Ρομποτικά Συστήματα ΙΙ: Εργασία 4

Ονοματεπώνυμο: Βραχωρίτη Αλεξάνδρα

Αριθμός Μητρώου: 1092793

4 Φεβρουαρίου 2026

1 Μοντελοποίηση συστήματος του double pendulum

Στην εκφώνηση δίνεται η δυναμική του συστήματος ως:

$$f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \ddot{q_1} \\ \ddot{q_2} \end{bmatrix} \tag{1}$$

και ορίζοντας $\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$ έχουμε

$$\dot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) - \mathbf{F}(\dot{\mathbf{q}})) \tag{2}$$

όπου

$$\mathbf{M} = \begin{bmatrix} I_1 + I_2 + l_1^2 m_2 + 2\ \mathrm{CoM}_2\ l_1 m_2 l_2 \cos q_2 & I_2 + \mathrm{CoM}_2\ l_1 m_2 l_2 \cos q_2 \\ I_2 + \mathrm{CoM}_2\ l_1 m_2 l_2 \cos q_2 & I_2 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

$$\mathbf{C} = \begin{bmatrix} -2\ \mathrm{CoM}_2\ \dot{q}_2\ l_1\ m_2\ l_2\ \sin q_2 & -\ \mathrm{CoM}_2\ \dot{q}_2\ l_1\ m_2\ l_2\ \sin q_2 \\ \mathrm{CoM}_2\ \dot{q}_1\ l_1\ m_2\ l_2\ \sin q_2 & 0 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

$$\mathbf{G} = \begin{bmatrix} -\mathrm{CoM}_1\ g\ m_1\ \sin q_1 - g\ m_2\big(l_1\ \sin q_1 +\ \mathrm{CoM}_2\ \sin(q_1 + q_2)\big) \\ -\mathrm{CoM}_2\ g\ m_2\ l_2\ \sin(q_1 + q_2) \end{bmatrix} \in \mathbb{R}^{2\times 1}$$

$$\mathbf{F} = \begin{bmatrix} d_1\ \dot{q}_1 + f_{c_1} \sin(100\dot{q}_1) \\ d_2\ \dot{q}_2 + f_{c_2} \sin(100\dot{q}_2) \end{bmatrix} \in \mathbb{R}^{2\times 1}$$

Λύνοντας την εξίσωση (2) ως προς u θα έχουμε:

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{3}$$

2 Βελτιστοποίηση τροχιάς

Για την βελτιστοποίηση της τροχιάς του διπλού εκκρεμούς (Double Pendulum) θα χρησιμοποιηθεί η μέθοδος σημειακής προσαρμογής (collocation) μέσω του κανόνα Hermite-Simpson. Η μέθοδος έχει περιγραφεί αναλυτικά στην Εργασία 3. Για την επίλυση του προβλήματος θα χρησιμοποιήσουμε τη βιβλιοθήκη `cyipopt`. Στόχος μας είναι το double pendulum να κάνει swing-up, με δεδομένο ότι $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$ και στόχος είναι το $\mathbf{x}_f = \begin{bmatrix} \pi & 0 & 0 & 0 \end{bmatrix}^T$. Ο επιλυτής που θα χρησιμοποιήσουμε για την βελτιστοποίηση της τροχιάς θα χρησιμοποιεί τους περιορισμούς που ορίστηκαν με την μέθοδο Herite-Simpson, αλλά και τα όρια στις ροπές των κινητήρων. Επιπλέον, αξιοποιεί το gradient και την hessian της συνάρτησης κόστους, καθώς και την jacobian και την hessian των περιορισμών. Παρακάτω παρουσιάζεται ο αλγόριθμος για την εύρεση της βέλτιστης τροχιάς.

Algorithm 1: Trajectory Optimization via Hermite-Simpson and `cyipopt` library

Input: Initial state $\boldsymbol{x}_0$, Intermediate state $\boldsymbol{x}_m$, Final state $\boldsymbol{x}_f$, Time step dt, Knot points S
Output: Optimized trajectory $\boldsymbol{z}^*$

