Bumbiņas lēkāšana

Juris Birznieks Ludvigs Miķelsons

2023. gada Decembrī

1 Darba mērķis

Izpētīt galda tenisa bumbiņas lēkāšanas fizikālos lielumus ar pašu veidotas datorprogrammas palīdzību. Ar programmas palīdzību atsevišķā failā saglabāt:

- video ar iezīmētu bumbiņas masas centru un trajektoriju;
- grafikus, kur attēlota bumbiņas atrašanās vietas atkarība no laika $x(t)$ un $y(t)$;
- grafikus, kur attēlota bumbiņas ātruma un paātrinājuma atkarība no laika $v(t)$ un $a(t)$.

2 Darbības pamatprincipi un algoritms

Video tiek ielasīts un sadalīts pa kadriem izmantojot CV2 bibliotēkas "VideoCapture" un "read" funkcijas. Tās nodrošina, ka katrs kadrs tiek ielasīts kā trīs dimensiju masīvs, kurā katram no divu dimensiju pikseļiem atbilst trīs krāsu RGB(sarkanā, zaļā un zilā) vērtības. Šajā masīvā, ieviešot Dekarta koordinātu sistēmu, kadra augstumu mēs uzskatām par y asi un kadra platumu - par x asi. Šādu interpretāciju arī var viegli iegūt attēlojot masīvu ar "matplotlib" palīdzību: kadrs tiek parādīts ar y asi gar kreiso malu un x asi gar attēla apakšmalu. X asij nulles punkts ir attēla kreisajā apakšējā stūrī, bet y asij - kreisajā augšējā. Tātad jebkura pikseļa vai punkta y koordinātas var uzzināt skaitot no kadra augšējās malas, kā arī bumbiņas krišanas virziens un arī gravitācijas paātrinājuma virziens ir y ass pozitīvajā virzienā, kas ir uz leju.

Pirmajā kadrā bumbiņa tiek maklēta pa visu attēlu izmantojot funkciju "ball_finder". Pēc visu pareizo punktu atrašanas, tiek iegūti maksimālie un minimālie punkti, kas teorētiski atbilst bumbiņas malām - y ass: augšējā un apakšējā; x ass: kreisā un labā.

Turpmāk šīs vērtības tiek izmantotas analizējot nākamos kadrus, lai ierobežotu meklēšanas laukumu kadrā, tādā veidā samazinot meklēšanas laiku un palielinot programmas ātrumu.

Maksimālās un minimālās x un y vērtības arī tiek padotas uz funkciju "center_finder", kas aprēķina vidējo vērtību un atgriež bumbiņas masas centra aptuvenās koordinātas.

Masas centrs (punkts sarkanā krāsā) un trajektorijas līnija (zilā krāsā) uz katra kadra tiek aprēķināti ar CV2 bibliotēkas funkcijām "cv2.Circle" un "cv2.line", un attēloti ar funkciju "write". Trajektorijas līnijas punkti tiek iegūti ar "for" cikla palīdzību, izmantojot iepriekšējo kadru punktus, kas ar katru kadru saglabājas masīvā "trajectory_points".

Funkcija "ball_finder_loop", masas centra un trajektorijas atrašana ir ievietotas "while" ciklā, kas ar katru iterāciju apstrādā jaunu kadru.

Lai būtu iespējams attēlot darbā prasītos grafikus, jāspēj atrast laiks t no bumbiņas kustības sākuma momenta un pikseļi jāpielīdzina reālām SI sistēmas mērvienībām, ko var izdarīt aprēķinot cik daudz ir viens pikselis metros. Laiku t var noteikt zinot filmēšanas kadru daudzumu sekundē (FPS). Laiks uz katru kadru (TPF) būs $\frac{1}{FPS}$. Sareizinot šo laiku ar video kadru daudzumu, iegūstam bumbiņas kustības kopējo reālo laiku. Tas ir svarīgi, jo daudzos no mūsu uzņemtajiem video atskaņošanas laiks un reālais bumbiņas kustības laiks atšķirās, tāpēc, ka šie video tika uzņemti izmantojot kameras lēnās kustībās funkciju (slow-motion). Tālāk, katram kadram atbilstošais reālais laiks tiek saglabāts masīvā "real_time_set". Šajā masīvā atrodot vērtību ar indeksu, kas atbilst kustības trajektorijas punktam ar maksimālo vērtību, kas savukārt atbilst laika momentam, kad bumbiņa atsitās pret zemi. Ievietojot šo laiku kinemātikas vienādojumā

$$y(t) = \frac{gt^2}{2}[m], \tag{1}$$

iegūstam reālo augstumu h, no kura tika izmesta bumbiņa. Reālais augstums, protams, neatbilst y vērtībām kadrā, jo vietas, kur bumbiņa tiek atlaista un kur bumbiņa atsitās, precīzi nesakrīt ar kadra apakšējo malu.

