SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Projektni zadatak iz kolegija *Primjena umjetne inteligencije*

OPTIMIZACIJA PARAMETARA MODELSKOG PREDIKTIVNOG UPRAVLJANJA POMOĆU GENETSKOG ALGORITMA

Rijeka, lipanj 2025.

Leonard Mikša
0069086808

Sadržaj

1. Uvod

Upravljanje dinamičkim sustavima u stvarnom vremenu predstavlja jedan od ključnih izazova u modernoj automatizaciji, posebno kada se radi o nelinearnim, više-varijabilnim dinamičkim sustavima. Modelsko prediktivno upravljanje (eng. *Model Predictive Control*; MPC) se u tom kontekstu ističe kao napredna strategija upravljanja koja koristi matematički model sustava za predviđanje budućih stanja i optimizaciju upravljačkih signala.

Iako MPC nudi visoku razinu fleksibilnosti i optimalnosti, njegovu izvedbu uvelike određuju pravilno podešeni parametri kao što su horizont predikcije i matrice težinskih faktora. Neoptimalno podešeni parametri mogu rezultirati sporim odzivima, povećanim pogreškama u stacionarnom stanju ili prevelikim opterećenjem na korišteni hardver.

Upravo iz tog razloga, u ovom radu implementira se i analizira primjena genetskog algoritma - heurističke metode optimizacije inspirirane prirodnim evolucijskim procesima - za automatsku optimizaciju parametara modelskog prediktivnog upravljanja. Ostvareno je upravljanje pneumatskim aktuatorom pomoću MPC algoritma, te se genetskim algoritmom optimiziraju vrijednosti horizonta predikcije p i elementa Q_{11} regulacijske matrice.

Cilj rada je demonstrirati kako se kombinacijom modelskog prediktivnog upravljanja i genetskog algoritma može postići preciznije, brže i robusnije upravljanje sustavom, uz istovremeno smanjenje potrebe za ručnim podešavanjem i subjektivnom evaluacijom performansi.

2. Model Predictive Control

Modelsko prediktivno upravljanje (eng. *Model Predictive Control; MPC*) je napredni algoritam upravljanja koji koristi dinamički model sustava u prostoru stanja kako bi predvidio i optimirao trajektorije varijabli stanja [1].

Ono što MPC čini superiornijim u odnosu na konvencionalne algoritme upravljanja poput klasične PID regulacije, LQR-a (eng. *Linear Quadratic Regulator*) i mnogih drugih, je relativno jednostavna primjenjivost na MIMO (eng. *Multiple Input Multiple Output*) sustave, prediktivnost u odnosu na reaktivnost, optimalnost u svakom koraku, itd.

Tablica 2.1 sažeto prikazuje karakteristike MPC regulacije u odnosu na PID i LQR. Neke od značajki MPC-a, poput mogućnosti ograničenja varijabli stanja i ulaza, su ostvarive i u PID i u LQR regulaciji, ali ih je značajno teže implementirati i po potrebi mijenjati.

-	PID	LQR	MPC
Upravljanje	Povratna veza	State-feedback	Predikcije
Model sustava	Nije potreban	Linearni model	Dinamički model
Ograničenja ulaza/stanja	Ne	Ne	Da
Hardverski zahtjevi	Niski	Relativno niski	Visoki
MIMO sustavi	Ne	Da	Da
Implementacija	Jednostavno	Relativno jednostavno	Kompleksno
Korištenje	Klasično upravljanje	Linearni sustavi	Nelinearni sustavi

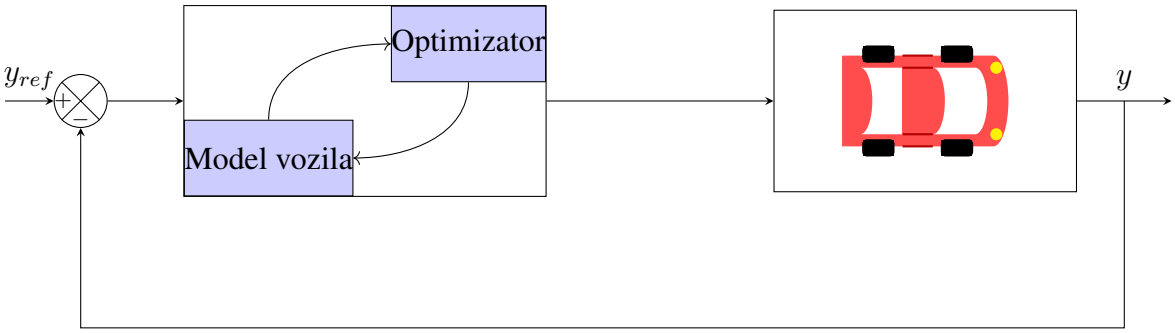
Tablica 2.1. Usporedba značajki PID, LQR i MPC regulacije

Modelsko prediktivno upravljanje se pokazuje izrazito pogodnim za upravljanje složenim, nelinearnim, više-varijabilnim sustavima u kojima je potrebno implementirati ograničenja. Neki od primjera primjene su procesno upravljanje u kemijskoj industriji, zrakoplovno inženjerstvo, autonomna vozila, upravljanje u robotici, itd.

2.1. Parametri algoritma i princip rada

Parametri modelskog prediktivnog upravljanja i princip rada bit će ukratko ilustrirani na primjeru autonomnog vozila.

S obzirom da je MPC implementiran na DSP-u (eng. *Digital Signal Processor*), jedan od temeljnih parametara je **vrijeme uzorkovanja** T_s. Budući da ovaj parametar nije specifičan posebno za MPC nego za sve digitalne algoritme upravljanja, njega nije potrebno posebno opisivati.

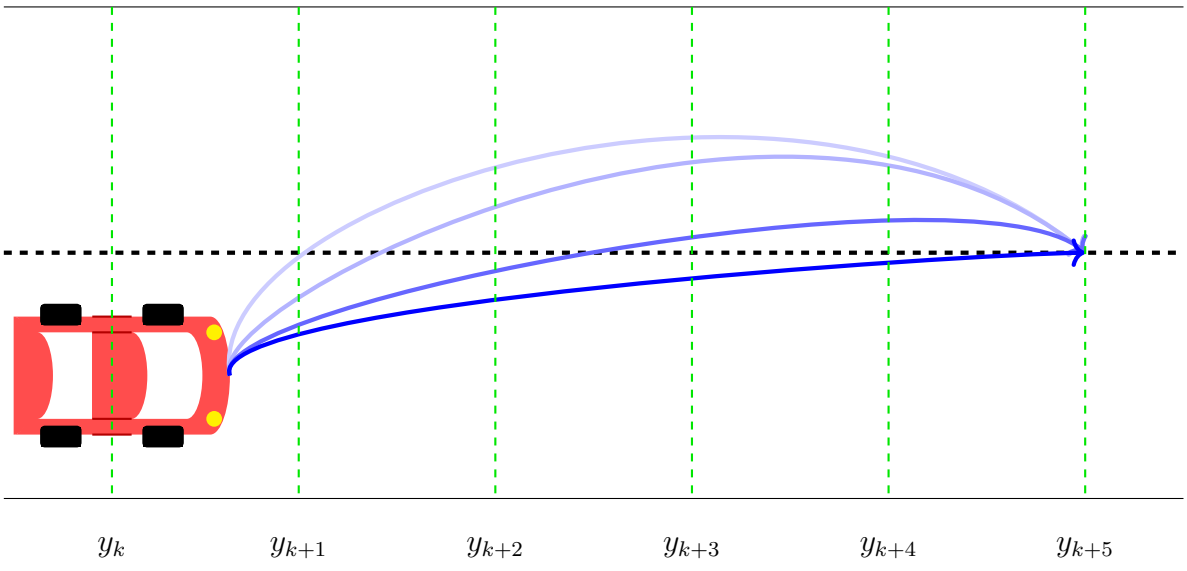


Slika 2.1. Ilustrativna shema upravljanja autonomnim vozilom korištenjem MPC

Kao što je prikazano na slici 2.1, MPC regulator koristi model (u prostoru stanja) autonomnog vozila kako bi predvidio ponašanje budućih stanja izlazne varijable (pomak, kut volana, brzina, akceleracija, itd.) y_{k+1}, y_{k+2}, ..., y_{k+p}.

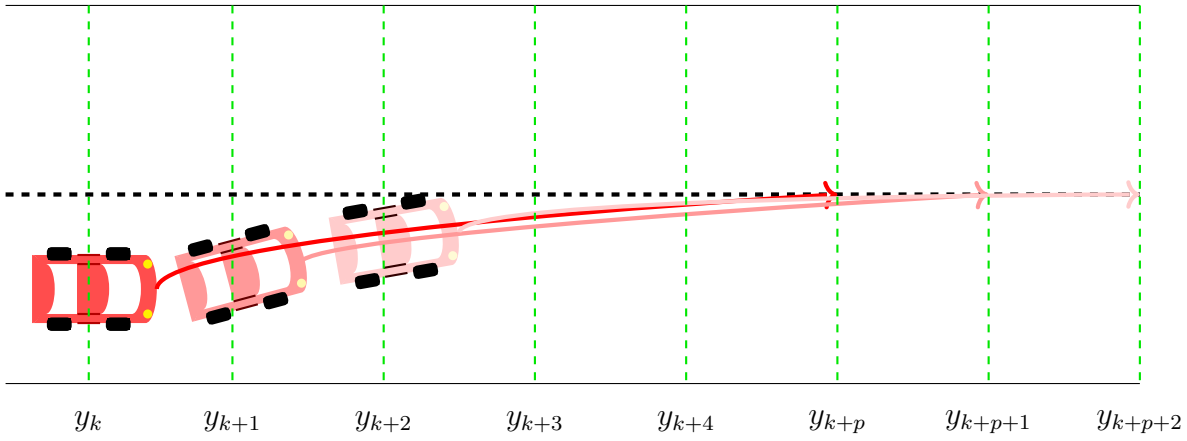
Parametar koji govori koliko daleko u budućnost MPC predviđa je ***horizont predikcije*** (eng. *prediction horizon*) i označava se sa p. Slika 2.2 prikazuje princip ovog parametra u slučaju regulacije kuta zakreta volana pri čemu je isprekidana horizontalna linija referenca lateralne pozicije automobila (sredina jedne trake na cesti).