// 1. Initial Guess Generation using Cubic Splines
$\boldsymbol{c}_{0a}, \boldsymbol{c}_{1a}, \boldsymbol{c}_{2a}, \boldsymbol{c}_{3a}, \boldsymbol{c}_{4a} \leftarrow$ calculate_coeffs$(\boldsymbol{x}_0, \boldsymbol{x}_m, \frac{2}{5}t_f)$
$\boldsymbol{c}_{0b}, \boldsymbol{c}_{1b}, \boldsymbol{c}_{2b}, \boldsymbol{c}_{3b}, \boldsymbol{c}_{4b} \leftarrow$ calculate_coeffs$(\boldsymbol{x}_m, \boldsymbol{x}_f, \frac{3}{5}t_f)$
$\boldsymbol{z}_{init} \leftarrow \varnothing$

for $k \leftarrow 0$ **to** $steps_a$ **do**
  $t \leftarrow k \cdot \frac{dt}{2}$
  $\boldsymbol{q}(t) \leftarrow$ cubic_spline$(t, \boldsymbol{c}_{0a}, \boldsymbol{c}_{1a}, \boldsymbol{c}_{2a}, \boldsymbol{c}_{3a})$
  $\dot{\boldsymbol{q}}(t) \leftarrow$ cubic_spline_dot$(t, \boldsymbol{c}_{0a}, \boldsymbol{c}_{1a}, \boldsymbol{c}_{2a}, \boldsymbol{c}_{3a})$
  $\boldsymbol{u} \leftarrow \mathbf{0}$
  Append $[\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{u}]$ to $\boldsymbol{z}_{\text{init}}$

for $k \leftarrow 0$ **to** $steps_b$ **do**
  $t \leftarrow k \cdot \frac{dt}{2}$
  $\boldsymbol{q}(t) \leftarrow$ cubic_spline$(t, \boldsymbol{c}_{0b}, \boldsymbol{c}_{1b}, \boldsymbol{c}_{2b}, \boldsymbol{c}_{3b})$
  $\dot{\boldsymbol{q}}(t) \leftarrow$ cubic_spline_dot$(t, \boldsymbol{c}_{0b}, \boldsymbol{c}_{1b}, \boldsymbol{c}_{2b}, \boldsymbol{c}_{3b})$
  $\boldsymbol{u} \leftarrow \mathbf{0}$
  Append $[\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \boldsymbol{u}]$ to $\boldsymbol{z}_{\text{init}}$

// 2. Define Bounds
bounds $\leftarrow (\mathbf{l}, \mathbf{u})$

// 3. Compute Gradient and Jacobian
$\nabla c \leftarrow$ grad(cost_function)
$\boldsymbol{J}_{\mathbf{f}} \leftarrow$ jacobian(constraints)
$\boldsymbol{H}_{\mathbf{c}} \leftarrow$ hessian(cost_function)
$\boldsymbol{H}_{\mathbf{f}} \leftarrow$ hessian(constraints)

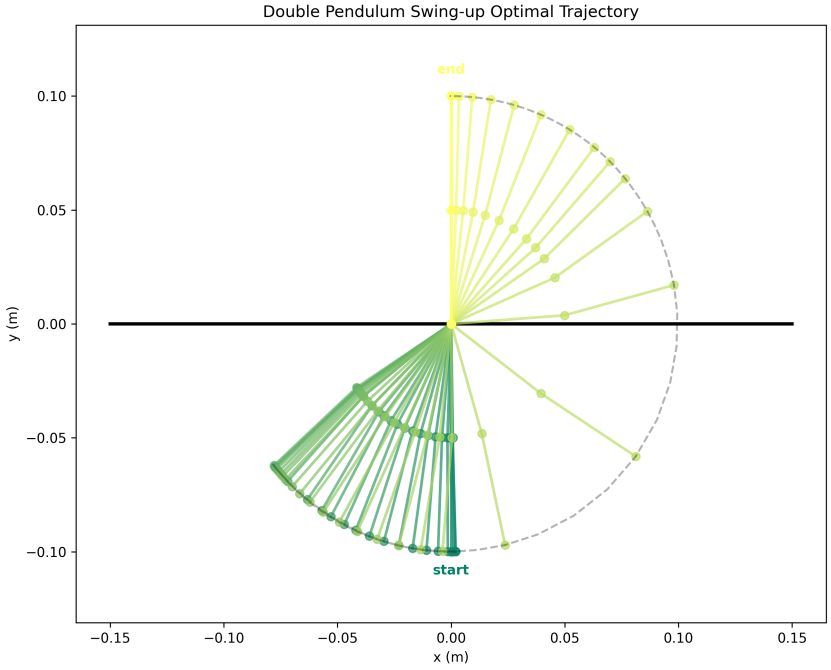
// 4. Nonlinear Programming (NLP) Solver
$\boldsymbol{z}^* \leftarrow$ minimize_ipopt(
  fun = cost,
  jac = ∇c,
  hess = $\boldsymbol{H}_{\mathbf{c}}$,
  $\boldsymbol{x}_0 = \boldsymbol{z}_{init}$,
  bounds = $(\boldsymbol{l}, \boldsymbol{u})$,
  constraints = {
    fun : constraints,
    jac : $\boldsymbol{J}_{\mathbf{f}}$
    hess : $\boldsymbol{H}_{\mathbf{f}}$ },
  options = {
    max_iter : 100,
    disp : 5,
    tol : e
  }
)

