CSCI 1430 Final Project Report
SubwaySurfer.CV

Pixel Pushers: Arun Kavishwar, Muskaan Patel, Pavani Nerella, Shipra Priyadarshini
Brown University

Abstract

Subway Surfers and Flappy Bird are 21st century cultural phenomenons. However, they are limited in that they require a user to be focused on a screen, which contributes to worldwide sedentary practices. We intend to remedy this by forcing full upper body movement to move your character. While we recognize this reduces accessibility, something that is typically frowned upon in our modern age, we are aiming to increase general mobility in a fun format using computer vision.

1. Introduction

Computer games are the most popular entertainment in modern societies and they target a variety of people of different ages. Given its user base and market, the field and genre of video games have made incredible advancements in the past decades. While there are cognitive benefits to gaming, such as better control of one's attention and improved spatial reasoning, there's a lot of discussion about the concerns towards online games encouraging harmful sedentary practices as discussed in the this article by Harvard. This further leads to health issues and injuries that come from activities that involve repeated use of muscles and tendons, causing pain and inflammation to people.
While all this points to the downside of gaming, recently there have been developments that separates the gamers from their screens whilst making it an immersive experience. For decades, generating visual feedback in electronic games involved interaction with a user interface or input device – such as a joystick, controller, keyboard, or motion sensing device. Newer innovations include moving from the traditional flat screen approaches to 3D game playing, ranging from camera relay to fancy VR lens. Our project is one such application of real time adaptive game playing. Specifically, playing popular mobile games by using a camera relay instead of devices.

This is an implementation of Subway Surfer, an endless runner video game using MoveNet for pose detection and estimation in images and videos. The game starts by tapping the touchscreen, while Jake (the game's starter character) or any other character sprays graffiti on a subway, and then gets caught in the act by the inspector and his dog, who starts chasing the character. While running, the player can swipe up, down, left, or right to avoid crashing into oncoming obstacles, including moving subways, poles, tunnel walls and barriers. The aim in this implementation is to play the game in real life, where upper body actions are relayed through the camera and translated into actions to play the game.

2. Related Work

The initial problem was of recognizing the player's pose on the screen, which included having to translate different positions of the player's body parts onto the screen. On these grounds, we looked into the concept of Pose estimation that refers to computer vision techniques to help detect human figures in images and videos. PoseNet, our model of choice created by Tensorflow, is very lightweight and can run on any webcam-equipped desktop, phone, or right from within a web browser. It is a simple model that helps estimate where key body joints are. PoseNet helps mapping the eyes, hands and other such features as required in our implementation.

This led to investigating a more advanced version called MoveNet, a more advanced pose-detection model in TensorFlow, which detects 17 keypoints of a body for pose recognition and estimation while also offering two ultra fast variants, known as Lightning and Thunder. In our implementation we incorporate the Lightning version due to it's ability to perform well in latency-critical applications, as opposed to Thunder, which focuses more on accuracy at the cost of speed. The original MoveNet Tensorflow implementation is referenced at MoveNetTF and BlogTF. Another site that helped us with PoseEstimation. Kaggle lend us a hand with Humanpose rec. One can watch the lightning version of this model be built in a tutorial here: MoveNet Lightning Youtube.

3. Method

The real time adaptive game-playing in this implementation involves detecting the player's pose and simultaneously integrating the pose changes into actions on screen, to translate into moves in the game. The first problem, detecting the player's pose, is solved by using MoveNet, a model that helps with pose estimation. The second problem, triggering actions based on specific poses, utilized frame-based pixel values crossing predefined axes. Each of these will be discussed in detail below.

We attempted to make these axes/lines inferred with RAFT, rather than hardcoded, but met many challenges on that front. Each of these will be discussed in detail below.

