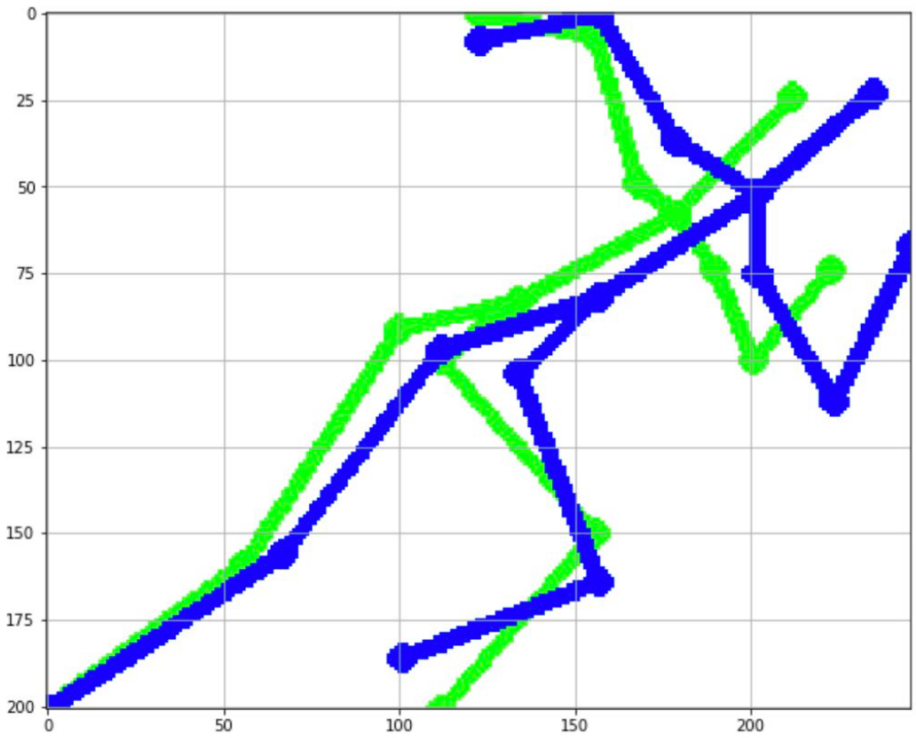
מימוש אלגוריתם השוואה בין פוזיציות אדם

Human Pose Comparison

אריאל ורבין, לירן מנצורי, ניתאי דלגושן



מבוא

אנו מעוניינים באלגוריתם המסוגל למדוד כמה מדויק אדם מחקה את תנועותיו של אדם אחר. ולכן, השלב השני בפרויקט הוא לממש אלגוריתם המקבל שתי פוזיציות ומחזיר ערך המייצג את הדמיון ביניהן.

נבחן גישות שונות לבצע השוואה בין 2 פוזיציות. בסופו של דבר אין מדד שיאמר לנו אם שיטה היא יותר טובה מהשנייה, אך למקרה הספציפי שלנו קיימים מספר דרישות שפונקציית ההשוואה צריכה לממש.

- פונקציית ההשוואה אינה אמורה להיות מושפעת מגודל השלד. כלומר אם אדם יעשה למשל פוזיציה מדויקת אך הוא עומד רחוק מן המצלמה, השלד שלו יראה קטן יותר, אמנם זה לא אמור להשפיע על ערך ההשוואה.

- פונקציית ההשוואה אינה אמורה להיות מושפעת מסיבוב של השלד. כלומר אם אדם יעשה למשל פונקציה מדויקת אך המצלמה שלו מוטה מעט, השלד שלו יראה גם כן מוטה, אמנם זה לא אמור להשפיע על ערך ההשוואה.

- משום שאנו משווים תנועות ריקוד, במקרים רבים רוב הkey-points אכן נמצאים במקום הנכון שלהם (האף, העניים, המותניים וכדומה) אמנם מספיק שהאדם יזיז את ידיו או רגליו בצורה לא נכונה, נרצה לקבוע כי הפוזיציה שהוא עושה לא מתאימה לתנועת הריקוד. במילים אחרות, גם כאשר רוב הפוזיציה שלו נכונה, נרצה לשים יותר דגש בחישוב ההשוואה דווקא בחלקים הלא טובים.

- נרצה למזער ככל הניתן את ההשפעה של מבני גוף שונים על ערך ההשוואה. למשל, אם שני הפוזיציות עושות את אותה הפוזיציה, אך פוזיציה אחת היא של אדם גבוה יותר בעל פרופורציות שונות, נרצה שערך ההשוואה יושפע באופן מינימלי מכך.

נרצה לבחון את תנאים אלו כאשר אנו בוחנים את הגישות השונות, לבסוף נבחר את השיטה המתאימה ביותר.

לאורך הדו"ח אנו משתמשים במושגים "השחקן" ו-"הרקדן". נדגיש כי משימתנו הסופית היא לבנות מערכת ניקוד אשר משווה את תנועות השחקן לתנועותיו של הרקדן, כלומר משווה אותן לתנועות ה-"נכונות".