Ar katram kadram aprēķinātā atbilstošā laika t un pikseļa garuma palīdzību ir iespējams attēlot bumbiņas masas centra pārvietojuma grafikus atkarībā no laika x un y virzienos. Tālāk, attiecīgi atvasinot izmantojot Numpy bibliotēkas funkciju gradient, ieguvām bumbiņas ātruma v un paātrinājuma a grafikus.

3 Kritiski svarīgi koda elementi

3.1 Funkcija "ball_finder"

Šī funkcija iterē caur visām attēla masīva "frame" RGB vērtībām katrā punktā, caur koordinātām x un y ar divu "for" ciklu palīdzību, un pārbauda pikseļa atbilstību manuāli ievadītajam RGB vērtību intervālam, kas atrodas masīvos "lower_rgb" un "upper_rgb" un kas aptuveni atbilst RGB vērtībām kadra laukumam, kurā ir attēlota bumbiņa. Šī funkcija atgriež visus pikseļus jeb punktus, kas atrodas manuāli ievadītajā intervālā. Teorētiski tiem ir jābūt uz bumbiņas, bet praksē ne vienmēr tā notiek. Pareizo RGB intervāla robežu meklēšanas procesā, bija daudz gadījumu, kad funkcija atgrieza punktus, kas kadrā neatrodas uz bumbiņas.

```python
#Finds position of the ball in the first frame by going trough the whole image
def ball_finder(image, rgb_low, rgb_high):
    pixels = []
    height, width, _ = image.shape
    for y in range(height):
        for x in range(width):
            pixel = image[y, x]
            if np.all(rgb_low <= pixel) and np.all(pixel <= rgb_high):
                pixels.append((x, y))
    return pixels
```

3.2 Funkcija "ball_finder_loop"

Kadru analīze tiek veikta izmantojot funkciju "ball_finder_loop", kas iterē cauri visām pikseļu vērtībām līdzīgi, kā "ball_finder", tikai šoreiz izmantojot mazāku kadra laukumu, ko nosaka atrastajām bumbiņas malu vērtībām attiecīgi pieskaitot un atņemot vēl 70 pikseļus, lai ietvertu laukumu, kurā bumbiņa visticamāk varētu atrasties, ņemot vērā iepriekšējā kadra atrašanās vietu. Lai novērstu pikseļu vērtību meklēšanu ārpus kadra robežām, pārbaudāmā laukuma apakšējās vērtības "end_y" definēšanā tiek izmantota funkcija "shape", kas atgriež kadra garumu. Gadījumā, ja meklēšanas intervālu robežas iziet ārpus kadra robežām, kā intervāla apakšējā robeža tiek pieņemta kadra apakšējā robeža.

```python
#Using info about previous position of the ball looks for the ball in a limited area
def ball_finder_loop(left, right, top, bot, image, rgb_low, rgb_high):
    pixels = []

    #makes sure that ball is looked for within dimensions of image
    end_y = min(bot + 250, image.shape[0])
    end_x = min(right + 20, image.shape[1])

    for y in range(top -250, end_y):
        for x in range(left - 20, end_x):
            pixel = image[y, x]
            if np.all(rgb_low <= pixel) and np.all(pixel <= rgb_high):
                pixels.append((x, y))
    return pixels
```

3.3 Funkcija "center_finder"

Masas centrs tiek iegūts aprēķinot vidējo vērtību starp bumbiņas malu punktiem gar x asi bumbiņas kreisajā un labajā pusē, un gar y asi bumbiņas augšā un apakšā. Pēc tam vidējām vērtībām attiecīgi pieskaita

attālumu no y ass līdz bumbiņas kreisajai malai un attālumu no x ass līdz bumbiņas augšējai malai. Pēc saskaitīšanas funkcija atgriež gan x, gan y koordinātas. Masas centra atrašanās vieta kadrā ir aptuvena, jo daudzos kadros programma atrod un pieskaita pikseļus, kas ir ārpus bumbiņas, vai arī neatrod visus pikseļus, kas attiecas uz bumbiņu.

```
#Finds the center coordinates given 4 points
def center_finder(top, bot, left, right):
    center_x = left + (right - left) / 2
    center_y = top + (bot - top) / 2

    return center_x,center_y
```

3.4 Pikseļa garuma aprēķināšana metros

Lai atrastu lielumu, kas atbilst viena pikseļa garumam metros reālais augstums tiek dalīts ar trajektorijas masīva lielāko vērtību t.i. trajektorijas zemākā punkta un trajektorijas sākuma punkta starpību, pēc formulas:

$$[h]h_{piksela} = \frac{h_{teoretiskais}}{\Delta p}[m], \tag{2}$$

kur p - pikseļu daudzums

```
#Finds time per frame given recording fps (playback fps doesn't matter)
fps = 60
tpf = 1/fps


real_time_full = time * tpf
real_time_set = np.linspace(0,real_time_full,time)
```

```
#Finds how many meters are per pixel
ptm = Augstums/(ball_movement_y[max_index]-trajectory_point_first)
```