3.1. MoveNet

MoveNet is an ultra fast and accurate model that detects 17 features of a body. It uses heatmaps to accurately localize human keypoints. It's a bottom-up estimation model, which means that, for every person on screen, it first detects the human joints of all persons, and then assembles these joints into poses for each person. When a picture or video comes in as input to the pose estimator model, it identifies the coordinates of those detected body parts as output and a confidence score indicating continuity of the estimations. Further, if it is unable to see a body feature (off screen, or hidden by something like hair), it estimates where it should be in relation to what it currently sees. Segmentation divides the pixels in the video frame into two subsets of the foreground target and the background region, and generates the object segmentation mask, which is the core problem of behavior recognition and video retrieval. In our implementation, we downloaded the trained weights and architecture and used it for our pose estimation. We used openCV to access video streams in Python.

```
1   def draw_connections(frame,
        keypoints, edges,
        confidence_threshold):
2   y, x, _ = frame.shape
3   shaped = np.squeeze(np.multiply(
        keypoints, [y,x,1]))
4   cv2.line(frame, (int(0), int(y *
        threshold)), (int(x), int(y *
        threshold)), (255,0,0), 4)
5   for edge, color in edges.items():
6       p1, p2 = edge
7       y1, x1, c1 = shaped[p1]
8       y2, x2, c2 = shaped[p2]
9
10      if (c1 > confidence_threshold)
            & (c2 > confidence_threshold
            ):
11          cv2.line(frame, (int(x1),
                int(y1)), (int(x2), int(
                y2)), (0,0,255), 2)
```

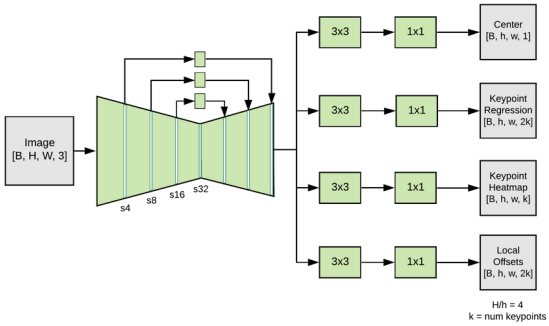

Figure 1. The MoveNet Architecture

3.2. Segmenting the screen to trigger events

In order to effectively analyze and use a stream of video data, it is important to automatically segment and track the objects of interest in the video. Video object segmentation and tracking are two basic tasks in field of computer vision. Since we had access to the keypoint locations in 3D space with MoveNet, we were able to grab specific values on an axis. We created thresholds in the 2D plane that acted as "triggers" for actions. For example, a horizontal line being crossed from below to above might trigger a "jump" action.

3.3. Flappy Bird Implementation

Putting the above concepts in practice, we were able to achieve smooth gameplay where the player's pose was detected and mapped to points (green dots) on the screen and any movement past the threshold lines translated into moves.

To begin with this implementation, we initially tried a simpler game that needed just one component to initiate the game playing. Flappy Bird is one such game in which the player controls the bird "Faby" to navigate through pairs of pipes that have equally sized gaps placed at random heights. Faby automatically descends and only ascends when the player taps the touchscreen and the ultimate goal is to keep it flapping and alive while getting past the obstacles. This "tap" action was the single interaction that needed to be translated. Hence we worked on getting a horizontal line to work as a threshold to trigger the ascend action for the bird.

This code snippet highlights the threshold-based algorithm we used to create event triggers. We were able to grab specific keypoints mapped to body parts by creating a bodyPartIndex. Pixel values are [y, x, z], so we're indexing

into the zeroth element because our threshold is a horizontal line.

```
    if prevFrame < threshold and
        keypoints_with_scores[0][0][
        bodyPartIndex][0] > threshold:
         timesFlapped += 1
         pg.click()
         print("FLAPPED! " + str(
             timesFlapped) + " times.")
```

Here's our team trying to play flappy bird, as you can see each time the player's nose points crosses the horizontal line, it is considered one flap that in turn triggers the bird to ascend the the game.