גישה נאיבית – מרחק אוקלידי

השיטה הראשונה שנראה היא השיטה המתבקשת כאשר חושבים על השוואה בין סט של קואורדינאטות. לפי שיטה זו, נחשב את המרחק האוקלידי בין כל זוג קואורדינטות מתאימות ונחזיר את ממוצע מרחקים אלו.
פורמלית, בהינתן $\{(x_i^1, x_i^2)\}_{i=1}^n, \{(y_i^1, y_i^2)\}_{i=1}^n$ שני פוזיציות (כאשר n הוא מספר הkey-points), המרחק ביניהם יהיה

$$Euclidean(\{(x_i^1, x_i^2)\}_{i=1}^n, \{(y_i^1, y_i^2)\}_{i=1}^n) = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(x_i^1 - y_i^1)^2 + (x_i^2 - y_i^2)^2}$$

כמובן, ששיטה זו אינה עונה על הדרישות שתיארנו ממקודם. שכן הגדלה או סיבוב של השלד ישפיע באופן ישיר על הערך.

נשים לב שניתן לנרמל קודם את הפוזיציות שיהיו בעלות גודל קבוע, ולסובב את השלד כך שהקו המחבר את עיניו יהיה מאונך לציר האנכי (למשל). תוסף זה אכן יספק פתרון עבור שתי הדרישות הראשונות. אמנם, השיטה עדיין שוויונית בין כל מרחקי הkey-points ואינה שמה דגש על חלקים פחות טובים, וכן היא מושפעת באופן רב משוני בפרופורציות של מבנה הגוף של האנשים. לכן, נחפש שיטה אחרת.

גישה ראשונה – Cosine Similarity

השיטה הבאה אותה נבדוק היא Cosine Similarity והיא מתוארת בהקשר של השוואת פוזיציות במאמר של קרישנה ראג' [1]. הרעיון המרכזי בשיטה הזו היא לשטח את השלד לווקטור באורך $2n$, ולחשב את קוסינוס הזווית אשר כלואה בין שני הווקטורים הללו.

פורמלית, בהינתן $\{(x_i^1, x_i^2)\}_{i=1}^n, \{(y_i^1, y_i^2)\}_{i=1}^n$ שני פוזיציות (כאשר n הוא מספר הkey-points), המרחק ביניהם יהיה

$$CosineSimilarity(\{(x_i^1, x_i^2)\}_{i=1}^n, \{(y_i^1, y_i^2)\}_{i=1}^n) = \cos(\theta_{X\to Y}) = \frac{\langle X, Y\rangle}{\|X\| \cdot \|Y\|}$$

כאשר $X = (x_1^1, x_1^2, x_2^1, x_2^2, x_3^1, x_3^2, \dots, x_{17}^1, x_{17}^2),\ A = (y_1^1, y_1^2, y_2^1, y_2^2, y_3^1, y_3^2, \dots, y_{17}^1, y_{17}^2)$.

אם שני הפוזיציות זהות או קרובות אחת לשנייה, גם לאחר שיטוח השלדים הווקטורים יהיו דומים אחד לשני ולכן הזווית $\theta_{X\to Y}$ תהיה קרובה ל0, כלומר הקוסינוס שלה יהיה קרוב ל1. לעומת זאת, ווקטורים מנוגדים (הפוכים לחלוטין) יגרמו לערך הקוסינוס להיות קרוב למינוס 1.

[1] Human Pose Estimation and Action Scoring using Deep Learning, OpenCV & Python, Krishna Raj R, 2020.

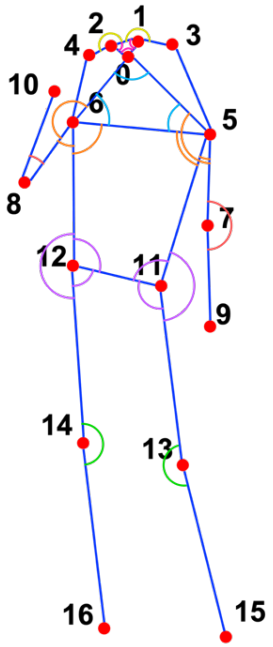
נשים לב שחישוב הזווית בין הווקטורים לא לוקח בחשבון את הנורמה שלהם, ולכן באופן שקול אפשר לחשוב על שיטה זו כאילו נרמלנו את השלדים לפני חישוב ההשוואה. נרמול השלדים בעצם פותר לנו את הדרישה הראשונה. וכן ניתן לסובב את השלדים באופן דומה למה שהצענו בשיטה הקודמת. נשים לב שהפעם שיטה זו כן שמה מעט דגש חזק יותר עבור הטעויות הגדולות בהשוואה בין השלדים, שכן קואורדינטה אחת לא טובה עשויה להיות מספיקה כדי להגדיל את הזווית בין הווקטורים באופן משמעותי.

