Έλεγχος διπλού ολοκληρωτή υπό την επίδραση της βαρύτητας σε μη ομαλό έδαφος

Υπολογιστικό Θέμα 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών

Ονοματεπώνυμο: Παπαδάκης Μιχαήλ
Αριθμός Μητρώου: 02118026
Ακαδημαϊκό έτος: 4^o
Ημερομηνία: 25/7/22

Επιβλέπων Καθηγητής: Κ. Κυριακόπουλος

Περιεχόμενα

Κατάλογος Σχημάτων

1. Εισαγωγή

Αντικείμενο του υπολογιστικού θέματος ήταν ο έλεγχος ενός δισδιάστατου διπλού ελεγκτή, ο οποίος κινείται υπό την επήρεια ενός συντηρητικού πεδίου, και συγκεκριμένα της βαρύτητας, ώστε να φτάσει στο επιθυμητό σημείο του χώρου εργασίας. Εισαγωγικό στάδιο αποτέλεσε η δημιουργία απλών navigation functions με σκοπό την εξοικείωση με το motion planning πρόβλημα. Στη συνέχεια ακολούθησε η μοντελοποίηση της δυναμικής, η οποία έγινε με διάφορες μεθοδολογίες. Ύστερα, υιοθετήθηκε ένας ελεγκτής που χρησιμοποιεί τεχνητά αρμονικά πεδία (Artificial Harmonic Potential Fields) για να οδηγήσει το σύστημα στον προορισμό. Τελικό στάδιο αποτέλεσε η διερεύνηση του κόστους του ελεγκτή αυτού.

Στην παρούσα έκθεση, αφού παρουσιασθούν συνοπτικά η εισαγωγική δουλειά στα navigation functions και μερικές πληροφορίες σχετικά με την δημιουργία των αρμονικών πεδίων, παρουσιάζεται αναλυτικά η δυναμική του συστήματος. Ύστερα, παρουσιάζεται ο ελεγκτής, διάφορες προσομοιώσεις και παρατηρήσεις σχετικά με το κόστος του, σε έναν απλό χώρο. Στο παράρτημα περιέχονται όλοι οι κώδικες μαζί με σχετικές τεχνικές πληροφορίες.

2. Abstract

The scope of this computation project was the motion planning of a two-dimensional double intergrator which moves under the influence of a conservative field, and specifically the gravitational potential field. The first part of the project included the creation of navigation functions in simple circular spaces. The next step was the modelling of the relevant dynamics, using different techniques. Having modelled the external dynamics, a controller was designed, using Artificial Harmonic Potential Fields (AHPF) in order to drive the system to the desired destination. The final part of the project included the investigation of the controller cost. In this report, the work concerning the navigation functions will be summurized, and some details about the creation of AHPFs will be discussed as they are later used on the controller. Subsequently, the dynamics modelling will be presented in detail. Finally, the controller design will be outlined together with remarks about its cost. The programms that were used throught the project are presented with greater detail in the relevant anex.

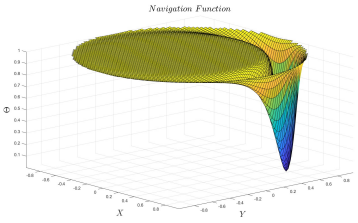
3. Μέθοδοι Πλοήγησης

Υπάρχουν διάφορες μεθοδολογίες για το motion planning των ρομποτικών συστημάτων. Υπάρχουν δυο κύριες κατηγορίες προσέγγισης του προβλήματος: οι δειγματοληπτικές μέθοδοι (πχ RRTs, MPC) και η δημιουργία κατάλληλων πεδίων πλοήγησης (navigation functions, Harmonic Potential Fields) [3]. Στο παρόν υπολογιστικό θέμα χρησιμοποιήθηκε η δεύτερη κατηγορία μεθόδων **η οποία είναι αναδραστική**. Στη συνέχεια θα παρουσιαστεί συνοπτικά η μέθοδος των navigation functions, και συγκεκριμένα ο έλεγχος ενός απλού ολοκληρωτή σε έναν κύκλο με εμπόδια, αφού αποτέλεσε το εισαγωγικό κομμάτι του υπολογιστικού θέματος. Ύστερα, θα παρουσιασθεί συνοπτικά η μέθοδος κατασκευής τεχνητών αρμονικών πεδίων αφού χρησιμοποιήθηκαν στον τελικό ελεγκτή.

3.1 Πλοήγηση σε 2 διαστάσεις με χρήση συναρτήσεων πλοήγησης

Το εισαγωγικό κομμάτι του υπολογιστικού θέματος ήταν η υλοποίηση ενός ελεγκτή που θα οδηγεί έναν απλό ολοκληρωτή στον προορισμό μέσα σε ένα κυκλικό workspace με σημειακά εμπόδια.

Το πρώτο βήμα σε αυτή την προσέγγιση, σύμφωνα με το paper [1], είναι να χρησιμοποιηθεί το navigation transformation ώστε να μετασχηματιστεί ένα workspace W σε ένα point-world P^n. Στον σημειακό κόσμο, τα εμπόδια του χώρου εργασίας (W) μετασχηματίζονται σε σημεία. Ωστόσο, αυτό το βήμα παραλήφθηκε, καθώς ο έλεγχος έπρεπε εξαρχής να γίνει σε κυκλικό δίσκο[1]. Ύστερα, πρέπει να βρεθεί το αρμονικό navigation function. Ωστόσο, αυτό είναι αδύνατο, από συνέπεια της αρχής του μεγίστου (και ελαχίστου). Έτσι, πρέπει να γίνει μετασχηματισμός σε έναν χώρο που μπορεί να δημιουργηθεί μια αρμονική συνάρτηση με ιδιότητες των συναρτήσεων πλοήγησης. Ο χώρος αυτός λέγεται harmonic domain. Αυτό γίνεται με τον εξής μετασχηματισμό (Γίνεται η θεώρηση πως ο harmonic domain είναι άπειρος, και δημιουργείται ένας μετασχηματισμός c που τον απεικονίζει πάνω σε μια σφαίρα διαμέτρου ρ. Ο αντίστροφος του c κάνει τον επιθυμητό μετασχηματισμό, από ένα sphere point world στον harmonic domain.):



Σχήμα 1: Navigation Function σε δίσκο με σημειακά εμπόδια

$$c^{-1}(x) = \frac{x}{\rho - ||x||} \tag{1}$$

Έτσι, κάθε μεταβλητή θέσης x από τον $2D$ point-world μετασχηματίζεται μέσω του μετασχηματισμού στην h. Το ίδιο πρέπει να γίνει και για κάθε σημειακό εμπόδιο αλλά και τον προορισμό ($p_i \to h_i$). Πρέπει να τονιστεί πως ο μετασχηματισμός αυτός είναι διαφορομορφισμός, και συνεπώς τόσο αυτός όσο και ο αντίστροφος ($c(p)$) είναι διαφορίσιμοι. Στην συνέχεια, στον harmonic domain, χρησιμοποιούνται πηγές στα εμπόδια ώστε να απωθήσουν το ελεγχόμενο σύστημα και καταβόθρα στον προορισμό, που δίνονται από τον τύπο:

$$\varphi_i = a_i \ln\left(||h - h_i||\right) \tag{2}$$

Ύστερα, επιλέγονται τα κέρδη των πηγών/καταβοθρών δυναμικού. Πρέπει να ισχύει η συνθήκη:

$$\lim_{||h|| \to \infty} \phi(h) = +\infty$$

Η οποία ικανοποιείται όταν επιλεχθούν τα παρακάτω κέρδη (ικανή συνθήκη):

$$\alpha_d = K > M + 1$$

$$a_i = -1$$

Ύστερα, γίνονται δύο ακόμα μετασχηματισμοί:

$$\sigma(x) \triangleq \frac{e^x}{1 + e^x} \tag{3}$$

[1]Η εύρεση του κατάλληλου navigation transformation είναι εν γένει δύσκολη, για αυτό και προτιμάται η προσέγγιση των τεχνητών αρμονικών πεδίων.

Όπου προβάλει τον αρμονικό χώρο σε έναν δίσκο με ακτίνα 1 (μάλιστα, προβάλει και το $\pm\infty$ στο $0, 1$ αντίστοιχα). Και τέλος:

$$\sigma_d \triangleq x^{2/k} \tag{4}$$

Έτσι, με όλους αυτούς τους μετασχηματισμούς (σχέσεις 1,2,3,4,), μπορεί να δημιουργηθεί το navigation function το οποίο δίνεται από την σχέση 5.

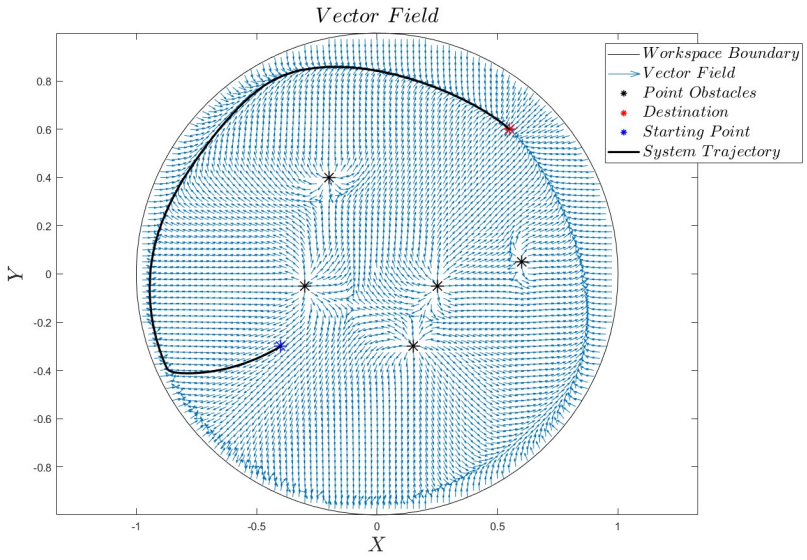
$$\Theta(\cdot) = [\sigma_d \circ \sigma \circ \varphi \circ c^{-1}](\cdot) \tag{5}$$

Η συνάρτηση αυτή δέχεται σημεία από τον $2D$ point-world (δηλαδή μετά το navigation tranformation) και δίνει το δυναμικό στο σημείο αυτό. Για έλεγχο ενός συστήματος μονού ολοκληρωτή, χρησιμοποιούνται οι σχέσεις:

$$\dot{x} = u \tag{6}$$

$$\dot{x} = u = -\nabla\Theta = -[\frac{\partial\Theta}{\partial x_1}, \frac{\partial\Theta}{\partial x_2}] \tag{7}$$

Στο σχήμα 1 παρουσιάζεται το navigation function που προκύπτει από την παραπάνω ανάλυση και στο σχήμα 2 φαίνεται η τροχιά ενός απλού ολοκληρωτή που ελέγχεται βάσει του παραπάνω navigation function.



Σχήμα 2: Έλεγχος απλού ολοκληρωτή με χρήση navigation function

3.2 Πλοήγηση σε 2 διαστάσεις με χρήση αρμονικών πεδίων

Στη μέθοδο των αρμονικών πεδίων δημιουργείται ένα πεδίο Φ στο W, η τιμή του οποίου δίνεται σε κάθε θέση από την συνεισφορά όλων των πηγών του χώρου. Αν οι πηγές του πεδίου θεωρηθούν σαν συναρτήσεις βάσης u_i (στην περίπτωση σημειακών πηγών[2] $u_i = ln(||p - p_i||)$), τότε το πεδίο σε κάθε θέση παράγεται από τον γραμμικό συνδυασμό των u_i (αρμονικών όρων).

$$\Phi(p; p_c) = w^T v(p; p_c) \tag{8}$$

με $p_c \triangleq [p_0^T, p_1^T, ..., p_K^T]$ τα κέντρα των πηγών και $w \triangleq [w_0, w_1, .., w_k]$ τα σχετικά βάρη, και έτσι $v(p; pc) = [v_0(p, p_0), v_1(p, p_1), ..., v_K(p, p_K)]^T$.

