



**Projektna naloga pri Matematičnem
modeliranju:
Lovljenje psa in lastnika**

Avtor: Luka Urbanč, 27231152

Mentor: doc. dr. Tadej Kanduč

Akademsko leto 2025/26

Ljubljana
1. julij 2026

1 Uvod

Del obveznosti pri predmetu Matematično modeliranje v tekočem letu je izdelava projektne naloge. Pri tem imamo na voljo dve nalogi: težjo in lažjo. Zase sem izbral težjo nalogo, pri kateri moramo napisati program, ki bo izračunal krivuljo, ki jo opiše pes, ki lovi lastnika.

2 Povzetek

Podrobneje si pogledamo gibanje po krožnici in opazujemo vpliv različnih spremenljivk na čas ulova pri zastavljenem problemu. Za problem pokažemo tudi zapletene povezave med spremenljivkami. Pri gibanju po premici predstavimo numerične rezultate in intuitivno rešitev problema iz vidika hitrosti, rešitev pa poiščemo tudi analitično, saj gre za preprosto gibanje. Pri obeh numerično poiščemo tudi optimalen t_k . Dalje obravnavamo poševno spiralo in predstavimo reševanje problema z reparametrizacijo, kar uporabimo tudi na Lissajousevi krivulji, za katero s pridobljenim znanjem pravilno prilagodimo parametre, da do ulova pride prej, kot v originalnem primeru. Delo uporablja podatke iz [1, 2, 3, 4, 5].

3 Predstavitev naloge

Nalogo najlažje razumemo preko prispodobe psa, ki lovi svojega lastnika oziroma obratno. Naj bo $I \subset \mathbb{R}$ interval in $q : I \rightarrow \mathbb{R}^2$ vsaj C^1 parametrizacija gibanja psa. Pes želi slediti svojemu lastniku in mu sledi po naslednjem načelu: v vsakem trenutku se bo lastnik premikal proti točki v prostoru, v kateri bi se lastnik nahajal, če bi se premikal premočrtno in bi njegovo gibanje trajalo t_k časa. Hitrost psa je ob nekem času $t \in I$ torej usmerjena proti točki $q(t) + t_k \cdot q'(t)$. Določiti moramo krivuljo $p : I \rightarrow \mathbb{R}^2$, po kateri se giblje pes. Pri tem je pomembno izpostaviti, da v nalogi nimamo nobenih omejitev glede velikosti hitrosti tako psa kot tudi lastnika. Pogoji so podani zgolj na njuni smeri. To pomeni, da imamo veliko svobode glede velikosti, kar bo vplivalo tudi na iskano krivuljo $p(t)$.

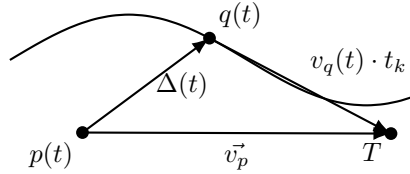
Tematsko lahko nalogo povzamemo kot iskanje sledilne krivulje (ang. *pursuit curves*), kar spada v področje sledilne analize (ang. *pursuit analysis*). Najbolj osnovni primer naloge s tega prodočja je iskanje krivulje, kjer se lovec (pri nas pes), usmeri proti trenutni legi plena (pri nas lastnika). Tako hitrost lovca ob času t kaže kar proti $q(t)$. Vidimo, da gre pri naši nalogi za nadgradnjo originalnega problema. Kot zanimivost omenimo, da se je področje začelo razvijati že leta 1732, ko je Pierre Bouguer zastavil in rešil problem iskanja krivulje, po kateri se piratska ladja premika za trgovsko ladjo, če se premika tako, da v vsakem trenutku kaže proti trgovski ladji. Tak primer dobimo lahko tudi pri naši nalogi, če le nastavimo $t_k = 0$.

4 Izpeljava

Denimo, da imamo torej dve točki, ki se gibljeta v prostoru. Označimo njuno gibanje z

$$q(t) = \begin{bmatrix} x_q(t) \\ y_q(t) \end{bmatrix} \quad \text{in} \quad p(t) = \begin{bmatrix} x_p(t) \\ y_p(t) \end{bmatrix},$$

kjer so x_p, y_p, x_q, y_q seveda vse ustrezne parametrizacije gibanja prve in druge točke. Predpostavimo, da gibanje točke $q(t)$ poznamo, torej imamo funkciji $x_q(t)$ in $y_q(t)$ in predpišemo, da se druga točka giblje tako, da je njena hitrost vedno usmerjena proti točki, kjer se bo prva točka nahajala čez t_k časa.



Na zgornji sliki je prikazan primer, kako dobimo smer hitrosti za krivuljo $p(t)$. Točka T označuje mesto, kjer bi se $q(t)$ nahajal čez t_k časa. Iz slike lahko enostavno zapišemo enačbo za točko $p(t)$. Vemo že, da za splošno gibanje delca velja enačba

$$s(t) = s(0) + \int_0^t \vec{v}(\tau) d\tau. \quad (1)$$

kjer je $\vec{v}(t)$ hitrost delca, torej $\vec{v}(t) = \dot{s}(t)$. Za našo točko velja, da je

$$\dot{p}(t) = \vec{v}_p(t) = \Delta(t) + v_q(t) \cdot t_k.$$

Razlika med položajema med obema točkama je kar

$$\Delta(t) = q(t) - p(t) = \begin{bmatrix} x_q(t) \\ y_q(t) \end{bmatrix} - \begin{bmatrix} x_p(t) \\ y_p(t) \end{bmatrix} = \begin{bmatrix} x_q(t) - x_p(t) \\ y_q(t) - y_p(t) \end{bmatrix},$$

torej je

$$\vec{v}_p(t) = \begin{bmatrix} x_q(t) - x_p(t) \\ y_q(t) - y_p(t) \end{bmatrix} + \begin{bmatrix} \dot{x}_q(t) \\ \dot{y}_q(t) \end{bmatrix} t_k.$$

Dobljeno hitrost lahko tedaj upoštevamo v enačbi (1) in dobimo, da je

$$\begin{aligned} \vec{p}(t) &= \vec{p}(0) + \int_0^t \vec{v}_p(t) dt = \\ &= \vec{p}(0) + \int_0^t q(t) - p(t) + \dot{q}(t)t_k dt, \end{aligned}$$

kar lahko zapišemo po komponentah kot

$$\begin{bmatrix} \dot{x}_p(t) \\ \dot{y}_p(t) \end{bmatrix} = \begin{bmatrix} x_p(0) \\ y_p(0) \end{bmatrix} + \int_0^t \begin{bmatrix} x_q(t) - x_p(t) + \dot{x}_q(t)t_k \\ y_q(t) - y_p(t) + \dot{y}_q(t)t_k \end{bmatrix} dt. \quad (2)$$

Enačbo lahko sedaj odvajamo in dobimo, da je

$$\dot{x}_p(t) = x_q(t) - x_p(t) + \dot{x}_q(t)t_k$$

in

$$\dot{y}_p(t) = y_q(t) - y_p(t) + \dot{y}_q(t)t_k.$$

Obe izpeljani enačbi sta linearni diferencialni enačbi prvega reda in opisujeta gibanje druge točke. Rešitvi teh enačb bosta torej komponenti krivulje $p(t)$, ki jo iščemo.