4 Darbības piemērs

Video, kas pieejams atverot saiti redzama programma darbībā uz viena no filmētajiem video lēnās kustības režīmā (slow-motion) ar 200 FPS. Katra kadra (izņemot pirmā) meklēšanas laukums ir 100x40 pikseļu liels ap masas centra atrašanās vietu iepriekšējā kadrā. Krāsu RGB vērtību intervāli ir manuāli uzstādīti ar vērtībām: R:[170; 255] G: [10, 150] B: [10, 80]. Uz bumbiņu ir pavērsti trīs gaismas avoti - labajā, kreisajā pusē un no apakšas. https://streamable.com/o35vcv

5 Rezultāti un secinājumi

Izdevās izveidot programmu, kas konkrētos video, kuros ir nodrošināts pietiekošs apgaismojums un pietiekoša krāsu atšķirība no apkārtējiem objektiem, spēj atrast bumbiņu katrā kadrā, izsekot tai, uzzīmēt tās trajektoriju un no šīs informācijas attēlot pārvietojuma, ātruma un paātrinājuma grafikus metros (attiecīgi $m, m/s, m/s^2$) atkarībā no laika sekundēs. RGB vērtību intervāli pirms programmas palaišanas tiek ievadīti manuali, kas nosaka, kas tieši tiks meklēts attiecīgajā video.

Pēc filmēšanas ar viedtelefona kameru saskārāmies ar problēmu, kas radās apgriežot video pašā telefonā. Izrādās, ka šajā gadījumā, kaut arī filmēšanas laikā telefonā tika uzlikts 60 FPS, pēc apgrišanas un video faila pārnešanas uz datoru, faila detaļās var redzēt, ka tas ir tikai 30 FPS, kas nozīmē, ka telefonā daudzi kadri tiek izņemti. Tāpēc tiek iegūts šads grafiks.

Programma diezgan labi var atrast un ļoti labi sekot līdzi bumbiņai vai citam noteiktas krāsas objektam.