Απαίτηση ενός τέτοιου πεδίου είναι να είναι ασφαλές, και στα όρια του workspace η επιθυμητή ταχύτητα να δείχνει στο εσωτερικό του[3]. Σύμφωνα με το [3], υπάρχει ένας ακέραιος N_0 τέτοιος ώστε, αν το πεδίο δείχνει στο εσωτερικό σε $N > N_0$ ισαπέχοντα σημεία p_j στα όρια του χώρου εργασίας ($p_j \in \partial W,\ \forall j = 1, 2, ..N$) , δηλαδή αν ισχύει [4]:

$$n^T(p_j)u(p_J) \geq 0,\ \forall j = 1, 2, ..., N > N_0 \tag{9}$$

τότε, το πεδίο ταχυτήτων δείχνει στα εσωτερικά του workspace σε κάθε σημείο του ορίου ∂W. Επιπλέον, η ταχύτητα σε κάθε σημείο δίνεται από τον τύπο:

$$u(p) = -\underbrace{||p - p_o||^2}_{T_0} \nabla\Phi = -||p - p_o||^2 \nabla v(p; p_c)w \tag{10}$$

όπου ο όρος T_0 αφερεί την μοναδικότητα (singularity) από τον προορισμό[5]. Από τις σχέσεις 9 και 10 προκύπτει ένα σετ N περιορισμών που πρέπει να ικανοποιηθούν. Επιπλέον, πρέπει ο προορισμός να είναι ελκυστικός, άρα $w_0 > 0$. Συνολικά οι $N + 1$ περιορισμοί γράφονται ως:

$$A_{N+1}^T w < \mathbf{0}_{(N+1)x1} \tag{11}$$

Για να βρεθεί ένα σετ w που ικανοποιεί τους παραπάνω περιορισμούς λύνεται ένα τετραγωνικό πρόβλημα βελτιστοποίησης με περιορισμούς:

$$min||w||^2$$

$$s.t.\ A_{N+1}^T w < \mathbf{0}_{(N+1)x1}$$

Στην πράξη ο αριθμός των περιορισμών αποτελεί input στο πρόγραμμα των αρμονικών πεδίων. Έχοντας τα βάρη, έχει βρεθεί το αρμονικό πεδίο ταχυτήτων που χρησιμοποιείται στον τελικό ελεγκτή.

---

[2] Το πρόγραμμα των αρμονικών πεδίων χρησιμοποιεί γραμμικές αρμονικές πηγές, αφού πλαισιώνει τον χώρο με panel.

[3] Με τον κατάλληλο ελεγκτή, όπως αυτός που χρησιμοποιήθηκε τελικά (σχέση 60), η απαίτηση για την κατεύθυνση του πεδίου είναι επαρκής για να εγγυηθεί την ασφάλεια του ελεκγτή.

[4] Γίνεται η παραδοχή πως το κάθετο διάνυσμα δείχνει στο εσωτερικό του χωρίου.

[5] Σε σημειακή πηγή ισχύει:

$$\frac{\partial v_d}{\partial p_d} = \frac{p - p_d}{||p - p_d||^2}$$

Οπότε πολλαπλασιάζοντας με $||p - p_o||^2$ αφαιρείται ο όρος από τον παρανομαστή ο οποίος μηδενίζεται όταν το σύστημα φτάσει στον προορισμό.

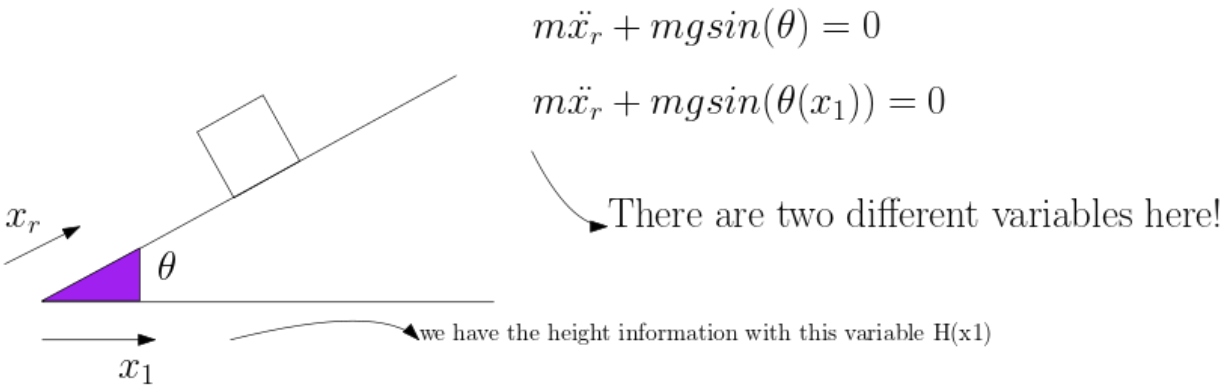
4. Δυναμική

4.1 Μονοδιάστατο Πρόβλημα με Σταθερή Κλίση

Αρχικά παρατίθενται οι εξισώσεις για το μονοδιάστατο πρόβλημα, με σταθερή κλίση :

$$m\ddot{x_r} = -mgsin(\theta) \tag{12}$$

Παρατηρείται πως η εξίσωση γράφεται ως προς την καμπυλόγραμμη συντεταγμένη x_r. Συνεπώς, η μετατόπιση Δx_r είναι διαφορετική ως προς την μετατόπιση στον οριζόντιο άξονα Δx_1 . Έτσι, πρέπει από την Δx_r να υπολογίζεται η Δx_1 ώστε να βρίσκεται σωστά το σημείο ως προς τον άξονα x_1 και να λαμβάνεται η πληροφορία του ύψους. Αυτό παρατηρείται και από το σχήμα 3:



Σχήμα 3: Μονοδιάστατη περίπτωση με σταθερή κλίση

Ισχύει:

$$x_1 = x_r cos(\theta) \tag{13}$$

Άρα:

$$m\frac{1}{cos(\theta)}\ddot{x_1} + mgsin(\theta) = 0 \tag{14}$$

$$m\ddot{x_1} + mgsin(\theta)cos(\theta) = 0 \tag{15}$$

Η μέθοδο lagrange καταλήγει σε παρόμοια αποτελέσματα. Έχουμε:

$$T = \frac{1}{2}m\dot{x_r}^2 = \frac{1}{2}m\frac{\dot{x_1}^2}{cos(\theta)^2} = T(\dot{x_1}) \tag{16}$$

$$V = mg(tan(\theta)x_1) + Vo = V(x_1) \tag{17}$$

Η μεταβλητή Lagrange είναι:

$$L = T - V \tag{18}$$

Η εξίσωση Lagrange γράφεται:

$$\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{x_1}}) - \frac{\partial L}{\partial x_1} = 0 \tag{19}$$

Οπότε προκύπτει:

$$\frac{\partial}{\partial t}(\frac{\partial T(\dot{x_1})}{\partial \dot{x_1}}) + \frac{\partial V(x_1)}{\partial x_1} = 0$$

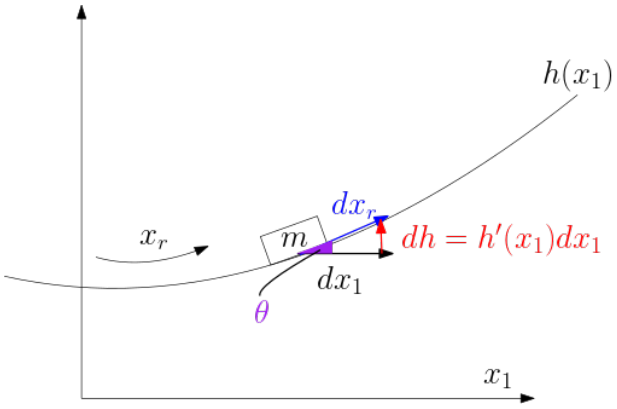
$$m\frac{\ddot{x_1}}{cos^2(\theta)} + mgtan(\theta) = 0 \tag{20}$$

Οπότε προκύπτει ξανά η σχέση 15.

4.2 Μονοδιάστατο Πρόβλημα με Μεταβλητή Κλίση

4.2.1 Αλλαγή Μεταβλητών

Ύστερα, έστω ότι έχουμε μεταβλητή κλίση, όπως φαίνεται στο σχήμα 4:



Σχήμα 4: Μονοδιάστατη περίπτωση με μεταβλητή κλίση

Αρχικά μπορούν να υπολογιστούν μερικά χρήσιμα μεγέθη:

$$sin(\theta) = \frac{dh}{dx_r} = \frac{h'(x_1)dx_1}{\sqrt{dx_1^2 + h'(x_1)^2 dx_1^2}} = \frac{h'(x_1)}{\sqrt{1 + h'(x_1)^2}} \tag{21}$$

$$cos(\theta(x_1)) = \frac{1}{\sqrt{1 + h'(x_1)^2}} \tag{22}$$

Η δυναμική δίνεται από την παρακάτω σχέση. Σε αντιστοιχία με την περίπτωση της σταθερής κλίσης, το x_r είναι η καμπυλόγραμμη συντεταγμένη.

$$m\ddot{x_r} = -mgsin(\theta(x_1)) \tag{23}$$

Κάνοντας αλλαγή συντεταγμένων προκύπτει:

$$dx_1 = cos(\theta(x_1))dx_r \xrightarrow{22}$$

$$dx_r = \sqrt{1 + h'(x_1)^2}dx_1 \tag{24}$$

Άρα

$$\frac{dx_r}{dt} = \sqrt{1 + h'(x_1)^2}\frac{dx_1}{dt}$$

$$\frac{d^2x_r}{dt^2} = \sqrt{1 + h'(x_1)^2}\frac{d^2x_1}{dt^2} + \frac{dx_1}{dt}\frac{2h'(x_1)h''(x_1)}{2\sqrt{1 + h'(x_1)^2}}\frac{dx_1}{dt}$$

$$\frac{d^2x_r}{dt^2} = \sqrt{1 + h'(x_1)^2}\frac{d^2x_1}{dt^2} + \frac{h'(x_1)h''(x_1)}{\sqrt{1 + h'(x_1)^2}}(\frac{dx_1}{dt})^2$$

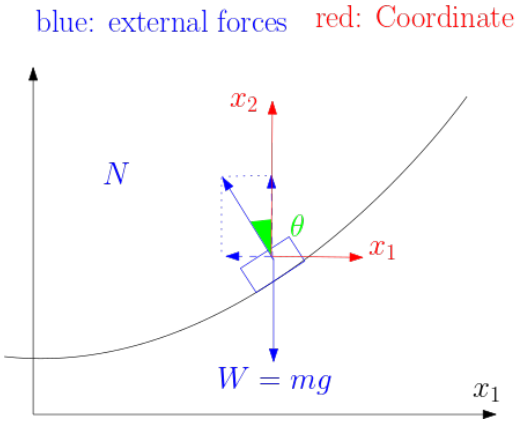
Και τελικά η 23 γίνεται:

$$m\sqrt{1 + h'(x_1)^2}\ddot{x_1} + m\frac{h'(x_1)h''(x_1)}{\sqrt{1 + h'(x_1)^2}}\dot{x}_1^2 + mg\frac{h'(x_1)}{\sqrt{1 + h'(x_1)^2}} = 0$$

$$\ddot{x_1} + \frac{h'(x_1)h''(x_1)}{1 + h'(x_1)^2}\dot{x}_1^2 + g\frac{h'(x_1)}{1 + h'(x_1)^2} = 0 \tag{25}$$