return $\boldsymbol{z}^*$

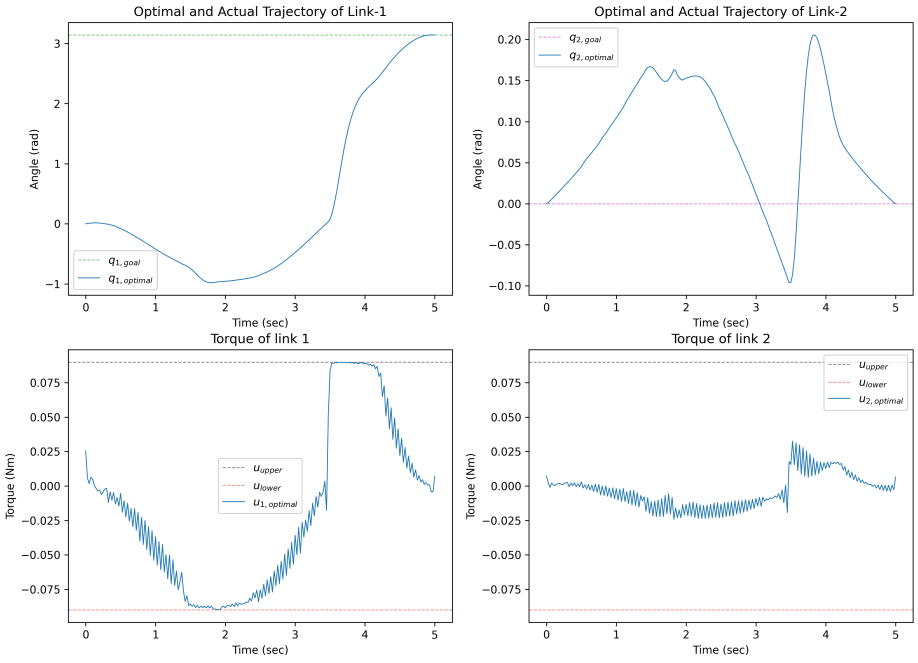
Θα εξετάσουμε 2 περιπτώσεις εδώ, μία για `torque_limit` $= 0.09$ και μία για `torque_limit` $= 0.15$ και θα συγκρίνουμε τη συμπεριφορά του συστήματος.

2.1 Περίπτωση #01: `torque_limit` $= 0.09$

Ο επιλυτής εδώ, για `tolerance` $= 10^{-3}$, βρίσκει λύση σε 22 iterations. Η βέλτιστη τροχιά που παράγει είναι:



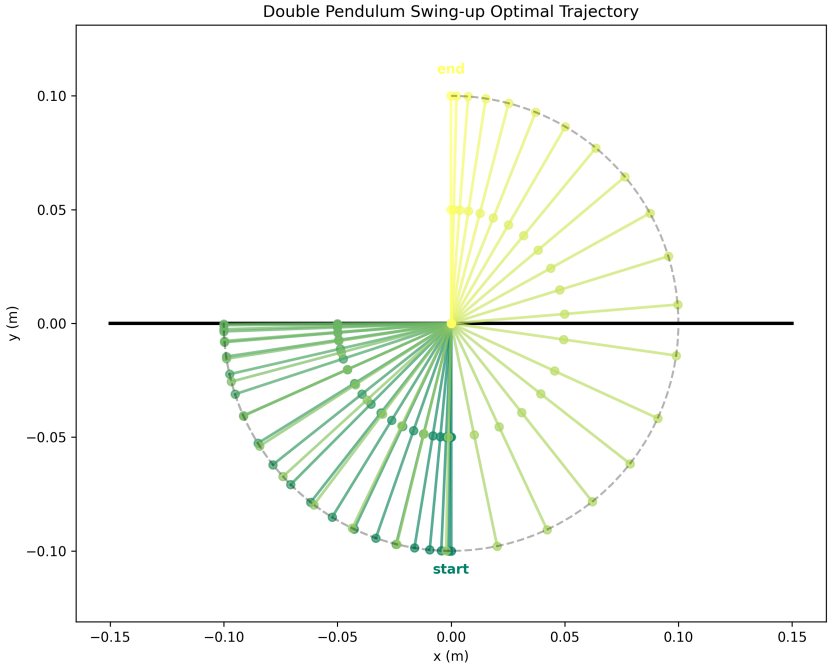
Εικόνα 2.1.1 Βέλτιστη τροχιά συστήματος για την Περίπτωση #01.