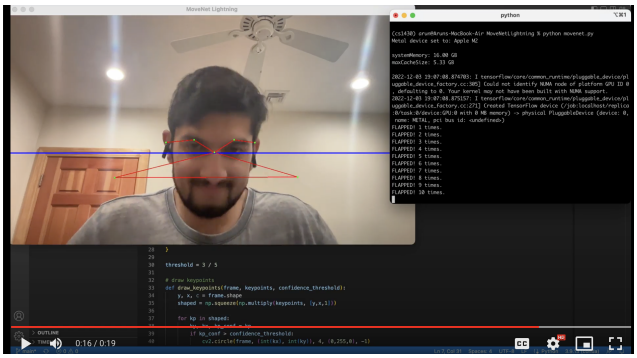

Figure 2. Flappy Bird: Flap count on the nose dropping below the line

3.4. Extension to Subway surfers

With the successful working of the action triggers on screen, we were able to go further and augment this for the more advanced moves that Subway Surfers required. In this implementation we had a pair of horizontal and vertical lines each, dividing the screen into nine sub areas. Our player is required to be in the center tile at the beginning of the game. The left and right vertical lines are used to make movement actions in the game if motion is detected from center to across the threshold line. Whereas, the top and bottom horizontal lines are used to make the character jump above a train or obstacle and duck in similar instances respectively. The code snippet below points to the same.

```
    # slide -> up to down
    if prevSlideFrame < slideThreshold
        and keypoints_with_scores[0][0][
        slideIndex][0] > slideThreshold:
         pg.press('s')
         print("Slide!")

    # jump -> down to up
    if prevNoseFrame > noseThreshold
        and keypoints_with_scores[0][0][
        noseIndex][0] < noseThreshold:
         pg.press('w')
         print("Jump!")
```

3.5. Experiments

As our problem dealt with have to detect a moving player/object on camera and have it translated into game moves, we looked into various other concepts on how to implement this. This section describes some of those attempts, including pixel velocity and RAFT.

For our first task of identifying the player on camera and tracking their motion, we came across a paper [1], which did video object segmentation by exploiting motions using the optical flow of objects and trained the model unsupervised. The concept of optic flow was that elements tend to be perceived as a group if they move in the same direction at the same rate. But this approach had it's challenges when it came real time video and latency. After looking and experimenting a bit we dropped the case and took up the threshold triggers as the initial game (Flappy Bird) only required single moves to be relayed one input at a time.

Another experiment in the same lines was to use RAFT aka Recurrent All-Pairs Field Transforms. It is another deep learning method to solve optical flow in an iterative manner. This paper [2] proposed a even more complex method of extracting pixel features and builds a multi-scale 4D correlation volumes for all pairs of pixels and then iterative updates the flow. We discovered RAFT quite late, and pursued it as a potential way to remove the hard-coded thresholds. Though, this model's results were real good and it was quite an interesting topic to learn and read about, we weren't able to go all the way to implement it given the time constraint.

4. Results

Pose recognition implemented with MoveNet helped us map the eyes, nose, and wrists as green dots which are joined to form the player's pose figure. The earlier stages of implementing pose recognition are demonstrated in Figure 3. The Flappy Bird game-playing (Figure 4) was done with single trigger event of crossing the horizontal line. We were able to achieve our target goal of passing a single obstacle with real-time gameplay.

Once we were successful with the Flappy bird , we further tried to subdivide it for 4 threshold lines (Figure 5) to extend our project to the other game, Subway Surfer.

Finally. the subway surfer game-play can be seen in Figure 6. We achieved our target goal of getting to a score of 1500, which means we stayed alive for at least 60sec.
Due to the exponential difficulty increase in-game as time progresses, staying alive this long was a testimony to our smooth implementation and effective relay measures.