4.2.2 Μέθοδος Newton

Η μέθοδος newton οδηγεί στα ίδια αποτελέσματα:



Σχήμα 5: Μονοδιάστατη περίπτωση με μεταβλητή κλίση - Μέθοδος Newton

Λαμβάνοντας τις εξισώσεις ισορροπίας σε κάθε κατεύθυνση:

$$X : m\ddot{x}_1 = -Nsin(\theta) \tag{26}$$

$$Y : m\ddot{y}_1 = Ncos(\theta) - mg \tag{27}$$

Κάνοντας τις κατάλληλες μετατροπές:

$$dy_1 = h'(x_1)dx_1 \rightarrow \dot{y}_1 = h'(x_1)\dot{x}_1 \rightarrow \ddot{y}_1 = h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2 \tag{28}$$

$$\xrightarrow[28]{27} N = \frac{m\ddot{y}_1 + mg}{cos(\theta)} = \frac{m(h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2) + mg}{cos(\theta)} \tag{29}$$

Χρησιμοποιώντας τις σχέσεις 26, 29:

$$\ddot{x}_1 = -\frac{sin(\theta)}{cos(\theta)} \cdot (h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2 + g) \xrightarrow[21]{22}$$

$$\ddot{x}_1 = -h'(x_1) \cdot (h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2 + g)$$

$$\ddot{x}_1(1 + h'(x_1)^2) = -h'(x_1)h''(x_1)\dot{x}_1^2 - h'(x_1)g$$

Και τελικά καταλήγουμε στην σχέση 25:

$$\ddot{x}_1 + \frac{h'(x_1)h''(x_1)}{1 + h'(x_1)^2}\dot{x}_1^2 + g\frac{h'(x_1)}{1 + h'(x_1)^2} = 0$$

Η μέθοδος Newton μας δίνει επιπλέον την συνθήκη αποκόλλησης:

$$N \geq 0 \rightarrow \frac{(h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2) + g}{cos(\theta)} \geq 0 \tag{30}$$

Και επειδή στο πρόβλημα $\theta \in (-\pi/2, \pi/2)$, η συνθήκη απλοποιείται περαιτέρω:

$$h'(x_1)\ddot{x}_1 + h''(x_1)\dot{x}_1^2 + g \geq 0 \tag{31}$$

4.3 Δισδιάστατο Πρόβλημα με μεταβλητή κλίση

4.3.1 Μέθοδος Lagrange - Πρώτη μέθοδος για Κινητική και Δυναμική ενέργεια

Έχοντας μελετήσει τις 2 απλοποιημένες περιπτώσεις, ακολουθεί η μελέτη της γενικότερης περίπτωσης. Θεωρώντας τη θέση του σωματιδίου στον χώρο να περιγράφεται από το διάνυσμα:

$$\vec{R} = [x_1, x_2, h(x_1, x_2)]^T \tag{32}$$

Αρχικά ορίζουμε το διάνυσμα $x = [x_1, x_2]^T$. Η ταχύτητα δίνεται από τον τύπο:

$$\dot{\vec{R}} = \vec{v} = \frac{\partial R}{\partial x} \cdot \frac{\partial x}{\partial t} \tag{33}$$

$$\vec{v} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} \end{bmatrix} \cdot \dot{x} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \nabla h^T \dot{x} \end{bmatrix} \tag{34}$$

Η κινητική και δυναμική ενέργεια γράφονται:

$$T = \frac{1}{2} m ||v||^2 = \frac{1}{2} m (||\dot{x}||^2 + (\nabla h^T \dot{x})^2) = T(x(t), \dot{x}(t)) \tag{35}$$

$$V = \int_{x_o}^{x} mg \nabla h(s) \cdot ds = mg(h(x) - h(x_0)) = V(x) \tag{36}$$

4.3.2 Μέθοδος Lagrange - Δεύτερη μέθοδος για Κινητική και Δυναμική ενέργεια

Η κινητική ενέργεια γράφεται ως:

$$T = \frac{1}{2} m ||x_r||^2 \tag{37}$$

Αυτή η κινητική ενέργεια έχει γραφτεί ως προς καμπυλόγραμμες συντεταγμένες, **με χρήση του διανύσματος** $x_r = [x_{r1}, x_{r2}]^T$. Ισχύει (για $i = 1, 2$), όπως αποδείχθηκε στην $1D$ περίπτωση:

$$dx_{r,i} = \sqrt{1 + \frac{\partial h}{\partial x_i}^2} \, dx_i \tag{38}$$

Έτσι, ο μετασχηματισμός που πρέπει να γίνει είναι:

$$x_r = Jx \tag{39}$$

όπου J [6]:

$$J = \begin{bmatrix} \frac{\partial x_{r1}}{\partial x_1} & \frac{\partial x_{r1}}{\partial x_2} \\ \frac{\partial x_{r2}}{\partial x_1} & \frac{\partial x_{r2}}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \sqrt{1 + \frac{\partial h}{\partial x_1}^2} & 0 \\ 0 & \sqrt{1 + \frac{\partial h}{\partial x_2}^2} \end{bmatrix} \tag{40}$$

Συνεπώς:

$$T = \frac{1}{2} x_r^T [m] x_r = \frac{1}{2} (Jx)^T [m] (Jx) = \frac{1}{2} m (Jx)^T (Jx) = \frac{1}{2} m ||(Jx)||^2$$

$$T = \frac{1}{2} m x^T J^2 x \tag{41}$$

Επειδή ο J είναι διαγώνιος, προκύπτει :

$$J^2 = \begin{bmatrix} 1 + \frac{\partial h}{\partial x_1}^2 & 0 \\ 0 & 1 + \frac{\partial h}{\partial x_2}^2 \end{bmatrix}$$

Έτσι, οι εκφράσεις 35 και 41 είναι ισοδύναμες.

[6] Επειδή $detJ \neq 0$ μπορούμε να πάμε από την μια βάση στην άλλη με αντιστοιχία 1-1

4.3.3 Μέθοδος Lagrange - Παραγώγιση Lagrangian

Αρχικά, οι παράγωγοι ακολουθούν τους παρακάτω κανόνες:

$$\frac{\partial}{\partial x}\left(x^T x\right) = 2x,\ \frac{\partial}{\partial a}\left(a^T b\right) = b,\ \frac{\partial}{\partial b}\left(a^T b\right) = a$$

Οπότε:

$$\frac{\partial T}{\partial \dot{x}} = m(\dot{x} + (\nabla h^T \dot{x})\nabla h)$$

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{x}}\right) = m(\ddot{x} + (\nabla h^T \dot{x}) \underbrace{\frac{\partial}{\partial t}(\nabla h)}_{T_1} + \underbrace{\frac{\partial}{\partial t}\left(\nabla h^T \dot{x}\right)}_{T_2} \nabla h \tag{42}$$

Παραγωγίζοντας αναλυτικότερα τους όρους T_1, T_2:

$$T_1:\ \frac{\partial}{\partial t}(\nabla h) = \frac{\partial}{\partial x}(\nabla h)\frac{\partial x}{\partial t} = J_{\nabla h}\dot{x} = H_h \dot{x}$$

Όπου $J_{\nabla h}$ η ιακωβιανή του ∇h. Ακόμα:

$$T_2:\ \frac{\partial}{\partial t}\left(\nabla h^T \dot{x}\right) = \nabla h^T \ddot{x} + \dot{x}^T \frac{\partial}{\partial t}(\nabla h) = \nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x} \tag{43}$$

Έτσι, αντικαθιστώντας τους όρους T_1, T_2 στην σχέση 42:

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial \dot{x}}\right) = m(\ddot{x} + (\nabla h^T \dot{x}) H_h \dot{x} + \left(\nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}\right)\nabla h) \tag{44}$$

Υπολογίζοντας τους υπόλοιπους όρους της Lagrangian:

$$\frac{\partial (T-V)}{\partial x} = \frac{1}{2} m \frac{\partial}{\partial x}(||\dot{x}||^2 + (\nabla h^T \dot{x})^2)) - mg\frac{\partial h}{\partial x}$$

$$\frac{\partial (T-V)}{\partial x} = m(\nabla h^T \dot{x}) \underbrace{\frac{\partial}{\partial x}(\nabla h^T \dot{x})}_{=T_2} - mg\nabla h = m(\nabla h^T \dot{x}) H_h \dot{x} - mg\nabla h \tag{45}$$

Άρα συνολικά, με χρήση των 45 και 44:

$$m(\ddot{x} + \left(\nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}\right)\nabla h) + mg\nabla h = 0 \tag{46}$$

Η σχέση 46 είναι η τελική εξίσωση κίνησης. Μπορεί ωστόσο να απλοποιηθεί. Αρχικά, χρησιμοποιείται η ταυτότητα:

$$\nabla h(\nabla h^T \ddot{x}) = (\nabla h \otimes \nabla h)\ddot{x} \tag{47}$$

όπου $\nabla h \otimes \nabla h$ είναι το outer product των ∇h. Έτσι, η 46 γράφεται:

$$(I_2 + \nabla h \otimes \nabla h)\ddot{x} + \left(\dot{x}^T H_h \dot{x}\right)\nabla h + g\nabla h = 0$$

Η έκφραση αυτή απλοποιείται περαιτέρω με χρήση του θεωρήματος Sherman - Morrison [2] σύμφωνα με το οποίο :

Sherman - Morrison theorem

Αν ο A είναι αντιστρέψιμος πίνακας, και τα u, v είναι διανύσματα στήλης, τότε αν ισχύει:

$$1 + v^T A^{-1} u \neq 0$$

Η έκφραση $(A + uv^T)$ αντιστρέφεται και ο αντίστροφος δίνεται από τον παρακάτω τύπο:

$$\left(A + uv^{\mathsf{T}}\right)^{-1} = A^{-1} - \frac{A^{-1}uv^{\mathsf{T}}A^{-1}}{1 + v^{\mathsf{T}}A^{-1}u}. \tag{48}$$

Οπότε, ο αντίστροφος του $(I_2 + \nabla h \otimes \nabla h)$ προκύπτει (Ισχύει $1 + ||\nabla h||^2 > 0, \forall x \in W$):

$$(I_2 + \nabla h \otimes \nabla h)^{-1} = I_2 - \frac{\nabla h \otimes \nabla h}{1 + ||\nabla h||^2} \tag{49}$$

Έτσι, κάνοντας μερικές πράξεις:

$$\ddot{x} + \big((\dot{x}^T H_h \dot{x}) + g\big) \underbrace{(I_2 - \frac{\nabla h \otimes \nabla h}{1 + ||\nabla h||^2})\nabla h}_{T_3} = 0$$

Και κάνοντας τις πράξεις στον όρο T_3:

$$T_3 : \ (I_2 - \frac{\nabla h \otimes \nabla h}{1 + ||\nabla h||^2})\nabla h = \frac{(1 + ||\nabla h||^2)\nabla h - (\nabla h \otimes \nabla h)\nabla h}{1 + ||\nabla h||^2} =^{47}\longrightarrow$$

$$T_3 : \ \frac{(1 + ||\nabla h||^2)\nabla h - (\nabla h^T \nabla h)\nabla h}{1 + ||\nabla h||^2} = \frac{(1 + ||\nabla h||^2)\nabla h - ||\nabla h||^2 \nabla h}{1 + ||\nabla h||^2} = \frac{\nabla h}{1 + ||\nabla h||^2}$$

Έτσι, προκύπτει η παρακάτω εξίσωση κίνησης, η οποία είναι γραμμένη σε βολική μορφή:

$$\ddot{x} + \frac{\dot{x}^T H_h \dot{x}}{1 + ||\nabla h||^2}\nabla h + \frac{g}{1 + ||\nabla h||^2}\nabla h = 0 \tag{50}$$

Μερικά σχόλια για τον τύπο: Ο όρος:

$$\frac{\dot{x}^T H_h \dot{x}}{1 + ||\nabla h||^2}\nabla h$$

είναι ο όρος που λαμβάνει υπόψη την καμπυλότητα του χωρίου. Ενώ ο όρος:

$$\frac{g}{1 + ||\nabla h||^2}\nabla h$$

είναι η επίδραση της βαρύτητας στο σύστημα.