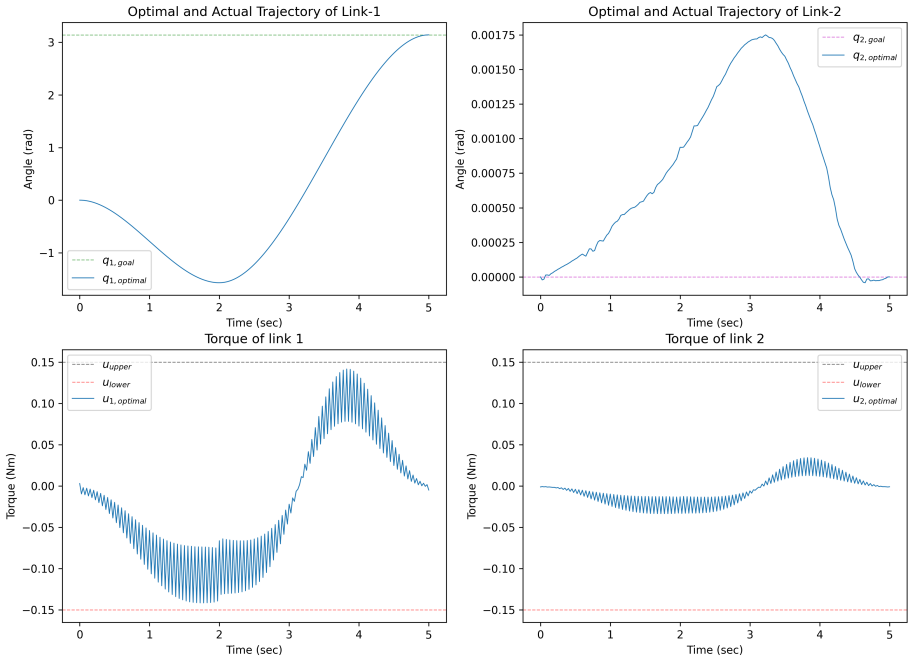
Εικόνα 2.1.2 Διαγράμματα της τροχιάς του ακρινού σημείου του double pendulum και της ροπής κάθε άρθρωσης.

2.2 Περίπτωση #02: `torque_limit` $= 0.15$

Σε αυτή την περίπτωση, για `tolerance` $= 10^{-4}$, βρίσκει λύση σε 12 iterations. Η βέλτιστη τροχιά που παράγει είναι:



Εικόνα 2.2.1 Βέλτιστη τροχιά συστήματος για την Περίπτωση #02.



Εικόνα 2.2.2 Διαγράμματα της τροχιάς του ακρινού σημείου του double pendulum και της ροπής κάθε άρθρωσης.

Κοιτάζοντας τα Διαγράμματα 2.1.1 και 2.2.1, με τις βέλτιστες τροχιές, παρατηρούμε ότι στην πρώτη περίπτωση όπου το όριο της ροπής είναι μικρό, το σύστημα ταλαντώνεται περισσότερο προκειμένου να φτάσει τον στόχο. Αντίθετα, όταν αυξάνεται το όριο οι κινητήρες μπορούν να δώσουν περισσότερη δύναμη, η τροχιά είναι πολύ πιο ομαλή, χωρίς πολλές ταλαντώσεις. Επιπλέον, στην 1η περίπτωση το tolerance είναι λιγότερο αυστηρό σε σχέση με τη 2η πρίπτωση, ώστε να μπορεί ο επιλυτής να δώσει σχετικά γρήγορα και εύκολα λύση.