Kot je bilo že omenjeno, takšna izpeljava ne postavlja nobenih dodatnih zahtev na velikost hitrosti, zgolj na njeno smer. Pri pregledu literature hitro opazimo, da klasični primeri običajno zahtevajo, da je hitrost obeh krivulj konstantna. Če bi želeli izpolniti predpostavo o konstantni velikosti hitrosti sledilne krivulje $p(t)$, moramo v enačbi (2) zgolj desno stran enačbe deliti z njeno normo. Tako zagotovimo, da je velikost odvoda $\dot{p}(t)$ konstantno enaka 1, z dodatnim množenjem desne strani s konstantnim faktorjem λ pa lahko velikost hitrosti do skalarja natančno tudi spreminjamo.

5 Reševanje v Matlabu

Za numeričen del reševanja naloge uporabljamo program Matlab, v katerem lahko enačbo (2) rešimo z uporabo enega od številnih vgrajenih funkcij za reševanje navedenih diferencialnih enačb. V našem programu uporabljamo `ode45`, kateri dodamo tudi dogodek ali *event*. Ta dodatna nastavitev dovolj programu, da preneha z iskanjem rešive ob izpolnitvi vnaprej določenega pogoja. Pri tej nalogi ta pogoj nastavimo na trk oziroma ulov: ko sta točki $q(t)$ in $p(t)$ bližje od neke določene vrednosti, program to zazna kot dogodek in preneha z reševanjem, sicer pa išče rešitev na podanem časovnem intervalu I . Poglavitna funkcija za reševanje naloge tega tipa se imenuje `lovi`, kjer dodatno lahko izberemo možnost, da se izriše celoteno gibanje vključno z zveznico med obema točkama na krivulji.

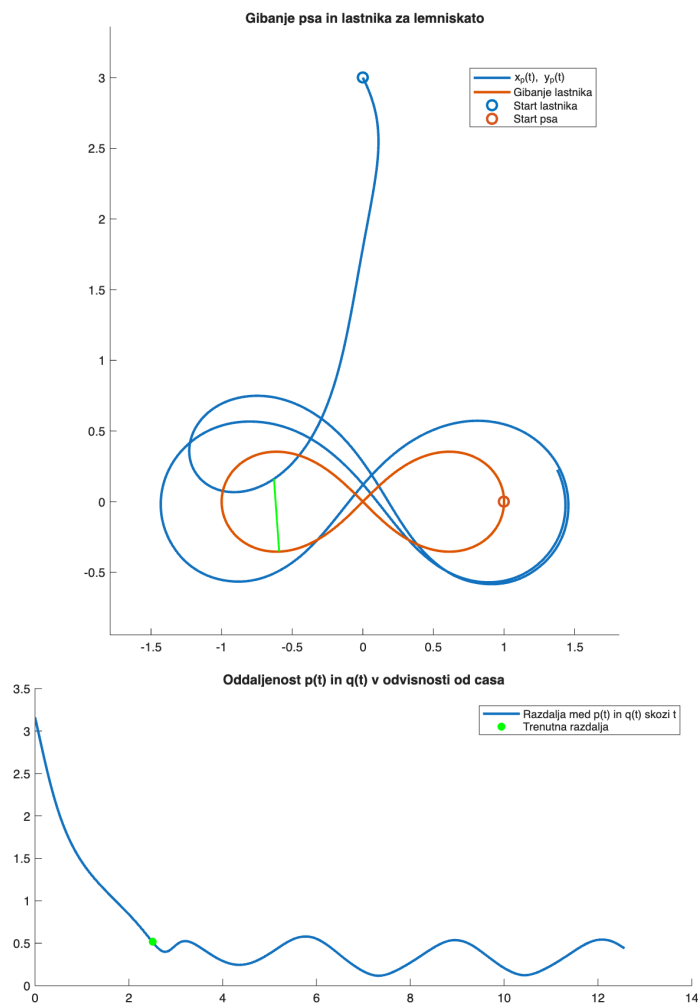
Na sliki 1 lahko bolje vidimo, katere informacije nam lahko delovanje programa pove. Spodnji primer prikazuje reševanje problema za lemniskato, ki je parametrizirana z

$$q(t) = \begin{bmatrix} \frac{\cos(t)}{1+\sin^2(t)} & \frac{\sin(t)\cos(t)}{1+\sin^2(t)} \end{bmatrix},$$

kjer smo za začetno točko lastnika izbrali $(0, 3)$ ter smo hitrost lastnika normalizirali s faktorjem 1.5 in upoštevali $t_k = 1$. Na sliki vidimo zveznico med trenutnima položajema točk, ki je označena z zeleno barvo. Označena razdalja ustreza tudi točki na spodnjem grafu, iz katerega razberemo, da gibanje lastnika glede na gibanje psa postane periodično po začetni fazi približevanja. Skaliranje hitrosti za faktor 1.5 pomeni tudi, da večja krivulja ni presenetljiva, saj je tudi hitrost večja.

Takšna periodičnost lastnika se je pravzaprav izkazala kot izjemno pogosta, ko smo gledali periodične parametrizacije $q(t)$. Pogosto so lahko razlogi za to

razmeroma velike vrednosti t_k , kjer je pojem *velikosti* močno odvisen od začetne krivulje in zanj ne poznamo konkretne klasifikacije.



Slika 1: Izpis programa za lemniskato

6 Dane parametrizacije

Ker je parametričnih krivulj ogromno in je vsaka zanimiva na svoj način, možnih nastavitev in spremenljivk v programu pa tudi ni malo, bomo v tem poglavju podrobneje pogledali nekaj primerov različnih krivulj.

6.1 Krožnica

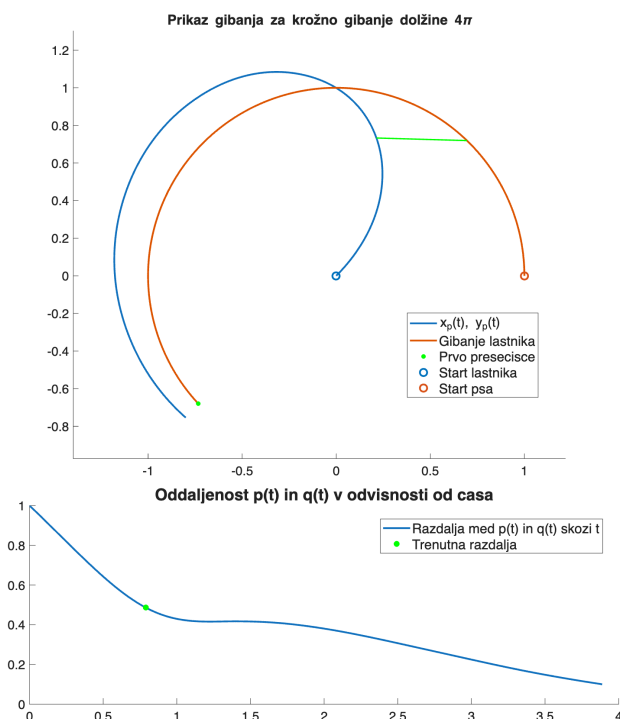
Začnimo s krožnico, ki jo za dan $r > 0$ parametriziramo kot

$$q(t) = \begin{bmatrix} r \cos(t) \\ r \sin(t) \end{bmatrix}. \quad (3)$$

Velika prednost krožnice je, da je njena parametrizacija že parametrizacija s konstantno velikostjo odvoda, poleg tega pa zaradi začetka lastnika znotraj kroga povečamo verjetnost, ko izbiramo na slepo, da se lastnik in pes približata. Poglejmo si naslednji primer. Denimo, da je začetna točka lastnika v izhodišču ter si pogledajmo, kako izgleda gibanje za vrednosti