U trenutnom vremenskom koraku k, MPC simulira trajektoriju automobila za slučaj zakreta volana od npr. 5°. Zatim ponavlja postupak za različite kuteve zakreta volana kako bi pronašao predviđenu putanju najbliže referentnoj poziciji. Za sistematičnost ovog postupka zaslužan je optimizator kao podsustav MPC regulatora, uzimavši u obzir i brzinu promjene kuta zakreta volana kako bi izbjegao trzaje i prenagla skretanja, te slična ograničenja.



Slika 2.2. Ilustracija horizonta predikcije autonomnog vozila na cesti; $p = 5$

U idućem koraku $k+1$, algoritam ponavlja postupak i pritom odbacuje prethodne optimizacije zbog mogućih poremećaja poput jakog vjetra, oštećenog kolnika, detektiranog semafora ili pješačkog prijelaza, itd.

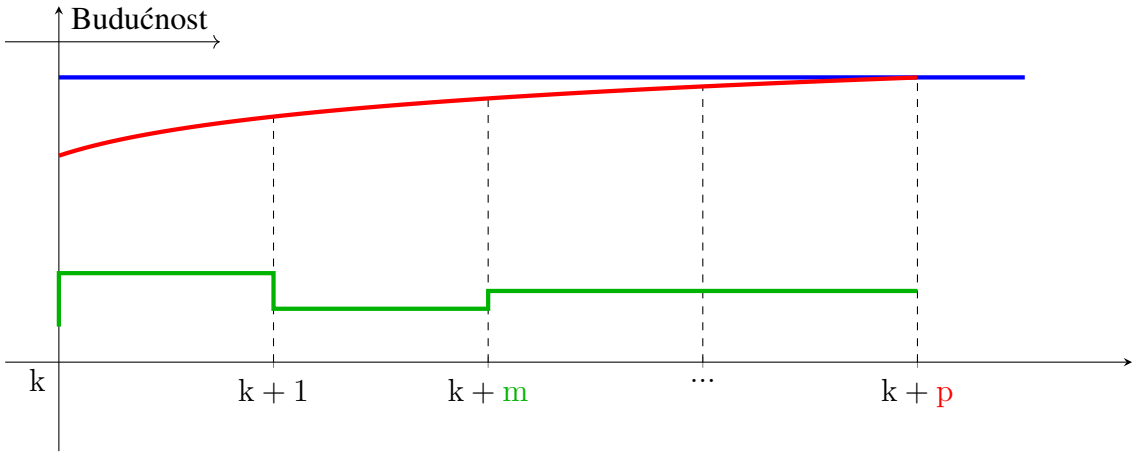


Slika 2.3. Illustracija iterativne prirode horizonta predikcije

Veći horizont predikcije je za samo upravljanje bolji, ali predstavlja opterećenje za mikroupravljač na kojem se algoritam izvršava u stvarnom vremenu. Premali p bi na primjeru autonomne vožnje mogao uzrokovati nemogućnost optimiranja kočenja vozila ispred semafora, dok preveliki p povlači prekomjerno odbacivanje prethodnih optimizacija.

Jedna od preporuka za odabir horizonta predviđanja je obuhvaćanje 20 do 30 uzoraka tranzijenta odziva sustava u otvorenoj petlji [2].

Idući važan parametar je ***horizont upravljanja*** (eng. *control horizon*) koji se označava sa *m*, a govori koliko daleko u budućnost MPC generira upravljačke signale na temelju predviđenih trajektorija stanja ostvarenih preko p.



Slika 2.4. Ilustracija horizonta upravljanja u usporedbi s horizontom predikcije

Dakle, m budućih upravljačkih signala nastoji forsirati objekt upravljanja u ostvarivanje trajektorije predviđene horizontom predviđanja. S obzirom da samo prvih nekoliko upravljačkih signala ima značajan utjecaj na ponašanje sustava, horizont upravljanja ne mora biti previsok, a nikako veći od horizonta predviđanja. Preporuka je da horizont upravljanja bude 10% do 20% vrijednosti horizonta predviđanja.

Idući važan parametar su **ograničenja**. Ograničenja se mogu implementirati i na ulaze i na varijable stanja. Sva primjenjena ograničenja uzimaju se u obzir u svakom koraku izvršavanja algoritma, i prilikom svih predviđanja koje MPC izvršava.

Na primjer, ako algoritam ustvrdi da je optimalna brzina za ostvarivanje optimalne trajektorije 150 km/h, a ograničenje brzine je postavljeno na 130 km/h, ta optimalna brzina neće biti uzimana u obzir.

Ograničenja mogu biti tvrda i meka (eng. *hard and soft constraints*). Tvrda ograničenja se ni u kom slučaju ne smiju prekršiti, a primjer je gore spomenuta maksimalna brzina vozila, puna traka na cesti, ili maksimalni kut zakreta robotskog manipulatora u robotici kako bi se osiguralo da robot ne izađe iz radnog prostora ili ne udari u neki drugi objekt.

Meka ograničenja postoje da se izbjegne neizvedivost optimiziranja, a ona dozvoljavaju kratkotrajno kršenje. Primjer mekog ograničenja je isprekidana traka na cesti koja dozvoljava preticavanje vozila ispred (naravno, tek nakon što je očitanjima ostalih senzora ustvrđena sigurnost preticavanja).

Posljednji važan parametar bitan za opseg ovog rada su **regulacijske matrice** $\mathbf{Q}$ i $\mathbf{R}$. Pomoću njih se parametrira sustav sukladno željenim performansama; matricom $\mathbf{Q}$ se podešava agresivnost upravljanja željenih varijabli stanja, a matricom $\mathbf{R}$ se penalizira potrošnja energije.

Njima se zapravo modificira *cost function* optimizacijskog problema definiran kao

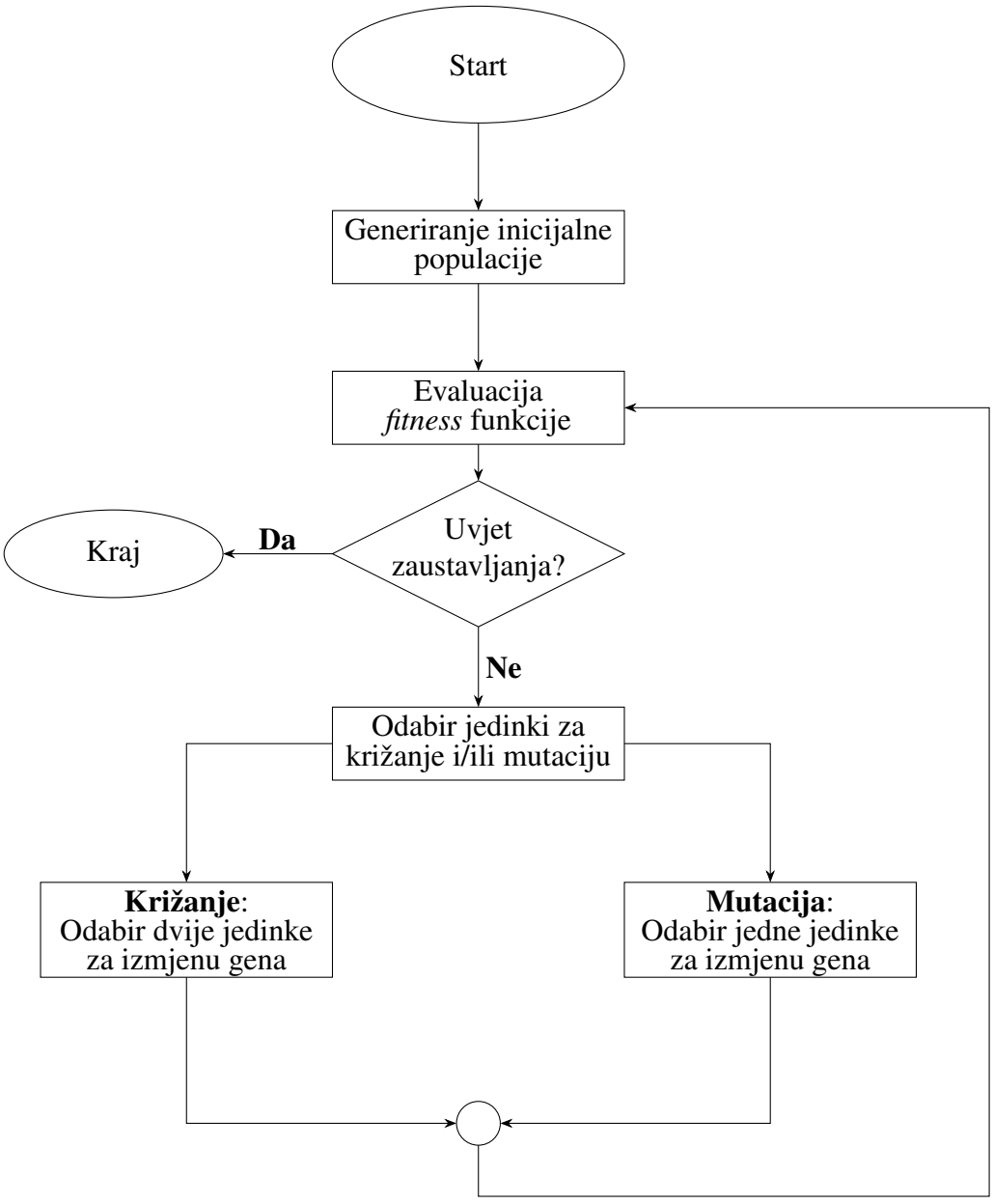
$$J = \int_0^\infty \left(x^T \mathbf{Q} x + u^T \mathbf{R} u\right) dt. \tag{2.1}$$

Jednadžbom (2.1) osigurana je minimizacija površine ispod krivulje odziva (što brži odziv sa što manjom greškom u odnosu na referentni signal).

3. Genetski algoritmi

Genetski algoritmi su heuristički optimizacijski algoritmi inspirirani principima genetike i prirodne selekcije [3], [4], [5]. Pojednostavljeno, genetski algoritam je metoda pretraživanja prostora rješenja s ciljem rješavanja optimizacijskog problema [6].

Osnovna struktura i princip rada genetskog algoritma prikazana je na slici 3.1 [3], [5].



Slika 3.1. Osnovna struktura genetskog algoritma

Inicijalna populacija može biti eksplicitno definirana ili generirana nasumično. Poželjno je generirati što više jedinki nasumično kako bi se pokrio što je veći mogući prostor koji se pretražuje.

