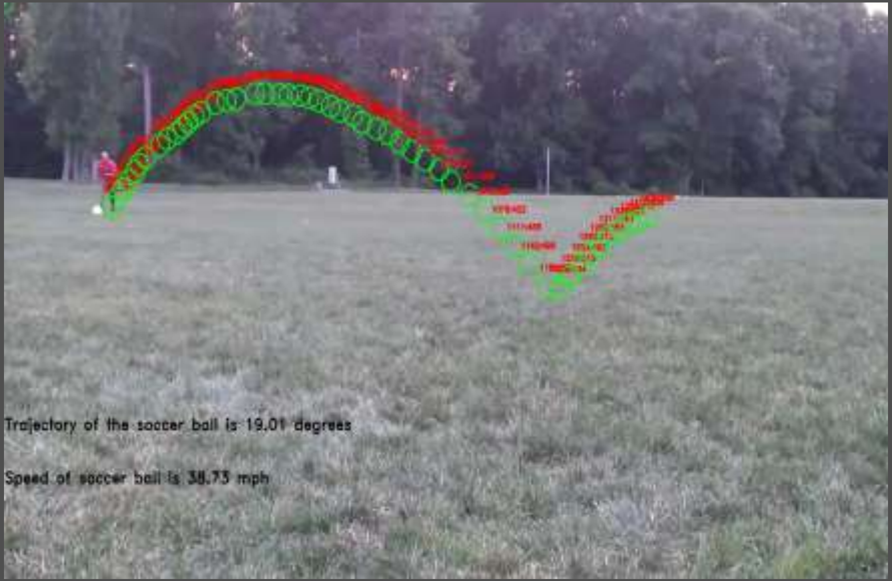
Scope Changes

Scope of the project was expanded.

In addition to tracking the trajectory of the soccer ball, we increased the scope of the project to include additional modules to calculate the speed and the angle of the trajectory of the soccer ball.

Input

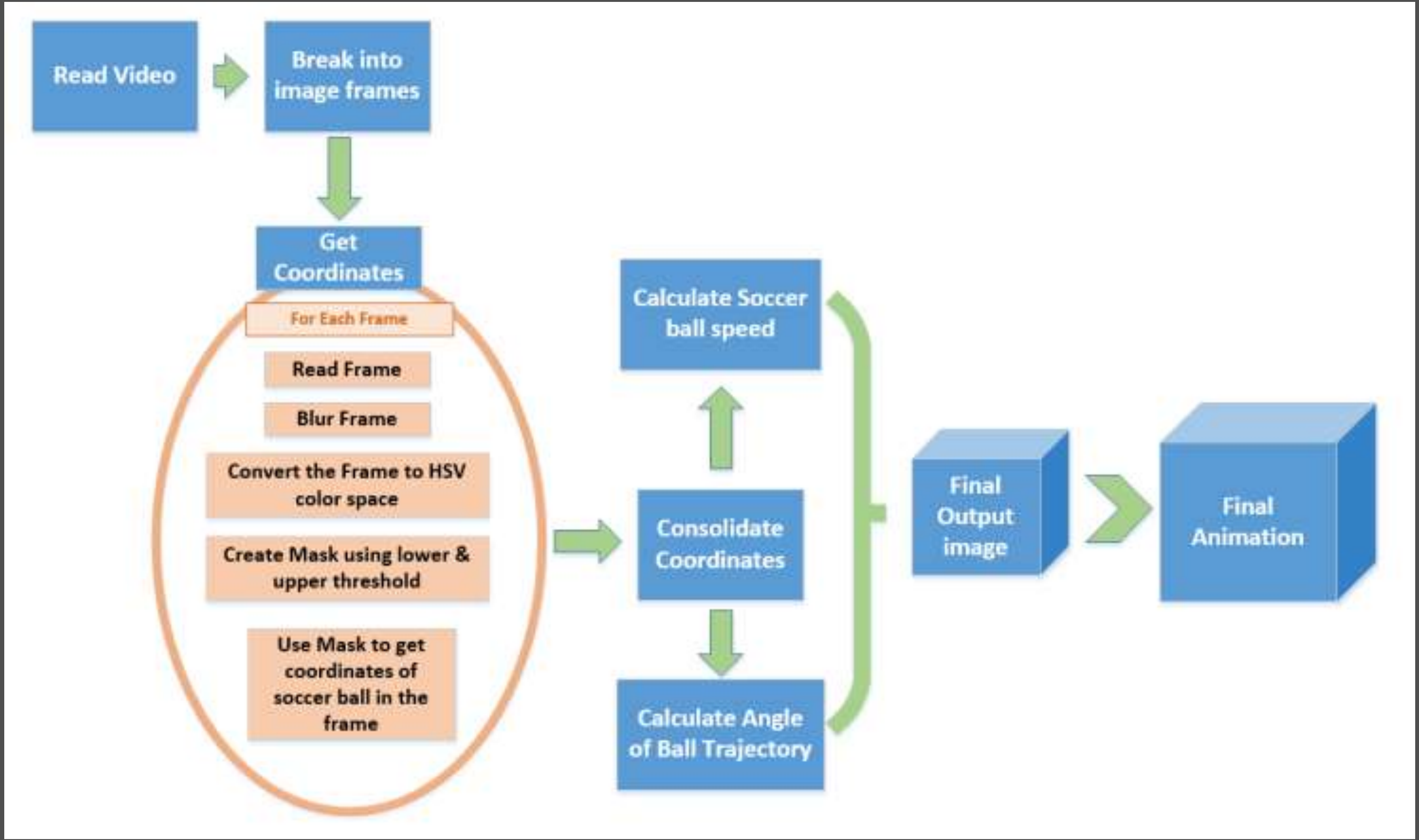
Output



Trajectory of the soccer ball is 19.01 degrees

Speed of soccer ball is 38.73 mph

Soccer Ball Assistant - Pipeline


Read Video
Break into image frames
Get Coordinates
For Each Frame
Read Frame
Blur Frame
Convert the Frame to HSV color space
Create Mask using lower & upper threshold
Use Mask to get coordinates of soccer ball in the frame
Consolidate Coordinates
Calculate Soccer ball speed
Calculate Angle of Ball Trajectory
Final Output image
Final Animation

Pipeline Explained

The complete flow of the project has been automated. All the steps shown in the pipeline diagram, in the previous slide, are implemented and can be run with a single line of command.

Overall the flow works as follows. A video is taken of a soccer player kicking a soccer ball. This video needs to be copied to a specified folder. Currently the code has been tested for AVI and MP4 formats only. More than one video can be placed on the specified folder. The pipeline can then be invoked by running the main python code from a command line. Everything else works automatically for the rest of the steps. Here are the automated steps explained sequentially in their order of execution:

Step: Read Video:
Reads all videos one by one from a pre-configured folder

Step: Break into image frames
The videos are broken into multiple frames using OpenCV.

CONTINUED…

Pipeline Explained (contd.)

Step: Get Coordinates

This step runs against each frame for each video. In addition, this process is made up of multiple sub-steps. The sub-steps are listed below as per their execution flow:

- Read frame
- Blur frame using Gaussian blur
- Convert the frame to HSV color space using OpenCV cvtcolor
- Create mask using lower and upper threshold
- Use mask to get the moments
- Use cv2.moment to get the centroid coordinates of soccer ball in the frame

Step: Consolidate Coordinates

Consolidate the coordinates from each frame for each video.

Step: Calculate soccer ball speed:

In this step the speed of the soccer ball is calculated by deriving it from the coordinate change per frame, as well as based on the number of frames per second (derived from video runtime). In addition the frame FOV, distance of the camera is also taken into consideration.

CONTINUED…

Pipeline Explained (contd.)

Step: Calculate angle of the ball's trajectory

In this step the angle of the ball is calculated based on the coordinates from each frame.

Step: Final Output Image

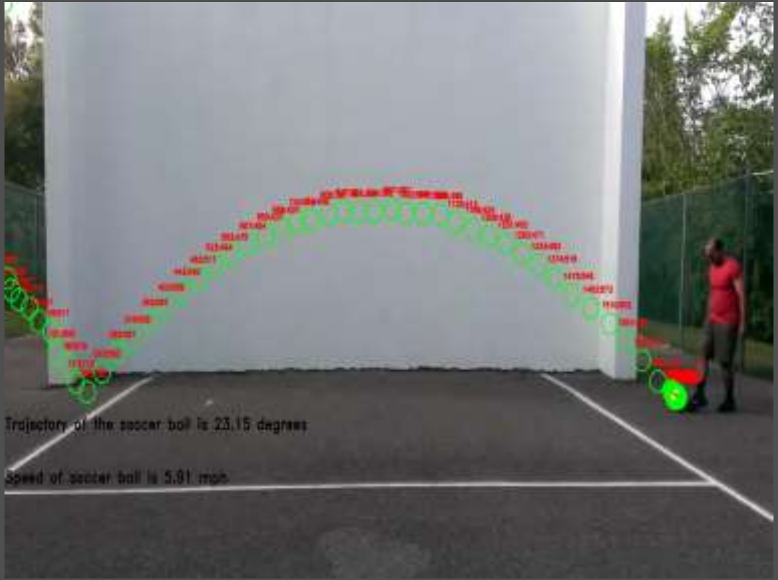
Here the speed, and the trajectory angle is drawn onto the first image frame from for each video. Then the coordinates are plotted on the image to show the path of the ball

Step: Final Animation

This is the final step where a dynamic video is created showing the path of the ball, along with its speed and trajectory.

- In this an animation of the ball's path and trajectory is constructed using Matplotlib's animation feature.
- The path of the plot is animated using the ball coordinates.
- The image from the previous step is used as the background of the dynamic plot.
- And finally the dynamically animated plot is saved as a video.

Demonstration: Input/Output results
Video # 1 Closeup Shot - Soccer Ball Moving from "Right to Left"

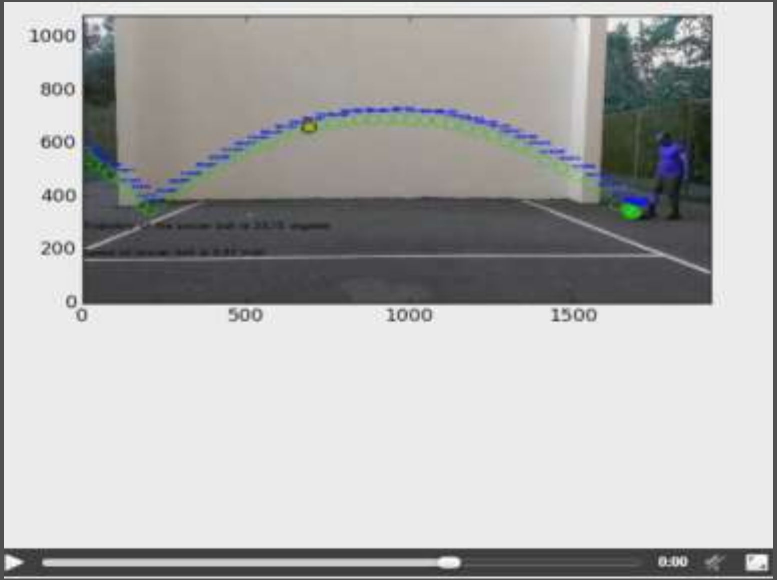


Trajectory of the soccer ball is 23.15 degrees

Speed of soccer ball is 5.91 mph

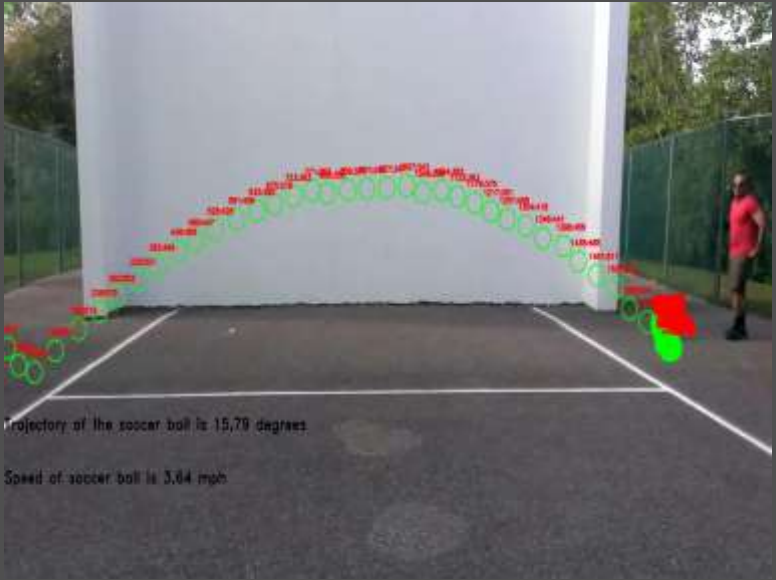
The input video # 1 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 1 Closeup Shot - Soccer Ball Moving from “Right to Left” (Animation)