4.3.4 Μέθοδος Newton

Λαμβάνοντας ισορροπία δυνάμεων:

$$m\vec{a} = -m\vec{g} + N\vec{n} \tag{51}$$

όπου το διάνυσμα $\vec{n}$ είναι το μοναδιαίο κάθετο διάνυσμα στην επιφάνεια, και συνεπώς η δύναμη $N\vec{n}$ είναι η κάθετη αντίδραση από το έδαφος. Σε μια παραμετροποιημένη επιφάνεια $S(u, v)$ (στην παρούσα περίπτωσή $u = x_1, v = x_2$), το κάθετο διάνυσμα δίνεται από τον τύπο:

$$\vec{n} = \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \tag{52}$$

οπότε προκύπτει ίσο με:

$$\vec{n} = \big[-\tfrac{\partial h}{\partial x_1}, \quad -\tfrac{\partial h}{\partial x_2}, \quad 1\big]^T$$

Πολλαπλασιάζοντας την παραπάνω ποσότητα με τον όρο $\frac{1}{\sqrt{1+||\nabla h||^2}}$ εξασφαλίζεται ότι είναι το μοναδιαίο κάθετο διάνυσμα. Έτσι, προκύπτει η παρακάτω εξίσωση:

$$m\vec{a} = m\begin{bmatrix}\ddot{x}_1\\ \ddot{x}_2\\ \ddot{x}_3\end{bmatrix} = \begin{bmatrix}0\\0\\-mg\end{bmatrix} + N \cdot \frac{1}{\sqrt{1 + ||\nabla h||^2}}\begin{bmatrix}-\frac{\partial h}{\partial x_1}\\ -\frac{\partial h}{\partial x_2}\\ 1\end{bmatrix} \tag{53}$$

Χρησιμοποιώντας την σχέση 43, προκύπτει:

$$\vec{v} = \begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \nabla h^T \dot{x}\end{bmatrix} \rightarrow \vec{a} = \begin{bmatrix}\ddot{x_1}\\ \ddot{x_2}\\ \nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}\end{bmatrix} \tag{54}$$

Έτσι, χρησιμοποιώντας την τρίτη σχέση από την εξίσωση 54:

$$m(\nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}) = -mg + N\frac{1}{\sqrt{1+||\nabla h||^2}} \rightarrow$$

$$N\frac{1}{\sqrt{1+||\nabla h||^2}} = m(\nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}) + mg \tag{55}$$

Και αντικαθιστώντας την σχέση 55 στις 2 πρώτες της 54, λαμβάνεται η σχέση 46. Οπότε, προκύπτει το ίδιο αποτέλεσμα. Ωστόσο, η μέθοδος του newton δίνει και την συνθήκη αποκόλλησης:

$$N \geq 0 = (\nabla h^T \ddot{x} + \dot{x}^T H_h \dot{x}) + g \geq 0 \tag{56}$$

4.4 Προσομοίωση Δυναμικού Συστήματος

Η προσομοίωση γίνεται στην matlab. Θεωρώντας ως διάνυσμα κατάστασης το παρακάτω:

$$y = [y_1, y_2, y_3, y_4]^T = [x_1, \dot{x}_1, x_2, \dot{x}_2]^T \tag{57}$$

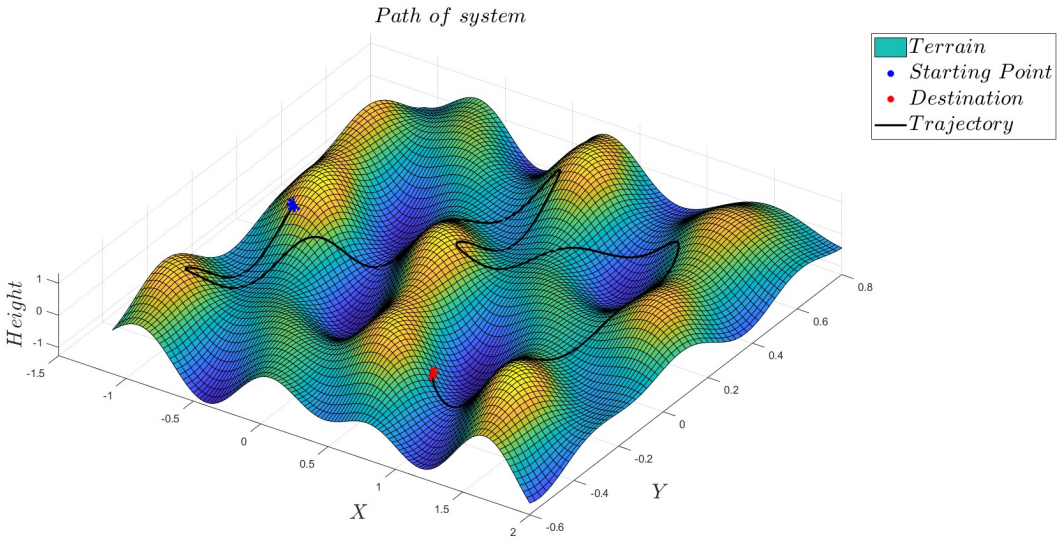
Η εξίσωση κίνησης γίνεται:

$$\dot{y} = \frac{d}{dt}\begin{Bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{Bmatrix} = \begin{Bmatrix} y_2 \\ f_1(y_1, y_3) \\ y_4 \\ f_2(y_1, y_3) \end{Bmatrix} \tag{58}$$

και η εξωτερική δύναμη f είναι ένα διάνυσμα $2x1$:

$$f = -\frac{\dot{x}^T H_h \dot{x}}{1+||\nabla h||^2}\nabla h - \frac{g}{1+||\nabla h||^2}\nabla h \tag{59}$$

Παρακάτω παρουσιάζεται αποτέλεσμα της προσομοίωσης:



Σχήμα 6: Προσομοίωση Δυναμικού Συστήματος

5. Έλεγχος Δυναμικού Συστήματος

5.1 Σχεδιασμός Ελεγκτή

Για να ελεγχθεί το σύστημα, υιοθετείται έλεγχος της μορφής:

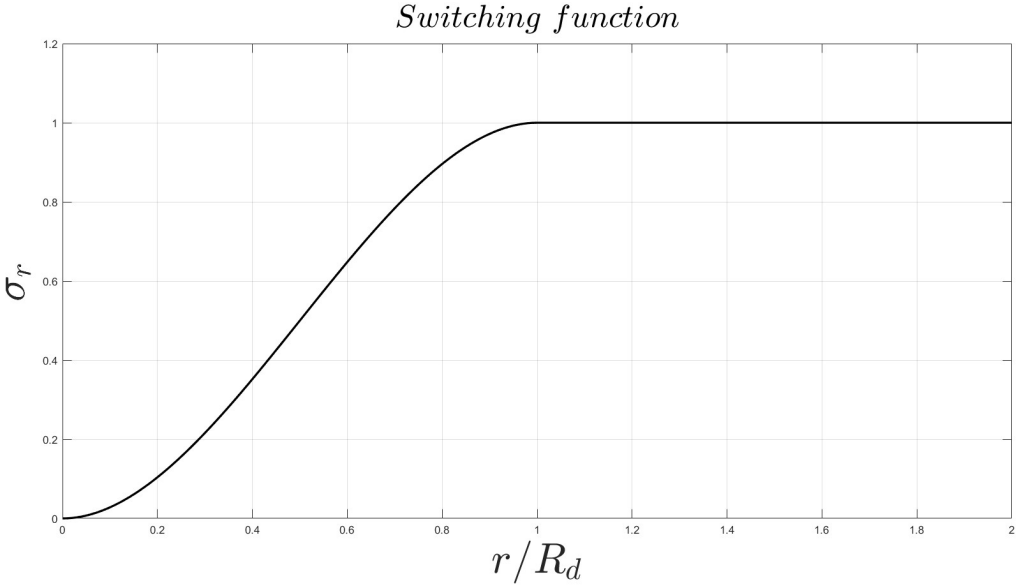
$$u = \frac{1}{\sigma_r}\nabla\Phi - k(\dot{x} - \dot{x}_d) + F_{comp} \tag{60}$$

όπου:

- Το Φ είναι το artificial harmonic potential field που προκύπτει από το πρόγραμμα των αρμονικών πεδίων.
- Το σ_r είναι ένα bumping function[7], που δίνεται από τον τύπο:

$$\sigma_r = \left\{ \begin{array}{ll} 1 - exp\left(-(\frac{r}{r-R_d})^2\right), & \text{for } r < R_d \\ 1, & \text{for } r \geq R_d \end{array} \right\} \tag{61}$$

όπου r η απόσταση από τα όρια του χωρίου, που δίνεται από το πρόγραμμα *distance.m,* και R_d είναι μια απόσταση-παράμετρος. Η συνάρτηση $\sigma_r(r)$ έχει τη μορφή που φαίνεται στο σχήμα 7.



Σχήμα 7: Bumping function

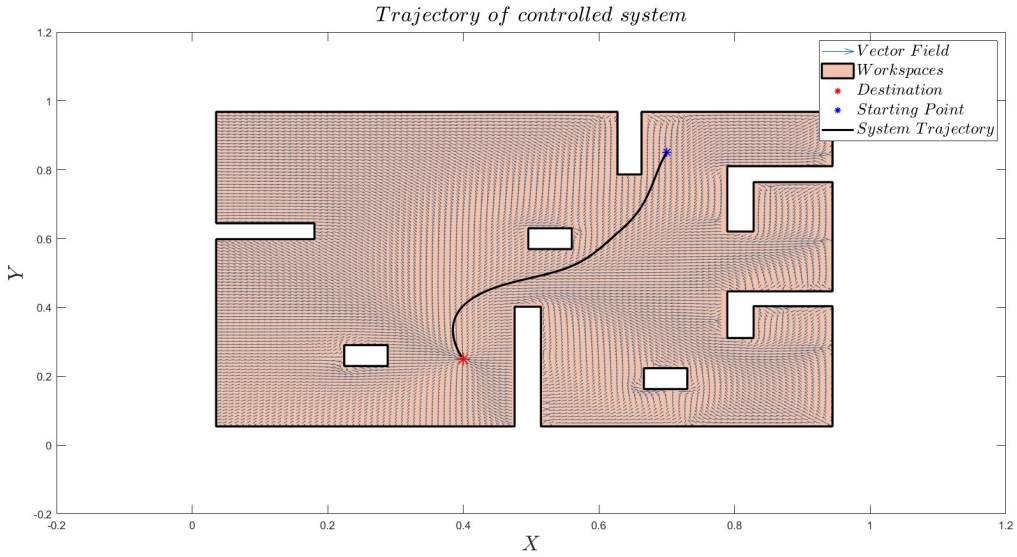
Παρατηρείται πως για $r > R_d$ ο όρος $1/\sigma_r$ δεν επηρεάζει τον έλεγχο, ενώ όταν $r \to 0$, δηλαδή όταν το σύστημα τείνει στα όρια του χωρίου, η δύναμη που ασκείται από τον ελεγκτή προς το εσωτερικό του χώρου εργασίας απειρίζεται ($\frac{1}{\sigma_r} \to \infty$). **Έτσι, εξασφαλίζεται η ασφάλεια του ελεγκτή.**

- Ο όρος $-k(\dot{x} - \dot{x}_d)$ έχει διπλή λειτουργία. Αρχικά, προσθέτει απόσβεση στο σύστημα, συγκεκριμένα το τμήμα $-k\dot{x}$. Επιπλέον, το διάνυσμα $\dot{x}_{correction} = \dot{x} - \dot{x}_d$ αποτελεί το διάνυσμα διόρθωσης προς την επιθυμητή ταχύτητα.
- Τέλος, ο όρος F_{comp} αντισταθμίζει τις αδρανειακές και βαρυτικές δυνάμεις.