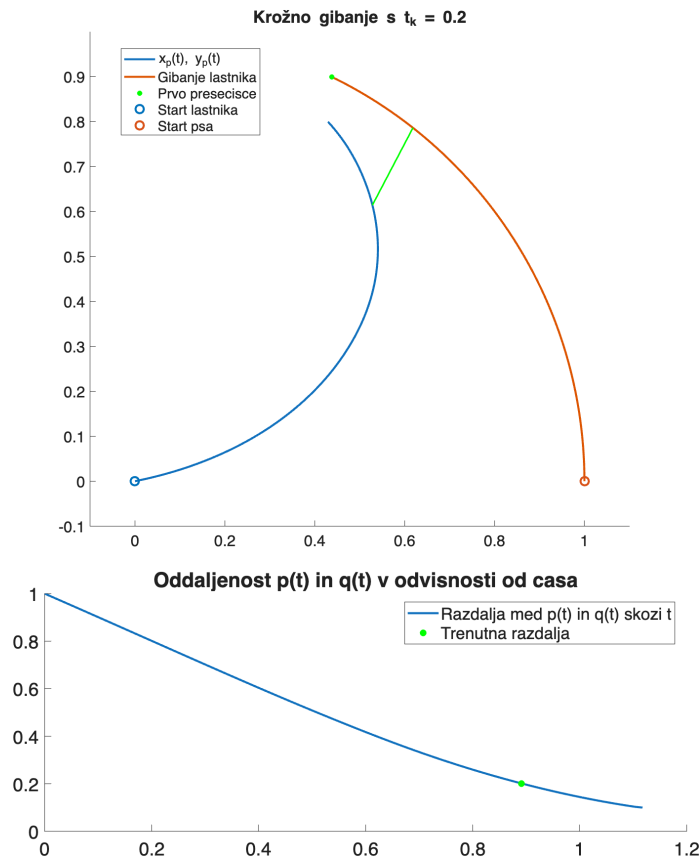
$$\lambda = 1, \quad t_k = 1, \quad |\dot{p}| = 1 \quad (4)$$

Ob iskanju rešitve uporabimo toleranco za trke 0.1 in simulacijo izvajamo 4π sekund. Seveda bi se rezultati spremenili, če bi povečali natančnost, a lahko vedno predpostavimo, da tako pes kot tudi lastnik nista točkasta, pač pa imata neko velikost, za ulov pa je dovolj, da se dotakneta. Na sliki 2 lahko vidimo potek rešitve. Lastnik psa ujame v času $t = 3.888$ s, iz grafa razdalje pa vidimo tudi, da ne preidemo v periodično gibanje lastnika, kot se je to zgodilo na sliki 1.



Slika 2: Krožno gibanje z nastavitvami kot v (4)

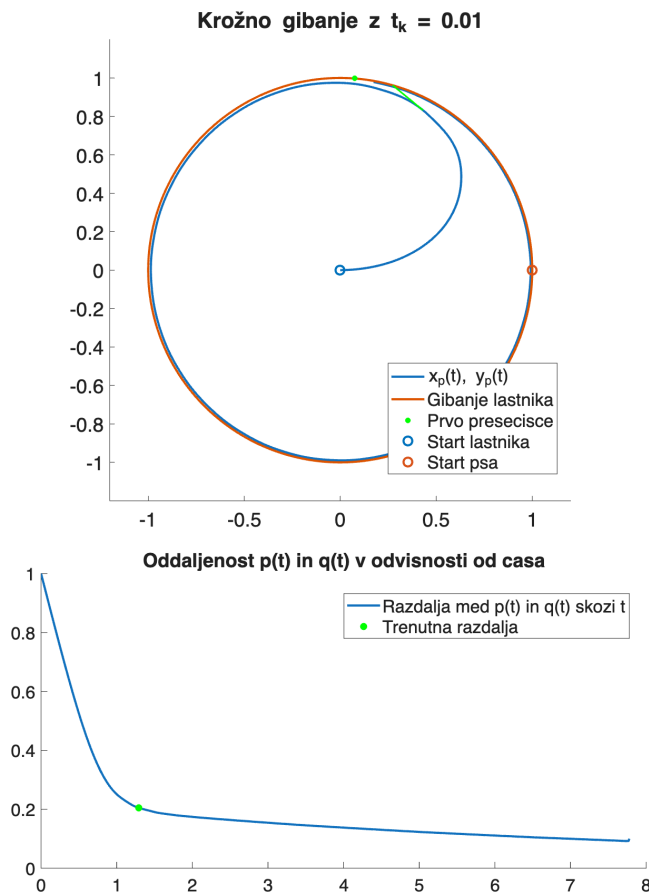
Spremenimo sedaj vrednost t_k in jo nastavimo na $t_k = 0.2$. Na sliki 3 opazimo, da se lastnik in pes srečata prej, kot v zgornjem primeru. Sedaj je čas za srečanje zgolj $t = 1.117$ s.



Slika 3: Krožno gibanje s $t_k = 0.2$

Ali to pomeni, da manjše vrednosti t_k ustrezajo hitrejšemu srečanju? Poskusimo sedaj s $t_k = 0.01$ in si oglejmo rezultate na sliki 4.

Vidimo, da manjši t_k ne pomeni nujno hitrejšega trka. V tem primeru trčita šele po $t = 7.778$ s. Primer s tako majhnim t_k pokaže, da je zveza med časom do trka in povezavami zapletena, zato jo narišemo. Na sliki 5 primerjamo različne vrednosti t_k in različne velikosti krogov ter za vsako možnost izmerimo čas do trka. Na grafu lahko opazimo izjemno zanimivo obliko, kjer so najmanjši časi približno na krožnici v vodoravni ravnini. Omenimo tudi, da za primere, ki trka ni bilo, smo čas nastavili na 11 sekund in s tem zagotovili polnost grafa. Problem iskanja najbolj ustreznega časa za dano konfiguracijo je zaradi možnosti, da do trka sploh ne pride, zelo zahteven za numerično reševanje, saj večina algoritmov potrebuje neke vrste zveznosti v funkciji, ki jo želimo minimizirati. Globalni