The input video # 1 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 2 Closeup Shot - Soccer Ball Moving from “Right to Left”

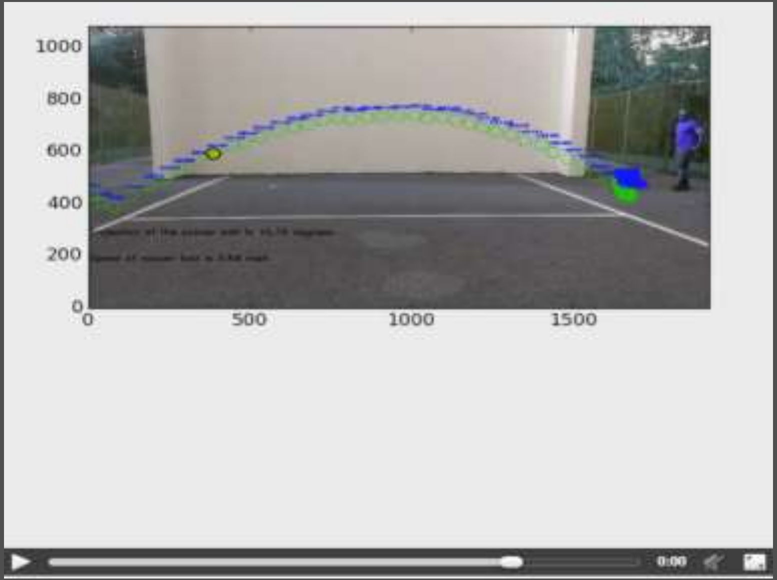


```
Trajectory of the soccer ball is 15.79 degrees

Speed of soccer ball is 3.64 mph
```

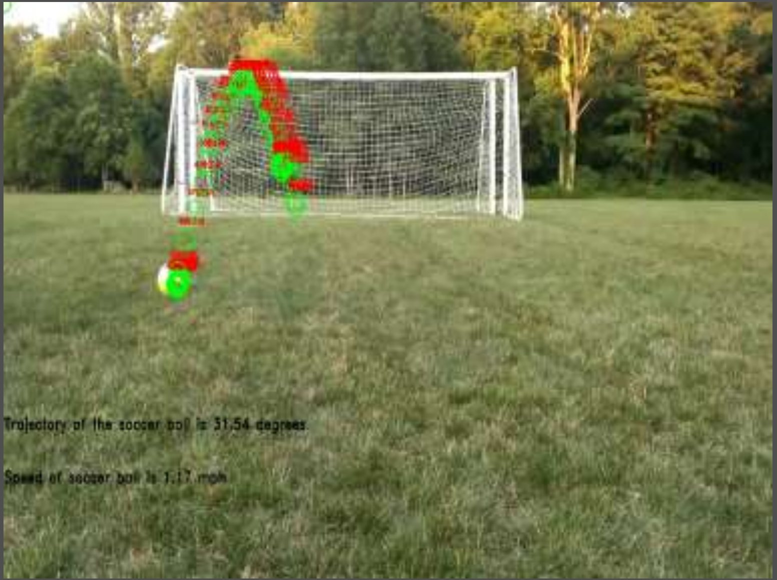
The input video #2 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 2 Closeup Shot - Soccer Ball Moving from "Right to Left" (Animation)



The input video #2 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 3 Closeup Shot - Soccer Ball Moving “Away” from Camera

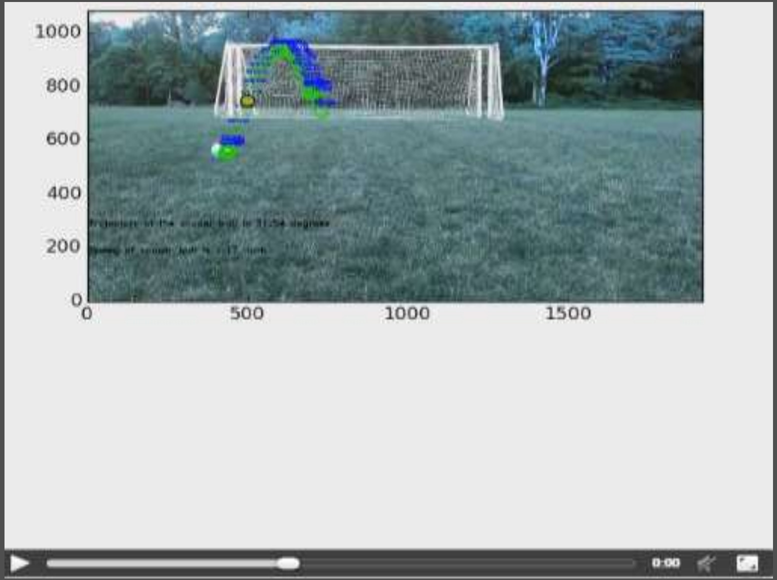


```
Trajectory of the soccer ball is 31.54 degrees.

Speed of soccer ball is 1.17 mph
```

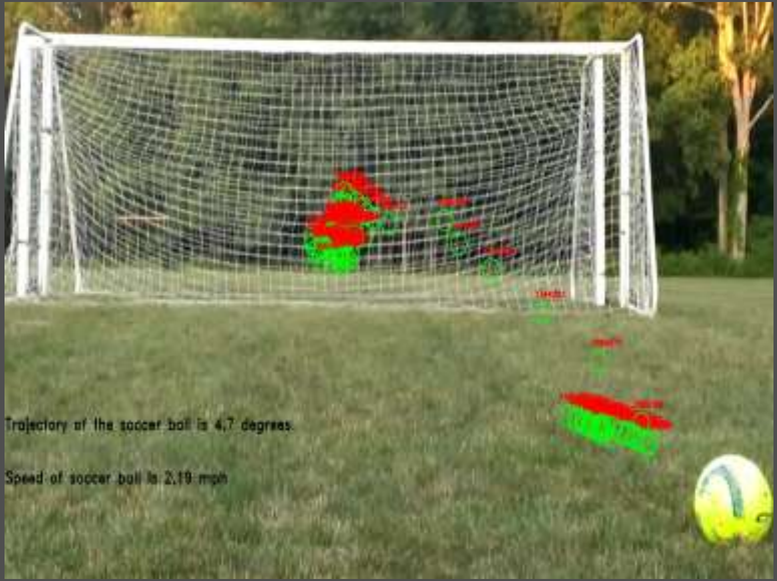
The input video #3 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 3 Closeup Shot - Soccer Ball Moving “Away” from Camera (Animation)



The input video #3 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 4 Long Shot - Soccer Ball Moving “Away” from Camera

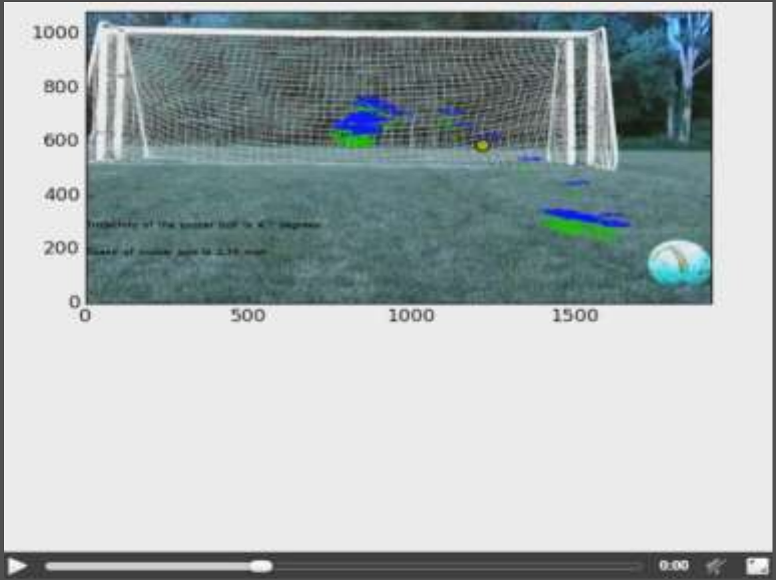


```
Trajectory of the soccer ball is 4.7 degrees.

Speed of soccer ball is 2.19 mph
```

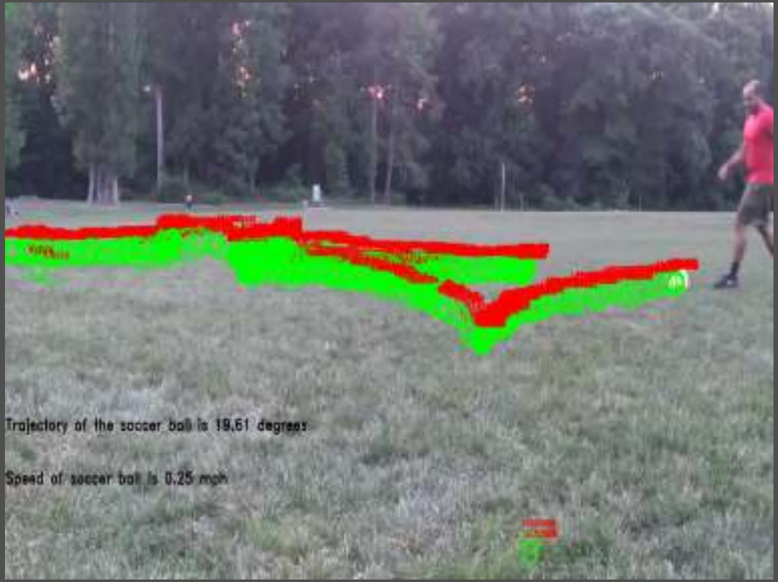
The input video #4 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 4 Long Shot - Soccer Ball Moving "Away" from Camera (Animation)



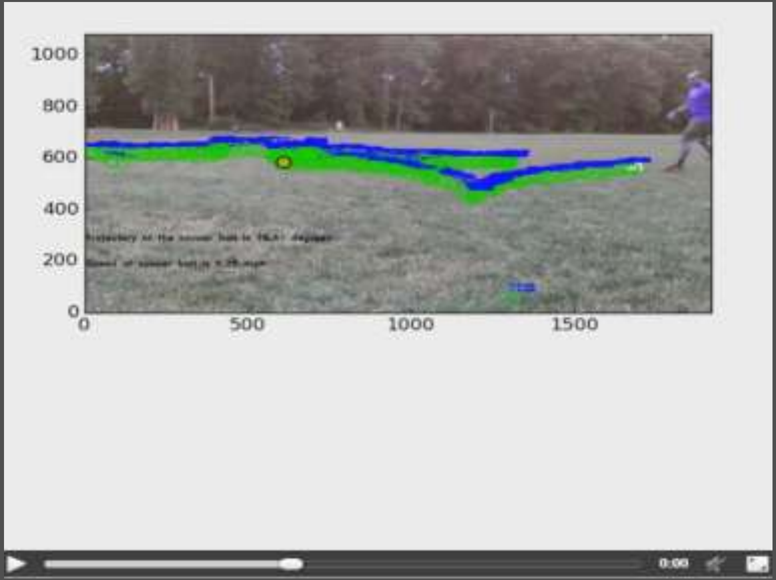
The input video #4 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 5 Long Shot - Soccer Ball Moving “All Around” the Camera FOV