Fitness funkcija ili funkcija podobnosti je ključna za uspješno rješavanje optimizacijskog problema jer ona definira *koliko je neko rješenje dobro*. Ako se radi o funkciji dvije varijable, tj. ako se optimiziraju dva parametra, tada se zapravo traži globalni minimum *fitness* funkcije u trodimenzionalnom prostoru rješenja. S obzirom na to, ključno je da ona bude što je moguće bolje definirana.

Uvjet zaustavljanja može biti definiran egzaktnim brojem generacija ili konvergencijom algoritma, a također bitno utječe na ishod samog optimizacijskog problema. Na primjer, ako je definiran egzaktan broj generacija, postoji mogućnost da optimizacija ne bude izvršena do kraja, a ako je uvjet zaustavljanja konvergencija algoritma, moguće je da pronađeno rješenje bude suboptimalno. Stoga je najbolji odabir kombinacija tih dvaju uvjeta, tj. postavljanje visokog broja generacija u kombinaciji s uvjetom za konvergenciju (npr. razlika u najboljim rješenjima u prethodnih X generacija manja od ε; $\varepsilon <<$).

Odabir jedinki za križanje i/ili mutaciju vrši se na više načina. Operacije za stvaranje iduće populacije **mutacija** i **križanje**, svaka od njih može imati svoje kriterije. Neki od kriterija odabira jedinki su odabir temeljen na ruletu (eng. *Roulette Wheel Selection*), stohastičko univerzalno uzorkovanje (eng. *Stochastic Universal Sampling*), odabir prema rangu (eng. *Rank Selection*), i turnirski odabir (eng. *Tournament Selection*).

U ovom radu korištena je **turnirska selekcija**, koja se obično koristi na velikom broju populacije, a odabire dvije ili tri nasumične jedinke i izabire najbolju između njih.

Za **križanje** najčešće se odabiru dvije jedinke koje imaju najoptimalnije vrijednosti *fitness* funkcije. Na taj način, te dvije jedinke prenose svoje dobre karakteristike na jedinku u idućoj generaciji.

Za **mutaciju**, odabire se jedna jedinka, najčešće nasumično. Mutacija osigurava raznolikost populacije i pretraživanje šireg prostora rješenja. Obično je stopa mutacije relativno mala, jer se s velikom stopom mutacije genetski algoritam svodi na nasumično pretraživanje.

Na novoj populaciji se ponovno evaluira *fitness* funkcija i provjerava uvjet zaustavljanja. Proces se ponavlja sve dok jedan od uvjeta zaustavljanja nije zadovoljen.

3.1. Prednosti, nedostaci i primjena genetskih algoritama

Prednosti genetskih algoritama uključuju paralelizam, mogućnost pretraživanja širokog prostora rješenja, otpornost na šum u podacima, mogućnost rješavanja više-dimenzionalnih optimizacijskih problema, itd. Pored navednog, genetski algoritmi zahvaljujući operacijama križanja i mutacije imaju otpornost na zaglavljivanje u lokalnim optimumima, te gotovo uvijek pronalaze **globalni** optimum [3].

Najzahtjevnije ograničenje genetskih algoritama je identificiranje i formiranje *fitness* funkcije, s obzirom da je upravo ona ključna za provjeravanje optimalnosti rješenja i definiranje samog optimizacijskog problema. Loše definirana *fitness* funkcija rezultirat će suboptimalnim, ili potpuno neoptimalnim rješenjima.

Ostali nedostaci uključuju mogućnost pojave prerane konvergencije, nejedinstvenost optimalnih parametara samog algoritma (veličina populacije, stope križanja i mutacije, broj generacija, uvjeti zaustavljanja, itd.), nemogućnost pronalaska egzaktnog optimalnog rješenja s obzirom da je metoda sama po sebi heuristička, itd.

Genetski algoritmi pronalaze svoju primjenu u raznim vrstama optimizacijskih problema, od robotike i algoritama upravljanja, strojnog učenja i obrade signala, pa sve do primjene u analizi podataka i predikcijama u izrazito nelinearnim dinamičkim sustavima.

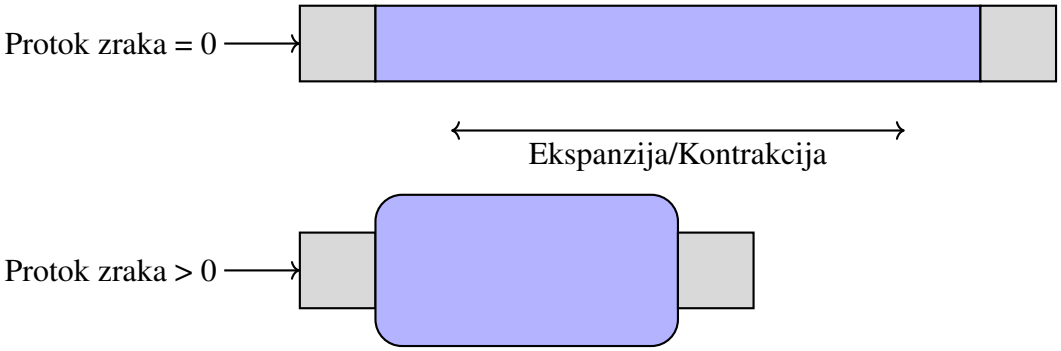
4. Primjena genetskog algoritma na Model Predictive Control

Kao što je spomenuto u potpoglavlju 2.1., modelsko prediktivno upravljanje ima razne parametre koji ga definiraju, a ovise o objektu upravljanja, zahtjevima na performanse, dostupnom hardveru, itd.

Parametri koji su spomenuti, a koje je cilj optimizirati, tj. pronaći optimalan balans između njih, su numeričke vrijednosti člana matrice $\mathbf{Q}$ Q_{11} i horizonta predikcije p. Naime, za manje vrijednosti horizonta predikcije potrebne su veće numeričke vrijednosti u matrici $\mathbf{Q}$ kako bi se ostvarile iste karakteristike odziva regulirane varijable stanja.

4.1. Objekt modelskog prediktivnog upravljanja

Objekt kojeg se regulira je pneumatski aktuator ilustriran na slici 4.1. S obzirom da modelsko prediktivno upravljanje zahtijeva model sustava u prostoru stanja, potrebno je opisati pneumatski aktuator diferencijalnim jednadžbama prvog reda.



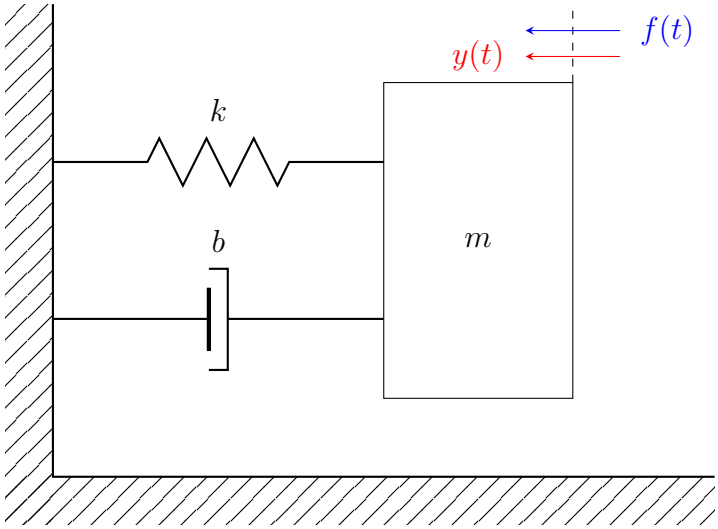
Slika 4.1. Princip rada pneumatskog aktuatora

Mehaničko gibanje pneumatskog aktuatora može se opisati jednostavnim masa-opruga-prigušnica modelom, prikazanim na slici 4.2. Diferencijalna jednadžba koja opisuje takav sustav definirana je 2. Newtonovim zakonom

$$f(t) - m\ddot{y}(t) - b\dot{y}(t) - ky(t) = 0, \tag{4.1}$$

gdje m predstavlja masu aktuatora, b faktor trenja, a k konstantu opruge. Budući da je jednadžba gibanja aktuatora diferencijalna jednadžba drugog reda, potrebno ju je redefinirati i zapisati kao dvije diferencijalne jednadžbe prvog reda.

Stoga uvodimo dvije nove varijable $x_1(t)$ i $x_2(t)$, koje su zapravo varijabla linearnog pomaka $y(t)$ proširena u dvije dimenzije.



Slika 4.2. Shema sustava masa-opruga-prigušnica

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix} \tag{4.2}$$

Slijede redefinirane diferencijalne jednadžbe prvog reda

$$\dot{x}_1(t) = x_2(t) \tag{4.3}$$

$$\dot{x}_2(t) = \frac{1}{m} f(t) - \frac{b}{m} x_2(t) - \frac{k}{m} x_1(t) \tag{4.4}$$

Naposljetku, silu ne tretiramo kao nezavisni ulaz, nego ju izražavamo preko treće varijable stanja $x_3(t)$ koja predstavlja protok zraka.

$$\dot{x}_2(t) = \frac{\gamma A}{m} x_3(t) - \frac{b}{m} x_2(t) - \frac{k}{m} x_1(t), \tag{4.5}$$

gdje je γ konstanta proporcionalnosti ventila koja povezuje protok zraka s tlakom; tlak je jednak umnošku konstante proporcionalnosti i protoka zraka, a A predstavlja poprečni presjek pneumatskog aktuatora.