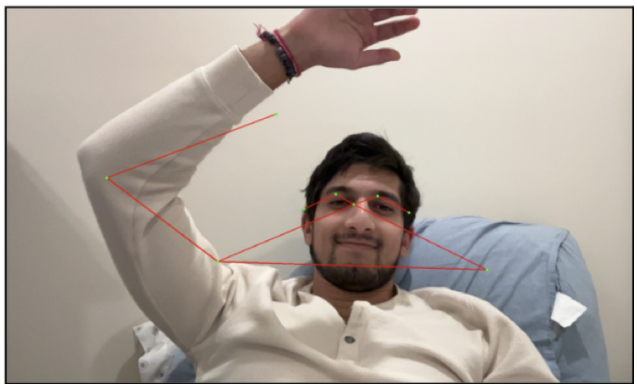

Figure 3. Pose recognition *A*run's eyes,nose and wrist detected in points and lines.

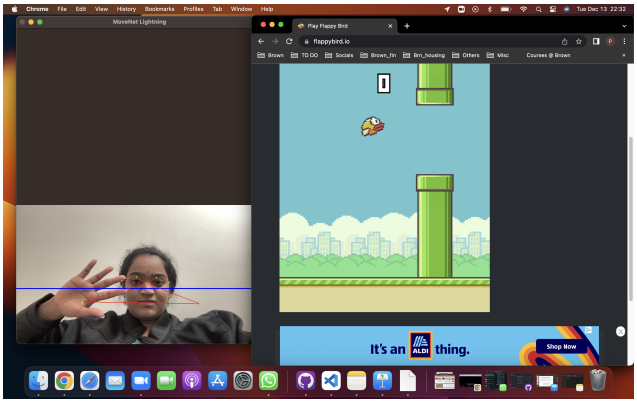

Figure 4. Flappy bird game being played by Pavani

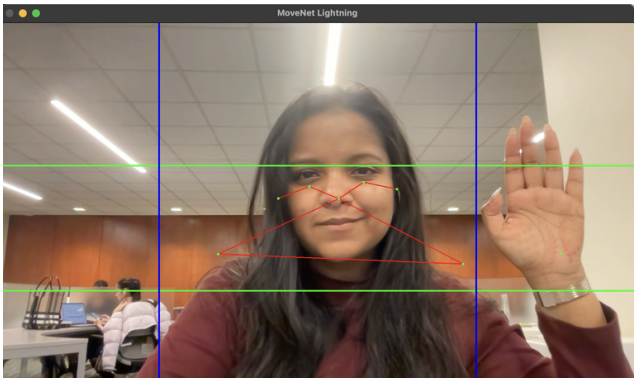

Figure 5. Pose recognition segmentation *M*uskan's pose recognised with screen segmentation thresholds.

4.1. Technical Discussion

Going into this project, we had predicted a number of challenges we would face. Firstly, we were unsure whether we had the hardware capability to do real-time pose estimation to play an infinite-scroller game like Flappy Bird or Subway Surfers. To avoid this hurdle, we did research into a number of pose estimation models– OpenPose, AlphaPose, HRNet, PoseNet, MoveNet Thunder,

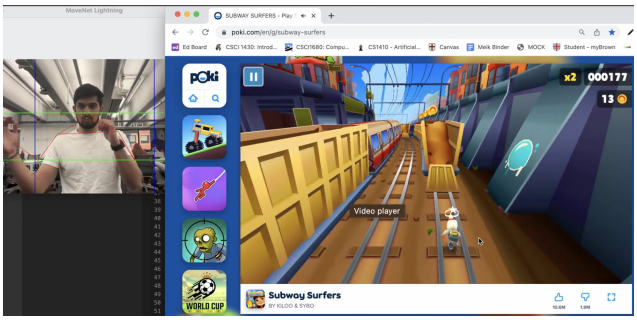

Figure 6. Pose recognition segmentation *M*uskan's pose recognised with screen segmentation thresholds.

and Movenet Lightning. We settled on Movenet Lightning, due to its understandability, effectiveness, and performance in low-latency systems. Additionally, the documentation for this model was great, and allowed us to work at a low level to extract specific keypoints for event triggers. MoveNet is based off of the lightweight MobileNetV2 architecture, with a small classification head for human keypoints. An interesting question is whether (or when) models like MoveNet could be used in critical and widely consumed software applications like CGI development, as we personally witnessed their ability to perform inference incredibly quickly.