Trajectory of the soccer ball is 19.61 degrees
Speed of soccer ball is 0.25 mph

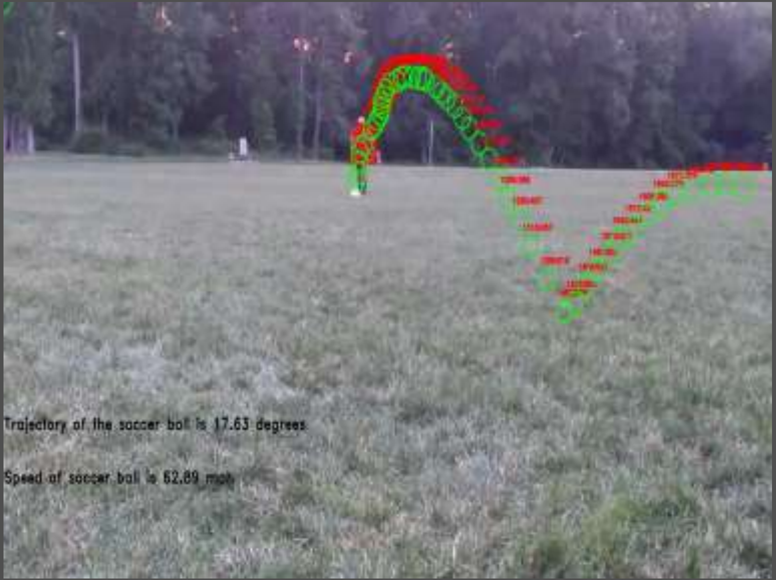
The input video #5 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 5 Long Shot - Soccer Ball Moving “All Around” the Camera (Animation)



The input video #5 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 6 Long Shot - Soccer Ball Moving “Towards” the Camera

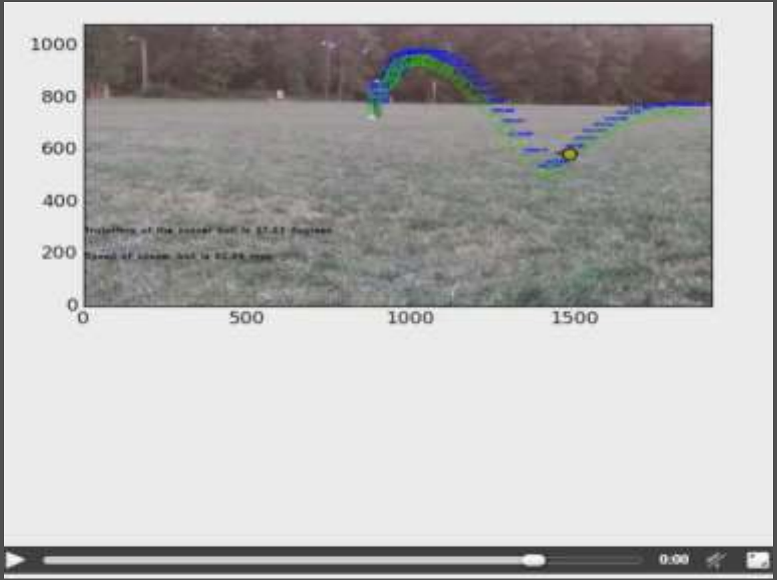


Trajectory of the soccer ball is 17.63 degrees

Speed of soccer ball is 62.89 mph

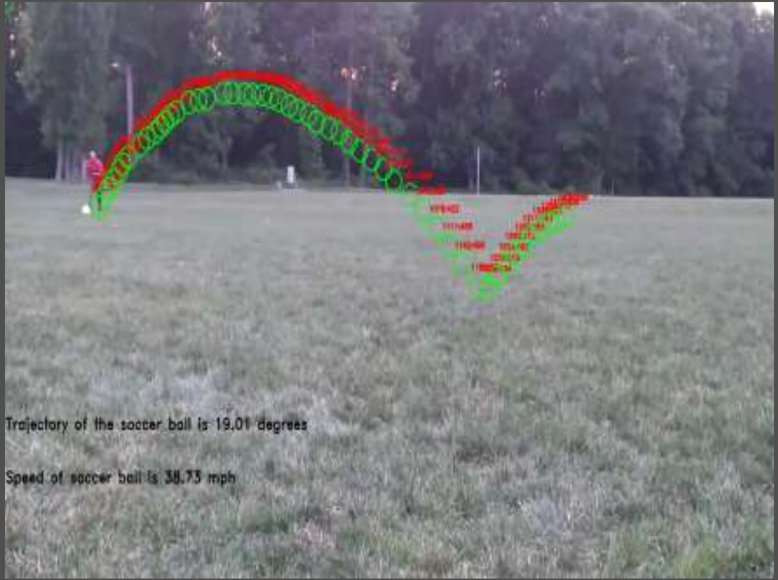
The input video #6 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 6 Long Shot - Soccer Ball Moving “Towards” the Camera (Animation)



The input video #6 is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here.

Demonstration: Input/Output results
Video # 7 Long Shot - Soccer Ball Moving “Towards” the Camera

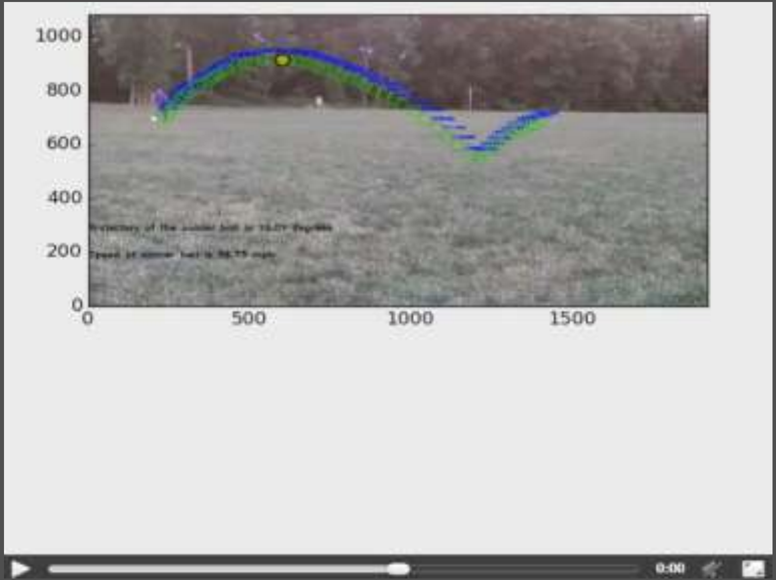


Trajectory of the soccer ball is 19.01 degrees

Speed of soccer ball is 38.73 mph

The input video #7 is shared in mp4 format here (avi format here), final output image link shared here and final output animation link is shared here.

Demonstration: Input/Animation Output results
Video # 7 Long Shot - Soccer Ball Moving “Towards” the Camera (Animation)



The input video #7 is shared in mp4 format here (avi format here), final output image link shared here and final output animation link is shared here.

Computation: Project Development

One of the initial decisions we as a team decided on this project was to shoot our own soccer videos. This decision was made to avoid too much stochasticity with external soccer videos given the limited time allocated to complete the project.

Initially, four soccer balls of different colors were used to shoot different videos. The color of balls were green, multi colored, black/white patches and blue/pink patches. We narrowed down on using two different approaches to track the soccer ball and pick the one which gave the best results. The first approach was feature matching using three different algorithms - SHIFT with ORB and Hamming distance, KNN and finally FLANN. The second approach was to create a mask for each frame in the video based on the HSV

Computation: Project Development

color space corresponding to the color of the ball in the video. Using either of these approaches, the plan was to get the (x,y) coordinates of the ball in each frame of the video and store in a numpy array. This coordinates (numpy array) would give the location of the ball for any given frame in the video and this would precisely help us track the path followed by the soccer ball in the video.

Additionally, to avoid biases in the videos, a second decision was made to shoot videos with the soccer ball moving in different directions from the camera and close-up and long shots as well. A total of seven videos were finally selected which includes 3 close-up and 4 long shots. Seven videos covered different perspectives of soccer ball traversing the camera including soccer ball

- moving from right to left while staying horizontally in the FOV of camera

Computation: Project Development

- right to left while moving away from camera ,
- left to right while moving away from camera,
- long shot with ball moving towards the camera at a very high speed
- and finally, slow dribbling of soccer ball all over the FOV of camera (towards, away, right to left and left to right from the camera).

After shooting for hours over a span of 3 days, we selected the best 7 videos to use for this project.

Shooting the videos was the only manual process in the whole pipeline. All other modules in the pipeline were completed automatic with no manual intervention needed. The next step was to narrow down on which of the two approaches to choose for this project.

Computation: Project Development

Approach # 1 - Feature Matching Approach:

Image of each of the four different colored ball was taken and these images served as the template for feature matching. Three different templates (large, medium and small image of the ball) were taken for each ball, summing up the total templates to 12.

Using OpenCV video reading function, the input videos were read one frame at a time. The first issue was the frames per second were too high for the long shot videos, kicking the soccer ball at a high speed. We decided convert these videos into slow motion (x1/4 slower) to control the frames per second. This helped. The second issue was the root cause of scrapping the feature matching

Computation: Project Development

approach. Because of the drastic change in scale of the soccer ball size in the video frame and blurriness caused by the speed it was travelling, all the feature matching algorithms gave poor results.

Fine tuning different parameters like the number of features used, number of pyramids used, edge threshold and patch size improved the results a little but was nowhere close to capturing coordinates of the soccer ball in each frame.

Changing the SHIFT algorithm from using Hamming Distance to KNN and FLANN did not help much and also reducing the number of key points matches proved futile. It was becoming very clear that the blurriness caused by the speed of the soccer ball, variance in scale and most importantly the

Computation: Project Development

background noise consisting of sky, trees and even grass close to the camera gave high number of false positive matches. The same results were consistently reproduced with all four ball colors and all videos with different FOV.

Having tested these feature matching approaches with number of different algorithms, videos and features (different balls), we decided to scrap the feature matching approach for this project.

Computation: Project Development

Approach # 2 - Creating Mask using HSV Color Space:

The reasoning behind this approach was to first capture the HSV color space of the soccer ball used in the video and then use the lower and upper threshold of the captured HSV color space to track the soccer ball in each frame of the input video.

Using open cv video reading function, the input videos were read one frame at a time. Once again, the issue was the frames per second were too high for the long shot videos, kicking the soccer ball at a high speed. We decided convert these (kicking the soccer ball at high speed) videos into slow motion (x1/4 slower) to control the frames per second. This was the only issue we

Computation: Project Development

faced with the mask creation approach and that was fixed immediately after we converted some videos into slow motion videos.

The approach works as follows: The input videos are fed into get_hsv_code.py to automatically calculate the HSV color space of the soccer ball in the video. This program outputs the lower and upper threshold of the HSV color space of the soccer ball.

Each frame in the input video is read one frame at a time and sent to the pipeline process. The process consists of blurring the frame to reduce high frequency noise and narrow down on prominent objects in the frames like human, soccer ball etc. The blurred frame is them converted to HSV color

Computation: Project Development

space. The frame in the HSV color space now is used to create the mask using the lower and upper limit HSV color space of the soccer ball which was an output from the get_hsv_code.py program.

The mask is then passed into open cv function moments to narrow down the coordinates of the soccer ball in a given frame. The coordinates are stored in a numpy array. This process is repeated for each frame in the video sequentially and all the coordinates are captured in the same numpy array.

The resulting numpy array now has the exact trajectory of the soccer ball in the video on a per frame basis.

Computation: Project Development

Calculation of speed of the soccer ball was one of the most challenging tasks in this project. For each video file, there were number of constants such as FOV, distance from which the video was shot, the video resolution (image width) and total run time of the video.