אמנם, יש מספר חסרונות לשיטה. החיסרון הראשון כמובן הוא שהוא אינו עונה על הדרישה הרביעית שכן שינוי בפרופורציה של השלד תשפיע על הניקוד. עוד חיסרון הוא שאין לנו חופש להגדיר כמה אנו מעוניינים לתת דגש על השגיאות הגדולות, בהשוואה לשגיאות הקטנות. כלומר, מלבד העובדה שהשיטה תתייחס יותר לשגיאות גדולות, היינו רוצים להיות מסוגלים להגדיר לה כמה להתייחס אליהם. חיסרון נוסף אשר שמנו לב אליו במימוש השיטה הוא שהערך שהשיטה מפיקה מאוד רגיש לשינויים בפוזיציה.

גישה שנייה – Angular Method – הרעיון שלנו

לבסוף, בחרנו ללכת עם שיטה שאנו מאמינים שהיא מאוד אינטואיטיבית, והיא להשוות בין שני הפוזיציות על סמך הזוויות שמופקות בין הkey-points השונים. כלומר, נגדיר את הקווים בין זוגות key-points שונים (למשל רגליים וברכיים, ידיים ומרפקים, כתפיים ומותניים וכדומה), ונחשב את הזוויות שבין הקווים הללו.

נשים לב שהשוואה מבוססת זוויות מתעלמת לחלוטין מגודל השלד (שכן רק מעניין אותנו הכיוון של הקווים בהשוואה לקווים אחרים, וגודלם לא משנה), וכן מתעלם לחלוטין מסיבובים של השלד (הזווית שאנו מחשבים היא בין קווים של השלד לבין לקווים אחרים, אז אם מסובבים את כל השלד ביחד, כל הזוויות נשמרות). לכן שתי הדרישות הראשונות מתקיימות אפילו מבלי הצורך לנרמל או לסובב את השלד כחישוב מקדים.



בשביל לממש את הרעיון, ראשית עלינו לבחור את הזוויות אותן נרצה לחשב, עלינו לוודא שאנו לא מפספסים זוויות שעלולות להיות מכריעות בתיאור הפוזיציה.

עוד נשים לב שמשום שהשיטה מבוססת על זוויות, היא כמעט ואינה מושפעת מהפרופורציות של מבנה הגוף של השלדים. ולכן השיטה מותאמת לדרישה הרביעית, שהיא גם מאוד חשובה עבורנו (אנו נשווה את תנועות השחקן לרקדן כלשהו שנצלם מראש, אזי לא נרצה לתת העדפה בניקוד לשחקנים שדומים במבנה הגוף שלהם לרקדן).

אנו בחרנו לחשב 24 זוויות המתוארות באיור. 2 זווית ברגליים, 2 זוויות בידיים, 6 זוויות במותניים, 6 זווית בכתפיים, 3 זוויות בין הכתפיים לאף, 3 זוויות בין העיניים לאף, ו2 זוויות בין העיניים לאוזניים.

הרעיון הראשוני הוא להחזיר את ממוצע ההפרשים בין כל זוג זוויות מתאימות. אולם כאמור שיטה זו לא תענה על הדרישה השלישית שכן אם רוב הזוויות יהיו טובות, זווית אחת לא טובה לא תשפיע בצורה ניכרת.

נדגיש כי המטרה שלנו היא לא לתת ציון מספרי נביאי על ההפרש בין הפוזיציות, אלא לתת ניקוד על תנועות הבנאדם. לכן, נרצה שאם השחקן עושה תנועה לא נכונה הציון שלו יהיה נמוך, ואם הוא עושה תנועה כמעת מדויקת הציון שלו יהיה כמעט מושלם. לכן, חישוב ממוצע רגיל של הפרשי הזוויות לא יביאו לנו תוצאה מספקת, כיוון שגם אם הבנאדם עומד כמו שצריך בהשוואה לרקדן, אך הידיים שלו עושות תנועה לא נכונה – עדיין הוא יקבל ציון גבוה (משום שכל הזוויות שלו ברגליים מדויקות). אמנם בהקשר של מתן ניקוד, השחקן אינו עושה את התנועה הנכונה ולכן אף על פי שזוויות רבות רצו מדויקות, נרצה להביא לו ציון נמוך.

גישה "קיצונית" לפתרון של זה תהיה לקחת את הפרש הזווית המקסימלי בלבד במקום ממוצע של כולם, אמנם כן נרצה לתת התייחסות לשאר הזוויות כמובן ולכן נלך על גישת "אמצע" בין חישוב ממוצע ההפרשים לחישוב ההפרש המקסימלי.