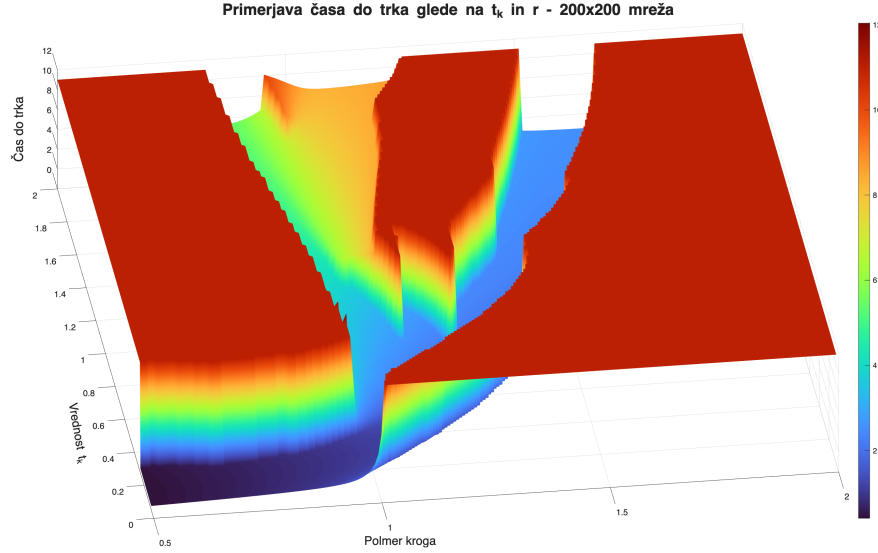


Slika 4: Krožno gibanje s $t_k = 0.01$

minimum časa, ki ga potrebujemo za trk, na zgornji mreži, ki upošteva zgolj vrednosti $r \in [0.5, 2]$ in $t_k \in [0.05, 2]$, je $t = 0.410$ s pri $r = 0.5$ in $t_k = 0.109$. Ker točne vrednosti $r = 1$ ni na grafu, lahko namesto minimuma za $r = 1$ pogledamo minimum čas za $r = 0.9975$, ki znaša $t = 1.083$ s za $t_k = 0.256$. Zgornje ugotovitve povzemimo v naslednji tabeli.

r	t_k	Čas do trka [s]	Opomba
1.0000	1.000	3.888	Začetna nastavitvev
1.0000	0.200	1.117	$t_k = 0.2$
1.0000	0.010	7.778	$t_k = 0.01$
0.5000	0.109	0.410	Globalni minimum na mreži
0.9975	0.256	1.083	Minimum za $r \approx 1$

Tabela 1: Povzetek numeričnih rezultatov za krožno gibanje.



Slika 5: Primerjava časa do trka, velikosti t_k in polmera kroga

Poleg polkrožne oblike grafa je zanimivo tudi, da se vrednosti znotraj optimalnega polkroga dvigajo z večjim polmerom kroga. Opazimo lahko tudi, da minimalne vrednosti ne sledijo polkrožni obliki strogo, saj obstaja še ena dolina manjših vrednosti. Oblike, ki jih vidimo na sliki 5 lahko utemeljimo s tem, da se zaradi dane parametrizacije v enačbi (3) s spreminjanjem vrednosti r spreminja tudi velikost hitrosti lastnika, saj je pri taki parametrizaciji hitrost ravno r . Vse skupaj nakazuje na kompleksno zvezo med nastavljenimi parametri za dani primer, ki bi trejala precej več prostora in časa za raziskovanje.

6.2 Premica

Poglejmo si še en enostaven in lahko razumljiv primer. Pri gibanju lastnika po premici lahko ugibamo, da bo pes ulovil lastnika, če bo le začel dovolj blizu, ali pa se premikal z dovolj veliko hitrostjo. Poglejmo parametrizacijo

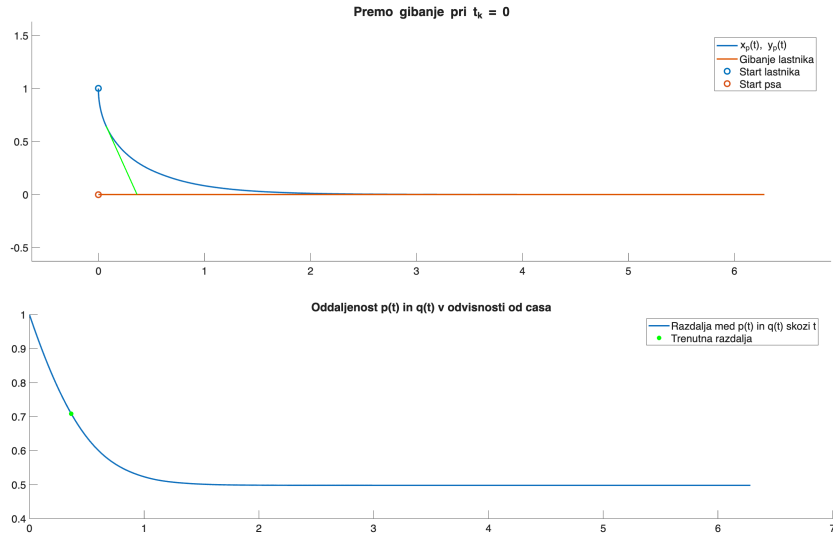
$$q(t) = [t, 0], \quad (5)$$

začetni položaj pa naj bo recimo točka $(0, 1)$. Poglejmo si primer, ko je

$$t_k = 1, \quad |\dot{p}| = 1, \quad \text{Toleranca} = 10^{-1}$$

in simulacijo izvajamo 2π časa. Gibanje prikazuje slika 6, vidimo pa, da po začetni fazi približevanja nastopi gibanje, kjer se obe točki premikata praktično v isto smer in z enako hitrostjo.

Obe krivulji imata namreč hitrost 1, kar razloži dejstvo, da se potem njuna medsebojna razdalja ustali blizu 0.5. Sklepamo, da bi potem z večanjem hitrosti



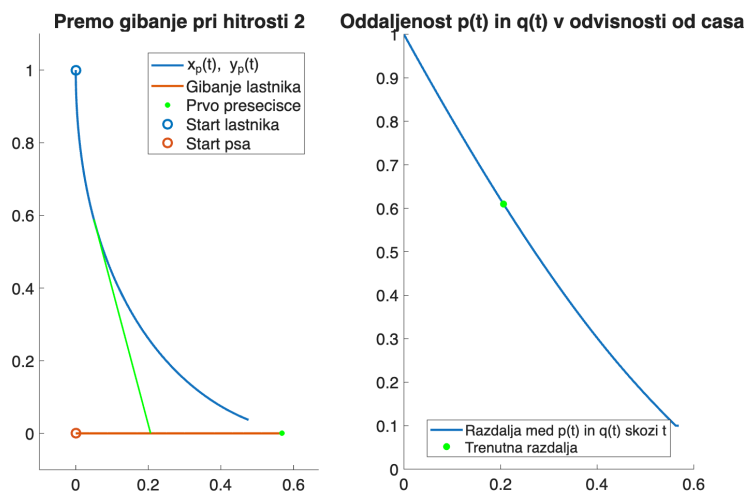
Slika 6: Premo gibanje za $t_k = 0$

psa dosegli ulov, zato povišamo hitrost in nastavimo $|\dot{p}| = 2$. Gibanje potem izgleda kot prikazuje slika 7.