Using these above mentioned constants, we can then calculate the distance in pixels travelled by the soccer ball from the start to the end of the video using each frame of the input video file. Pixel range is then calculated by subtracting the start and end of the final coordinates in the soccer ball tracker numpy array.

soccer ball_speed = ((pixel_range * pixels_travelled) / video_run_time) * 0.681818

(note: constant value 0.681818 was used for feet and mph as the metrics)

Computation: Project Development

Angle of trajectory of the soccer ball is calculated using the final coordinates contained in the numpy array. Once all the coordinates are accumulated in the numpy array, the task is to simply calculate the angle(s) between two n-dimensional vectors.

This achieved using the formula:

angle = math.acos(dotproduct(x, y) / (length(x) * length(y)))

angle_in_degrees = np.degrees(angle)

Where x and y are soccer ball coordinates from each frame contained in the numpy array.

Details: What worked?

The approach using creating a mask using HSV color space of the soccer ball worked the best.

The reasoning behind this approach was to first capture the HSV color space of the soccer ball used in the video and then use the lower and upper threshold of the captured HSV color space to track the soccer ball in each frame of the input video.

Using OpenCV video reading function, the input videos were read one frame at a time. Once again, the issue was the frames per second were too high for the long shot videos, kicking the soccer ball at a high speed. We decided convert these (kicking the soccer ball at high speed) videos into slow motion (x1/4 slower) to control the frames per second. This was the only issue we faced with the mask creation approach and that was fixed immediately after converting some videos into slow motion videos.

Details: What worked?

The approach works as follows: The input videos are fed into get_hsv_code.py to automatically calculate the HSV color space of the soccer ball in the video. This program outputs the lower and upper threshold of the HSV color space of the soccer ball.

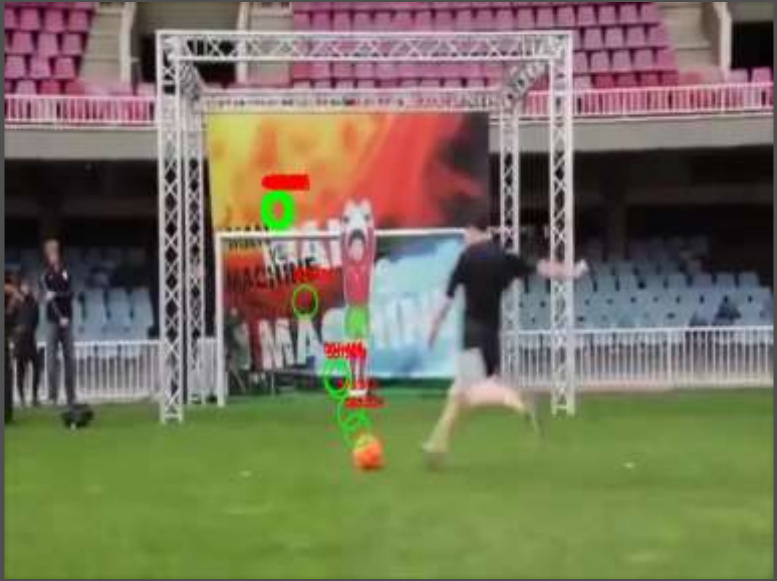
Each frame in the input video is read one frame at a time and sent to process in the pipeline. The process consists of blurring the frame to reduce high frequency noise and narrow down on prominent objects in the frames like human, soccer ball etc. The blurred frame is them converted to HSV color space. The frame in the HSV color space now is used to create the mask using the lower and upper limit HSV color space of the soccer ball which was an output from the get_hsv_code.py program.

Details: What worked?

The mask is then passed into open cv function *moments* to narrow down the coordinates of the soccer ball in a given frame. The coordinates are stored in a numpy array. This process is repeated for each frame in the video sequentially and the all the coordinates are captured in the same numpy array. The resulting numpy array now has the exact trajectory of the soccer ball in the video on a per frame basis.

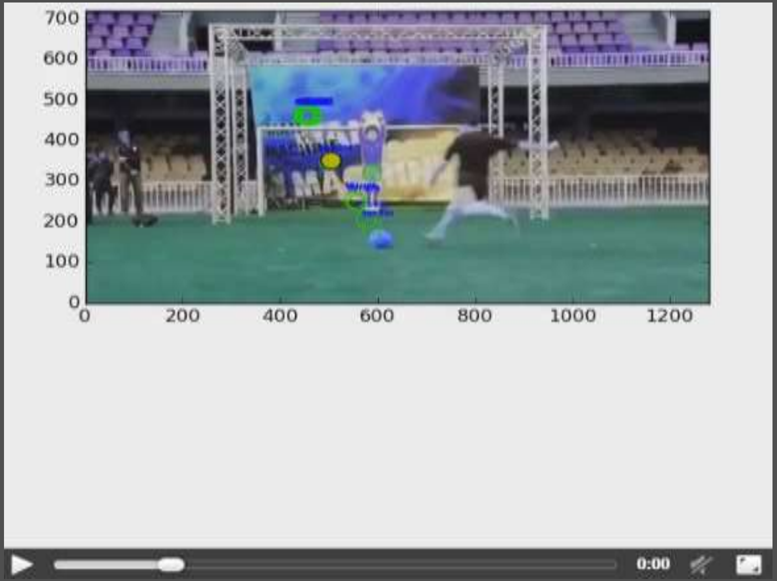
To test the robustness of the Soccer Ball Assistant, we decided to use an external video. This video on YouTube is of a Robotic Goal Keeper trying to prevent goals from a real human being. This video has high amount of background noise because all the objects in the video were almost the same color as the Orange Soccer Ball. The chairs were red, The Robotic Goal Keeper was also swearing an Orange shirt, the card board behind goal was also Orange. Soccer Ball Assistant proved robust to this high level of noise and was able to successfully track the soccer ball in the video. The next four slides shows the phenomenal ball tracking by this Soccer Ball Assistant.

Details: What worked? Testing the Soccer Ball Assitant with an External Video with Large Amount of Background Noise (Input & Output Images)



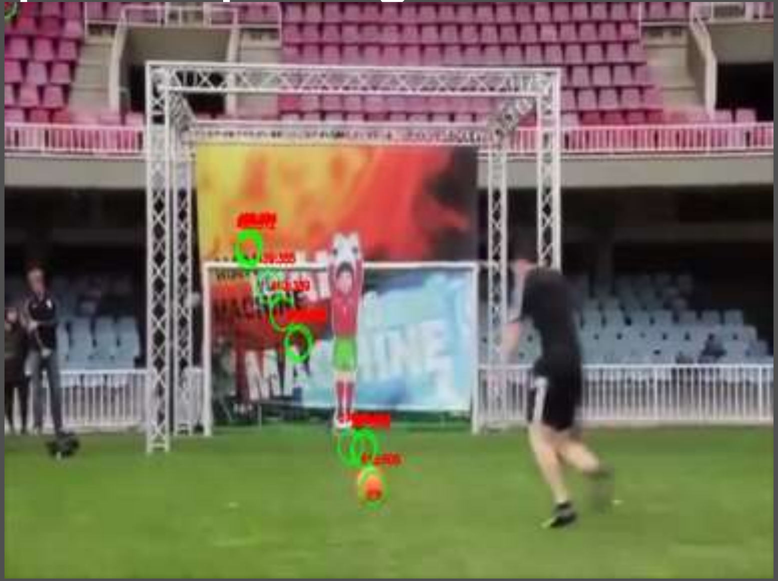
The input video # 1 of Robotic Goali is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here. Original YouTube video can be found here.

Details: What worked? Testing the Soccer Ball Assitant with an External Video with Large Amount of Background Noise (Input & Animation Output Images)



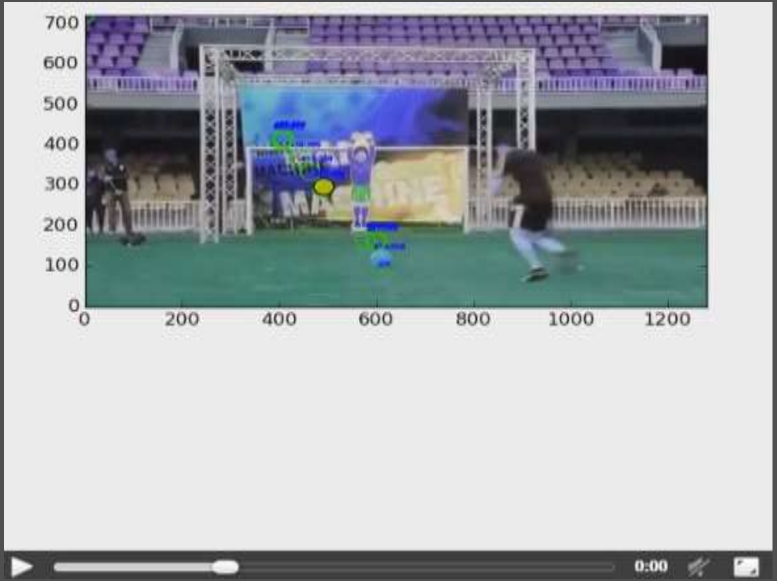
The input video # 1 of Robotic Goali is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here. Original YouTube video can be found here.

Details: What worked? Testing the Soccer Ball Assitant with an External Video with Large Amount of Background Noise (Input & Output Images)



The input video # 2 of Robotic Goali is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here. Original YouTube video can be found here.

Details: What worked? Testing the Soccer Ball Assitant with an External Video with Large Amount of Background Noise (Input & Animation Output Images)



The input video # 2 of Robotic Goali is shared in mp4 format here (avi format here), final output image link is shared here and final output animation link is shared here. Original YouTube video can be found here.

Details: What did not work? Why?

Our first approach of using Feature Matching techniques using algorithms like SHIFT with ORB() with Hamming Distance, KNN and FLANN gave very poor results. This because of the drastic change in scale of the soccer ball size in the video frame and blurriness caused by the speed it was travelling, all the feature matching algorithms gave poor results.

Fine tuning different parameters like the number of features used, number of pyramids used, edge threshold and patch size improved the results a little but was nowhere close to capturing coordinates of the soccer ball in each frame.

Also reducing the number of key points matches proved futile. It was becoming very clear that the blurriness caused by the speed of the soccer ball, variance in scale and most importantly the background noise consisting of sky, trees and even grass close to

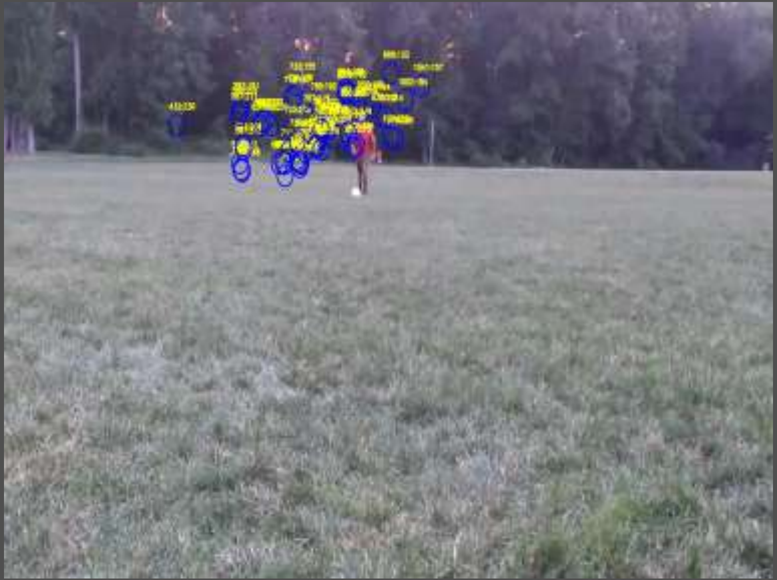
Details: What did not work? Why?

the camera gave high number of false positive matches. The same results were consistently reproduced with all four ball colors and all videos with different FOV.