הרעיון, במקום חישוב ממוצע ההפרשים, נחשב את ממוצע ההפרשים בחזקת p, וניקח שורש p. פורמלית, נסמן X וY להיות אוסף הזוויות המתאימות של שני השלדים, אזי

$$AngularMethod(\{(x_i^1, x_i^2)\}_{i=1}^n, \{(y_i^1, y_i^2)\}_{i=1}^n) = \sqrt[p]{\frac{1}{24}\sum_{i=1}^{24} |X_i - Y_i|^p}$$

כאשר p זה קבוע שאנו יכולים לבחור. נשים לב ש$p = 1$ נותן לנו ממוצע רגיל, ו$p \to \infty$ נותן לנו את ההפרש המקסימלי: בה"כ נניח $|X_0 - Y_0|^p$ הוא ההפרש המקסימלי. אזי לפי סנדוויץ',

$$\sqrt[p]{\frac{1}{24}\sum_{i=1}^{24} |X_i - Y_i|^p} \le \sqrt[p]{\frac{1}{24}\sum_{i=1}^{24} |X_0 - Y_0|^p} = \sqrt[p]{\frac{24}{24}\cdot |X_0 - Y_0|^p} = |X_0 - Y_0| \to_{p\to\infty} |X_0 - Y_0|$$

ומנגד,

$$\sqrt[p]{\frac{1}{24}\sum_{i=1}^{24} |X_i - Y_i|^p} \ge \sqrt[p]{\frac{1}{24}|X_0 - Y_0|^p} = \frac{1}{\sqrt[p]{24}}|X_0 - Y_0| \to_{p\to\infty} \frac{1}{1}\cdot |X_0 - Y_0| = |X_0 - Y_0|$$

ולכן, נבחר $p > 1$ חיובי כלשהו. השיטה עונה על הדרישה השלישית, וכן יש לנו חופש בבחירת p כדי להחליט כמה משקל הנוסחה תיתן להפרשים הגדולים בהשוואה להפרשים הקטנים.

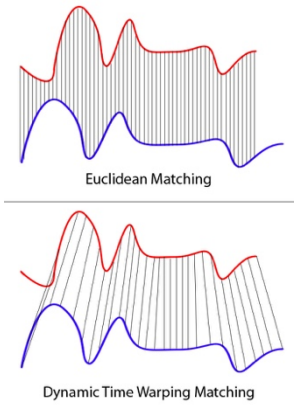
לאחר בחינת השיטות השונות ראינו שהשיטה הזוויתית הייתה טובה במיוחד עבור ערכים של p בין 5 ל8. ולכן בחרנו ללכת איתה.

השוואה של פוזיציה לסדרה של פוזיציות

למרות שיש לנו כעת כלי להשוואה בין 2 פוזיציות, בפועל המטרה הסופית שלנו היא לתת ניקוד לשחקן לאורך זמן המשחק. כלומר, בכל רגע אנו צריכים להשוות את השחקן לרקדן ולהפיק ניקוד.

הגישה הנאיבית להכללה זו תהיה להשוות את פוזיציית השחקן זמן t, לפוזיציית הרקדן בזמן t ולהחזיר ניקוד עבור כל פריים. אמנם, שיטה זו היא מאוד מחמירה שכן היא דורשת מהשחקן תזמון מושלם בהשוואה לרקדן. אנו נרצה להביא לשחקן ניקוד גבוה גם אם התזמון שלו לא מושלם.

בחיפוש אחר גישה להכללה זו נתקלנו ברעיון הנקרא Dynamic Time Warping, או DTW.



DTW הוא אלגוריתם למדידת דמיון בין שני סדרות של נתונים המייצגות רצפים, שעשויות להיות שונות מבחינת מהירות או זמן. המטרה של האלגוריתם הוא להתעלם מהבדלים בתזמון ובעצם "למתוח" או "לדרוס" את הסדרות כדי להתאים אותם בצורה המיטבית, אפילו אם אורכן שונה. DTW הוא אלגוריתם נפוץ במיוחד במשימות כמו מערכות זיהוי דיבור, גילוי תנועה בווידאו וכדומה.

האלגוריתם בעצם לוקח את אחד הסדרות "ומעוות" את התזמון שלה כך שיתאים לסדרה השנייה.

אמנם, יש שני עקרונות אשר הופכים את רעיון זה ללא רלוונטי עבורנו.

- ראשית, DTW הוא אינו אלגוריתם מקוון (online) שכן הוא דורש לקבל את שתי הסדרות במלואן. משום שאנו מעוניינים למצוא אלגוריתם שעובד בreal-time, ונותן לשחקן ניקוד בזמן אמת, אלגוריתם זה אינו מתאים לנו.
- סיבה נוספת לחוסר ההתאמה היא שמטרת האלגוריתם היא להתעלם באופן מכוון משינויים בתזמון בין שתי הסדרות. אמנם אנו כן נרצה לסלוח מעט לאי דיוקים של השחקן בתזמון, עדיין בהקשר שלנו יש חשיבות כלשהי לתזמון של השחקן. בריקוד הזמנים והקצב חשובים ולכן אם השחקן זז מהר מידי או לאט מידי כן נרצה להעניש את הניקוד שלו על כך.

לכן, נעבור לחפש פתרון אחר.