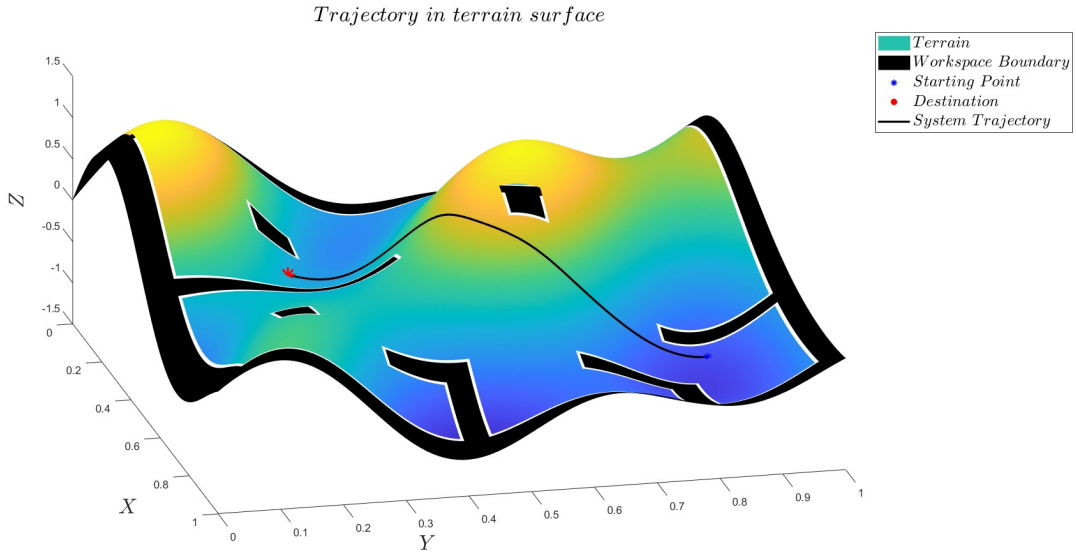
[7] Η συγκεκριμένη συνάρτηση είναι C^∞ αλλά όχι αναλυτική.

5.2 Προσομοίωση Ελεγκτή

Στη συνέχεια παρουσιάζεται η τροχιά ενός συστήματος σε έναν τυχαίο γνωστό χώρο υπό την επίδραση του ελεγκτή αυτού, στα σχήμα 8 και 9.



Σχήμα 8: Τροχιά ελεγμένου συστήματος

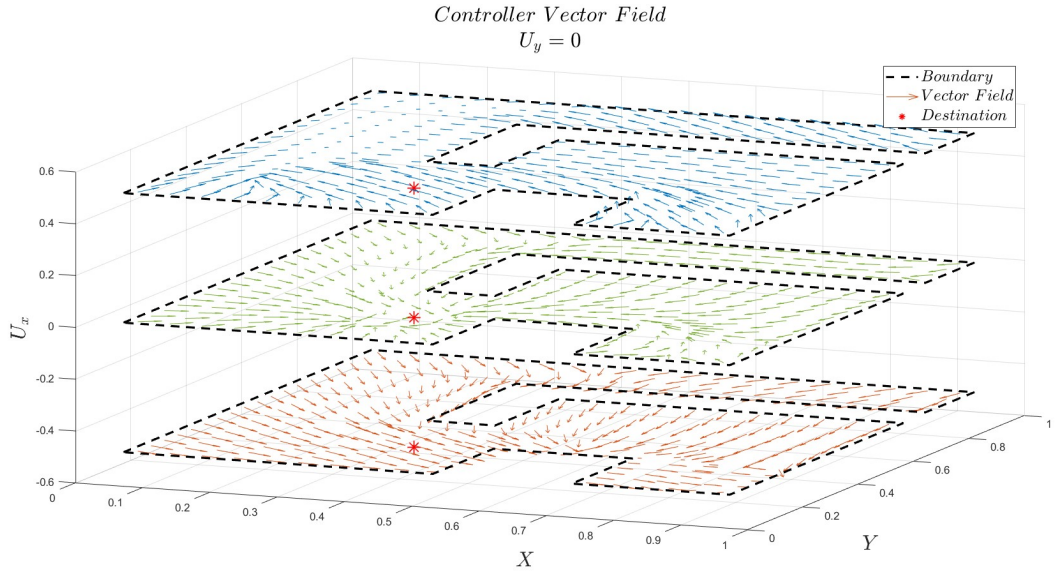


Σχήμα 9: Τροχιά ελεγμένου συστήματος πάνω στην επιφάνεια του εδάφους

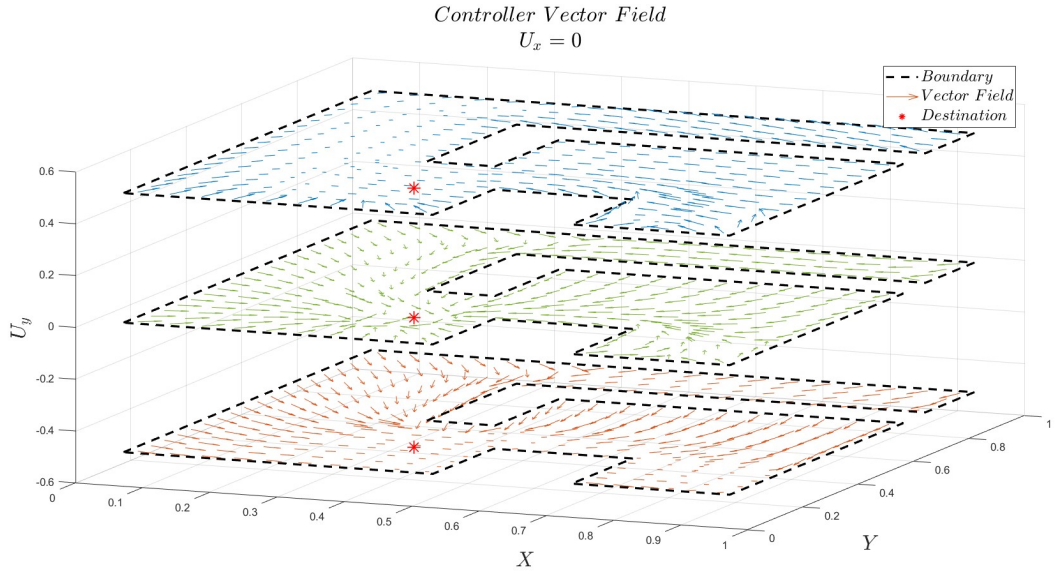
Ο ελεγκτής αυτός μπορεί να θεωρηθεί ως ένα διανυσματικό πεδίο $F : \mathbb{R}^4 \to \mathbb{R}^2$, που λαμβάνει την κατάσταση και δίνει την δύναμη επενέργησης:

$$F(\boldsymbol{x}) = F(x_1, \dot{x}_1, x_2, \dot{x}_2) = (u_1, u_2) \tag{62}$$

Καθώς η είσοδος στο πεδίο είναι τεσσάρων διαστάσεων, δεν μπορεί να αναπαρασταθεί στον τρισδιάστατο χώρο. Για αυτό παγώνεται μια μεταβλητή και λαμβάνεται η τομή του διανυσματικού χώρου του ελεγκτή. Ακολουθούν δύο παραδείγματα στα σχήματα 11 και 10.



Σχήμα 10: Διανυσματικό πεδίο ελεγκτή.Τομή για $U_{y0} = 0m/s$.



Σχήμα 11: Διανυσματικό πεδίο ελεγκτή.Τομή για $U_{x0} = 0m/s$.

5.3 Κόστος Ελεγκτή

Το κόστος του ελεγκτή δίνεται από τον τύπο:

$$L = \int_{t=0}^{\infty} a||x - x_d||^2 + b||\dot{x}||^2 + c||u||^2 dt \tag{63}$$

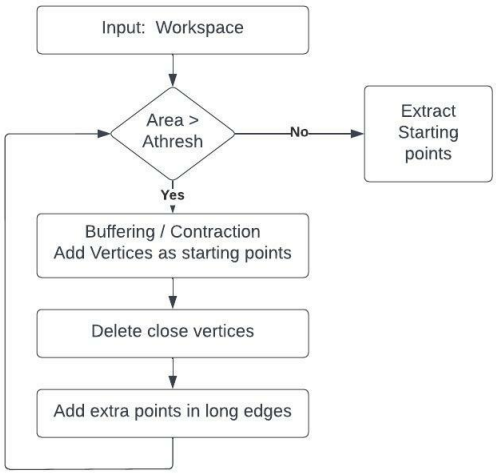
όπου:

- Ο όρος $a||x - x_d||^2$ λαμβάνει υπόψη την απόσταση από τον στόχο κάθε χρονική στιγμή
- Ο όρος $b||\dot{x}||^2$ λαμβάνει υπόψη την ταχύτητα του συστήματος
- Ο όρος $c||u||^2$ λαμβάνει υπόψη το κόστος επενέργησης.

Οι παράμετροι $a,\ b,\ c$ αποτελούν τα σχετικά βάρη. Στη συνέχεια παρουσιάζονται διαγράμματα για διάφορα σχετικά βάρη.

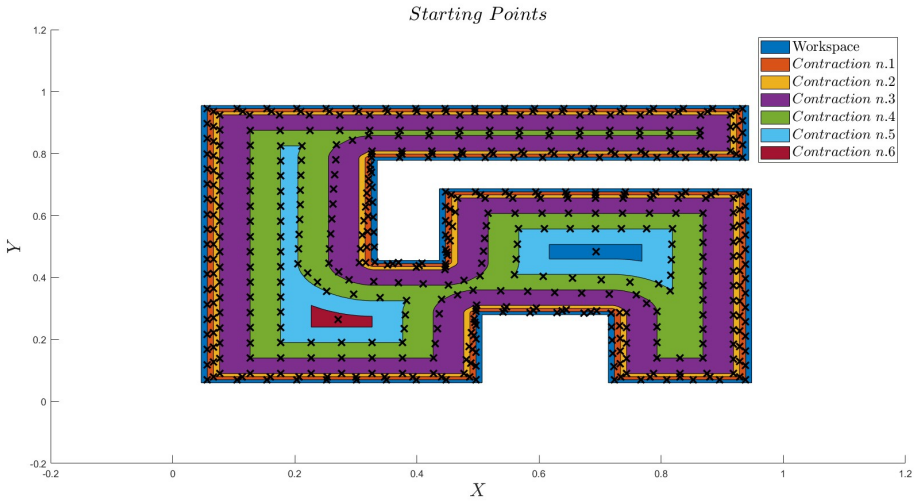
5.3.1 Επιλογή Αρχικών Σημείων

Για να υπολογισθεί το κόστος του ελεγκτή, πρέπει να υπολογισθούν οι τροχιές του συστήματος από διάφορα σημεία του χώρου εργασίας. Ύστερα πρέπει να γίνει παρεμβολή της τιμής του κόστους ώστε να δημιουργηθεί η επιφάνεια κόστους[8]. Όμως, ένας τυχαίος χώρος έχει διάφορες διαμορφώσεις, και για να μπορέσει να γίνει η παρεμβολή πρέπει να υπάρχουν αρκετά σημεία σε κάθε διαμόρφωση του χωρίου. Επιπλέον, ένα άλλο ζητούμενο από αυτή την ανάλυση είναι η διερεύνηση φαινομένων απειρισμού στα όρια του χώρου εργασίας. Αυτό απαιτεί ακριβέστερη ¨διακριτοποίηση' των αρχικών σημείων κοντά στα όρια του χώρου εργασίας. Έτσι, δημιουργήθηκε ένα πρόγραμμα, το *DOE.m* το οποίο λαμβάνει με συστηματικό τρόπο σημεία μέσα στο χωρίο.