Another big problem, that was difficult to overcome, was to pattern match with balls that did not have clear features, and were mono-colored in nature. This lack of feature made it near impossible to get any matching with the frames.

Details: What did not work? Why?

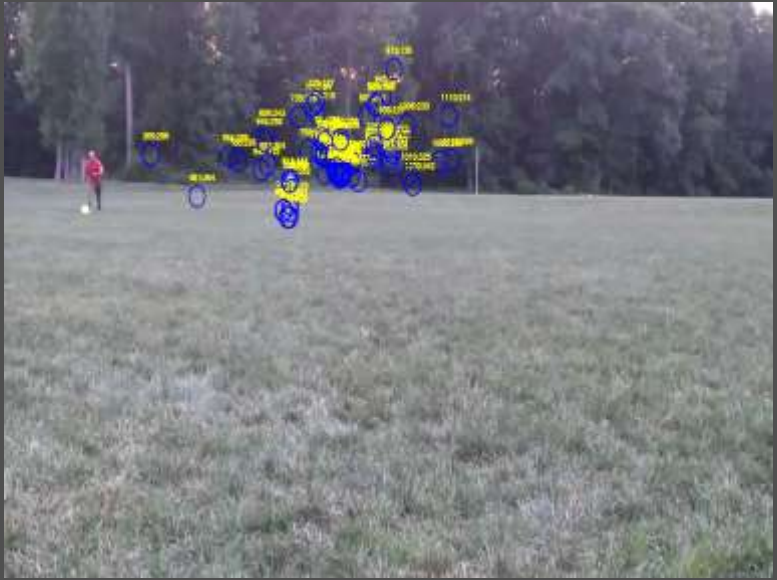
Soccer Ball Moving “Towards” the Camera (Bad Feature Matching – Unused Approach)



The input video #6 is shared in mp4 format here (avi format here), final output image link with not bad feature matching is shared here. This approach was not used in the project pipeline and scrapped after these bad feature matching results.

Details: What did not work? Why?

Soccer Ball Moving “Towards” the Camera (Bad Feature Matching – Unused Approach)



The input video #7 is shared in mp4 format here (avi format here), final output impage link with bad feature matching is shared here shared. This approach was not used in the project pipeline and scrapped after these bad feature matching results.

Computation: Code Explanation

There are four python programs created for the Soccer Ball Assistant final project.

- soccer_ball_tracker.py: This is the main program which incoorporates necessary algorithms and functions need to track the soccert ball, calculate the speed and trajectory of the soccer ball and plot the results in the first frame of the video file.

- util.py: This is the utility program used to calculate angle of the trajectory of the soccer ball and create final animation based on the coordinates of the soccer ball in each frame in the input video file.

- get_hsv_code.py: This is another utility program used before the start of pipeline to get the hsv color space of each of the four soccer balls. This helped in creating the lower and upper threshold needed for creating the mask.

- feature_matching.py: This program is not used but this was the initial attempt to track the soccer ball based on feature matchings algorithms like SHIFT wth ORB() and Hamming Distance, KNN and FLANN.

Computation: Code Explanation

Function: if __name__ == '__main__‘ in soccer_ball_tracker.py. This is the main parent function which initiates the pipeline and calls the different functions used in this project.

- All the existing files in the output folder are cleared.
- The process then checks the OS the program is running on and decides on using .avi or .mp4 input video files based on the OS.
- The process then iterates through the input folder and kicks of the “Soccer Ball Assistant” pipeline for every videofile in input directory one file at a time.

Function: __main__ code snippet

```
# remove all existing images in out directory
  files = glob('output/*')
  for f in files:
      os.remove(f)

  # Check OS, windows only supports .avi files and not mp4 (mostly)
  if platform.system() == 'Windows':
      file_extension = "avi"
  else:
      file_extension = "mp4"

  video_files = fnmatch.filter(os.listdir(video_dir), '*.' + file_extension)

  for video_file in video_files:
      file_name = video_file.split('.')[0]
      coordinates, soccer_ball_speed = process_video(video_file, file_name)
      # load the first frame in the video
      image = cv2.imread(os.path.join("output", file_name + "_first.jpg"), cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR)
      # draw the ball trajectory on the first frame
      drawOutput(coordinates, image, soccer_ball_speed)
      util.create_animation(coordinates, image, file_name)

  print("Process completed successfully")
```

Computation: Code Explanation

Function: process_video() in soccer_ball_tracker.py

- This function takes two input parameters video_file name and the video file name without the extension (.mp4 or .avi). The function returns the coordinates of the soccer ball and the speed of the soccer ball.
- Open CV function cv2.VideoCapture is initiated to capture the frames in the video using the read functoin.
- For each frame in the each video, the process calls track_soccer_ball() function to get the corresponding coordinates of the soccer ball in the frame and builds a consolidated list of coordinates in a numpy array
- Finally, it calculates the speed of the soccer ball using different specifications obtained from the function get_speed_specs()

Function: process_video() code snippet

```
def process_video(video_file, file_name):
 global video_dir
   coordinates = np.empty((0, 2), int)
    video_path = os.path.join(video_dir, video_file)
    capture = cv2.VideoCapture(video_path)
    # get techincal specs needed to calculate soccer ball speed
    pixels_travelled, video_run_time = get_speed_specs(capture, file_name)
    print("Processing Video... {}").format(video_file)
    print("")
    first = 1
    while True:
        okay, image = capture.read()
        if first:
            cv2.imwrite(os.path.join("output", file_name + "_first.jpg"), image)
            first = 0
        if okay:
            centroid = track_soccer_ball(image, file_name)
            if not centroid:
                break
            else:
                coordinates = np.vstack((coordinates, np.array([[centroid[0], centroid[1]]])))
            if cv2.waitKey(1) & 0xFF == 27:
                break
        else:
            print("End of Video {}").format(video_file)
            print("")
            break
```

CONTINUED ..

Function: process_video() code snippet (contd.)

```
# calculate speed
pixel_range = abs(coordinates[0, 0] - coordinates[-1, 0])
# use constant value 0.681818 for feet and mph and a constant value 3.6 for meters and kph
soccer_ball_speed = ((pixel_range * pixels_travelled) / video_run_time) * 0.681818
soccer_ball_speed = round(soccer_ball_speed, 2)
print("Speed of the soccer ball is {} mph").format(soccer_ball_speed)
print("---------------------------")
print("")
return coordinates, soccer_ball_speed
```

Computation: Code Explanation

Function: track_soccer_ball in soccer_ball_tracker.py

- This function takes two input parameters, image frame in the video file (one frame at a time) and the name of video file without the extension (.mp4 or .avi) and return the coordinates of the soccer ball in the frame.
- Based on the video file name, the process decides on the HSV threshold to be used and then blurs the image to reduce high frquency noise.
- The blurred image with the HSV thresholds are then used to create the mask which represent the soccer ball in the given frame.
- cv2.moments function is then used to find the centroids of the soccer ball.

Function: track_soccer_ball() code snippet

```
def track_soccer_ball(image, file_name):
 # Blur the image to reduce noise
 blur = cv2.GaussianBlur(image, (5, 5), 0)
 # Convert BGR to HSV
 hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

 if file_name.find("closeup_shot") != -1:
        lower_threshold = lower_dict.get("closeup_shot")
        upper_threshold = upper_dict.get("closeup_shot")
    elif file_name.find("long_shot") != -1:
        lower_threshold = lower_dict.get("long_shot")
        upper_threshold = upper_dict.get("long_shot")
    elif file_name.find("robotic_goalee") != -1:
        lower_threshold = lower_dict.get("robotic_goalee")
        upper_threshold = upper_dict.get("robotic_goalee")
    else:
        pass

 # threshold the HSV image to get only for the chosen color and create the mask
 mask = cv2.inRange(hsv, lower_threshold, upper_threshold)
 # blur the mask
 bmask = cv2.GaussianBlur(mask, (5, 5), 0)
```

CONTINUED…

Function: track_soccer_ball() code snippet (contd.)

```
# Take the moments to get the centroid
    moments = cv2.moments(bmask)
    m00 = moments['m00']
    centroid_x, centroid_y = None, None
    if m00 != 0:
        centroid_x = int(moments['m10'] / m00)
        centroid_y = int(moments['m01'] / m00)
    # initialize centroid
    ctr = (-1, -1)
    # use centroid if it exists
    if centroid_x != None and centroid_y != None:
        ctr = (centroid_x, centroid_y)
        # put black circle in at centroid in image
        cv2.circle(image, ctr, 4, (0, 0, 0))
    # force image display, setting centroid to None on ESC key input
    if cv2.waitKey(1) & 0xFF == 27:
        ctr = None
    return ctr
```

Computation: Code Explanation

Function: get_speed_specs( ) in soccer_ball_tracker.py

- This function takes two input parameters, open cv video capture object and the video file name without the extension (.mp4 or .avi) and returns pixels travelled by soccer ball and the video run time.
- The process captures the video file resolution (image width and height), frames per second and total run time using cv2 functions.
- The FOV and distance of the input video is stored in a dictionary and those constants are retrieved.
- Pixels travelled by the soccer ball is then calculated using the formula:

frame_width_ft = 2 * (math.tan(math.radians(fov * 0.5)) * distance) and pixels_travelled = frame_width_ft / float(image_width)

Function: get_speed_spec() code snippet

```
def get_speed_specs(capture, file_name):

    # print video resolution
    image_width = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
    image_height = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)
    image_resolution = [image_width, image_height]
    print("Video resolution Frame Count is {} (width x height)").format(image_resolution)
    print("")

    # print frames per second
    fps = capture.get(cv2.CAP_PROP_FPS)
    print "Frames per second using video.get(cv2.CAP_PROP_FPS) : {0}".format(fps)
    print("")

    # print frame count
    frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
    print("Total Frame Count is {}").format(frame_count)
    print("")
```

CONTINUED…

Function: get_speed_spec() code snippet (contd.)

```
# total run time
video_run_time = round(frame_count / fps, 2)
 print("Total video run time is {}").format(video_run_time)
 print("")

 # get fov for the video
  fov = fov_dict.get(file_name)

  # get distance from which the video was shot
  distance = distance_dict.get(file_name)

  # calculate the the width of the image at the distance specified
  frame_width_ft = 2 * (math.tan(math.radians(fov * 0.5)) * distance)
  pixels_travelled = frame_width_ft / float(image_width)

  return pixels_travelled, video_run_time
```

Computation: Code Explanation

Function: drawOutput() in soccer_ball_tracker.py

- This function takes three input parameters, numpy array containing the coordinates of the soccer ball in each frame in the video, first image in the video frame and the speed of the soccer ball. The output image is stored in the output folder.
- The first frame of the input video is used as the base output image.
- The coordinates are plotted on the base output image using cv2.circle() function.
- The speed of the soccer ball is printed the output image using cv2.putText() function.

Function: drawOutput() in code snippet

```
def drawOutput(coordinates, img2, soccer_ball_speed):

 rows2 = img2.shape[0]
 cols2 = img2.shape[1]
 out = np.zeros((rows2, cols2, 3), dtype='uint8')
 out = img2

 # For each pair of points we have between both images draw circles, then connect a line between them
    for pt in coordinates:
        x = pt[0]
        y = pt[1]
        cv2.circle(out, (int(x), int(y)), 20, (0, 255, 0), thickness=2)
        font = cv2.FONT_HERSHEY_SIMPLEX
        txt = str(int(x)) + ":" + str(int(y))
        cv2.putText(out, txt, (int(x) - 20, int(y) - 30), font, 0.5, (0, 0, 255), 2)

 # print soccer ball trajectory on the image
    angle_in_degrees = util.calculate_angle(coordinates)
    angle_in_degrees = round(angle_in_degrees, 2)
    angle_text = "Trajectory of the soccer ball is " + str(angle_in_degrees) + " degrees“
    print("Trajectory of the soccer ball is {} degrees").format(angle_in_degrees)
    print("")
```

Function: drawOutput() code snippet (contd.)

```
# print soccer ball speed on the image
 font = cv2.FONT_HERSHEY_DUPLEX
 cv2.putText(out, angle_text, (1, 800), font, 1, (0, 0, 0), 2)
 font = cv2.FONT_HERSHEY_DUPLEX
 speed = "Speed of soccer ball is " + str(soccer_ball_speed) + " mph“
 cv2.putText(out, speed, (1, 900), font, 1, (0, 0, 0), 2)

 # show the image
 cv2.imshow('Matched Features', out)
 cv2.imwrite(os.path.join("output", file_name + ".jpg"), out)
 cv2.waitKey(0)
 cv2.destroyWindow('Matched Features')

 return
```

Computation: Code Explanation

Function: create_animation () in util.py

- This function takes 3 input parameters – the coordinates of soccer ball in the video, the first frame of the video (which is used as the base output image) and the video file name without the extension (.mp4 or .avi). The output animation is stored in the output folder.