הרעיון שלנו – Time-Windowed Pose Matching

לאחר חיפושים אחר פתרון מתאים החלטנו להציע רעיון אשר קראנו לו Time-Windowed Pose Matching או TWPM בקיצור. אלגוריתם זה מותאם למשימות real-time ונותן לנו חופש בכמה אנו רוצים להעניש שגיאות בתזמונים של השחקן.

אופן החישוב

האלגוריתם, כמו שהשם מרמז, מתבסס על הגדרה של חלון. נגדיר ε להיות אורך החלון (בשניות), ו$\omega: \left[-\frac{\varepsilon}{2}, \frac{\varepsilon}{2}\right] \to \mathbb{R}^+$ להיות פונקציית משקל. דרישה בסיסית של פונקציית המשקל היא שהיא תהיה פונקציה קמורה וכן ש$\omega(0) = 0$.

נסמן S להיות סדרת הפוזיציות של הרקדן, כלומר התנועות "הנכונות" לאורך הריקוד. וכן נסמן את האופרטור $S[t]$ להחזיר לנו את הפוזיציה של הרקדן בסדרת הפוזיציות, שהזמן המתאים בה היא בוצעה הוא הכי קרוב לזמן t. עוד נסמן $S[t_1 : t_2]$ להיות תת סדרה של פוזיציות הרקדן, המכילה את כל הפוזיציות של הרקדן אשר בוצעו בין הזמנים t_1 ו$t_2 > t_1$.

כעת, עבור כל פוזיציה T_t של השחקן המגיעה בזמן t, ניקח את החלון המתאים $S\left[t - \frac{\varepsilon}{2} : t + \frac{\varepsilon}{2}\right]$ של פוזיציות הרקדן שהתרחשו בסביבה של אותו הזמן – t. בעזרת אלגוריתם ההשוואה בין 2 פוזיציות, נשווה בין T_t לכל אחת מן הפוזיציות בחלון הזמן של הרקדן $S\left[t - \frac{\varepsilon}{2} : t + \frac{\varepsilon}{2}\right]$ ונקבל סדרה של תוצאות אותה נסמן $A\left[t - \frac{\varepsilon}{2} : t + \frac{\varepsilon}{2}\right]$. כל איבר בסדרה A הוא תוצאת השוואה בין T_t לפוזיציה כלשהי של הרקדן בחלון.

כעת, נעניש את תוצאות ההשוואה המתאימות לקצוות החלון באמצעות פונקציית המשקל ω. כלומר, לכל ערך השוואה $A[\alpha + t] \in A\left[t - \frac{\varepsilon}{2} : t + \frac{\varepsilon}{2}\right]$ (כאשר $\alpha \in \left[-\frac{\varepsilon}{2} : \frac{\varepsilon}{2}\right]$) נוסיף לו את הערך $\omega(\alpha)$. ולבסוף, עבור זמן t נחזיר את ערך ההשוואה הסופי,

$$TWPM(T_t) \coloneqq \min_{\alpha \in [-\varepsilon/2, \varepsilon/2]} \big(A[\alpha + t] + \omega(\alpha)\big) = \min_{\alpha \in [-\varepsilon/2, \varepsilon/2]} \big(Compare(T_t, S[t + \alpha]) + \omega(\alpha)\big)$$

ניתן לציין את הדרישה שω תהיה פונקציה קמורה, שכן המטרה של פונקציה זו היא להעניש את תוצאות ההשוואה שמתקבלות בקצוות החלון. הרעיון מאחורי העונש הזה הוא להפחית מהציון של השחקן אם התזמון שלו גרוע מספיק, אך במידה מסוימת. הדרישה ש$\omega(0) = 0$ נובעת מכך שההתאמה בין השחקן לרקדן במרכז החלון מייצגת תזמון מושלם ולכן אנו לא אמורים להעניש את הרקדן עבור התאמה זו.

יתרונות השיטה

- **ניקוד בזמן אמת**: סדרת הפוזיציות של הרקדן כמובן שמורה ומוכנה מראש. בכל רגע בריצת האלגוריתם שמגיעה פוזיציה חדשה של השחקן, אנו לוקחים אותה ומפיקים עבורה ניקוד. אין לנו צורך לדעת מה השחקן יעשה בעתיד עבור חישוב הניקוד לזמן הנוכחי. אמנם, נשים לב שאנו כן מסתכלים "לעתיד" בסדרת הפוזיציות של הרקדן – שכן אנו מנצלים את העובדה שהיא דווקא כן נתונה מראש, בניגוד לסדרת הפוזיציות של השחקן.

- **פשוטה ומהירה**: בהתאם לשיטת ההשוואה מבוססת הזוויות שבחרנו בפרק הקודם, ניתן לחשב מראש את הזוויות של כל הפוזיציות של הרקדן כמעין "עיבוד מקדים". ובזמן ריצת האלגוריתם, ניתן לחשב את הזוויות של הפוזיציה הנוכחית של השחקן פעם אחת ויחידה, ולהשתמש בהן לאורך כל חישוב הציונים בחלון. כך אנו נמנעים מחישובים כפולים ומתן הניקוד מתבצע במהירות.