3 Υλοποίηση του αλγορίθμου Time-Varying LQR (TVLQR)

Ο αλγόριθμος TVLQR αποτελεί επέκταση του κλασικού LQR για προβλήματα πεπερασμένου ορίζοντα (finite horizon). Η βασική του διαφοροποίηση έγκειται στο γεγονός ότι το σύστημα γραμμικοποιείται δυναμικά σε κάθε χρονικό βήμα k γύρω από την βέλτιστη τροχιά $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$, επιτρέποντας στον ελεγκτή να διαχειρίζεται μη γραμμικές δυναμικές. Η διαδικασία επίλυσης του προβλήματος γενικά περιλαμβάνει δύο στάδια: (α) το Backward Pass, όπου υπολογίζονται αναδρομικά οι πίνακες $\mathbf{P}_k$ και τα βέλτιστα κέρδη $\mathbf{K}_k$ βάσει των πινάκων κόστους $\mathbf{Q}$ και $\mathbf{R}$, και (β) το Forward Pass, όπου εφαρμόζεται ο νόμος ελέγχου ώστε να υπολογιστεί το νέο διάνυσμα κατάστασης. Σε πρώτη φάση, το μόνο που χρειάζεται να μας επιστρέφει ο ελεγκτής είναι τα κέρδη $\boldsymbol{K_s}$, άρα χρειαζόμαστε μόνο το στάδιο (α).

Algorithm 2: Time-Varying LQR Controller

Input: Initial state $\boldsymbol{x}_0$, Reference trajectory $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$, Cost matrices $\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{Q}_F$, Horizon S
Output: Nominal gains $\boldsymbol{K_s}$

// 1. Backward Pass: Riccati Recursion
$\boldsymbol{P}_{S-1} \leftarrow \boldsymbol{Q}_F$
for $k \leftarrow (S-2)$ **to** -1 **do**
    // Linearize system around $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$
    $\boldsymbol{A}_k, \boldsymbol{B}_k \leftarrow \text{linearize}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)$
    // Compute Optimal Gain and Value Function Matrix
    $\boldsymbol{K}_k \leftarrow (\boldsymbol{R} + \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1} \boldsymbol{A}_k$
    $\boldsymbol{P}_k \leftarrow \boldsymbol{Q} + \boldsymbol{A}_k^\top \boldsymbol{P}_{k+1} (\boldsymbol{A}_k - \boldsymbol{B}_k \boldsymbol{K}_k)$
return $\boldsymbol{K_s}$

4 Έλεγχος μέσω του TVLQR

Αντίστοιχα, το στάδιο (β) θα υλοποιηθεί ξεχωριστά, αξιοποιώντας τα κέρδη που προέκυψαν στο στάδιο (α), προκειμένου το σύστημά μας να κάνει swing-up, να παραμένει σε αυτή τη θέση ή να επιστρέφει ξανά εκεί αν υπάρξει εξωτερική διαταραχή.

Algorithm 3: Real-time TVLQR trajectory tracking with recovery

Input: Reference trajectory $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$, Nominal gains $\boldsymbol{K_s}$

// 2. Forward Pass
while *true* **do**
    $\boldsymbol{x} \leftarrow$ current state vector
    $\boldsymbol{u} \leftarrow \bar{\boldsymbol{u}}_k - \boldsymbol{K}_k(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)$
    Apply control $\boldsymbol{u}$
    if $k < S-2$ **then**
        $k \leftarrow k+1$
    else
        $\boldsymbol{q}_{\text{diff}} \leftarrow$ compute angular error
        $k \leftarrow \arg\min(\boldsymbol{q}_{\text{diff}})$
        $k \leftarrow \min(k, S-2)$

Ενδιαφέρον έχει το γεγονός ότι για την βελτιστοποίηση της τροχιάς του συστήματος, η δυναμική του συστήματος περιλάμβανε το μοντέλο της τριβής $\mathbf{F}$, ενώ ο ελεγκτής TVLQR όχι. Αν το λαμβάναμε υπόψη και

στον TVLQR, το σύστημα περνούσε αμέσως στην αστάθεια και αυτό είναι λογικό, γιατί το μοντέλο τριβής εμφανίζει μη γραμμικότητες οι οποίες δύσκολα μοντελοποιούνται σωστά. Έτσι, όταν εμείς το εμπιστευόμαστε πλήρως σε όλα τα στάδια του ελέγχου, οι ροπές που προκύπτουν off-line βασίζονται σε αυτό, ενώ στην πραγματικότητα δεν μπορεί να ισχύει.