- Animation module available in matplotlib helped create the final animation for each input video file.

Function: create_animation() in util.py code snippet

```
def create_animation(coordinates, image, file_name):

   length,width = image.shape[0], image.shape[1]
    global coords, ax
    coords = smoothen_coords(coordinates, length)
   ax = plt.axes(xlim=(0, width), ylim=(0, length))
   anim = animation.FuncAnimation(fig, animate,
   init_func=init,
   frames=len(coordinates),
   interval=50,repeat=True, blit=True)

    plt.imshow(image, zorder=0, extent=[0.5, width, 0.5, length])
    plt.show()

    anim.save(os.path.join("output", "animation_" + file_name + ".mp4"),fps=len(coordinates),extra_args=['-vcodec', 'h264', '-pix_fmt', 'yuv420p'])

    return
```

Computation: Code Explanation

Function: init() in util.py

This function is the standard prototype used for matplotlib animation module to define the patch center used for the animation.

Function: init() in util.py code snippet

```
def init():

    patch.center = (-200, -200)

    ax.add_patch(patch)

    return patch,
```

Computation: Code Explanation

Function: animate() in util.py

This function is the standard prototype function used for matplotlib animation module to calculate the patch center for all the coordinates (one frame at a time) used for the animation.

Function: animate() in util.py code snippet

```
def animate(i):

   x , y = coords[i]

   if((x != -1) or (y != -1)):

       patch.center = (x + 5, y + 5)

   return patch,
```

Computation: Code Explanation

Function: smoothen_coords() in util.py

- This function takes two inputs, the coordinates of the soccer ball in the video and the width of the image (used as the base output image) and it returns the smoothed out coordinates.
- This function helps eliminate noise created by patch center used by matplotlib animation module and smoothes the soccer ball flow in the animation based on the actual coordinates of the soccer ball in video frame.

Function: smoothen_coords() in util.py code snippet

```
def smoothen_coords(coords1, width):

  coords = coords1.copy()
  coords[:,1] = abs(coords1[:,1] - width)
  fcoords = np.unique((coords > 0).nonzero()[0])

 for idx , val in enumerate(fcoords):

   if(idx < (len(fcoords) - 1)):

      start = fcoords[idx]
      stop = fcoords[idx + 1]
      diff = abs(start - stop)
      xst = coords[start][0]
      xend = coords[stop][0]
      linspX = np.linspace(xst, xend, num=diff)
      yst = coords[start][1]
      yend = coords[stop][1]
      linspY = np.linspace(yst, yend, num=diff)

      coords[start: stop,0] = linspX
      coords[start: stop,1] = linspY

  return coords
```

Computation: Code Explanation

Function: calculate_angle() in util.py

This function takes only one input which is the coordinates of the soccer ball in the input video file.

The process calculate the angle(s) between two n-dimensional vectors which is the coordinates of the soccer ball that was accumulated in the numpy array.

```
angle = math.acos(dotproduct(x, y) / (length(x) * length(y)))

angle_in_degrees = np.degrees(angle)
```

Where x and y are soccer ball coordinates from each frame contained in the numpy array.

Function: calculate_angle() in util.py code snippet

```
def calculate_angle(coordinates):

    x, y = coordinates[:,0], coordinates[:,1]

    try:

       angle = math.acos(dotproduct(x, y) / (length(x) * length(y)))

       angle_in_degrees = np.degrees(angle)

    except:

        angle = 0

        angle_in_degrees = 0

    return angle_in_degrees
```

Computation: Code Explanation

Function: dotproduct() in util.py

This function takes the x, y coordinates of soccer ball in the input video frame as an input and returns the dot product of those two coordinates.

Function: dotproduct() in util.py code snippet

```
def dotproduct(x, y):

  return sum((a*b) for a, b in zip(x, y))
```

Computation: Code Explanation

Function: length() in util.py

This function takes either x or y coordinate of soccer ball in the input video frame as the input and returns the square root of the dot product of that one coordinate.

Function: length() in util.py code snippet

```
def length(v):

  return math.sqrt(dotproduct(v, v))
```

Computation: Code Explanation

Function: main() in get_hsv_code.py

- This function reads a video file using the OpenCV VideCapture function and displays the video in the HSV color space using the cv2.COLOR_BGR2HLS_FULL conversion.

- The user can click on any point on this video to display the corresponding HSV color space of that object in the video frame and also the lower/upper threshold of that HSV color space.

- The user can click on the soccer ball in the video to get the HSV color space for that particular. This is helpful function to use when different color soccer balls are used and also identify the HSV color space of the soccer ball in a noisy video.

Function: main() in get_hsv_code.py code snippet

```
def main():
    capture = cv2.VideoCapture(video_dir)
    while True:
        _, frame = capture.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HLS_FULL)
        if colors:
            cv2.putText(hsv, str(colors[-1]), (10, 50), cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 0), 2)
           cv2.imshow('frame', hsv)
           cv2.setMouseCallback('frame', on_mouse_click, hsv)
          if cv2.waitKey(1500) & 0xFF == ord('q'):
              break
     capture.release()
     cv2.destroyAllWindows()

    minb = min(c[0] for c in colors)
    ming = min(c[1] for c in colors)
    minr = min(c[2] for c in colors)
    maxb = max(c[0] for c in colors)
    maxg = max(c[1] for c in colors)
    maxr = max(c[2] for c in colors)
    print minr, ming, minb, maxr, maxg, maxb
    lb = [minb,ming,minr]
    ub = [maxb,maxg,maxr]
    print lb, ub
```

Computation: Code Explanation

Function: on_mouse_click() in get_hsv_code.py

This function uses open cv function cv2.EVENT_LBUTTONUP to get the coordinates of the point in the video that the user had clicked.

Function: on_mouse_click() in get_hsv_code.py code

```
def on_mouse_click (event, x, y, flags, frame):

    if event == cv2.EVENT_LBUTTONUP:

        colors.append(frame[y,x].tolist())
```

Details: What would you do differently?

One of the features we defnitely wanted to implement if we had more time was to automatically detect any color soccer ball, get it’s corresponding HSVcolor space and the calculate the lower and upper HSV threshold limits. The rest of the pipeline would remain the same.

Incorporating this feature would have enabled this pipeline to be used for any soccer video on the web. More background noise (crowd etc) in the actual world cup soccer games would have been a different task to handle but that would have been a great challenge.

Our team also wanted to try different algorithms for feature matching to get better result. Better feature matching algorithm would have also helped immensely to detect any colored soccer ball and invariant to blurriness of the ball with high speed.

Details: What would you do differently?

Speed calculation currently does not taken into account the depth of the video/image. If there was more time for this final project, we would have put in additional effort to implement some of the concepts of “Depth & Defocus” from Coded Photography. Using these Coded Photography concepts would have helped us accurately calculate the speed of the soccer ball especially when the ball was moving away from the camera in an almost straight line instead of sideways.

Resources

- Matplotlib Animation
- Matplotlib Animation Tutorial
- HSL and HSV Color Space
- PyImageSearch
- Open CV Feature Matching
- Open CV feature matching with ORB descriptors
- Richard Szeliski (2010) Computer Vision: Algorithms and Applications, Springer
- The Laplacian Pyramid as a Compact Image Code, paper by Peter J. Burt and Edward H. Adelson
- .avi to mp4 and .mp4 to .avi convertor
- Convert videos to slow motion

Teamwork (required for teams only)

The original division of labor for the “Soccer Ball Assistant” Project was as follows:

- Take videos of soccer shots in a controlled environment – Balaji Sundaresan
- Coding to break the video file into individual frames – Syed Nasar
- Feature detection (of the soccer ball) and reconstruction of the new image showcasing the path of the ball’s trajectory – Balaji Sundaresan and Syed Nasar both shared the coding responsibilities since this was huge model. This approach was scarped from the final pipeline because of lack of good matching.
- HSV color space threshold detector coding – Balaji Sundaresan
- Blurring Image and Centroid detection – Syed Nasar

Teamwork (required for teams only)

- Mask Creation and Coordinates aggregation – Balaji Sundaresan
- Coding for calculating of Soccer Ball Trajectory Angle – Syed Nasar
- Coding for calculating Soccer Ball Speed – Balaji Sundaresan
- Coding to capture operating system and use different version of video file – Syed Nasar
- Coding for final output image – Balaji Sundaresan
- Coding for final animation – Syed Nasar
- Embedding images and url links to the final report – Balaji Sundaresan
- Embedding code snipperts to the final report – Syed Nasar
- Final report write-up – Balaji Sundaren & Syed Nasar

Teamwork (required for teams only)

The division of labor worked great for our team because of the clear and detailed project scope we had developed before we began this project. The use of github for code sharing and google drive for sharing video files, images and power point presentation facilitated in the colllaboration effort.

We also used Slack for instant communication and Google hangout for team meetings and collaboration.

Teamwork (required for teams only)

- Do you feel that the actual division of labor to develop your project, code, and report was equitable?

✓ Yes, the actual division of labor o develop the “Soccer Ball Assistant” project, code and report was equitable.

✓ My team mate and I resonanted well with the needs of the project and time restrictions that came with such a huge project. This in turn helped us collaborate in writing a detailed project scope and allocation of labor. We had daily meetings in the evening to go over the progress of project and we were constantly clarifying issues and changes using the Slack App and Text messaging. In the end, we had immense fun working together while learning so many things on this project in a short duration.