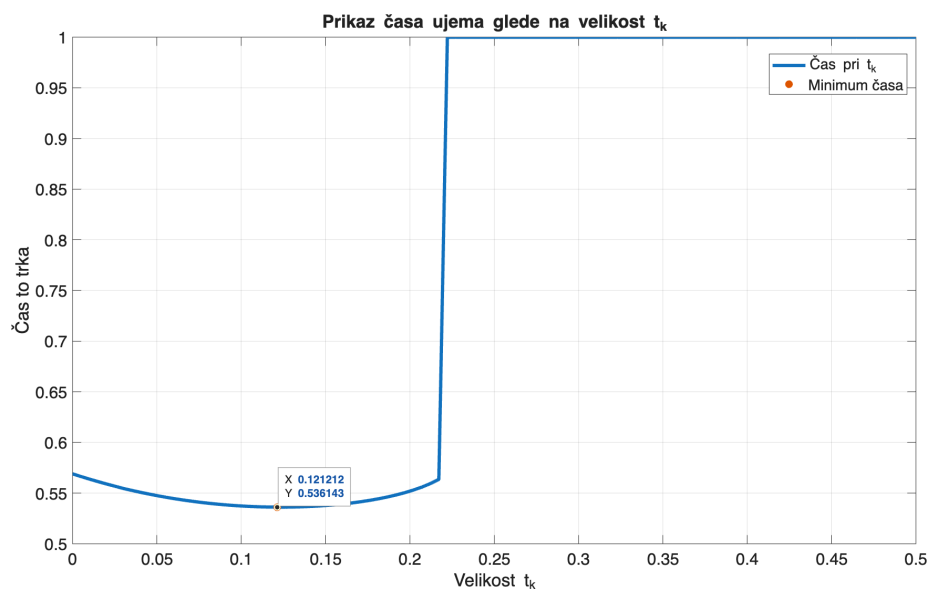
Zgornji sklep je bil pravilen, saj je sedaj do trka prišlo zelo zgodaj, namreč pri $t = 0.569$ s. Sklepamo torej, da se ob enaki medsebojni hitrosti njuna razdalja po začetni fazi približevanja ustali in ostane konstantna v neskončnost. Če je pes hitrejši od lastnika, ga za ustrezno majhne t_k ujame v končnem času, če pa je pes počasnejši od lastnika, pa ga seveda nikol ne ujame in mu lastnik pobegne.

Poglejmo si optimalno vrednost t_k , da bo čas ulova čim hitrejši. V ta namen narišemo spodnji graf, na katerem vidimo, da minimum za čas ulova nastopi pri vrednosti $t_k = 0.121$, znaša pa $t = 0.536$ s. Opazimo tudi, da je graf relativno simetričen okoli minimuma, ki pa je jasno definiran. Vse točke z vrednostjo časa 1, so konfiguracije, kjer ulova ni bilo, vrednost 1 pa je izbrana zato, da graf ohrani pregledno skalo. Kot vidimo na grafu velja, da pes ne ulovi lastnika za $t_k > 0.22$. To potrdijo tudi nadaljni preizkusi z več testnimi točkami in širšim naborom možnih vrednosti, a ker to spet kvari preglednost, smo se odločili možne vrednosti t_k omejiti.

Omenimo tudi, da je risanje takšnih prikazov lahko časovno zelo potratno, saj lahko traja dolgo, preden se rešitev izračuna. Tak prikaz, kjer smo pogledali le 100 možnih točk, je vseeno potreboval kar nekaj minut za izvedbo.



Slika 7: Premo gibanje s hitrostjo 2



Slika 8: Spreminjanje časa ulova zaradi t_k

Pri minimumu pa zaznamo podobnost s prejšnjim primerom. Velja namreč, da je minimum časa ulova dosežen pri razmeroma majhnih t_k , a vendar ne pri ničelnih oziroma skoraj ničelnih vrednostih t_k . Iz tega sklepamo, da naiven način lovljenja, ko se premikamo proti trenutni legi tarče, ni optimalen in se splača gledati malenkost v prihodnost, vsaj v primerih, ko so krivulje gibanja razmeroma enostavne in je hitrost konstantna.

Ker je gibanje po premici izjemno enostavno, lahko za ta primer celo analitično rešimo enačbe. Sistem (2) zapišemo po komponentah ob upoštevanju dane parametrizacije iz enačbe (5) dobimo

$$\begin{aligned}\dot{x} &= t - x + t_k \\ \dot{y} &= 0 - y + 0\end{aligned}$$

Najprej rešimo enačbo za $y(t)$ in dobimo, da je

$$y(t) = Ae^{-t},$$

kjer A določimo iz začetnega pogoja. Ker je $y(0) = 1$, je potem $A = 1$ in se zato rešitev za y komponento gibanja glasi kar

$$y(t) = e^{-t}.$$

Za reševanje prve enačbe lahko uporabimo variacijo konstante, ki po krajšem računu pove, da je

$$x(t) = e^{-t}(1 - t_k) + t - 1 + t_k,$$

kjer smo za začetni pogoj upoštevali, da je $x(0) = 0$. Če narišemo grafa numerične in analitične rešitve se ta tudi ujemata, kar potrdi naš račun. Ker je $y(t) = e^{-t}$, takoj vidimo, da se točki nikoli ne bosta srečali že zaradi dejstva, da $e^{-t} = 0$ nima rešive v končnem času. Tukaj vidimo, da toleranca pri zausavitvenem pogoju ne odpravi samo numeričnih napak, ampak tudi omogoča ulov, ko ta analitično ni možen. Spomnimo pa se, da so krivulje lahko poljubno zapletene, zaradi česar je iskanje analitičnih rešitev pogosto nemogoče. Gibanje po premici je v tem pogledu bolj izjema kot pravilo.

6.3 Poševna spirala

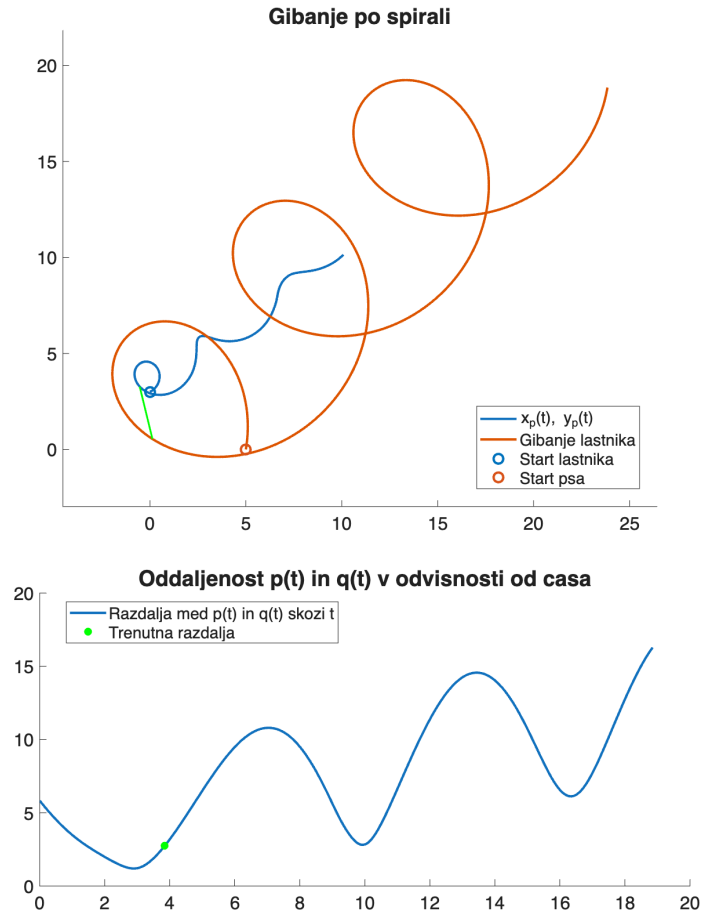
Pogljemo si primer bolj kompliciranega gibanja. Vpeljemo parametrizacijo

$$q(t) = \begin{bmatrix} t + 5 \cos(t) \\ t + 5 \sin(t) \end{bmatrix}, \quad (6)$$

in za začetno točko izberemo recimo točko $(0, 3)$. Če sledimo nastavitvam pri zgornjih primerih in nastavimo

$$|\dot{p}| = 1, \quad t_k = 1,$$

ter pustimo, da gibanje traja 6π sekund, dobimo gibanje na sliki 9.