Treća jednadžba koja opisuje model pneumatskog aktuatora je jednadžba pneumatske dinamike koju aproksimiramo diferencijalnom jednadžbom prvog reda

$$\tau \dot{q}(t) + q(t) = K u(t), \tag{4.6}$$

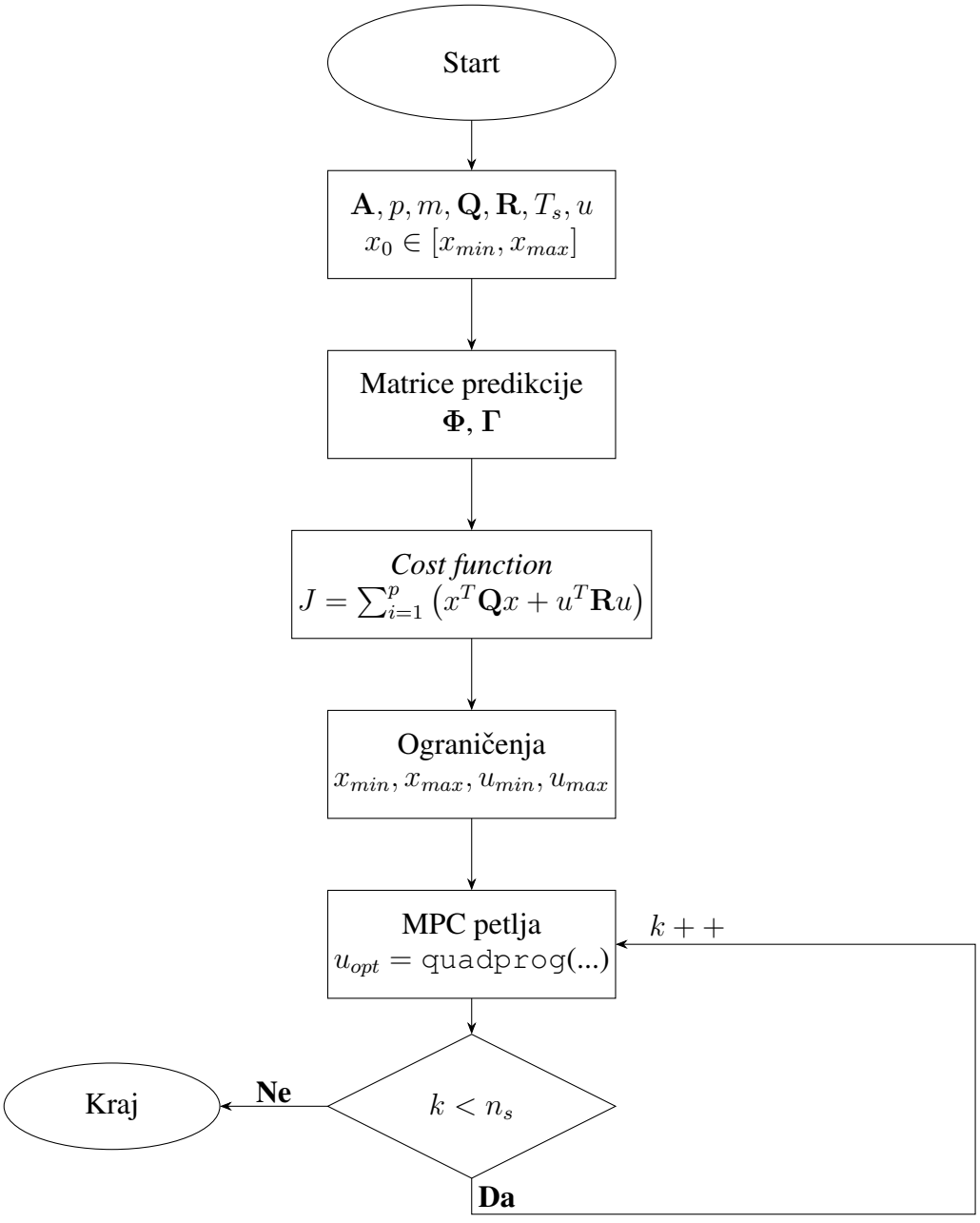
gdje $q(t)$ predstavlja protok zraka, τ vremensku konstantu ventila, K pojačanje ventila, a $u(t)$ napon ventila.

$$\dot{q}(t) = -\frac{1}{\tau} q(t) + \frac{K}{\tau} u(t) \tag{4.7}$$

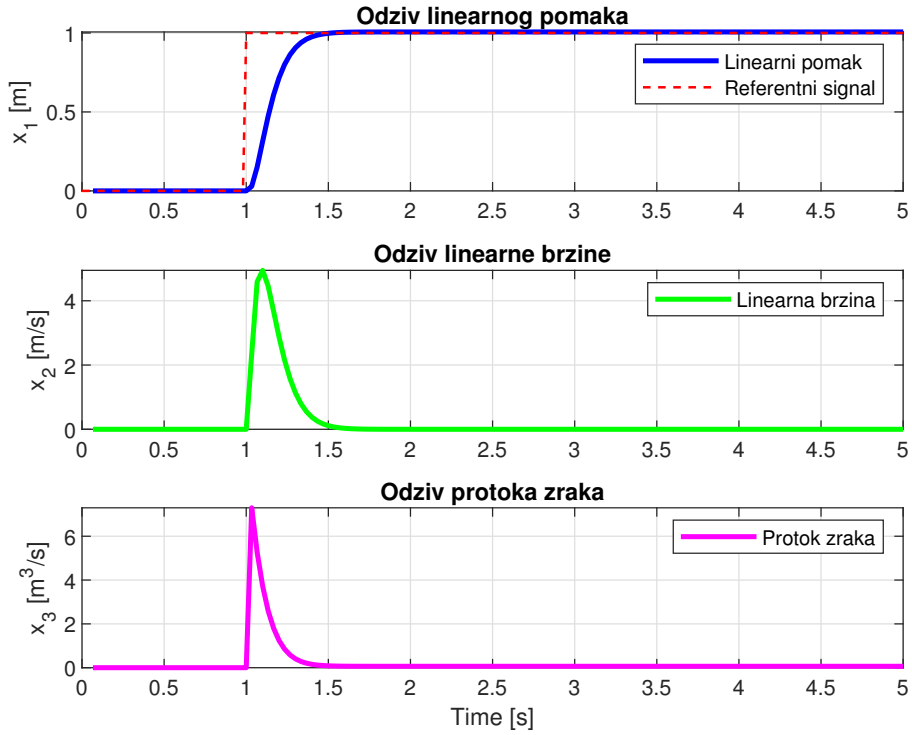
Slijedi model sustava u prostoru stanja

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{q}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{k}{m} & -\frac{b}{m} & \frac{\gamma A}{m} \\ 0 & 0 & -\frac{1}{\tau} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ q(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{K}{\tau} \end{bmatrix} u(t) \tag{4.8}$$

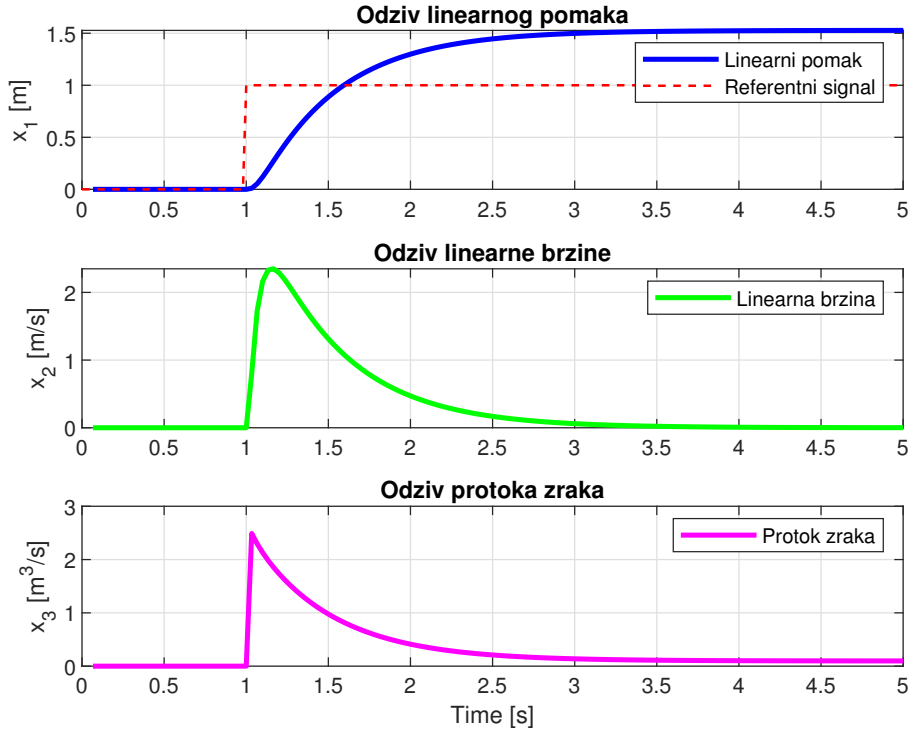
U nastavku je prikazan dijagram toka izvršavanja modelskog prediktivnog upravljanja.



Slika 4.3. Pojednostavljeni dijagram toka MPC algoritma



Slika 4.4. Odzivi varijabli stanja pneumatskog aktuatora, $Q_{11} >> p$



Slika 4.5. Odzivi varijabli stanja pneumatskog aktuatora, $Q_{11} \approx p$

Na slici 4.4 vidljivo je da MPC algoritam uspješno prati referentni signal, ali pod uvjetom da su horizont predikcije i numerička vrijednost matrice $\mathbf{Q}$ Q_{11} izbalansirani. Naime, u ovom slučaju je nužno da parametar Q_{11} bude značajno veći od horizonta predikcije kako bi se postiglo agresivnije upravljanje linearnim pomakom.

U protivnom, vrijeme porasta odziva linearnog pomaka bit će veće i pogreška u stacionarnom stanju će biti prisutna, što je prikazano na slici 4.5. U ovom slučaju, horizont predikcije je premali, a agresivnost upravljanja linearnim pomakom nedovoljna.

4.2. Formiranje *fitness* funkcije

S obzirom da je cilj odrediti optimalan balans između parametara Q_{11} i p, *fitness* funkcija bit će definirana kao funkcija dvije varijable koju je moguće prikazati u trodimenzionalnom prostoru $f(p, Q_{11})$. Kako bi se, ovisno o tim parametrima, odredila kvaliteta odziva linearnog pomaka pneumatskog aktuatora, potrebno je tu kvalitetu ocijeniti pokazateljima poput greške u stacionarnom stanju i vremena porasta odziva.

4.2.1. Analiza odziva linearnog pomaka pneumatskog aktuatora

Analizirani pokazatelji odziva linearnog pomaka pneumatskog aktuatora su greška u stacionarnom stanju i vrijeme porasta. Nadvišenje nije uzeto u obzir jer je ovaj sustav prigušen, pa će odziv u cijelom prostoru rješenja biti aperiodski.

Matrica rješenja dobivena u MATLAB-u u svakoj iteraciji MPC algoritma je dimenzija 3×150, odnosno broj varijabli stanja $\times$ broj koraka simulacije koji je omjer ukupnog trajanja simulacije i vremena uzorkovanja. Varijabla stanja od interesa je linearni pomak, pa iz te matrice izdvajamo prvi redak, dok drugi i treći (linearna brzina i protok zraka) nisu u fokusu interesa.

Greška u stacionarnom stanju računa se kao apsolutna vrijednost razlike posljednjeg uzorka u vektoru linearnog pomaka i posljednjeg uzorka u vektoru referentnog signala
$e = \lim_{t \to \infty} |x(t) - x_{ref}(t)|$.