Σχήμα 12: Συνοπτικό λογικό Διάγραμμα προγράμματος επιλογής αρχικών σημείων

Ειδικότερα, ο χώρος εργασίας αρχικά συστέλλεται πολύ λίγο, ώστε να δημιουργηθεί ένα όμοιο του workspace χωρίο, εσωτερικά του. Εδώ πρέπει να τονισθεί, πώς με αυτή την μέθοδο υπάρχουν σίγουρα (ρυθμίζοντας και το offset της συστολής) αρκετά σημεία σε κάθε διαμόρφωση του χώρου, ώστε η διαμόρφωση αυτή να μπορεί να αναπαρασταθεί με ένα πολύγωνο [9] . Ύστερα, όλες οι κορυφές γίνονται υποψήφια σημεία εκκίνησης, αφού πρώτα πολύ κοντινά σημεία εκφυλιστούν σε ένα. Παράλληλα, σε πολύ μακρινά vertices (σε μεγάλα ευθύγραμμα τμήματα), εισέρχονται και άλλα σημεία εκκίνησης ενδιάμεσα από αυτά. Ύστερα, η διαδικασία αυτή επαναλαμβάνεται έως ότου το εμβαδόν των εσωτερικών πολυγώνων (συσταλμένων χωρίων), ξεχωριστά το καθένα, να είναι μικρότερο από μια σταθερά (A_{Thresh}). Τότε, λαμβάνεται το κεντροειδές (centroid) των πολυγώνων αυτών σαν τελευταίο σημείο εκκίνησης. Τόσο τα διάφορα offsets (αρχικά το offset είναι μικρότερο για να παρατηρηθεί ο απειρισμός του ελεγκτή), όσο και οι αποστάσεις για εκφυλισμό των κοντινών κορυφών και εμπλουτισμό των μακρινών, είναι παράμετροι που μπορούν να τροποποιηθούν, μέχρι η δειγματοληψία στο εσωτερικό του χωρίου να είναι ικανοποιητική. Στο σχήμα 12 παρουσιάζεται ένα συνοπτικό λογικό διάγραμμα[10]. για να γίνει περισσότερο κατανοητή η λειτουργία του προγράμματος, ενώ στο σχήμα 13 παρουσιάζεται το αποτέλεσμα του προγράμματος με τα σημεία εκκίνησης. Παρατηρείται πως πράγματι στα όρια γίνεται πυκνή δειγματοληψία.



Σχήμα 13: Αρχικά σημεία που επιλέγονται από το πρόγραμμα *DOE.m*

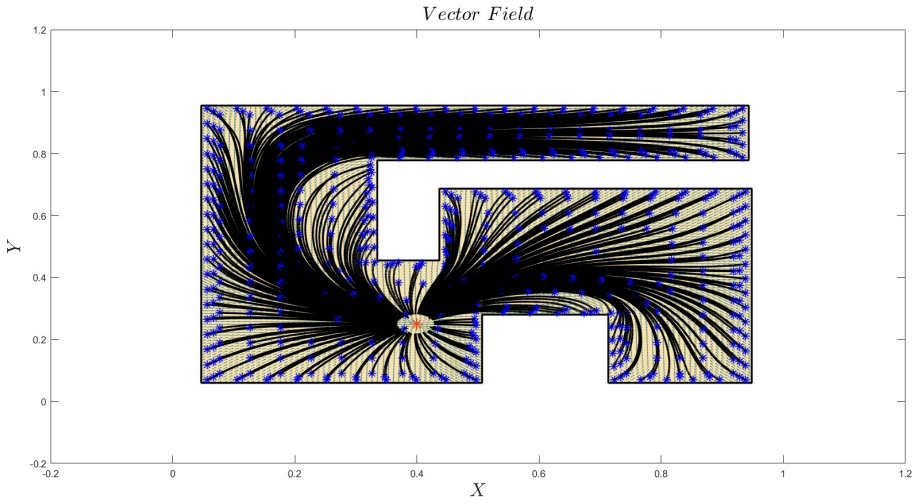
[8] Η παρεμβολή γίνεται ούτως η άλλως από την συνάρτηση surf της matlab.

[9] Σε αρχικές δοκιμές -χρησιμοποιώντας την εντολή *meshgrid* και ελέγχοντας αν τα σημεία είναι στον χώρο εργασίας- σε κάποιους διαδρόμους λαμβάνονταν starting points κατά μήκος μιας γραμμής. Οπότε, με την εντολή *surf*, δεν μπορούσε να γίνει παρεμβολή, και στα γραφήματα του κόστους οι διάδρομοι αυτοί δεν φαίνονταν.

[10] Στο παράρτημα, υπάρχει αναλύτικότερο λογικό διάγραμμα, στο σχήμα 22

5.3.2 Υπολογισμός Κόστους

Έχοντας τα υποψήφια αρχικά σημεία, προσομοιώνονται οι τροχιές του συστήματος και υπολογίζεται αριθμητικά το κόστος από τον τύπο 63. Αρχικά παρουσιάζονται οι τροχιές των σημείων στο σχήμα 14. Τα σημεία εκκίνησης είναι αυτά τα οποία φαίνεται να επιλέγονται στο σχήμα 13.



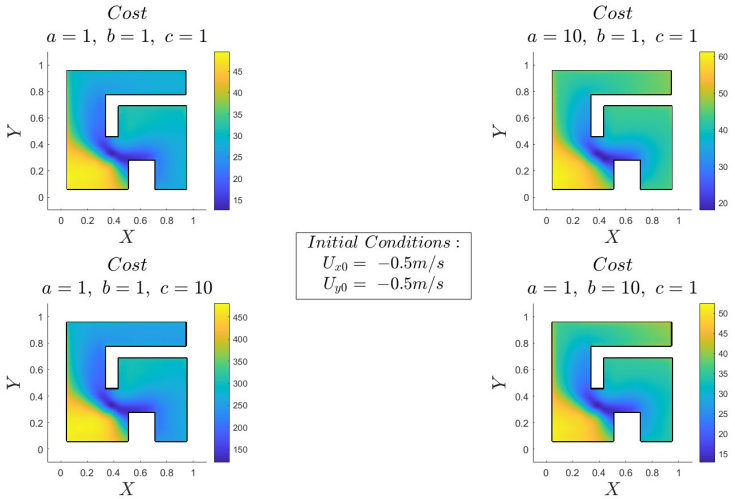
Σχήμα 14: Τροχιές σημείων που επιλέγονται από το πρόγραμμα *DOE.m*

Ύστερα, εφόσον έχει υπολογιστεί το κόστος σε κάθε σημείο, γίνεται παρεμβολή του κόστους σε ένα structured grid (που προκύπτει από την εντολή *meshgrid* της matlab) για να προκύψουν τα τελικά διαγράμματα. Η παρεμβολή γίνεται με την εντολή *griddata* και τη μέθοδο *v4*. Τα αποτελέσματα της παρεμβολής έχουν συγκριθεί και με τη μέθοδο *nearest* για επαλήθευση.

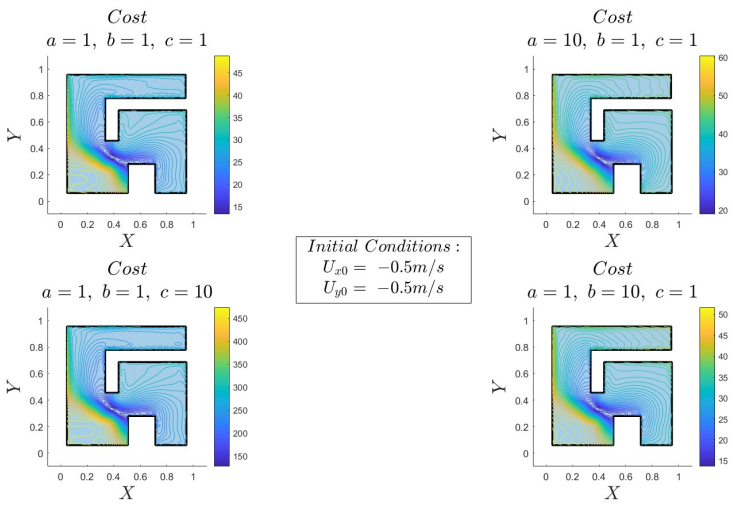
Ακολουθούν μερικά γραφήματα με την επιφάνεια κόστους όπως υπολογίστηκε για αρχικές συνθήκες:

$$u_{x0} = u_{y0} = -0.5m/s$$

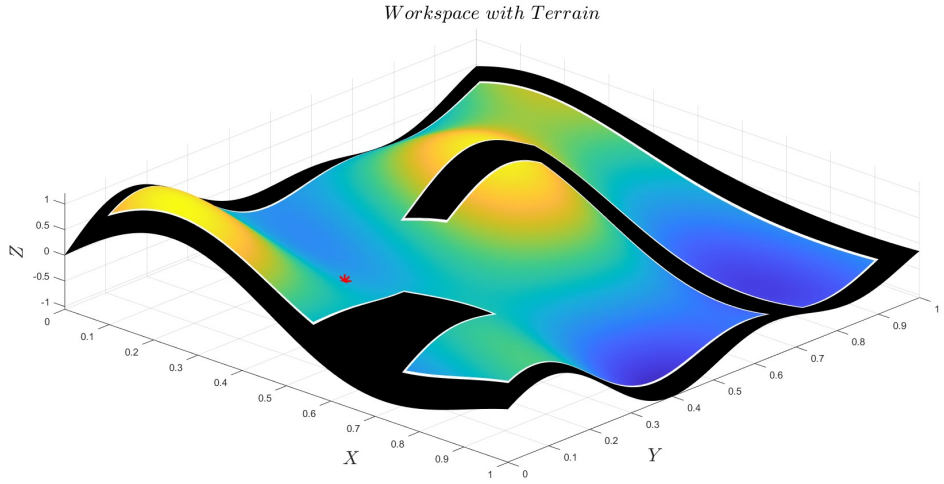
και την επίδραση της βαρύτητας στα σχήματα 15 και 16. Παρατηρείται πως στην κάτω αριστερή περιοχή το κόστος είναι πολύ αυξημένο, καθώς ο ελεγκτής πρέπει συνεχώς να αντισταθμίζει τις δυνάμεις που προκύπτουν λόγω της κλίσης. Η κλίση του εδάφους φαίνεται στο σχήμα 17.