Artifacts (code & videos)

The code , the input videos, final output images/plots and animation are all available in the following github link:

https://github.com/sbalajis/Final_Project

All the input videos and final output images/plot and animation are also shared in the following google drive link:

https://drive.google.com/folderview?id=0BwrKQqvXmFTuOTFRaGZGY0RuRm8&usp=sharing

Our Code

In the following slides we have copied the code from our project. We have not provided any headers to those slides. In case you want to refer to the actual code, feel free to refer to our Github repository here - https://github.com/sbalajis/Final_Project

soccer_ball_tracker.py

```
# Name: Balai Sundaresan & Syed Nasar
# Email: bsundaresan@gatech.edu & syed.nasar@gatech.edu

import cv2
import numpy as np
from glob import glob
import os, fnmatch
import util
import platform
import math

template_dir = os.path.join("source", "template")
video_dir = os.path.join("source", "videos")
frames_dir = "frames"
output_dir = "output"

# dictionary contains the lower threshold for the videos based on how the shots were taken
lower_dict = {'closeup_shot': np.array([36, 100, 220]), 'long_shot': np.array([30, 50, 182])}

# dictionary contains the upper threshold for the videos based on how the shots were taken
upper_dict = {'closeup_shot': np.array([80, 255, 255]), 'long_shot': np.array([60, 220, 255])}

# dictionary contains the distance in feet from which the video was shot
distance_dict = {'balaji_closeup_shot_1': 25, 'balaji_closeup_shot_2': 25, 'balaji_closeup_shot_3': 30,
                 'balaji_long_shot_1': 30, 'balaji_long_shot_2': 20, 'balaji_long_shot_3': 60,
                 'balaji_long_shot_4': 60}

# dictionary contains the field of fov of the video
fov_dict = {'balaji_closeup_shot_1': 60, 'balaji_closeup_shot_2': 60, 'balaji_closeup_shot_3': 60,
            'balaji_long_shot_1': 60, 'balaji_long_shot_2': 80, 'balaji_long_shot_3': 120,
            'balaji_long_shot_4': 120}
```

```
def track_soccer_ball(image, file_name):
    """ This function reads each frame in the video , creates a mask and then finds the centroid coordinates of the soccer ball in the frame
    Args::
          Image as numpy array
    Returns: (x,y) coordinates of centroid if found
                (-1,-1) if no centroid was found
                None if user hit ESC
    """

    # Blur the image to reduce noise
    blur = cv2.GaussianBlur(image, (5, 5), 0)

    # Convert BGR to HSV
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    # Define threshold the HSV image for the chosen color

    # FORMULA
    # sensitivity = 15
    # lower_white = np.array([0,0,255-sensitivity])
    # upper_white = np.array([255,sensitivity,255])

    if file_name.find("closeup_shot") != -1:
        lower_threshold = lower_dict.get("closeup_shot")
        upper_threshold = upper_dict.get("closeup_shot")
    elif file_name.find("long_shot") != -1:
        lower_threshold = lower_dict.get("long_shot")
        upper_threshold = upper_dict.get("long_shot")
    else:
        pass
```

```
# threshold the HSV image to get only for the chosen color and create the mask
mask = cv2.inRange(hsv, lower_threshold, upper_threshold)

# blur the mask
bmask = cv2.GaussianBlur(mask, (5, 5), 0)

# Take the moments to get the centroid
moments = cv2.moments(bmask)
m00 = moments['m00']
centroid_x, centroid_y = None, None
if m00 != 0:
    centroid_x = int(moments['m10'] / m00)
    centroid_y = int(moments['m01'] / m00)

# initialize centroid
ctr = (-1, -1)

# use centroid if it exists
if centroid_x != None and centroid_y != None:
    ctr = (centroid_x, centroid_y)

    # put black circle in at centroid in image
    cv2.circle(image, ctr, 4, (0, 0, 0))

# force image display, setting centroid to None on ESC key input
if cv2.waitKey(1) & 0xFF == 27:
    ctr = None
return ctr
```

```
# threshold the HSV image to get only for the chosen color and create the mask
mask = cv2.inRange(hsv, lower_threshold, upper_threshold)

# blur the mask
bmask = cv2.GaussianBlur(mask, (5, 5), 0)

# Take the moments to get the centroid
moments = cv2.moments(bmask)
m00 = moments['m00']
centroid_x, centroid_y = None, None
if m00 != 0:
    centroid_x = int(moments['m10'] / m00)
    centroid_y = int(moments['m01'] / m00)

# initialize centroid
ctr = (-1, -1)

# use centroid if it exists
if centroid_x != None and centroid_y != None:
    ctr = (centroid_x, centroid_y)

    # put black circle in at centroid in image
    cv2.circle(image, ctr, 4, (0, 0, 0))

# force image display, setting centroid to None on ESC key input
if cv2.waitKey(1) & 0xFF == 27:
    ctr = None
return ctr
```

```
def readImages(image_dir):
    """ This function reads in input images from a image directory

    Args:
        image_dir (str): The image directory to get images from.

    Returns:
        images(list): List of images in image_dir. Each image in the list is of
                      type numpy.ndarray.
    """
    extensions = ['bmp', 'pbm', 'pgm', 'ppm', 'sr', 'ras', 'jpeg',
                  'jpg', 'jpe', 'jp2', 'tiff', 'tif', 'png']

    search_paths = [os.path.join(image_dir, '*.' + ext) for ext in extensions]
    image_files = sorted(reduce(list.__add__, map(glob, search_paths)))
    images = [cv2.imread(f, cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR)
              for f in image_files]

    return images
```

```
def drawOutput(coordinates, img2, soccer_ball_speed):
    """ This function creates a new output image that concatenates the two images together

    Args:
        soccer ball coordinates in the video, first frame of the viode and soccer ball speed
    Returns:
        none
    """
    #
    # (a.k.a) a montage
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]
    out = np.zeros((rows2, cols2, 3), dtype='uint8')
    out = img2

    # For each pair of points we have between both images
    # draw circles, then connect a line between them
    for pt in coordinates:
        x = pt[0]
        y = pt[1]
        cv2.circle(out, (int(x), int(y)), 20, (0, 255, 0), thickness=2)

        font = cv2.FONT_HERSHEY_SIMPLEX
        txt = str(int(x)) + ":" + str(int(y))
        cv2.putText(out, txt, (int(x) - 20, int(y) - 30), font, 0.5, (0, 0, 255), 2)

    # print soccer ball trajectory on the image
    angle_in_degrees = util.calculate_angle(coordinates)
    angle_in_degrees = round(angle_in_degrees, 2)
    angle_text = "Trajectory of the soccer ball is " + str(angle_in_degrees) + " degrees"
    print("Trajectory of the soccer ball is {} degrees").format(angle_in_degrees)
    print("")
    # print soccer ball speed on the image
    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(out, angle_text, (1, 800), font, 1, (0, 0, 0), 2)
    font = cv2.FONT_HERSHEY_DUPLEX
    speed = "Speed of soccer ball is " + str(soccer_ball_speed) + " mph"
    cv2.putText(out, speed, (1, 900), font, 1, (0, 0, 0), 2)

    # show the image
    cv2.imshow('Matched Features', out)
    cv2.imwrite(os.path.join("output", file_name + ".jpg"), out)
    cv2.waitKey(0)
    cv2.destroyWindow('Matched Features')
    return

def get_speed_specs(capture, file_name):
    """ This function calculates all the techincal specs for speed calculation

    Args:
        captured video, file name
    Returns:
        Number of pixels traversed in a given time by the soccer ball, video run time
    """

    # print video resolution
    image_width = capture.get(cv2.CAP_PROP_FRAME_WIDTH)
    image_height = capture.get(cv2.CAP_PROP_FRAME_HEIGHT)
    image_resolution = [image_width, image_height]
    print("Video resolution Frame Count is {} (width x height)").format(image_resolution)
    print("")
```

```
  # print frames per second
  fps = capture.get(cv2.CAP_PROP_FPS)
  print "Frames per second using video.get(cv2.CAP_PROP_FPS) : {0}".format(fps)
  print("")

  # print frame count
  frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
  print("Total Frame Count is {}").format(frame_count)
  print("")

  # total run time
  video_run_time = round(frame_count / fps, 2)
  print("Total video run time is {}").format(video_run_time)
  print("")

  # get fov for the video
  fov = fov_dict.get(file_name)

# get distance from which the video was shot
  distance = distance_dict.get(file_name)

  # calculate the the width of the image at the distance specified
  frame_width_ft = 2 * (math.tan(math.radians(fov * 0.5)) * distance)
  pixels_travelled = frame_width_ft / float(image_width)

  return pixels_travelled, video_run_time
```

```
def process_video(video_file, file_name):
    """ This function process a given video and ouputs the coordinates and speed of the ball

    Args:
        video_file, file name
    Returns:
        coordinates, soccer_ball_speed
    """
    global video_dir
    coordinates = np.empty((0, 2), int)

    video_path = os.path.join(video_dir, video_file)
    capture = cv2.VideoCapture(video_path)

    # get techincal specs needed to calculate soccer ball speed
    pixels_travelled, video_run_time = get_speed_specs(capture, file_name)

    print("Processing Video... {}").format(video_file)
    print("")
    first = 1

    while True:
        okay, image = capture.read()
        if first:
            cv2.imwrite(os.path.join("output", file_name + "_first.jpg"), image)
            first = 0
        if okay:
            centroid = track_soccer_ball(image, file_name)
            if not centroid:
                break
            else:
                coordinates = np.vstack((coordinates, np.array([[centroid[0], centroid[1]]])))
            if cv2.waitKey(1) & 0xFF == 27:
                break
        else:
            print("End of Video {}").format(video_file)
            print("")
            break
```

```
   # calculate speed
    pixel_range = abs(coordinates[0, 0] - coordinates[-1, 0])
    # use constant value 0.681818 for feet and mph and a constant value 3.6 for meters and kph
    soccer_ball_speed = ((pixel_range * pixels_travelled) / video_run_time) * 0.681818
    soccer_ball_speed = round(soccer_ball_speed, 2)
    print("Speed of the soccer ball is {} mph").format(soccer_ball_speed)
    print("---------------------------")
    print("")
    return coordinates, soccer_ball_speed


if __name__ == '__main__':

    # remove all existing images in out directory
    files = glob('output/*')
    for f in files:
        os.remove(f)

    # Check OS, windows only supports .avi files and not mp4 (mostly)
    if platform.system() == 'Windows':
        file_extension = "avi"
    else:
        file_extension = "mp4"

    video_files = fnmatch.filter(os.listdir(video_dir), '*.' + file_extension)

    for video_file in video_files:
        file_name = video_file.split('.')[0]
        coordinates, soccer_ball_speed = process_video(video_file, file_name)
        # load the first frame in the video
        image = cv2.imread(os.path.join("output", file_name + "_first.jpg"), cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR)
        # draw the ball trajectory on the first frame
        drawOutput(coordinates, image, soccer_ball_speed)
        util.create_animation(coordinates, image, file_name)

    print("Process completed successfully")
```

util.py

```
# Name: Balai Sundaresan & Syed Nasar
# Email: bsundaresan@gatech.edu & syed.nasar@gatech.edu

from matplotlib import pyplot as plt
from matplotlib import animation
import os
import math
import numpy as np
import cv2
import StringIO
import pickle



fig = plt.figure()
fig.set_dpi(100)
fig.set_size_inches(7, 9)

ax = plt.axes(xlim=(0, 200), ylim=(0, 200))
patch = plt.Circle((-200, -200), 20.75, fc='y')

coords = []



def init():
    patch.center = (-200, -200)
    ax.add_patch(patch)
    return patch,

def animate(i):
   x , y = coords[i]
   if((x != -1) or (y != -1)):
       patch.center = (x + 5, y + 5)
   return patch,
def smoothen_coords(coords1, width):

  coords = coords1.copy()
  coords[:,1] = abs(coords1[:,1] - width)


  fcoords = np.unique((coords > 0).nonzero()[0])

  for idx , val in enumerate(fcoords):
    if(idx < (len(fcoords) - 1)):
      start = fcoords[idx]
      stop = fcoords[idx + 1]
      diff = abs(start - stop)

      xst = coords[start][0]
      xend = coords[stop][0]

      linspX = np.linspace(xst, xend, num=diff)

      yst = coords[start][1]
      yend = coords[stop][1]

      linspY = np.linspace(yst, yend, num=diff)

      coords[start: stop,0] = linspX
      coords[start: stop,1] = linspY

  return coords
```

```
def create_animation(coordinates, image, file_name):
    length,width = image.shape[0], image.shape[1]

    #print "length,width = ", length,width
    global coords, ax
    coords = smoothen_coords(coordinates, length)

    ax = plt.axes(xlim=(0, width), ylim=(0, length))

    anim = animation.FuncAnimation(fig, animate,
                               init_func=init,
                               frames=len(coordinates),
                               interval=50,repeat=True,
                               blit=True)
    plt.imshow(image, zorder=0, extent=[0.5, width, 0.5, length])
    plt.show()
    anim.save(os.path.join("output", "animation_" + file_name + ".mp4"),fps=len(coordinates),extra_args=['-vcodec', 'h264', '-pix_fmt', 'yuv420p'])
    return


def dotproduct(x, y):
  return sum((a*b) for a, b in zip(x, y))

def length(v):
  return math.sqrt(dotproduct(v, v))

def calculate_angle(coordinates):
     x, y = coordinates[:,0], coordinates[:,1]
     try:
        angle = math.acos(dotproduct(x, y) / (length(x) * length(y)))
        angle_in_degrees = np.degrees(angle)
     except:
         angle = 0
         angle_in_degrees = 0
     return angle_in_degrees
```

```
if __name__ == '__main__':
  print("Testing util")
  image = cv2.imread("output/balaji_closeup_shot_1.jpg", cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR) # Read in color

  with open('output/coordinates_balaji_closeup_shot_1.pickle', 'rU') as handle:
    coords = pickle.load(handle)

  create_animation(coords, image, "testfile")
```