Za izračun vremena porasta, najprije je potrebno odrediti u kojem uzorku dolazi do promjene u step funkciji i zatim preko dobivenog indeksa odrediti vrijeme u kojem je došlo do promjene. Amplituda step funkcije računa se kao razlika posljednje vrijednosti u vektoru referentnog signala i vrijednosti u uzorku prije određenog indeksa. Vrijednost amplitude je potrebna kako bi se odredilo 10% i 90% odziva čime je vrijeme porasta definirano.

Vremenski trenuci u kojima je došlo do prijelaza donje granice od 10% i gornje granice od 90% određuju se na sličan način kao i vrijeme promjene step funkcije. Razlika ta dva vremenska trenutka predstavlja vrijeme porasta odziva linearnog pomaka.

U nastavku slijedi dio koda kojim je implementiran izračun greške u stacionarnom stanju i vremena porasta odziva linearnog pomaka pneumatskog aktuatora. Ova sekcija se izvršava u svakom koraku genetskog algoritma i nakon svakog koraka MPC algoritma unutar genetskog algoritma.

```
%% Izračun parametara odziva linearnog pomaka:
pomak = x_trajectory(1, :);    %Izdvajanje pomaka iz matrice rješenja

steady_state_error = abs(pomak(end) - ref(end));

%Detekcija rastućeg brida:
step_idx = find(diff(ref) ~= 0, 1, 'first') + 1;
step_time = t(step_idx);
step_size = ref(end) - ref(step_idx - 1);
%Izračun gornje i donje granice za vrijeme porasta:
low = ref(step_idx - 1) + 0.1 * step_size;
upp = ref(step_idx - 1) + 0.9 * step_size;
%Izračun vremenskih indeksa u kojima odziv prelazi 10% i 90%:
idx_start = find(pomak(step_idx:end) >= low, 1, 'first') + step_idx - 1;
idx_end   = find(pomak(step_idx:end) >= upp, 1, 'first') + step_idx - 1;

%Izračun vremena porasta:
if ~isempty(idx_start) && ~isempty(idx_end)
  rise_time = time(idx_end) - time(idx_start);
else
  rise_time = NaN;
end
```

4.2.2. Formiranje *dataseta* i interpolacija

Idući korak je formiranje *dataseta*, odnosno skupa podataka na temelju kojeg će se formirati *fitness funkcija*. Parametri odziva su izračunati izvršavanjem MPC algoritma za ekvidistantne točke parova vrijednosti Q_{11} i p na rasponu $p \in [5, 65]$ s korakom 10 i $Q_{11} \in [500, 10000]$ s korakom 2000.

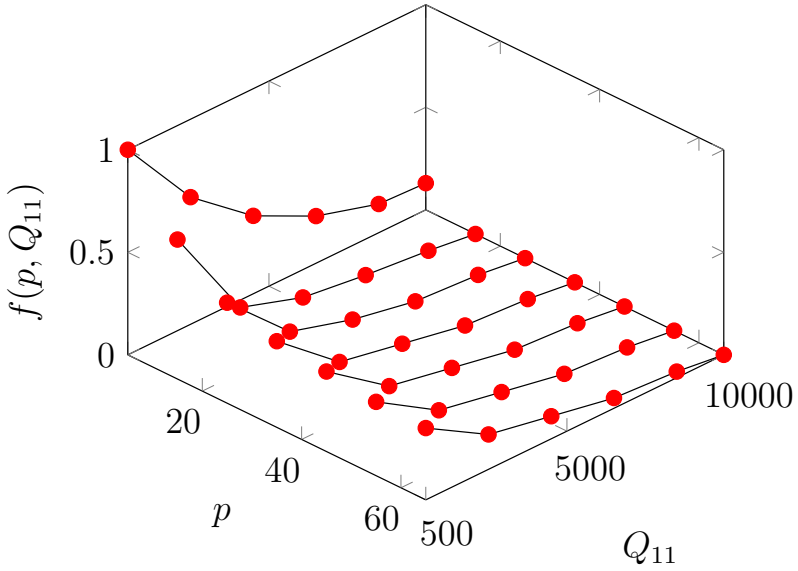
Na taj način dobivaju se diskretne točke u trodimenzionalnom prostoru koje predstavljaju egzaktne točke na kojima leži ploha *fitness* funkcije. Vrijednost *fitness* funkcije je normalizirana; $f(p, Q_{11}) \in [0, 1]$.

Normaliziranjem očitanih vrijednosti greške u stacionarnom stanju i vremena porasta

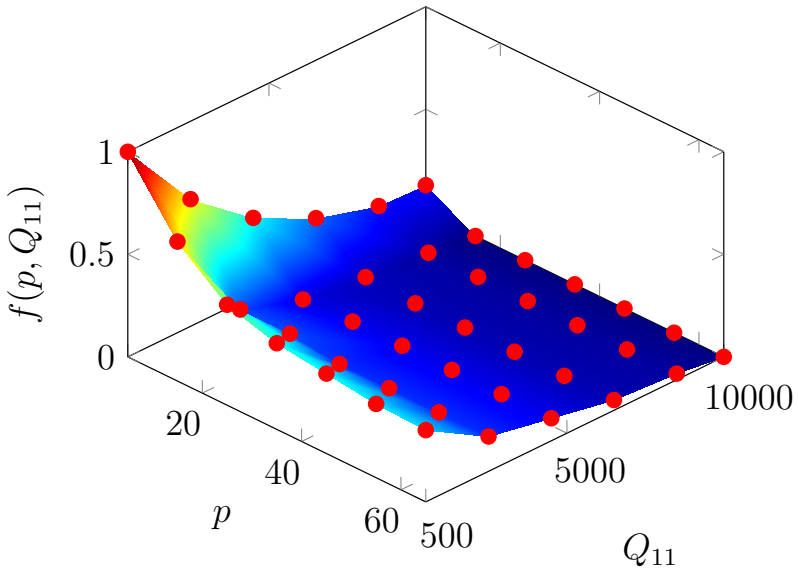
$$e_{NORM} = \frac{e - \min(e)}{\max(e) - \min(e)}, \tag{4.9}$$

$$t_{rNORM} = \frac{t_r - \min(t_r)}{\max(t_r) - \min(t_r)}, \tag{4.10}$$

vrijednosti *fitness* funkcije možemo definirati zbrajanjem normaliziranih pokazatelja kvalitete pomnoženim s težinskim faktorom. Pošto je greška u stacionarnom stanju teži kriterij od vremena porasta signala, njen težinski faktor je jednak 1, a za vrijeme porasta 0.5.



Slika 4.6. Diskretne točke fitness funkcije



Slika 4.7. Interpolirana fitness funkcija

Slika 4.6 prikazuje trodimenzionalni prikaz *fitness* funkcije u egzaktnim točkama, a slika 4.7 trodimenzionalni prikaz interpolirane *fitness* funkcije. Interpolacija je ostvarena definiranjem mreže na prostoru rješenja s malim koracima po x i y osi, te korištenjem MATLAB funkcije `scatteredInterpolant` koja interpolira zadani diskretni skup točaka u 2D ili 3D prostoru.

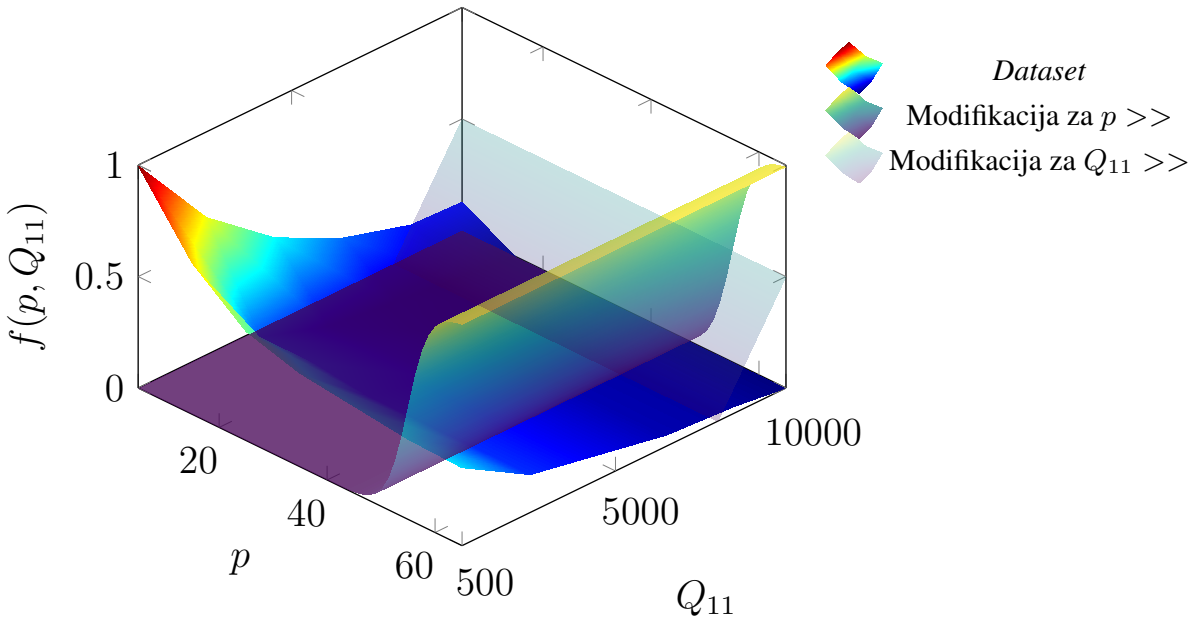
```
data = readtable('mpc_fitness_table.csv'); %Učitavanje dataseta
%Normaliziranje greške u stacionarnom stanju i vremena porasta:
norm_error = (data.Tracking_Error - min(data.Tracking_Error)) ./ (max(data.
  Tracking_Error) - min(data.Tracking_Error));
norm_rise  = (data.Rise_Time - min(data.Rise_Time)) ./ (max(data.Rise_Time)
  - min(data.Rise_Time));

fitness = 1 * norm_error + 0.5 * norm_rise; %Definiranje fitness funkcije
%Interpolacija:
F_interp = scatteredInterpolant(data.p, data.Q11, fitness, 'linear');
[p_fine, q_fine] = meshgrid(5:1:65, 500:100:10000);
fitness_base = F_interp(p_fine, q_fine);
```