Σχήμα 15: Επιφάνεια κόστους επενεργητή με αντιστάθμιση βαρύτητας, με αρχικές συνθήκες ταχύτητας $u_{x0} = u_{y0} = -0.5m/s$

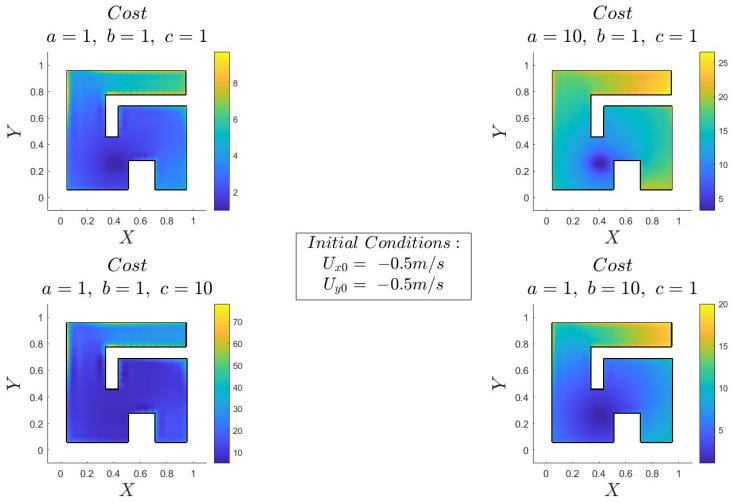


Σχήμα 16: Ισοσταθμικές γραμμές κόστους επενεργητή με αντιστάθμιση βαρύτητας, με αρχικές συνθήκες ταχύτητας $u_{x0} = u_{y0} = -0.5m/s$

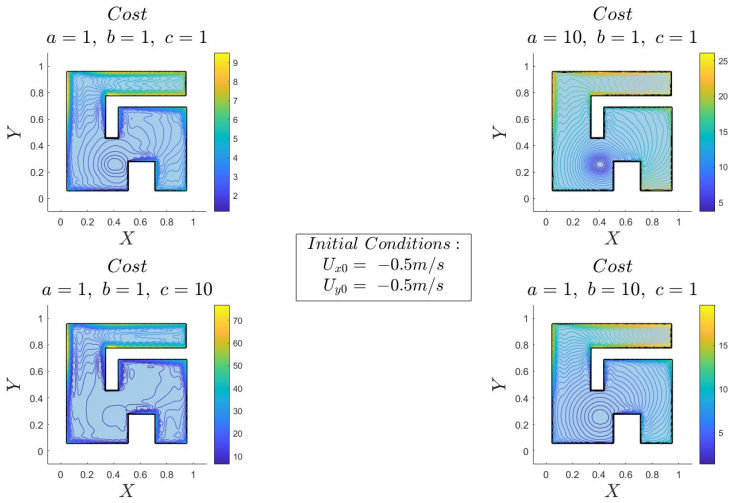


Σχήμα 17: Επιφάνεια εδάφους στον χώρο εργασίας

Αμελώντας την βαρύτητα και τις αδρανειακές δυνάμεις από την προσομοίωση (γεγονός που σημαίνει πως αφαιρείται και ο όρος αντιστάθμισης F_{comp} από τον ελεγκτή), προκύπτουν οι επιφάνειες που φαίνονται στα σχήματα 18 και 19. Παρατηρείται πως κοντά στα όρια, το κόστος απειρίζεται, όπως είναι αναμενόμενο λόγω του όρου $\frac{1}{\sigma_r}$.



Σχήμα 18: Επιφάνεια κόστους επενεργητή χωρίς αντιστάθμιση βαρύτητας, με αρχικές συνθήκες ταχύτητας $u_{x0} = u_{y0} = -0.5m/s$



Σχήμα 19: Ισοσταθμικές γραμμές κόστους επενεργητή χωρίς αντιστάθμιση βαρύτητας, με αρχικές συνθήκες ταχύτητας $u_{x0} = u_{y0} = -0.5m/s$

6. Αναφορές

[1] Savvas G. Loizou. Closed form navigation functions based on harmonic potentials. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 6361–6366, 2011.

[2] Sherman Formula in Wikipedia:. `https://en.wikipedia.org/wiki/Sherman-Morrison_formula`.

[3] Panagiotis Rousseas, Charalampos P. Bechlioulis, and Kostas J. Kyriakopoulos. Trajectory planning in unknown 2d workspaces: A smooth, reactive, harmonics-based approach. *IEEE Robotics and Automation Letters*, 7(2):1992–1999, 2022.

7. Προγράμματα Matlab

7.1 Navigation Functions

7.1.1 NavigationFunctionSimulation.m

Αρχικά, παρουσιάζεται ο γενικός κώδικας (”main”) στον οποίο ο χρήστης βάζει τα στοιχεία του χώρου εργασίας, δηλαδή:

- τον επιθυμητό προορισμό,
- το σημείο εκκίνησης και
- τη θέση των εμποδίων.

Πέρα από το section των εισόδων (*input* στο πρόγραμμα), ο χρήστης δεν αλληλεπιδρά παραπάνω με το πρόγραμμα.

Στη συνέχεια το πρόγραμμα πλοτάρει το workspace, καλεί την συνάρτηση *navigationFUN.m* που δίνει τη συνάρτηση πλοήγησης και το πεδίο ταχυτήτων (αρνητικό gradient) και κάνει τα αντίστοιχα γραφήματα. Στο τέλος, γίνεται η προσομοίωση του συστήματος, με βάση αυτό το πεδίο ταχυτήτων. Έχει δημιουργηθεί και ένα event function, το *reach_target.m*, ώστε η προσομοίωση να σταματάει όταν το σύστημα πλησιάσει αρκετά τον στόχο.

7.1.2 navigationFUN.m

Η συνάρτηση δέχεται σαν όρισμα τα:

- $input$: θέση στον χώρο
- Pi: θέση εμποδίων
- Pd: προορισμός
- Ki, Kd: Παράμετροι του πεδίου που ορίζονται στο κυρίως πρόγραμμα

και υπολογίζει το πεδίο ταχυτήτων και το navigation function.

7.1.3 reach_target.m

Η συνάρτηση δέχεται σαν όρισμα τα:

- t: χρόνος (για να έχει συμβατό format με τα ode functions της matlab).
- y: θέση συστήματος
- Pd: προορισμός

και εξετάζει αν το σύστημα βρίσκεται κοντά στον προορισμό

7.1.4 Κώδικες

NavigationFunctionSimulation.m:

```
%% Info
%The user provides the parameters of the Circular Workspace (destination,
%starting point and obstacle positions). Then the programm calls
%NavigationFun.m to compute the navigation function and it simulates the
%trajectory of the simple intergrator

%% input
%Obstacle positions [x1,x2,...,xn  ; y1,y2,...,yn]
Pi = [ -0.2  -0.3 0.15 0.25 0.6; 0.4 -0.05 -0.3 -0.05 0.05];
%Destination
```

```
Pd = [ 0.55; 0.6];
%Starting Point
Ps = [-0.4; -0.3];

Ki = 1;
Kd = Ki*size(Pi,2)+1;
%% Workspace plot
tt = linspace(0,2*pi);

figure(1)
plot(cos(tt),sin(tt),'k','DisplayName','$Workspace \ Boundary$') %contour
hold on
scatter(Pi(1,:),[Pi(2,:)],150,'k','Marker','*','LineWidth',1,'DisplayName','
    $Point \ Obstacles$') %point-obstacle
hold on
scatter(Pd(1),Pd(2),150,'r','Marker','*','LineWidth',1,'DisplayName','
    $Destination$') % destination
hold on
scatter(Ps(1),Ps(2),150,'b','Marker','*','LineWidth',1,'DisplayName','
    $Starting \ Point$') %starting point
hold on
title('$Workspace$','Interpreter','latex','FontSize',20)
xlabel('$X $','Interpreter','latex','FontSize',20)
ylabel('$Y$','Interpreter','latex','FontSize',20)
legend('Interpreter','latex',FontSize=15,Location='bestoutside')
axis equal %no distortion

%% Vector Field
Xx = -1:0.025:1;
Yy = -1:0.025:1;

npoints = length(Xx);
[X,Y] = meshgrid(Xx,Yy);

nav = zeros(npoints,npoints);
grad_fieldY = nav;
grad_fieldX = nav;

for row = 1:size(X,1)
    for cols = 1: size(X,2)
        if X(row,cols)^2+Y(row,cols)^2 <=1
            [GRAD, nav_calced] = navigationFUN([X(row,cols),Y(row,cols)],Pi,
                Pd,Ki,Kd);
            nav(row,cols) = nav_calced;
            grad_fieldX(row,cols)  = GRAD(1)/sqrt(GRAD(1)^2 + GRAD(2)^2) ;
            grad_fieldY(row,cols)  = GRAD(2)/sqrt(GRAD(1)^2 + GRAD(2)^2) ;
        else
            nav(row,cols) = nan;
            grad_fieldX(row,cols)= nan;
            grad_fieldY(row,cols) =nan;
        end
    end
end

%Navigation Function
figure(25)
surf(X,Y,nav,'FaceColor','texturemap')
```

```
title('$Navigation \ Function$','Interpreter','latex','FontSize',20)
xlabel('$X $','Interpreter','latex','FontSize',20)
ylabel('$Y$','Interpreter','latex','FontSize',20)
zlabel('$\Theta$','Interpreter','latex','FontSize',20)
view(45,20);
axis equal

%Workspace with velocity Field
figure(31)
plot(cos(tt),sin(tt),'k','DisplayName','$Workspace \ Boundary$') %contour
hold on
quiver(X,Y,-grad_fieldX,-grad_fieldY,1,'Color','#0072BD','DisplayName','
    $Vector \ Field$')
hold on
scatter(Pi(1,:),[Pi(2,:)],150,'k','Marker','*','LineWidth',1,'DisplayName','
    $Point \ Obstacles$') %point-obstacle
hold on
scatter(Pd(1),Pd(2),150,'r','Marker','*','LineWidth',1,'DisplayName','
    $Destination$') % destination
hold on
scatter(Ps(1),Ps(2),150,'b','Marker','*','LineWidth',1,'DisplayName','
    $Starting \ Point$') %starting point
hold on
title('$Vector \ Field$','Interpreter','latex','FontSize',20)
xlabel('$X$','Interpreter','latex','FontSize',20)
ylabel('$Y$','Interpreter','latex','FontSize',20)

%% simulation
final_fun = @(t,y) reach_target(t,y,Pd);
options = odeset('Events',final_fun );
[t,y] = ode45(@(t,y) -navigationFUN(y,Pi,Pd,Ki,Kd)' ,[0 1000],Ps,options);

plot(y(:,1),y(:,2),'k','LineWidth',2,'DisplayName','$System \ Trajectory$')
legend('Interpreter','latex',FontSize=15,Location='best')
hold off
axis equal %no distortion
```

navigationFUN.m:

```
function [GRADIENT,Var4,Var2] = navigationFUN(input,Pi,Pd,Ki,Kd)
%input is the position in 2D space.
%Pi is a 2xM matrix with the positions of the point obstacles. The first
%row is the x-direction, and the second the y-direction of each obstacle.
%Pd is a 2x1 vector with the destination point
%Ki is the gain of the sources- obstacles
%Kd is the gain of the sink-destination

x = input(1); %x-direction
y = input(2); %y-direction
M = size(Pi,2); %number of point obstacles

%% transformations
radius = 1;
Var1 = [x,y]/( radius - sqrt(x.^2+y.^2) ) ; % to harmonic domain x,y
HarPd = Pd'/( radius - norm(Pd) );          % to harmonic domain,
```

```
    destination
for i = 1:M
    HarPi(i,:) = [Pi(1,i),Pi(2,i)]/( radius - norm([Pi(1,i),Pi(2,i)]) ) ;
%to harmonic domain, obstacles
end

%harmonic potential - sources and sinks ----------------------
Var2 = Kd*log( norm(Var1 - HarPd ));
for i = 1:M
    Var2 = Var2 - Ki*log( norm( Var1 - HarPi(i,:) )) ;
end
%-------------------
Var3 = exp(Var2)./(exp(Var2)+1) ; %to [0 1];
Var4 = Var3.^(2/Kd); % for non-degenerate destination

%% gradient calculation
der4 = 2/Kd * ( Var3).^((2-Kd)/(Kd));

der3 = exp(Var2)./(1+exp(Var2) ).^2;

%harmonic potential - sources and sinks ----------------------
der2 = Kd*(Var1 - HarPd)/norm(Var1 - HarPd)^2;
for i = 1:M
    der2 = der2 - Ki*(Var1 - HarPi(i,:))/norm(Var1 - HarPi(i,:))^2;
end
%-------------------
der1 = [radius - y^2/sqrt(x^2+y^2),radius - x^2/sqrt(x^2+y^2)]/ (radius -
    sqrt(x^2+y^2) )^2 ;

%% output
GRADIENT  = der4*der3*der2.*der1 ;
end
```

reach_target.m:

```
function [value,isterminal,direction] = reach_target(t,y,Pd)

if norm(Pd-y) < 1e-6 %accuracy
    value = 0;
else
    value =1;
end
isterminal = 1; %terminate if true
direction= 0;
end
```

7.2 Δυναμική Προσομοίωση

7.2.1 gravitySimulation.m

Η συνάρτηση αυτή δέχεται τα όρια κατά x, y που θα έχει στο γράφημα για να αναπαραστήσει το έδαφος. Επιπλέον, λαμβάνει σαν είσοδο την αρχική κατάσταση του συστήματος (θέση και ταχύτητα κατά x, y) και τον χρόνο προσομοίωσης. Ύστερα, πλοτάρεται το έδαφος και γίνεται η δυναμική προσομοίωση καλώντας την συνάρτηση *gravity2D.m*. Τέλος, παρουσιάζεται ένα animation της τροχιάς που ακολουθεί το σύστημα πάνω στο έδαφος.