get_hsv_code.py

```
# Name: Balai Sundaresan & Syed Nasar
# Email: bsundaresan@gatech.edu & syed.nasar@gatech.edu

import cv2
import numpy as np
import os

video_dir = "source/videos"
video_file = "balaji_long_shot_4.avi"
video_dir = os.path.join(video_dir, video_file)
colors = []

def on_mouse_click (event, x, y, flags, frame):
    if event == cv2.EVENT_LBUTTONUP:
        colors.append(frame[y,x].tolist())


def main():
    capture = cv2.VideoCapture(video_dir)

    while True:
        _, frame = capture.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HLS_FULL)
        if colors:
            cv2.putText(hsv, str(colors[-1]), (10, 50), cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 0), 2)
        cv2.imshow('frame', hsv)
        cv2.setMouseCallback('frame', on_mouse_click, hsv)

        if cv2.waitKey(1500) & 0xFF == ord('q'):
            break

    capture.release()
    cv2.destroyAllWindows()
    minb = min(c[0] for c in colors)
    ming = min(c[1] for c in colors)
    minr = min(c[2] for c in colors)
    maxb = max(c[0] for c in colors)
    maxg = max(c[1] for c in colors)
    maxr = max(c[2] for c in colors)
    print minr, ming, minb, maxr, maxg, maxb

    lb = [minb,ming,minr]
    ub = [maxb,maxg,maxr]
    print lb, ub

if __name__ == "__main__":
    main()
```

feature_matching.py

```
# Name: Balai Sundaresan & Syed Nasar
# Email: bsundaresan@gatech.edu & syed.nasar@gatech.edu


import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from glob import glob
import time

# Import ORB as SIFT to avoid confusion.
try:
  from cv2 import ORB as SIFT
except ImportError:
  try:
    from cv2 import SIFT
  except ImportError:
    try:
      SIFT = cv2.ORB_create
    except:
      raise AttributeError("Version of OpenCV(%s) does not have SIFT / ORB."
                      % cv2.__version__)


# This magic line appears to be needed with OpenCV3 to prevent the feature
# detector from throwing an error...
cv2.ocl.setUseOpenCL(False)

template_dir = "source/template"
video_dir = "source/videos"
frames_dir = "frames"
output_dir = "output"
```

```
def print_error(code):
  if code == 1:
    print "Error: NOTE - Make sure proper versions of ffmpeg or gstreamer is installed. \n\
    Sometimes, it is a headache to work with Video Capture mostly due to wrong installation of ffmpeg/gstreamer.\n\
    Help url: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html"
  return


def save_image(fullName, img):
  cv2.imwrite(fullName, img)
  return

def save_plot(plt, plotName):

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    plt.savefig("{}/{}.png".format(output_dir, plotName))   # save the figure to file
    return

def findMatchesBetweenImages(image_1, image_2):
  """ Return the top 4 list of matches between two input images.

  This function detects and computes SIFT (or ORB) from the input images, and
  returns the best matches using the normalized Hamming Distance.
  """
  # matches - type: list of cv2.DMath
  matches = None
  # image_1_kp - type: list of cv2.KeyPoint items.
  image_1_kp = None
  # image_1_desc - type: numpy.ndarray of numpy.uint8 values.
  image_1_desc = None
  # image_2_kp - type: list of cv2.KeyPoint items.
  image_2_kp = None
  # image_2_desc - type: numpy.ndarray of numpy.uint8 values.
  image_2_desc = None
```

```
# Initiate SIFT detector

orb = cv2.ORB_create(nfeatures=10,scaleFactor = 1.2, nlevels=20, edgeThreshold=32, WTA_K = 2,patchSize=32)
#orb = cv2.ORB_create() # cardboard

#  Get keypoints and descriptors for both images
image_1_kp, image_1_desc = orb.detectAndCompute(image_1,None)
image_2_kp, image_2_desc = orb.detectAndCompute(image_2,None)

# Create a Brute Force Matcher, using the hamming distance (and set crossCheck to true)
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Compute the matches between both images
matches = bf.match(image_1_desc,image_2_desc)

# Flann testing
#flann_params = dict(algorithm=1, trees=4)
#flann = cv2.FlannBasedMatcher()
#matches = flann.knnMatch(image_1_desc,image_2_desc,k=2)

# Knn testing
#bf = cv2.BFMatcher()
#matches = bf.knnMatch(image_1_desc,image_2_desc,k=2)
### Here the filtering attempt ###

# Task 4: Sort the matches based on distance so you get the best matches
matches = sorted(matches, key = lambda x:x.distance)

# Task 5: Return the image_1 keypoints, image_2 keypoints, and the top 10 matches in a list
matches = matches[:4]

return matches, image_2_kp, image_1_kp
```

```
def extract_ball_location(parent_image, images):
    """
    Returns a numpy array consisting of coordinates for the ball location in each frame.
    """
    coordinates = np.empty((0,2),int)
    for index, image in enumerate(images):
        matches, ball_coordinates, template_coordinates = findMatchesBetweenImages(parent_image, image)
        coordinates_list = []
        for match in matches:
            # Get the matching keypoints for each of the images
            img1_idx = match.queryIdx
            srcPoint = template_coordinates[img1_idx].pt
            img2_idx = match.trainIdx
            dstPoint = ball_coordinates[img2_idx].pt
            (x,y) = ball_coordinates[img2_idx].pt
            diff = abs(srcPoint[1] - dstPoint[1])
            if( diff < threshold):
                coordinates_list.append((x, y))

 if coordinates_list:
            avg_coordinates = average_of_tuples(coordinates_list)
            coordinates = np.vstack((coordinates, np.array([[avg_coordinates[0], avg_coordinates[1]]])))

    # END OF FUNCTION.
    return coordinates

def average_of_tuples(xs):
    """
    Returns an average of list of tuples
    """
    N = float(len(xs))
    return tuple(sum(col)/N for col in zip(*xs))
```

```
def remove_noise(images):
    """
    Apply erosion and dilation to avoid noise using erode() and dilate() functions
    """
    output = np.zeros((len(images), images[0].shape[0], images[0].shape[1],
                  images[0].shape[2]), dtype=np.uint8)

    # WRITE YOUR CODE HERE.

    # END OF FUNCTION.
    return output

def readImages(image_dir):
    """ This function reads in input images from a image directory

    Args:
        image_dir (str): The image directory to get images from.

    Returns:
        images(list): List of images in image_dir. Each image in the list is of
                  type numpy.ndarray.
    """
    extensions = ['bmp', 'pbm', 'pgm', 'ppm', 'sr', 'ras', 'jpeg',
              'jpg', 'jpe', 'jp2', 'tiff', 'tif', 'png']

    search_paths = [os.path.join(image_dir, '*.' + ext) for ext in extensions]
    image_files = sorted(reduce(list.__add__, map(glob, search_paths)))
    images = [cv2.imread(f, cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR)
          for f in image_files]

    return images
```

```
def get_frames(video_dir, out_dir):

    print "Getting frames from", video_dir
    camera = cv2.VideoCapture(video_dir)
    print("Total Frame Count is {}").format(int(camera.get(cv2.CAP_PROP_FRAME_COUNT)))
    print ""
    count = 0

    success,frame = camera.read()
    if(success == False):
      print "Error: Camera read failed"
      print_error(1)
      exit(-1)
    while success:
        if (frame == None):
          print "Error: No frame found"
          print_error(1)
          exit(-1)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imwrite(out_dir + str(count) + ".png", frame)
        time.sleep(0.1)
        # grab the current frame
        success, frame = camera.read()
        count += 1
    camera.release()
    cv2.destroyAllWindows()
    return
```

```
def make_transparent(arr):
    """
    Linear normalization
    http://en.wikipedia.org/wiki/Normalization_%28image_processing%29
    """
    arr = arr.astype('float')
    # Do not touch the alpha channel
    for i in range(3):
        minval = arr[...,i].min()
        maxval = arr[...,i].max()

        arr[...,i] = arr[...,i] * 0.009
        #if minval != maxval:
        #    arr[...,i] -= minval
        #    arr[...,i] *= (255.0/(maxval-minval))
    return arr
def drawOutput(coordinates, img2):

    # Create a new output image that concatenates the two images together
    # (a.k.a) a montage
    rows2 = img2.shape[0]
    cols2 = img2.shape[1]

    out = np.zeros((rows2,cols2,3), dtype='uint8')
    out = make_transparent(img2)
    out = img2
    #out.fill(255)
    # For each pair of points we have between both images
    # draw circles, then connect a line between them
    for pt in coordinates:
        #print "pt", pt
        x = pt[0]
        y = pt[1]
        cv2.circle(out, (int(x),int(y)), 20, (255, 0, 0), thickness = 2)

        font = cv2.FONT_HERSHEY_SIMPLEX
        txt = str(int(x)) + ":"+ str(int(y))
        cv2.putText(out, txt , (int(x) - 20,int(y) - 30), font, 0.5 ,(0,255,255), 2)
        #cv2.circle(out, (int(x2)+cols1,int(y2)), 4, (255, 0, 0), 1)

        #cv2.line(out, (int(x),int(y)), (int(x2)+cols1,int(y2)), (255, 0, 0), 1)


  # Show the image
    cv2.imshow('Matched Features', out)
    cv2.imwrite('Matched_Features.jpg', out)
    cv2.waitKey(0)
    cv2.destroyWindow('Matched Features')

    # Also return the image if you'd like a copy
    #return out
```

```
if __name__ == "__main__":

    video_file = "balaji_1.avi"
    template_name = "green.jpg"
    parent = os.path.join(template_dir, template_name)
    out_dir = os.path.join(frames_dir, "frame_")
    #video_dir = os.path.join(os.getcwd(), "videos", video_file)
    video_dir = os.path.join(video_dir, video_file)
    # Remove all existing images in out directory
    files = glob('frames/*')
    for f in files:
        os.remove(f)


    # Get frames from video and load image list
    get_frames(video_dir, out_dir)

    # Load list of images in out directory
    images = readImages(frames_dir)
```

```
# Load parent image used to match
parent_image = cv2.imread(parent, cv2.IMREAD_UNCHANGED | cv2.IMREAD_COLOR) # Read in color
parent_image = cv2.imread(parent, 0) # Read in gray scale

# Get the parent image dimensions
parent_height, parent_width = parent_image.shape[:2]
thresFactor = 0.5
threshold = (parent_height + parent_width) * thresFactor
print "threshold = ", threshold, parent_height, parent_width


# Get average of coordinates of the football in each frame
coordinates = extract_ball_location(parent_image, images)
print coordinates
drawOutput(coordinates, images[0])
x, y = coordinates[:,0], coordinates[:,1]
#plt.plot(x,y)

plt.scatter(x, y,  alpha=0.5)
save_plot(plt,"trajectory")
plt.show()
print("Process completed successfully")
"""
from mpl_toolkits.mplot3d.axes3d import Axes3D
xs, ys = np.meshgrid(x, y)
zs = xs*2 + ys*2
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(xs, ys, zs, rstride=1, cstride=1)
plt.show()
"""
```

Credits and Thanks

- I would like to thank Farzaan Nasar, my son, who was the first one to suggest taking up this project. He is a huge soccer fan plus he plays soccer regularly.
- He also modelled for some of the videos, that we initially used for feature detection, but could not use in our final videos.
- My teammate’s girlfriend Daniela Lisnic who helped shoot the videos.