4.2.3. Dodatne modifikacije

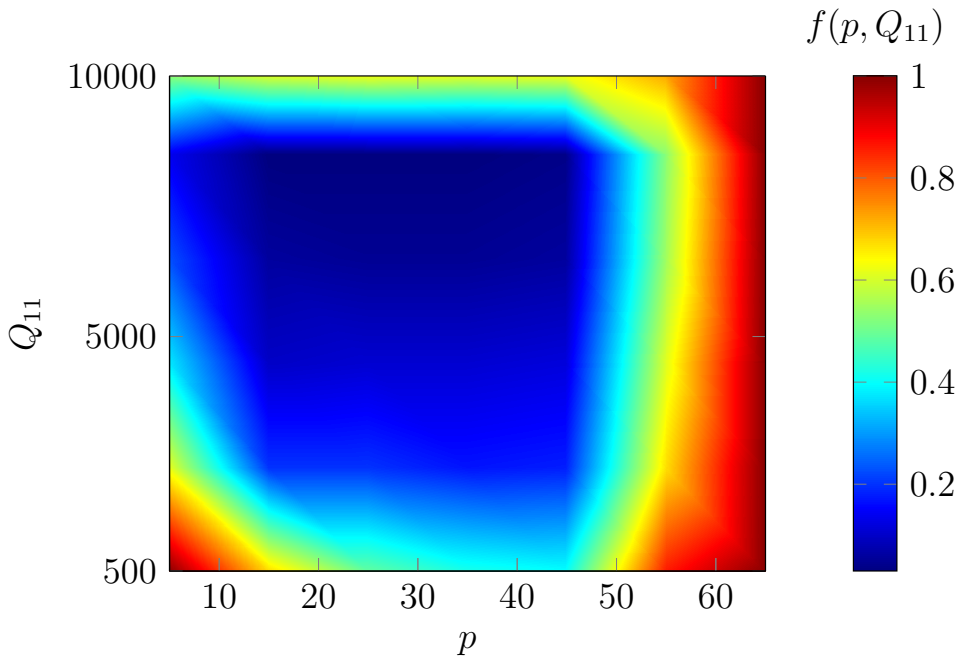
Kako je već prethodno spomenuto, veći horizont predikcije generalno daje bolje rezultate, ali rezultira većim zahtjevima za računalnim resursima. Naime, u svakom koraku izvršavanja algoritma modelskog prediktivnog upravljanja, optimizacijski problem se rješava p puta. Stoga je potrebno modificirati *fitness* funkciju ovisno o dostupnom mikroupravljaču i njegovim mogućnostima. *Fitness* funkciju u ovom slučaju modificiramo za $p > 55$, tako da osiguramo da genetski algoritam ne traži optimalna rješenja s tim vrijednosti horizonta predikcije.

Posljednja modifikacija *fitness* funkcije je 'podizanje' trodimenzialnog prikaza za Q_{11} >>. Ovo u stvarnosti nije potrebno jer numerička vrijednost Q_{11} ne utječe na sam hardver, ali je svejedno ostvareno zbog postizanja konveksnosti funkcije i bolje vizualizacije.

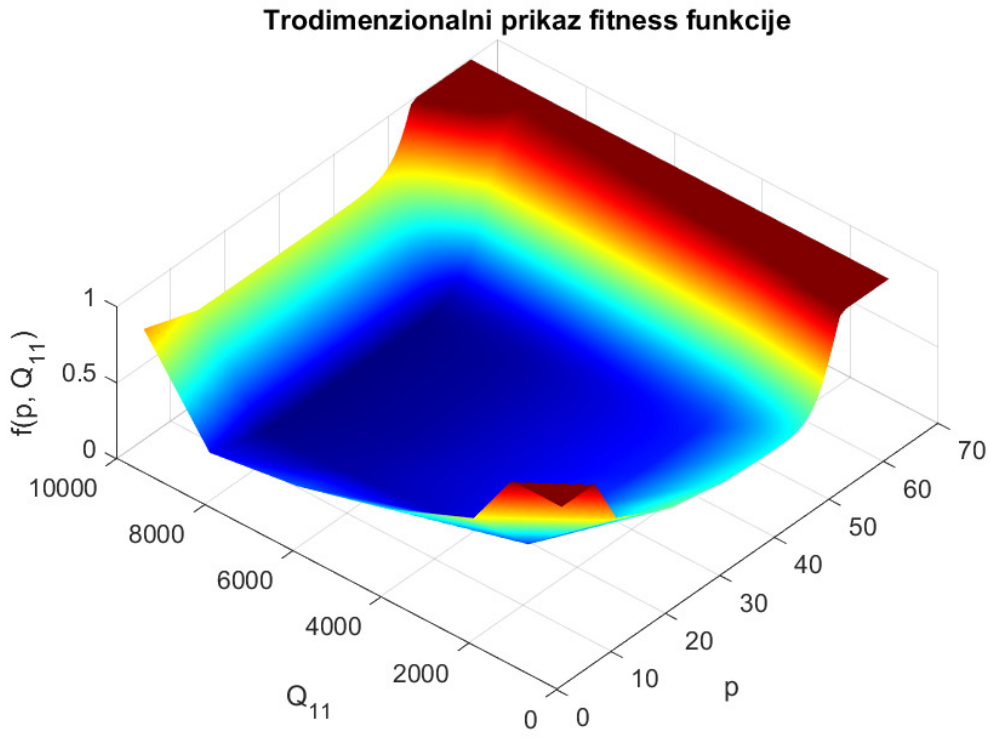


Slika 4.8. Modificirana fitness funkcija

Slika 4.8 prikazuje dodatne modifikacije na *fitness* funkciji sa slike 4.7, a slika 4.9 prikazuje prikaz konačne *fitness* funkcije odozgo.

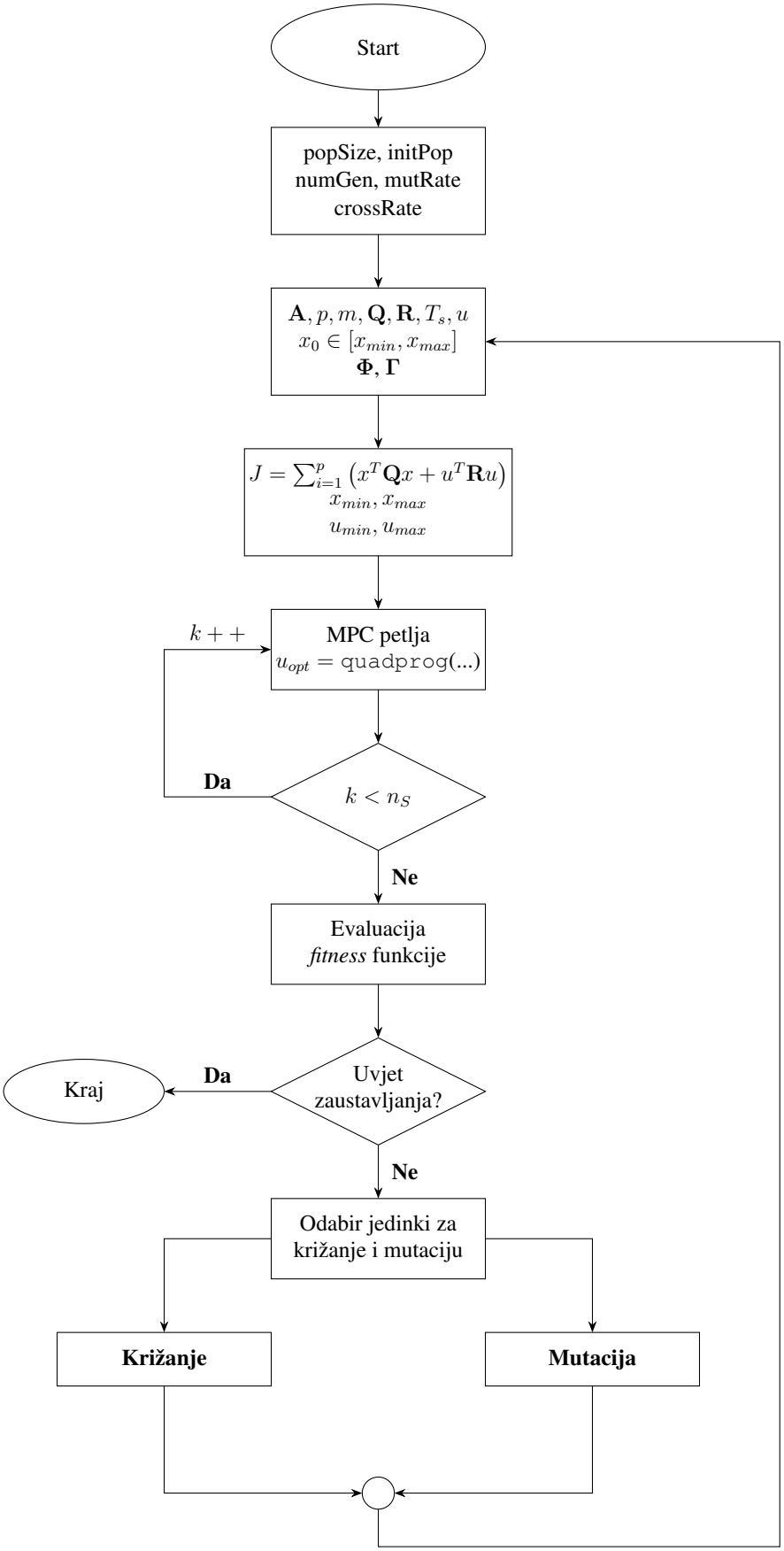


Slika 4.9. Prikaz konačne fitness funkcije odozgo



Slika 4.10. Prikaz fitness funkcije dobivene u MATLAB-u

4.3. Implementacija genetskog algoritma za optimizaciju parametara MPC algoritma



Slika 4.11. Dijagram toka genetskog algoritma za optimizaciju MPC parametara

Algoritam modelskog prediktivnog upravljanja implementiran je u MATLAB-u s ugrađenim opcijama tvrdih ograničenja i na ulazni signal i na sve tri varijable stanja. Genetski algoritam je nadogradnja postojećeg, a kako je već navedeno, optimiziraju se horizont predikcije p i član regulacijske matrice Q_{11}.