Slika 9: Gibanje po spirali

Vidimo, da oddaljenost narašča s časom, saj je hitrost psa omejena na 1, medtem ko hitrost lastnika nima nobenih omejitev. V prejšnjih dveh primerih je bila tudi ta konstantno enaka 1, saj smo izbrali take parametrizacije. Parametrizacija v enačbi (6) je tudi delno periodična v smislu, da se zanke ponavljajo. Zato, podobno kot pri kroženju, postane gibanje psa tudi periodično po začetni fazi približevanja. To vidimo na spodnjem grafu na sliki 9, kjer se pes najprej približa, nato pa se začne njegovo periodično gibanje. Zanimivo je to, da minimum medsebojne razdalje nastopi ravno med fazo približevanja, ne pa med kasnejšim gibanjem. Imamo torej primer, kjer je parametrizacija krivulje taka, da je hitrost neenakomerna, reparametrizacija krivulje pa ni možna, vsaj ne analitično. Rešitev za ta problem bo numerična reparametrizacija krivulje lastnika.

6.3.1 Postopek numerične reparametrizacije

Naj bo $I = [a, b] \subset \mathbb{R}$ in $\gamma : I \rightarrow \mathbb{R}^2$ gladka parametrizacija neke poti. Njena hitrost ob danem času je podana kot $|\dot{\gamma}(t)|$, želimo pa si, da bi bila ta hitrost konstantno enaka neki predpisani vrednosti, ki jo označimo z v . Predstavljamo si, da bi lahko naš začetni časovni interval raztegnili in skrčili tako, da bi prepotovana pot ustrezala konstantni velikosti hitrosti. Sklepamo, da to pomeni, da interval raztegnemo, ko je hitrost originalne parametrizacije velika in jo tako zmanjšamo, pri nizkih hitrostih pa ga skrčimo in tako povečamo prepotovano pot. To pomeni, da iščemo tako domeno, da bo γ v eni enoti te domene opisala krivuljo, ki ima dolžino loka v enot dolžine, to pa mora seveda veljati za vsak začetni trenutek. Poglejmo si to idejo nekoliko bolj formalno. Pot parametrične krivulje γ je podana kot

$$s(t) = \int_a^t \|\dot{\gamma}(\tau)\| d\tau.$$

Pri dodatni predpostavki, da je parametrizacija regularna, torej je $\|\dot{\gamma}(t)\| > 0$ za vsak $t \in I$, je potem $s(t)$ strogo naraščajoča funkcija, zato obstaja njen inverz, torej funkcija $t : [0, L] \rightarrow [a, b]$, da velja $t(s) = s^{-1}(s)$. Za kompozitum funkcije t s funkcijo γ , ki ga označimo z Γ , potem velja, da je $\Gamma : [0, L] \rightarrow \mathbb{R}^2$ s predpisom $\Gamma(s) = \gamma(t(s))$. Hkrati je $\|\dot{\Gamma}\| = 1$ povsod, saj je po verižnem pravilu

$$\dot{\Gamma}(s) = \dot{\gamma}(t(s)) \cdot \dot{t}(s),$$

hkrati pa je tudi

$$\dot{t}(s) = \frac{1}{\dot{s}(t)} = \frac{1}{\|\dot{\gamma}(t(s))\|}.$$

Zgornjo vrednost torej vstavimo v račun, velikost odvoda pa potem sledi z uporabo absolutne vrednosti. Če želimo, da se telo giblje s hitrostjo, ki ni enaka 1, pač pa recimo v , moramo zgolj skalirati čas. Velja torej, da je

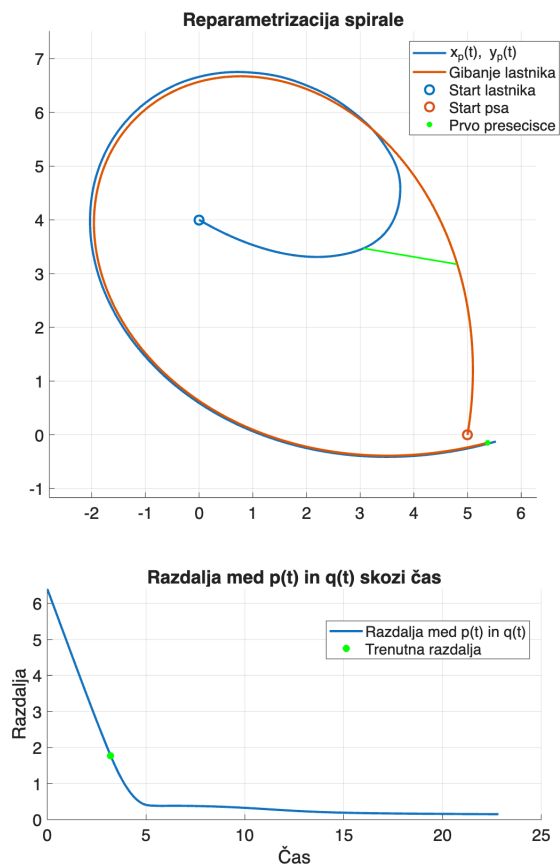
$$\gamma(t) = \Gamma(vt).$$

Pogosto pravimo, da je γ parametrizirana s časom, medtem ko je Γ parametrizirana z opravljeno potjo.

Zgornjo izpeljavo lahko relativno enostavno implementiramo tudi numerično. Težji del je zgolj poiskati inverz funkcije $s(t)$, kar pa lahko naredimo z numeričnim iskanjem ničel, torej bo $t(s)$ tista vrednost, kjer bo $s(t) - s = 0$, kar v matlabu posiščemo s pomočjo **fzero**. Funkcija, ki rešuje probleme, kjer želimo reparametrizirati gibanje lastnika, se imenuje **resi gibanje**, sledi pa ravno izpeljavi, ki smo jo navedli zgoraj. Najprej torej določimo funkcijo $t(s)$, s tem lahko potem definiramo novo krivuljo $q_s(s)$ kot $q_s(s) = q(t(s))$, nato pa rešimo gibanje (od tod ime) za novo parametrizacijo na enak način, kot smo to počeli do sedaj. Risanje rešitve je prav tako nespremenjeno.