- **חופש בבחירת עונש לטעויות תזמון**: השיטה לוקחת בחשבון ומענישה שגיאות בתזמון של השחקן, וכן יש לנו חופש בבחירת ω. אנו בחרנו להגדיר את בחירת ω בעזרת 3 פרמטרים.
  - `dont_punish` מתייחס לפרק זמן במרכז החלון שבו המשקל יהיה 0, כלומר חלון קטן יותר שאם השחקן נמצא בו אז הניקוד שלו לא יוענש כלל. הפרמטר נמדד בשניות.
  - `shift` מתייחס לכמה נסיט את פרק הזמן של dont_punish שמאלה, שכן אם השחקן טועה זה כנראה כי הוא מאחר ולא מקדים, ונרצה "לסלוח" לאיחורים קטנים של השחקן, שכן הוא מחקה את תנועותיו של הרקדן אז סביר שיהיה לו דיליי קצר. הפרמטר נמדד בשניות.
  - `punish_factor` מתייחס לקצב הגדילה של ω באיזורים שמחוץ לפרק הזמן שבו $\omega = 0$, כלומר כמה אנו רוצים לעניש את הקצוות.

- **התאמה טבעית למחשבים בעלי יכולות חישוביות שונות**: האלגוריתם בכל פעם מחכה שתגיע פוזיציה חדשה של השחקן, וברגע שהיא באה הוא מודד את הזמן שבו היא הגיעה ומחשב את הניקוד. עבור מחשבים איטיים יותר, הביצועים של מודל הלמידה לחישוב הפוזיציה עלולים לקחת יותר זמן, ותדירות ההגעה של פוזיציות חדשות מהשחקן יכולה להשתנות. השיטה מתייחסת לכל פוזיציה של השחקן והזמן שלה בנפרד ובאופן בלתי תלוי בשאר הפוזיציות. כלומר אין לנו צורך במעקב אחרי ביצועי המחשב והתאמה של האלגוריתם לכך, שכן זה מתבצע בצורה טבעית. הזרימה של האלגוריתם מבטיחה שלא יהיה "פקק" של פוזיציות המחכות לקבל ניקוד מה שבעצם שומר על הניקוד בזמן אמת. לכן היא מותאמת למחשבים שונים.

גישה זו אכן עונה על הדרישות שלנו ומתאימה לנו, ולכן נבחר בה.

המרה של ערך ההשוואה לניקוד

משום שמדובר באלגוריתם השוואה, "ציון מושלם" כרגע הוא 0, וגדל ככל שהפוזיציה פחות נכונה. אנו מעוניינים לתת ניקוד (בין 0 ל100) עבור הפוזיציה ולכן נצטרך להפוך את הסקלה. לאחר ניסוי וטעיה ראינו שערך ההשוואה מקבל ערכים באזור 0-15 עבור פוזיציות מושלמות, ובאזור ה50 ומעלה עבור פוזיציה גרועה. לכן, בחרנו מיפוי נאיבי של ערך ההשוואה ל6 ערכים שונים:

```
if comp_value < 15:
    return 100  # Excellent
if comp_value < 20:
    return 90  # Great
if comp_value < 25:
    return 70  # Good
if comp_value < 30:
    return 50  # OK
if comp_value < 40:
    return 10  # Nah

return 0  # X
```

תיאור ששת הערכים השונים (100 – מצוין, 90 – מעולה, 70 – טוב, 30 – בסדר, 20 – לא כל כך, ו- 0 – לא טוב) מתאימים לפידבק שיופיע על המסך של השחקן בזמן המשחק.

ניתוח מהירות השחקן

לאחר בדיקת האלגוריתם שמנו לב לבעיה – משום שהאלגוריתם לא לוקח בחשבון את מהירות השחקן, מתרחש מצב בו גם אם השחקן לא מזיז את גופו כלל, והרקדנית באופן רגעי עשתה פוזיציה שנראית דומה לשחקן, השחקן יקבל ניקוד גבוה באותו הרגע.

אמנם ניתן לטעון כי הניקוד של השחקן מוצדק ברגע הזה שכן הוא עשה פוזיציה דומה לרקדן, ראינו לנכון להתייחס לכך ולמנוע זאת. לשם כך, פיתחנו שיטה למדוד עד כמה השחקן בתנועה בכל רגע. כלומר, עבור כל רגע במשחק אנו מפיקים ערך המייצג כמה השחקן בתנועה. לערך זה קראנו *Movement Score*.

חישוב ציון התנועה

אנו מחשבים את ציון התנועה בהתבסס על המהירות הזוויתית של זוויות שונות בפוזיציית הרקדן. ע"י הנוסחה, $v_\theta = \frac{\Delta\theta}{\Delta t} = \frac{\theta_t - \theta_0}{t}$.