Najprije se definiraju parametri genetskog algoritma koji uključuju veličinu populacije, broj generacija, stopu mutacije i stopu križanja. Definira se i prostor rješenja postavljanjem krajnjih granica x i y osi, odnosno p i Q_{11} osi. Zatim se generira inicijalna populacija na način da svih 20 jedinki ima niske vrijednosti p i Q_{11} tako da se može vidjeti postepeni napredak po generacijama.

```
popSize = 20;              %Veličina populacije
numGenerations = 20;       %Broj generacija
mutationRate = 0.5;        %Stopa mutacije
crossoverRate = 0.95;      %Stopa križanja
pRange = [5, 65];          %Raspon horizonta predikcije
QRange = [500, 10000];     %Raspon Q11

%Inicijalna populacija:
populationP = pRange(1) + 10 * rand(popSize, 1);
populationQ = QRange(1) + 800 * rand(popSize, 1);
% populationX = pRange(1) + (pRange(2) - pRange(1)) * rand(popSize, 1);
% populationY = QRange(1) + (QRange(2) - QRange(1)) * rand(popSize, 1);
```

Inicijalna populacija se u prvom koraku izvršavanja algoritma koristi u izvršavanju dijela modelskog prediktivnog upravljanja. Kao što je vidljivo sa slike 4.11, MPC petlja je unutarnja petlja cjelokupnog algoritma, odnosno izvršava se n_S puta (u ovom slučaju 150) u svakoj generaciji genetskog algoritma.

Dio koda vezan za samo upravljanje nije prikazan, s obzirom da on nije u fokusu ovog rada. U nastavku je dan okvirni hodogram MPC algoritma u MATLAB-u u obliku komentiranih sekcija koda.

```
%% Definiranje referentnog signala (vektor u)

%% Model sustava u prostoru stanja (matrice A, B, C i D)

%% Početni uvjeti

%% Definiranje placeholdera (vektora i matrica) za spremanje rezultata

%% MPC petlja (for k = 1:steps)

%% Prikaz rezultata

%% Analiza pokazatelja kvalitete odziva
```

Idući korak je evaluacija *fitness* funkcije, koja je definirana kao funkcija koja očitava interpolirane vrijednosti trodimenzionalne plohe dobivene kako je opisano u potpoglavlju 4.2.. Ova mogućnost ostvarena je korištenjem MATLAB funkcije `interp2`. Najbolje jedinke u trenutnoj populaciji spremaju se u vektorske varijable koje će se kasnije koristiti i u odabiru za križanje i mutaciju, i u interaktivnom prikazu rezultata.

```
fitnessFinal = min(1.0, fitness_base + 1./1+exp(-0,5*(p-55)) + 0.6*max(0, q
  -8500)/(10000-8500)); %Konačna (modificirana) fitness funkcija
fitnessFunction = @(p, Q11) interp2(p_fine, q_fine, fitnessFinal, p, Q11);

fitnessValues = fitnessFunction(populationP, populationQ);
%Spremanje najboljih jedinki (p i Q11):
[bestFitness, bestIdx] = min(fitnessValues);
bestFitnessHistory(gen) = bestFitness;
bestpQHistory(gen, :) = [populationP(bestIdx), populationQ(bestIdx)];
```

Postavljena su dva uvjeta zaustavljanja - jedan je maksimalni broj generacija koji je definiran na početku koda varijablom `numGenerations`, a drugi je konvergencija genetskog algoritma, odnosno neostvarivanje značajnog napretka u prethodnih nekoliko generacija.

```
%Inicijalno definirani placeholderi izvan GA petlje:
noImprovementLimit = 5;       %Broj generacija bez napretka
lastBestFitness = -Inf;       %Prethodna najbolja vrijednost
stagnationCount = 0;          %Varijabla stagnacije

%Provjera napretka:
if abs(bestFitnessHistory(gen) - lastBestFitness) < 1e-4
  stagnationCount = stagnationCount + 1;
else
  stagnationCount = 0;            %Resetiranje ako dođe do napretka
end
lastBestFitness = bestFitnessHistory(gen);

%Zaustavljanje algoritma u slučaju prekoračenja limita:
if stagnationCount >= noImprovementLimit
  disp('Stopping early due to no improvement.');
  break;
end

%Zaustavljanje u slučaju maksimalnog broja generacija:
if gen >= numGenerations
  break;
end
```

Odabir jedinki za križanje i mutaciju vrši se turnirskom selekcijom na način da se odaberu dvije jedinke koje se zatim uspoređuju. Nakon definiranja *placeholdera* za novu populaciju, unutar `for` petlje se nasumično odabiru dvije jedinke MATLAB funkcijom `randi` koja vraća vektor 2×1

s vrijednostima u rasponu od 1 do `popSize`. Bolji kandidati se određuju na temelju njihove podobnosti funkcijom `min`, te se ugrađuju u varijablu nove populacije koja se u idućim koracima ažurira i križanjem i mutacijom.

Za evoluciju iduće generacije korišteno je tzv. aritmetičko križanje (eng. *arithmetic crossover*) koje se može interpretirati linearnom interpolacijom gena između dvije odabrane jedinke [7], te Gaussova mutacija koja jedinkama odabranim za mutaciju dodjeljuje uniformno distribuiranu slučajnu vrijednost. Stope mutacije su adaptivne - snažnije mutacije na početku, a slabije mutacije jednom kad najbolja jedinka prijeđe nizak prag *fitness* funkcije od 0.035.

```
%Turnirska selekcija:
newPopulationp = zeros(popSize, 1);
newPopulationQ = zeros(popSize, 1);

for i = 1:popSize
  candidates = randi([1 popSize], 2, 1);              %Odabir dvije jedinke
  [~, bestIdx] = min(fitnessValues(candidates));      %Evaluacija
  newPopulationp(i) = populationP(candidates(bestIdx));
  newPopulationQ(i) = populationQ(candidates(bestIdx));
end

%Aritmetičko križanje:
for i = 1:2:popSize-1
  if rand < crossoverRate
    alphaCrossover = rand;          %Faktor križanja
    newPopulationp(i) = alphaCrossover * newPopulationp(i) + (1 -
  alphaCrossover) * newPopulationp(i+1);
    newPopulationp(i+1) = (1 - alphaCrossover) * newPopulationp(i) +
  alphaCrossover * newPopulationp(i+1);
    newPopulationQ(i) = alphaCrossover * newPopulationQ(i) + (1 -
  alphaCrossover) * newPopulationQ(i+1);
    newPopulationQ(i+1) = (1 - alphaCrossover) * newPopulationQ(i) +
  alphaCrossover * newPopulationQ(i+1);
   end
end

%Gaussova mutacija:
if lastBestFitness < 0.035
  mutationMask = rand(popSize, 1) < mutationRate;
  mutationAmountp = 1 * randn(popSize, 1);
  mutationAmountQ = 10 * randn(popSize, 1);
  newPopulationp = newPopulationp + mutationMask .* mutationAmountp;
  newPopulationQ = newPopulationQ + mutationMask .* mutationAmountQ;
else
  mutationMask = rand(popSize, 1) < mutationRate;
  mutationAmountp = 10 * randn(popSize, 1);
  mutationAmountQ = 1500 * randn(popSize, 1);
```

```
34      newPopulationp = newPopulationp + mutationMask .* mutationAmountp;
35      newPopulationQ = newPopulationQ + mutationMask .* mutationAmountQ;
36    end
37
38    %Zadržavanje vrijednosti u rasponu prostora rješenja:
39    newPopulationp = max(pRange(1), min(pRange(2), newPopulationp));
40    newPopulationQ = max(QRange(1), min(QRange(2), newPopulationQ));
41
42    %Ažuriranje populacije:
43    populationP = newPopulationp;
44    populationQ = newPopulationQ;
```

Snažnijim mutacijama se osigurava raznolikost u populaciji i sprječavanje prerane konvergencije, ali preintenzivno mutiranje se svodi na nasumično pretraživanje. Stoga se u početku dozvoljava intenzivnija mutacija jedinki kako bi se pretražio što širi prostor rješenja, a kasnije se intenzitet mutacije smanjuje kako bi algoritam konvergirao u minimum *fitness* funkcije.

U ovom slučaju, *fitness* funkcija je vrlo jednostavna u smislu pronalaska minimuma, s obzirom da populacija ima gotovo linearan pad prema globalnom minimumu po plohi *fitness* funkcije; stoga ni parametriranje genetskog algoritma nije kompliciranо. Međutim, ako *fitness* funkcija ima više lokalnih minimuma, pogotovo onih koji su po svojoj vrijednosti dosta blizu globalnom minimumu, postoji velika vjerojatnost da će genetski algoritam rezultirati suboptimalnim rješenjem.

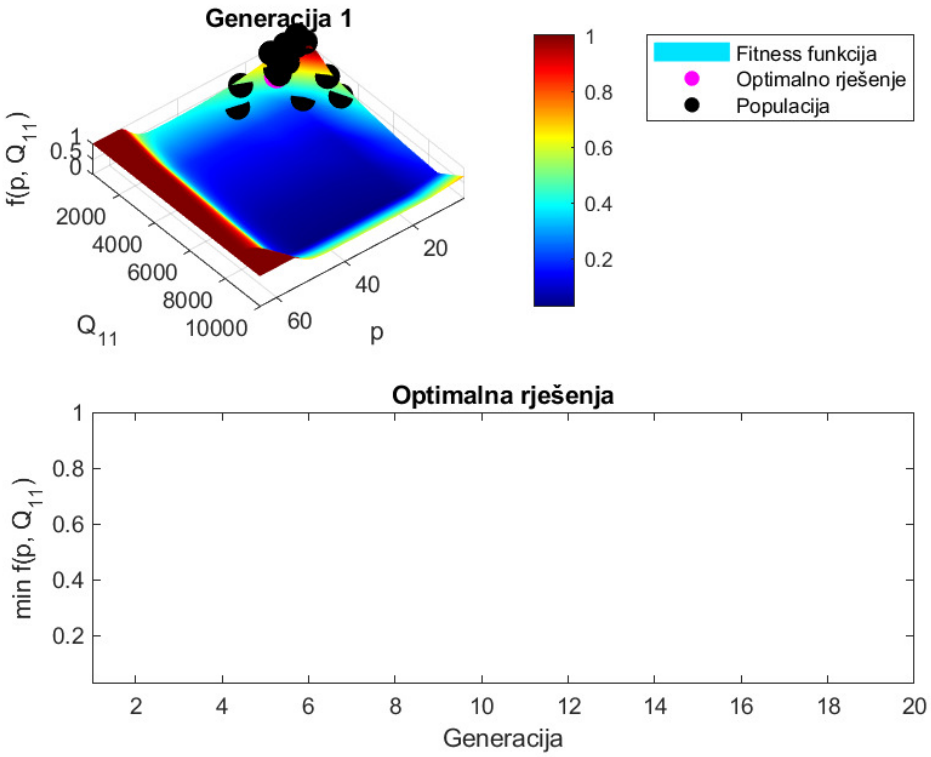
Ovo se u određenoj mjeri može izbjeći pažljivim odabirom stopa mutacije i križanja, uvjeta zaustavljanja, te detaljnom analizom performansi genetskog algoritma. Međutim, *fitness* funkcija je u ovoj konkretnoj primjeni prilično jednostavna pa ne dolazi do opisanih problema i algoritam uspješno konvergira u minimum.