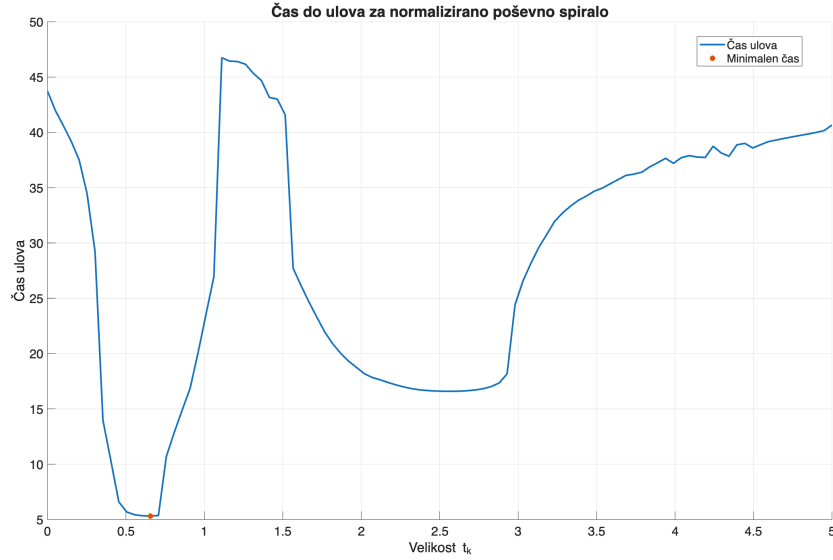
Poglejmo si torej rezultate, če spiralo z variabilno hitrostjo reparametriziramo. Nastavimo enake nastavitve kot prej, torej da je hitrost obeh točk ravno 1, $t_k = 1$, dodatno pa malenkost zmanjšamo toleranco na 0.15, da pride do

trka. Če bi toleranco pustili pri enaki vrednosti, bi se točki sicer približali, a se zaradi enake hitrosti obeh, spet nikoli ne bi ujeli. Dokler sta točki še malenkost narazen, se pes še bliža lastniku tudi v normalni smeri, ko pa postaneta skoraj vzporedni pa je pomembna praktično samo še velikost hitrosti. Gibanje za reparametrizirano krivuljo je prikazano na sliki 10.



Slika 10: Gibanje po reparametrizirani spirali

Ker delamo na reparametrizirani krivulji, ki jo izračunamo preko numeričnega iskanja ničle funkcije $s(t)$, je takšno reševanje računsko bolj zahtevno od uporabe funkcije `lovi`, zato smo minimum iskali na intervalu med 0 in 5 z le 100 vmesnimi točkami. Rezultati so prikazani na spodnjem grafu, kjer je označena tudi minimalna vrednost. Pes lastnika ujame v $t = 5.317$ s, kar je minimalni čas pri vrednosti $t_k = 0.656$. Podobno kot pri krožnici in premici pa ponovno vidimo, da optimalen t_k ni zelo blizu 0, vendar je od ničle malenkost odmaknjen.



Slika 11: Primerjava časov ulovov za reparametrizirano poševno spiralo

6.4 Lissajousova krivulja

Obpravnavajmo sedaj še eno periodično krivuljo, ki jo parametriziramo z

$$q(t) = \begin{bmatrix} \sin(2t) \\ \sin(3t) \end{bmatrix}. \quad (7)$$

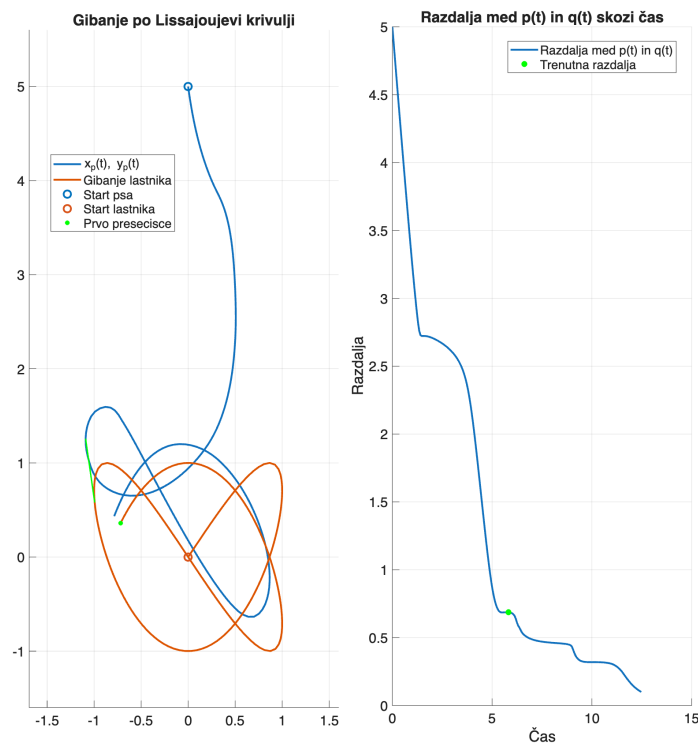
Krivulja ima spet bolj neenakomerno hitrost, zato bomo ta primer reševali s funkcijo **resi gibanje**. Poglejmo si, kako izgleda gibanje za tak primer, ko si za začetno točko izberemo $(0, 5)$. Na sliki 12 opazimo, da lahko v primeru enake hitrosti pes zelo dobro sledi svojemu lastniku in ga razmeroma hitro ulovi.

Poskusimo upoštevati dosedanje ugotovitve glede parametra t_k in ga nastavimo na $t_k = 0.5$, torej malenkost stran od ničle. Dodatno še povečajmo hitrost psa iz 1 na 2 enoti. S temi spremembami pričakujemo, da bo pes ulovil lastnika mnogo prej, kot v prejšnjem primeru.

Kot je razvidno iz slike 13, smo sklepali pravilno. Pes je v taki konfiguraciji potreboval zgolj $t = 2.619$ s, da je ujel svojega lastnika, medtem ko je v prejšnjem primeru potreboval več kot 12 sekund, kar predstavlja bistveno razliko.

7 Komentarji in sklepi

Tekom izdelave projektne naloge so se mi posvetili številni komentarji in ideje. Nekatere izmed teh sem vključil, kot na primer reparametrizacija krivulj, medtem ko drugih, recimo dinamične hitrosti, nisem raziskoval. Področje je gotovo zelo široko, možnosti za specifične scenarije pa je ogromno. Vsako krivuljo,



Slika 12: Gibanje po Lissajousovi krivulji

ki nam pade na pamet, lahko analiziramo na različne načine in individualno skušamo razumeti ter opisati povezavo med nastavljenimi parametri.