אמנם, שמנו לב לבעיה בזוויות קטנות (למשל זוויות בפנים) בהם המודל מעט רועש בהסקת ה-keypoints. אמנם רעש זה קטן במיוחד, במחשבים מהירים מתעוררת הבעיה שאם הזמן בין ניתוח כל פוזיציה הוא קטן במיוחד, יוצא שאנו מחלקים בt קטן, והרעש הזה מקבל משמעות עצומה.
כדי להתמודד עם זה, בחרנו לחסום את החישוב כך שהמהירות של השחקן תמדד פעם ב90 מילישניות בלבד. כלומר Δt חסום ב0.09.

לאחר שאנו מחשבים את המהירות הזוויתית של כל הזוויות, נחשב את ציון התנועה ע"י,

$$MovementScore = \sqrt[p]{\frac{1}{24}\sum_{i=1}^{24} v_{\theta_i}^{p}}$$

גם כאן, בדומה לאלגוריתם ההשוואה, בחרנו להשתמש בפקטור p כדי שציון התנועה יושפע באופן חזק יותר ממהירויות גבוהות.

עדכון הציון בהתבסס על ציון התנועה

האלגוריתם ישווה את ציון המהירות של השחקן לחלון קצר של ציוני מהירות הרקדן באותו הרגע. הוא ייקח את ציון מהירות הרקדן הדומה ביותר בחלון זה (כדי להתמודד עם רעשים בחישוב ציון המהירות).

אם הפרש ציוני המהירות גבוה מthreshold מסוים, נעניש את ציון השחקן.

ביצוע עיבוד לניקוד

השלב האחרון אותו נצטרך לכסות הוא ביצוע של עיבוד לציון, וחישוב "הציון הכולל" של השחקן. ביצוע העיבוד לניקוד מתחלק לשני חלקים: ייצוב הניקוד (score stabilization) וקירוב הניקוד הכולל (total-score approximation).

Score Stabilization

ייצוב הניקוד מתייחס לשקלול של הניקוד הנוכחי עם ההיסטוריה של מספר הציונים האחרונים שהוציא, והחזרה של ניקוד שמושפע גם מהיסטוריית האחרונה של ציוני השחקן. כך, אם למשל השחקן הוציא ציונים מושלמים ולרגע הניקוד שלו קפץ ל0 (למשל אם הוא התפספס ברגע מסוים) – הרגע הזה כמעט ולא יורגש שכן במקום להחזיר 0, נחזיר שקלול כלשהו של 0 יחד עם כמה מהציונים האחרונים שקיבל.

לייצוב הניקוד יש מספר יתרונות.

- ייצוב של הניקוד יקל על ההבנה של השחקן לגבי הביצועים שלו, אם ציוני השחקן שיוצגו לו בזמן המשחק יותר מידי יקפצו בין ערכים, זה יכול לבלבל את השחקן כך שהוא לא יבין אם הוא רוקד טוב או לא.

- יתרון נוסף הוא כפי שתיארנו, במצבים בהם לרגע השחקן מתפספס במודל הלמידה, במצב כזה השחקן יקבל ניקוד 0. אמנם, לא נרצה להציג לו זאת כיוון שאם השחקן היה עד כה בעל ביצועים מעולים, סביר להניח שהנפילה הזו היא אינה אשמתו. לכן, "נטשטש" את הנפילה הזו בעזרת השקלול שלה עם הציונים הקודמים של השחקן.

- עוד יתרון של ייצוב הניקוד עובד דווקא במקרה ההפוך: במצבים בהם הניקוד של השחקן היה נמוך, ופתאום לרגע הייתה קפיצה לציון גבוה במיוחד. נשים לב שבאופן מתן הציון אנו לא מביאים לידי ביטוי בהשוואה את המהירות והתנועתיות של השחקן. לכן, כחלק מהמאמץ לשמור על הניקוד בזמן אמת – עולה בעיה שיכול להיות רגע בו הרקדן עובר מתנועה אחת לאחרת ובמעבר הפוזיציה של הרקדן נראית כלא עושה כלום. עבור רגע כזה השחקן יקבל ניקוד מושלם גם אם הוא לא רוקד. היינו רוצים לעדן ולטשטש קפיצות חיוביות חדות כאלו בניקוד.

את אופן השקלול של הציון עם הציונים הקודמים נעשה באופן הבא: עבור הזמן הנוכחי, ניקח את כל הציונים שהגיעו עד לפני פחות משתי שניות (למשל). ונחזיר ממוצע ממושקל בין הציון הנוכחי, לכל הציונים בהיסטוריית הציונים הקודמים שהתרחשה ב2 שניות האחרונות.