Na samom kraju petlje genetskog algoritma, vrijednosti parametara modelskog prediktivnog upravljanja (horizonta predikcije p i člana upravljačke matrice $\mathbf{Q}$ Q_{11}) se ažuriraju za iduću generaciju genetskog algoritma i iteraciju petlje modelskog prediktivnog upravljanja. Funkcijom `ceil` osigurano je zaokruživanje vrijednosti na prvu iduću cjelobrojnu vrijednost.

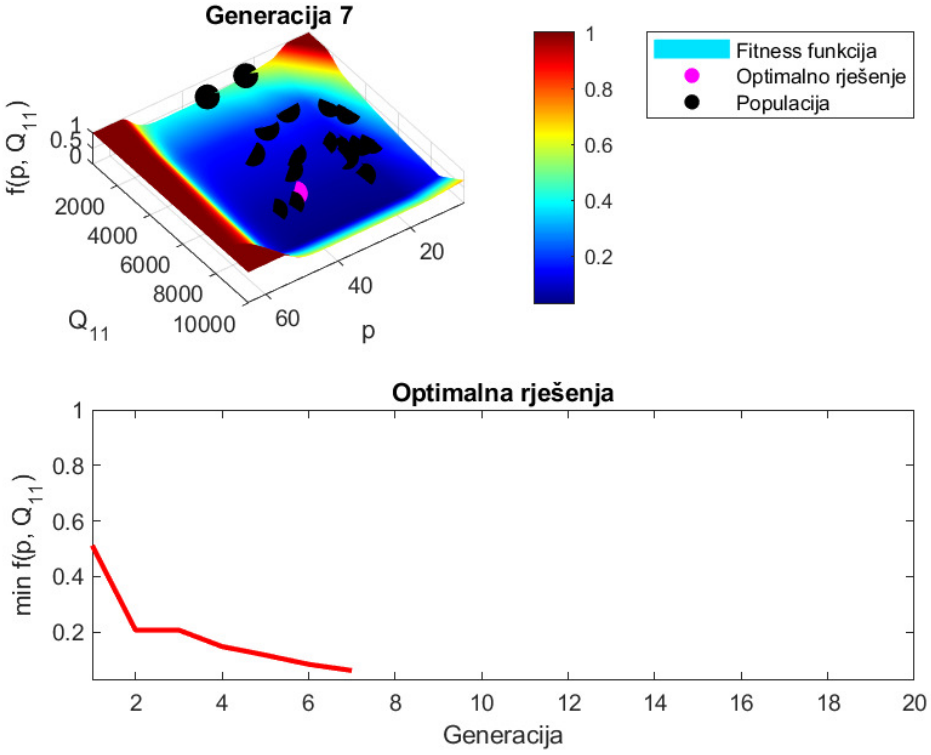
```
1  %Ažuriranje p i Q11:
2  p = ceil(bestpQHistory(gen, 1));
3  Q11 = ceil(bestpQHistory(gen, 2));
```

4.4. Rezultati

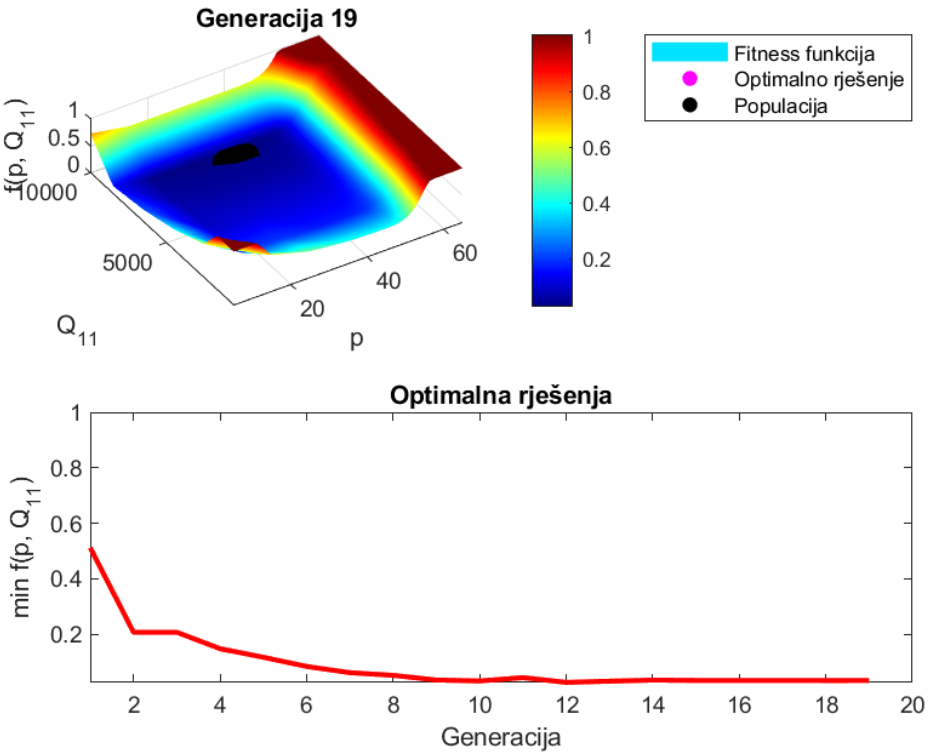
Kao što je ranije napomenuto, inicijalna populacija je namjerno generirana na najnepovoljnijem području *fitness* funkcije, gdje su numeričke vrijednosti i horizonta predikcije i člana upravljačke matrice malene. Slika 4.12 prikazuje inicijalnu populaciju jedinki genetskog algoritma, a slika 4.5 pripadujući odziv varijabli stanja pneumatskog aktuatora.



Slika 4.12. Prikaz inicijalne populacije genetskog algoritma



Slika 4.13. Prikaz sedme generacije genetskog algoritma



Slika 4.14. Prikaz posljednje generacije genetskog algoritma

Slike 4.12, 4.13 i 4.14 prikazuju napredak genetskog algoritma po generacijama. Na donjem podgrafu svake od slika vidljiv je postepeni napredak prema pronalasku globalnog minimuma *fitness* funkcije, a na slici 4.14 vidljivo je da su gotovo sve jedinke konvergirale u globalni minimum nakon 19 generacija.

Dobiveni optimalni par parametara modelskog prediktivnog upravljanja je

$$p = 35 \qquad Q_{11} = 7915$$

a pripadajući odziv varijabli stanja pneumatskog aktuatora prikazan je na slici 4.4. S obzirom na logiku kojom je formirana *fitness* funkcija, slijedi da su ovo optimalne vrijednosti za postizanje:

- najmanje pogreške u stacionarnom stanju
- najmanjeg vremena porasta odziva linearnog pomaka pneumatskog aktuatora
- optimalnog opterećenja na mikroupravljač

5. Zaključak

Kroz ovaj rad demonstrirana je primjena genetskog algoritma za optimizaciju parametara modelskog prediktivnog upravljanja. Optimizacijom horizonta predikcije p i elementa regulacijske matrice Q_{11} dobivene su vrijednosti kojima se ostvaruju brzo i precizno praćenje referentnog signala, minimalnu pogrešku u stacionarnom stanju, te prihvatljivo opterećenje mikroupravljača.

Rezultati su pokazali da genetski algoritam učinkovito konvergira prema globalnom minimumu definirane *fitness* funkcije. Time je potvrđena njegova robusnost i pogodnost za rješavanje optimizacijskih problema uz mogućnost vizualizacije napretka optimizacije, što doprinosi intuitivnom razumijevanju samog genetskog algoritma.

Osim toga, analiza odziva sustava kroz sve faze genetske evolucije ukazuje na značajnu prednost automatiziranog pristupa optimizaciji u odnosu na ručno podešavanje. Time se potvrđuje vrijednost i potencijal daljnje integracije evolucijskih algoritama u području upravljanja, osobito kod nelinearnih i složenih sustava.

Najveći problem za primjenu kombinacije ovih dvaju algoritama na neki stvarni objekt upravljanja je računalna zahtjevnost. Naime, modelsko prediktivno upravljanje je samo po sebi računalno zahtjevan algoritam, s obzirom da u svakom ciklusu izvršavanja MPC predviđa p koraka unaprijed. Dodavanje genetskog algoritma, koji je također iterativan i pretražuje potencijalno velik prostor rješenja, čini *real-time* implementaciju teško izvedivom.

Međutim, optimizacija parametara *offline*, te primjena u inertnim objektima upravljanja u kemijskoj i petrokemijskoj industriji gdje dominantnu ulogu imaju regulacija topline i protoka se ističu kao potencijalne primjene kombinacije ovih dvaju algoritama.

S druge strane, sustavi koji su izrazito dinamički i zahtijevaju brzo izvršavanje algoritma s malim vremenom uzorkovanja, poput robotskih sustava, biomedicinskih procesa i elektromotornih pogona se ističu kao primjeri u kojima kombinacija ovih dvaju algoritama čini upravljanje i optimizaciju presporima.

Bibliografija

[1] J. B. Rawlings, D. Q. Mayne, M. M. Diehl: *"Model Predictive Control: Theory, Computation, and Design"*, Nob Hill Publishing, 2017.

[2] MATLAB Documentation

[3] S. N. Sivanandam i S. N. Deepa: *"Introduction to Genetic Algorithms"*, Springer, 2008.

[4] S. Baressi Šegota: *"Determining the Energy-Optimal Path of Six-Axis Industrial Robotic Manipulators Using Machine Learning and Memetic Algorithms"*, doktorska disertacija, Sveučilište u Rijeci, Tehnički fakultet, 2025.

[5] N. Anđelić: *Predavanja iz kolegija Mehatronički sustavi*, Sveučilište u Rijeci, Tehnički fakultet, 2024.

[6] R. Son, J. Cho, K. Huh: *"Genetic Algorithm based Parameters Optimization of Model Predictive Control in Lane Keeping"*, Korean Society of Automotive Engineers, 2017.

[7] S Interneta: *"Crossover Operators in Genetic Algorithm"*, Geek Culture, 2021.