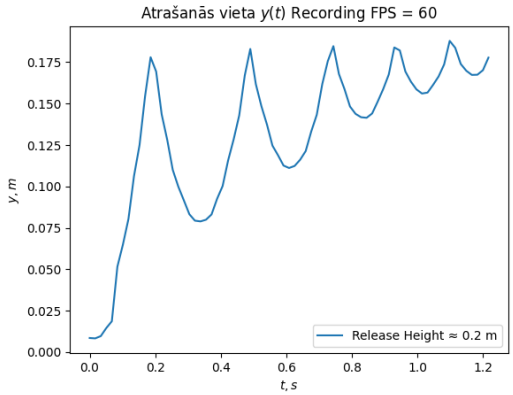

Figure 1: Atrašanās vietas grafiks atkarībā no laika, ar izņemtiem kadriem

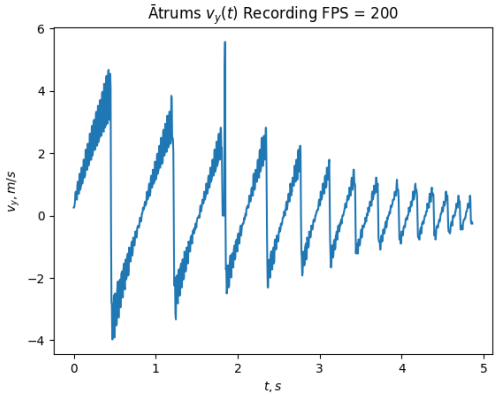

Figure 2: Ātrums atkarībā no laika

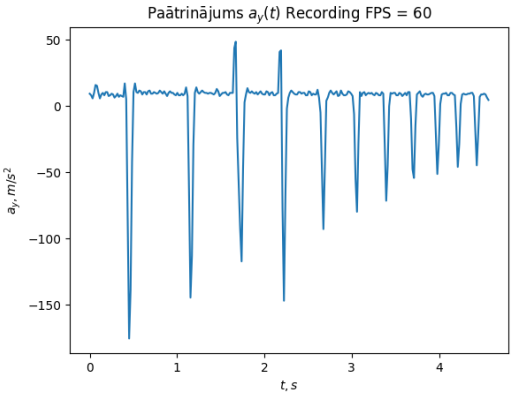

Figure 3: Paātrinājums atkarībā no laika

Sākumā, ar mūsu pirmajām koda versijām, apgaismojuma daudzums bija ļoti svarīgs, lai programma atrastu bumbiņu, bet pēc dziļākas konkrētu krāsu un toņu izpētes un precīzāku RGB vērtību uzzināšanas, pielāgojām programmu, iegūstot to, ka tā var atrast bumbiņu arī ar mazāku apgaismojuma daudzumu, kad bumbiņai apakšējā daļā ir ēna. Labajā attēlā ir redzams, ka ēna uz bumbiņas gandrīz vai precīzi sakrīt ar grīdas krāsu. No tā secinājām, ka oranžā krāsa no brūnās atšķiras tikai ar sarkanās jeb R vērtības intensitāti. Lai atšķirtu oranžo no brūnās krāsas ir jāpalielina R vērtības intensitāte.

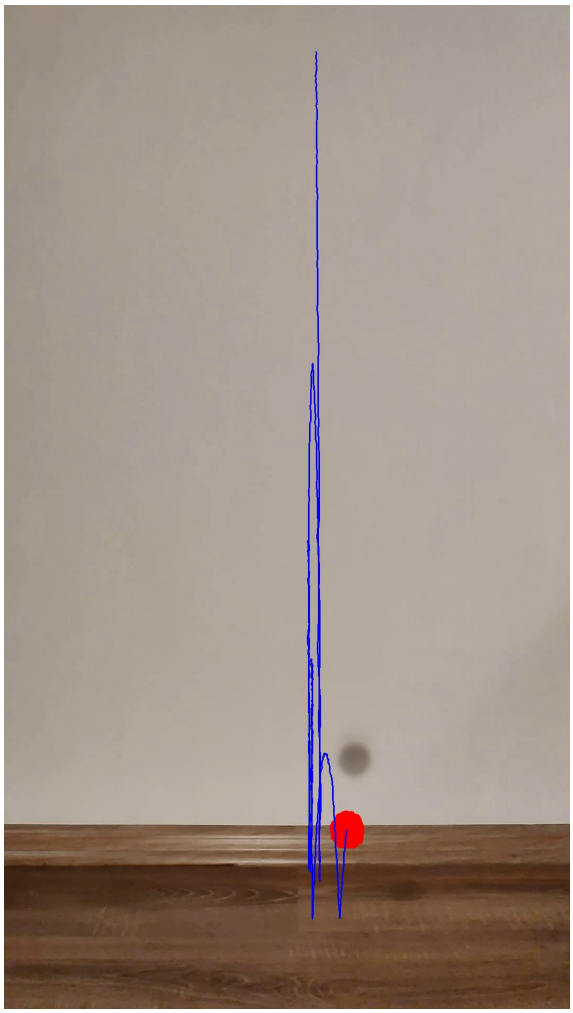

Figure 4: Vairāk gaismas

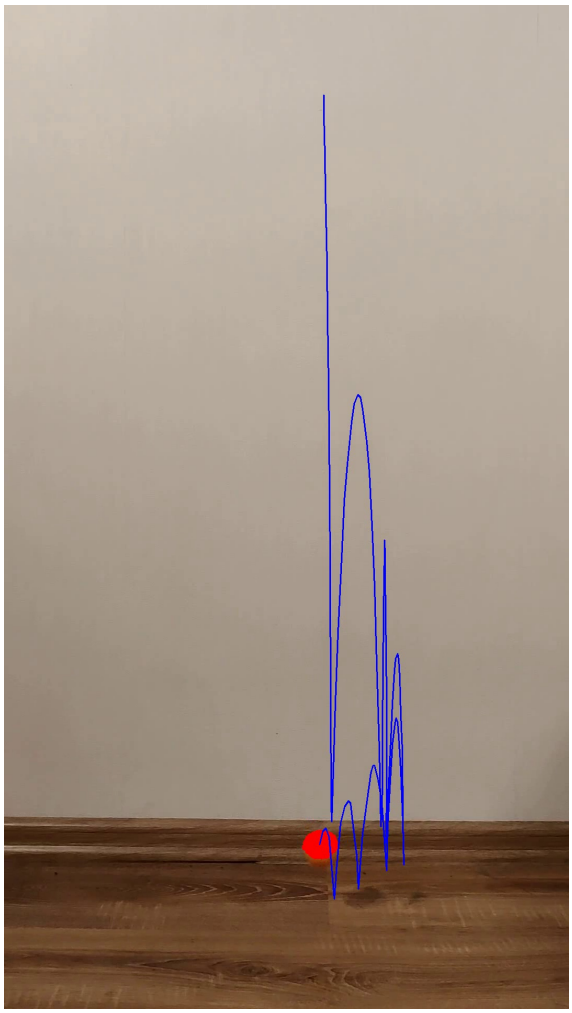

Figure 5: Mazāk gaismas