נשים לב שבחרנו להגדיר כמה ציונים נחשב בשקלול לפי הזמן שבהם הם חושבו, ולא בחרנו מספר קבוע של ציונים (כלומר, היינו יכולים למשל לקבוע שהייצוב יתבצע בעזרת הניקוד הנוכחי ו10 תוצאות הניקוד

האחרונות שחושבו). הסיבה לכך היא שבחירת מספר קבוע של ציונים לא תניב את אותו אפקט הייצוב בין מחשבים עם יכולות חישוב שונות. למשל, במחשבים מהירים, 10 תוצאות ניקוד יכולות להגיע בפרק זמן של חצי שנייה – ותוך חצי שנייה כבר "נשכח" שהיה לשחקן ציונים מעולים לפני רגע. אמנם, במחשבים איטיים יותר, 10 תוצאות ניקוד יכולות להגיע בפרק זמן של 5 שניות. לכן ככל שהמחשב יותר מהיר נרצה לקחת יותר תוצאות ניקוד לחישוב השקלול.

אנו בחרנו למשקל את הציונים כך: הניקוד הנוכחי יקבל משקל של 0.5 בחישוב, וה0.5 הנותרים יתחלקו שווה בשווה בין הציונים האחרונים שהתקבלו בשתי שניות האחרונות.

Total Score Approximation

לאורך המשחק עלינו לחשב את הניקוד הכולל של השחקן, בין היתר כדי לשמור שיאים ולהכריע מי ניצח עבור משחקים של דו-קרב. עלינו לבחור שיטה אוניברסלית שתפיק את הניקוד הכולל בסדר גודל קבוע כלשהו.

נשים לב, למשל, שחיבור פשוט של כל ערכי הניקוד של השחקן לאורך המשחק לא תהווה גישה טובה, שכן עבור מחשבים מהירים יותר, ערכי הניקוד מגיעים בקצב גבוה ולכן סדר הגודל של הניקוד הכולל יהיה גדול מאוד. יש בעיה בכיוון ההפוך עבור מחשבים איטיים. עוד נשים לב שגם בין מחשב לעצמו, כאשר הוא יותר פנוי (פחות תהליכים שצריכים זמן מעבד וכדומה) אזי ערכי הניקוד יבואו בקצב מהיר יותר והשחקן יקבל ניקוד כולל גבוה יותר. ולכן על מנת לשבור שיאים, השחקן יוכל לסגור את כל שאר התוכנות במחשב שלו כדי "לרמות" ולהעלות את קצב יצירת הניקוד. נרצה למנוע זאת.

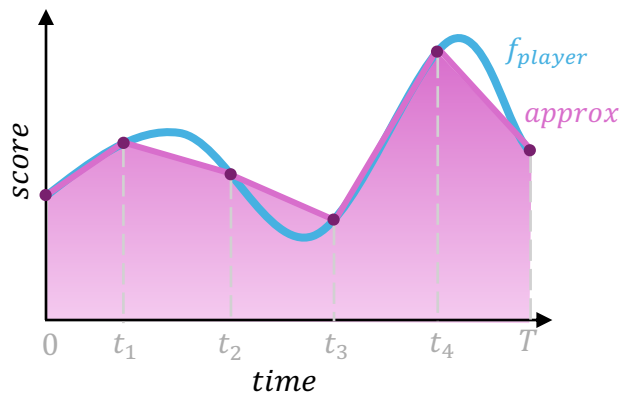
במקום, נחשוב על חישוב הניקוד הכולל כמעין "קירוב" של פונקציה המתארת בצורה רציפה את ביצועי השחקן לאורך המשחק. את הפונקציה הזו אנו לא יכולים לחשב במלואה, אך אנו בעצם "דוגמים" ממנה ערכים בתדירות המתאימה ליכולות החישוביות של המחשב.

כלומר אם נסמן f_{player} להיות הדיוק של השחקן בכל זמן לאורך המשחק, בכל פעם שאנו מקבלים את הפוזיציה הבאה של השחקן בזמן t_i, אנו מחשבים את הניקוד עבור זמן t_i כלומר אנו מחשבים את $f_{player}(t_i)$. האינטרוול בין t_i לt_{i+1} יכול להשתנות בהתאם ליכולת החישובית של המחשב.

לכן, נגדיר את הניקוד הכולל להיות השטח שמתחת לגרף f_{player}, כלומר אם המשחק לוקח T שניות, אזי

$$totalScore_{player} \coloneqq \int_0^T f_{player}(t)dt$$

נקרב את שטח זה ע״י חיבור הנקודות $\left(t_i, f_{player}(t_i)\right)$ שדגמנו מהפונקציה, וחישוב השטח שמתחת לפונקציה זו. נשים לב שככל שהמחשב בעל ביצועים חזקים יותר, אזי אנו נדגום יותר נקודות ולכן הקירוב שלנו יהיה יותר מדויק.



באופן מתמטי, השטח שמתחת לפונקציה הוורודה באיור ניתן לחישוב על ידי,

$$totalScoreApprox = \sum_i \frac{f_{player}(t_i) + f_{player}(t_{i+1})}{2} \cdot (t_{i+1} - t_i)$$

כלומר חיבור הטרפזים.