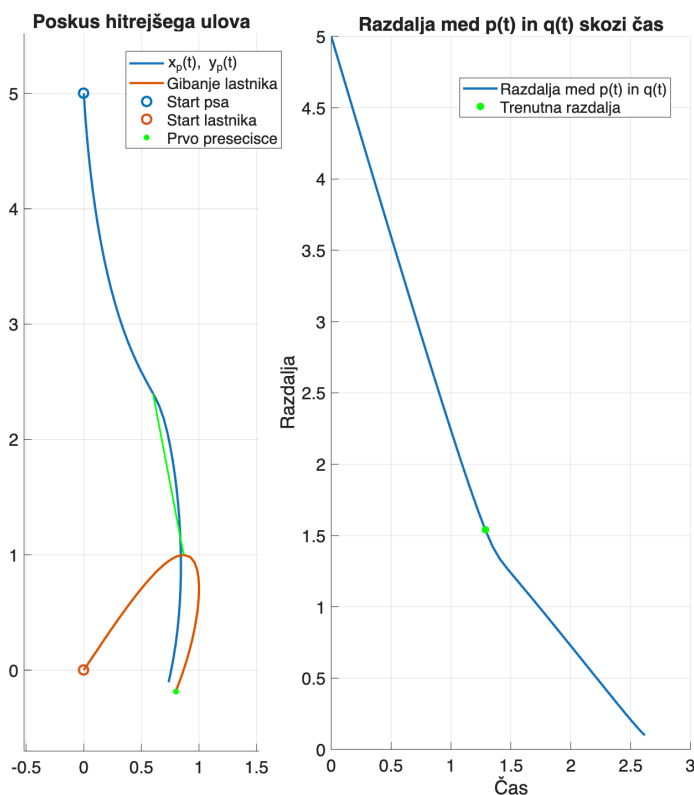
Na tem mestu se splača omeniti, da so grafi sicer narisani s priložnimi programi, a so prav vsi potem ročno dodelani in prilagojeni za poročilo.

7.1 Omejitve dela

Seveda imamo za izdelavo vsake projektne naloge na voljo zgolj končno časa, zato se je potrebno na neki točki zadovoljiti z opravljenim delom. V projektni nalogi sem sicer prikazal 5 različnih krivulj in skušal na zanimiv način prikazati vpliv različnih spremenljivk na rešitev gibanja. Kljub temu je veliko krivulj, ki bi bile same po sebi zanimive za obravnavo, a niso dobile te možnosti.

Pogosto sem naletel tudi na težave glede časovne zahtevnosti izvajanja. Graf na sliki 5 se mi zdi izjemno zanimiv in predstavlja dober vir informacij o zastavljenem problemu, a bi risanje takšnega grafa pri vseh primerih porabilo več časa, kot bi to bilo smiselno. Res sem bil presenečen nad polkrožno obliko v omenjenem grafu in se mi zdi razlog nastanka zelo zanimiv.

Omejitev je bila tudi količina strani, ki jih imamo na voljo za pisanje naloge. Dodatne strani so mi prišle zelo prav in sem z njimi lahko predstavil tudi repa-



Slika 13: Gibanje z manjšim t_k in večjo hitrostjo

rametrizacijo, kar sicer ne bi bilo možno, res pa je, da so vsi grafi precej veliki in vsaka krivulja dobi kar nekaj svojih grafov.

7.2 Možne izboljšave

Prepričan sem, da bi lahko marsikatero poglavje bolj pregledno in bolj jasno zasnoval in oblikoval. Poleg dodatne razširitve obravnavanih krivulj, bi lahko dodatno ustvaril več preizkusov zvez med spremenljivkami. Zanimivo bi blo opazovati primere, ko je začetna točka dlje stran od krivulje. S tem bi lahko opazovali, kako dobro se obnesejo veliki oziroma majhni t_k , ko je potrebno zgolj grobo približevanje. V luči te ideje bi bilo prav tako zelo zanimivo spreminjati hitrost psa, da bi ta recimo pridobival na hitrosti, če bi gibanje trajalo dlje, saj bi s tem povečali možnost za ulov.

Zanimivo bi bilo poskusiti gibanje analitično rešiti še za kakšne druge krivulje, ne zgolj za premico. Analitično rešitev bi lahko poiskali za polinomske krivulje, zato bi bila takšna obravnava zanimiva. Žal pa analitično reševanje v splošnem ne bi bilo možno, ko bi imeli opravka s trigonometričnimi funkcijami, saj bi enačbe postale precej zahtevne za ročno reševanje.

Seveda bi lahko izdelal tudi interaktivne programe za prikaz rešitev, a se zaradi hitrosti iskanja rešitev tega nisem lotil, saj so enako učinkoviti, kot če bi zgolj ponovno pogнал kodo, ki pa potrebuje nekaj sekund za izvajanje. Kot interaktivni del projekta lahko štejemo animacije, ki jim lahko sledimo ob klicu funkcij, grafični vmesniki pa niso pretirano koristni.

7.3 Sklepi

Tekom primerov smo opazovali različne konfiguracije gibanja iz katerih lahko grobo sklepamo o lastnostih problema. Pri vseh primerih smo opazili, da idealni t_k ni 0, niti ni zelo blizu 0, pač pa se optimalen t_k nahaja malenkost stran od ničle. To je sicer intuitiven rezultat, saj neka mera napovedovanja gibanja gotovo pomaga pri lovljenju. Izpostavil bi, da je ta sklep osnovan na podlagi obravnavanih primerov in ne splošne rešitve. Zelo verjetno je, da je optimalen t_k močno odvisen od hitrosti točke na krivulji in njene ukrivljenosti oziroma nepredvidljivosti. Sklepamo torej, da je pri močno ukrivljenih krivuljah bolj ustrezen majhen t_k , pri manj ukrivljenih pa utegne biti ustrezen tudi malenkost večji t_k .

Naslednja ugotovitev, ki bi jo lahko tudi uganili, je, da je pogosto ulov zelo odvisen od hitrosti psa. Ta učinek je močnejši, bolj kot se obe točki gibljeta na isti krivulji, torej ko je njuno gibanje skoraj vzporedno. V limitnem primeru, ko se obe gibljeta v isto smer na isti premici, lahko upravičeno in pravilno sklepamo, da bo pes ujel lastnika zgolj, če bo hitrejši od svojega lastnika, sicer pa mu ta pobegne.

8 Zaključek

V projektni nalogi smo predstavili izpeljavo in matematično ozadje tipičnega problema iz sledilne analize. Z uporabo dveh različnih funkcij za reševanje problema smo podrobneje obravnavali štiri krivulje in na primerih spoznali povezavo med hitrostjo, velikostjo t_k , lastnosti o minimalnih časih ulovov ter dodatno obravnavali reparametrizacijo krivulje preko dolžinskega parametra. Prostora za izboljšavo je veliko. Verjetno je najbolj zanimiva nadgradnja ravno relativno spreminjanje hitrosti psa glede na čas trajanja simulacije.

Literatura

- [1] Thales Azevedo in Anderson Pelluso. “Space pirates: A pursuit curve problem involving retarded time”. V: *American Journal of Physics* 90.10 (2022), str. 730–735.
- [2] MathWorks. *ODE Event Location*. <https://www.mathworks.com/help/matlab/math/ode-event-location.html>. [Online; dostop 1 julija 2026]. 2026.

- [3] Wikipedia contributors. *Arc length* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Arc_length. [Online; dostop 1 julij 2026]. 2026.
- [4] Wikipedia contributors. *Lissajous curve* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Lissajous_curve. [Online; dostop 1 julij 2026]. 2026.
- [5] Wikipedia contributors. *Pursuit curve* — *Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Pursuit_curve. [Online; dostop 1 julija 2026]. 2026.