A tradeoff we made was reducing project complexity for ease of use. A challenge we faced was determining how to segment the screen. After looking into pixel velocity for a bit, and determining that it added an unnecessary layer of complexity and would likely create jitter and accidental movements in our result, we decided to create a threshold-based solution. The pixel velocity solution, as described in section 3.5, would have made it much easier to play– a player wouldn't have to center their hands after every move. However, codifying the thresholds into the openCV capture, along with working at a low level to extract specific features from the MoveNet model, worked very well and effectively. Though it reduced project complexity and accessibility, it definitely improved performance.

5. Conclusion

In conclusion, we were successful in implementing real-time-adaptive, camera-relayed gameplay for two popular mobile games, Flappy Bird and Subway Surfer. The pose estimation done with MoveNet is translated into event triggers on threshold passes. This compact implementation is smooth, has very low latency and performs well with a single person in frame. Movement Games are a great way to encourage gross motor skill development. Great for use in groups or individually. Through the use of movement games, the following can be encouraged: faster reflexes, motor skills, co-

ordination, balance, body strength. Lastly though, there are some concerns about the potential negative effects of excessive gaming and its addictive nature. Overall, it can provide a fun and engaging way for people to socialize and unwind, especially when made interactive as with this project. As the technology continues to improve, it will be interesting to see how virtual game playing will evolve and what new possibilities it will offer in the future.

6. Broader Social Impact Discussion

This technology allows players to have a more immersive and interactive gaming experience, as they are able to physically interact with their environment and see their own actions reflected in the game. In the current world, people become more and more sedentary as technology makes lives easier. However, there are many health problems associated with a sedentary lifestyle, such as obesity and heart disease, which can cripple lives. Games like ours can be particularly beneficial for individuals who may not otherwise have access to or be interested in traditional forms of exercise. Our game can be easily modified to account for the whole body (i.e. moving both knees up/jumping to jump in game), and that can really create a full body workout. This approach can help people move out of the sedentary lifestyle while still enjoying the games that made them sedentary in the first place.

While Subway Surfers on its own is relatively benign, the technology we are using certainly has broader social implications. The fact that we can track movements to play infinite scroller games in real time is a demonstrative ability of the sheer speed at which these models can inference. This can make applications like weapons much more dangerous. Imagine a weapon that had "auto-lock" functionality, with a camera on its scope that was able to find a person's key point and target that specifically. This would make dangerous weaponry more accessible, removing the barrier of having to aim and creating a more lethal world, especially in countries where gun use is not regulated heavily.

On a happier note, this technology can also be used to improve athletes' skills. For example, a gymnast could perfectly track their body shape as they vault over a bar. They could use the data to see how effective their movement was, avoid fouls, and maybe even create an algorithm to find optimizations that couldn't be found otherwise, such as moving back the position of the pole or creating more torque. This would create a higher skill ceiling for athletes, leading to the advancement of humans in sports and exercise in general.

References

[1] Charig Yang., Hala Lamdouar., Erika Lu., Andrew Zisserman and Weidi Xie, "Self-supervised Video Object Segmentation by Motion Grouping,", Visual Geometry Group, University of Oxford, ICCV 2021 3

[2] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in European conference on computer vision. Springer, 2020, pp. 402–419 3

Team contributions

Arun Kavishwar Worked on the exploring PoseNet and MoveNet implementations to get the player's pose detected along with the threshold triggerring for Flappy bird and it's extension into the Subway Surfer's game and also on the report.

Muskaan Patel Worked on the MoveNet implementation for flappy bird, screen segmentation for both games and research on the the optical flow experiment. Also designed the poster and prepared the presentation.

Pavani Nerella Worked on getting the click triggering to link the event trigger, the subway surfer extension for screen segmentation, reseaching and RAFT code trials. She also got the poster printed and wrote final report.

Shipra Priyadarshini Worked on the RAFT experiment research along with the MoveNet work for the subway surfer screen trigger functions, Flappy bird code debugging along with final report work.