7.2.2 gravity2D.m

Η συνάρτηση αυτή δέχεται τη θέση του συστήματος και βγάζει την δύναμη σύμφωνα με τη σχέση 59.

7.2.3 Κώδικες

gravitySimulation.m:

```
%This programm plots the ground surface and then, given the initial
%conditions specified in the following section, simulates the trajectory of
% the system

%it is assumed there is no friction

%% Input: initial Conditions
PosXY = [-0.825,0.05];
Velocity = [0,0];
simTime = 5; %seconds

%plot limits
xlimits = [-1.1,2];
ylimits = [-0.6,0.8];

%% Terrain Plot
meshx = xlimits(1):0.025:xlimits(2);
meshy = ylimits(1):0.025:ylimits(2);
sizeofvarx = length(meshx);
sizeofvary = length(meshy);

[X,Y] = meshgrid(meshx,meshy);

lamda = 10; %scaling factor
H = @( X,Y ) 0.5*sin(0.5*lamda*X)+0.3*sin(1.3*lamda*Y)+0.4*sin(0.7*lamda*(X+
    Y))+0.2*sin(0.8*lamda*X+0.9*lamda*Y) ;

figure(WindowState="maximized")
s1 = surf(X,Y,H(X,Y),'DisplayName','$Terrain$');
hold on

%% Simulation
[t,y] = ode45(@(t,y) gravity2D(y),[0,simTime],[PosXY(1);Velocity(1);PosXY(2)
    ;Velocity(2)]);
XX = y(:,1);
YY = y(:,3);

instances = length(t);
Height = H(XX,YY); %trajectory height

%% plot
```

```
sc1 = scatter3(XX(1),YY(1),Height(1)+1e-2,'b','SizeData',200,'Marker',"*","LineWidth",3);
hold on
sc2 = scatter3(XX(end),YY(end),Height(end)+1e-2,'r','SizeData',200,'Marker',"*","LineWidth",3);
hold on

%for animation
view([-169.8875,71.9777])
%pause(10) %for video

for iT = 1:instances-1
    p2 = plot3(XX(1:iT),YY(1:iT),Height(1:iT),'LineWidth',2,'Color','k');
    pause(t(iT+1)-t(iT))
end

title('$Path \ of \ system$','Interpreter','latex','FontSize',20)
xlabel('$X$','Interpreter','latex','FontSize',20)
ylabel('$Y$','Interpreter','latex','FontSize',20)
zlabel('$Height$','Interpreter','latex','FontSize',20)
pause(1)
view(33.5,74)
legend('$Terrain$','$Starting \ Point$','$Destination$','$Trajectory$',Interpreter='latex',FontSize=20)
```

gravity2D.m:

```
function [yt] = gravity2D(y)
g = 9.81;

lamda = 10;%scaling factor
%IT MUST BE EQUAL TO THE HEIGHT FUNCTION

X = 10*y(1);Y = 10*y(3);
%X = y(1);Y = y(3);

%% Grads calcs

% % H = @(X,Y) 0.5*sin(0.5*X)+0.3*sin(1.3*Y)+0.4*sin(0.7*(X+Y))+0.2*sin(0.8*X+0.9*Y) ;
DH_DX =     0.25*cos(0.5*X) +    0.4*0.7*cos(0.7*(X+Y)) +    0.2*0.8*cos(0.8*X+0.9*Y) ;
DH_DXX=-0.25*0.5*sin(0.5*X) -0.4*0.7*0.7*sin(0.7*(X+Y)) - 0.2*0.8*0.8*sin(0.8*X+0.9*Y) ;

DH_DY =     0.3*1.3*cos(1.3*Y) +    0.4*0.7*cos(0.7*(X+Y)) +    0.2*0.9*cos(0.8*X+0.9*Y);
DH_DYY=-0.3*1.3*1.3*sin(1.3*Y) -0.4*0.7*0.7*sin(0.7*(X+Y)) - 0.2*0.9*0.9*sin(0.8*X+0.9*Y) ;
DH_DXY= -0.4*0.7*0.7*sin(0.7*(X+Y)) - 0.2*0.9*0.8*sin(0.8*X+0.9*Y);

%% output
gradH = [DH_DX;DH_DY];
Hess = lamda*[DH_DXX, DH_DXY; DH_DXY DH_DYY];

```

```
24 force = -1/( 1 + norm(gradH)^2 )*(g +  ([y(2),y(4)]* Hess * [y(2);y(4)])    )
       *gradH ;
25
26 yt(1,1) = y(2);
27 yt(2,1) = force(1);
28 yt(3,1) = y(4);
29 yt(4,1) = force(2);
```

7.3 Ελεγκτής

7.3.1 ControllerCostCalc.m

Το πρόγραμμα αυτό στήνει το αρμονικό πεδίο στον χώρο και υπολογίζει την επιφάνεια κόστους με διάφορα βάρη στη συνάρτηση 60. Ο χρήστης δίνει το επιθυμητό προορισμό στο χώρο εργασίας και τις αρχικές συνθήκες του ρομπότ. Επίσης, έχει την επιλογή να πλοτάρει την επιφάνεια του εδάφους πάνω στο workspace αν θέλει, θέτοντας το flag: *TerrainPlot* ίσο με ένα. Η διαδικασία που ακολουθείται χοντρικά είναι η εξής:

- Αρχικά δημιουργείται το αρμονικό πεδίο από τους αντίστοιχους κώδικες
- Ύστερα, γίνονται διάφορα γραφήματα του πεδίου. Ο χρήστης πρέπει να βεβαιωθεί πως οι παράμετροι που έχει θέσει στο αρχείο *input_panel.txt* δίνουν ένα ικανοποιητικό πεδίο, όπου τα διανύσματα δεν δείχνουν στο εξωτερικό του.
- Στη συνέχεια, καλείται η συνάρτηση *DOE.m* και διαλέγονται τα αρχικά σημεία.
- Έπειτα γίνεται η προσομοίωση για κάθε σημείο και υπολογίζεται το κόστος του.
- Μετά, γίνεται η δισδιάστατη παρεμβολή των σημείων στα οποία υπολογίστηκε το κόστος με ένα πολύ πυκνό πλέγμα (που προέρχεται από την εντολή meshgrid).
- Τέλος, γίνονται τα διαγράμματα κόστους.

7.3.2 potentialFIELD.m

Αυτός ο κώδικας χρειάζεται για να διαβάζει τα διανύσματα του αρμονικού πεδίου και να επιστρέφει και τρισδιάστατα διανύσματα. Επιλέχθηκε να μην χρησιμοποιηθεί η δεύτερη δυνατότητα, διότι τα γραφήματα δεν ήταν όμορφα.

7.3.3 reach_targetFULL.m

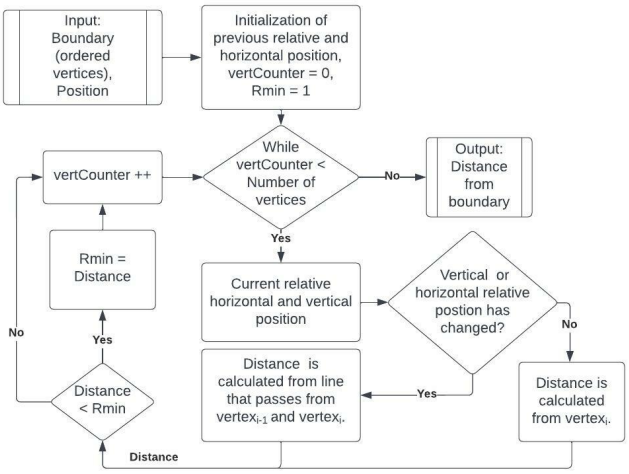
Αντίστοιχος κώδικα με την περίπτωση των navigation functions. Ωστόσο, εδώ η κατάσταση του συστήματος έχει 4 διαστάσεις, αφού το σύστημα είναι διπλός ολοκληρωτής.

7.3.4 controller.m

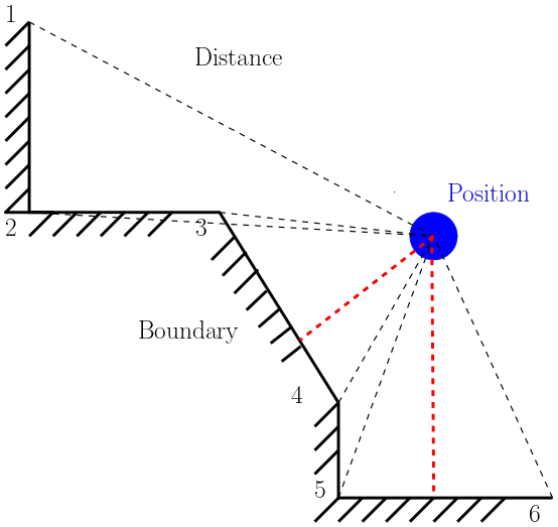
Το πρόγραμμα αυτό υλοποιεί τον προτεινόμενο ελεγκτή από την εξίσωση 60. Δέχεται σαν όρισμα την θέση του ρομπότ, και το αντικείμενο *robot*, από το οποίο καλείται η μέθοδος *input_next()* με σκοπό να ληφθεί η επιθυμητή ταχύτητα από το πεδίο ταχυτήτων. Επιπλέον, λαμβάνονται πληροφορίες για τα όρια του χωρίου. Ύστερα καλεί την συνάρτηση *distance.m* και βρίσκει την απόσταση του ρομπότ από τα όρια του χωρίου, ώστε να υπολογισθεί η τιμή του bump function. Στη συνέχεια καλείται η συνάρτηση *gravity2D.m* ώστε να παρθούν πληροφορίες για τις εξωτερικές δυνάμεις, οι οποίες πρέπει να αντισταθμιστούν. Τέλος, υπολογίζεται η δύναμη επενέργησης, η οποία αποτελεί την έξοδο του προγράμματος.

7.3.5 distance.m

Η συνάρτηση αυτή υπολογίζει την απόσταση του συστήματος από τα όρια του χωρίου. Η συνάρτηση αυτή εκμεταλλεύεται το γεγονός πως οι κορυφές του χωρίου είναι ήδη σε σειρά. Οπότε ξεκινώντας από την πρώτη κορυφή που είναι στο όριο του χώρου, υπολογίζει την διανυσματική απόσταση από την δοσμένη θέση με την σειρά. Ωστόσο, όπως φαίνεται και στο σχήμα 21, όταν η σχετική οριζόντια ή κάθετη θέση της κορυφής υπό εξέταση, ως προς την θέση του συστήματος, αλλάζει, τότε η απόσταση από τα όρια βρίσκεται από το ευθύγραμμο τμήμα που ενώνει την προηγούμενη και επόμενη κορυφή. Η παραπάνω διαδικάσια παρουσιάζεται και στο λογικό διάγραμμα στο σχήμα 20.



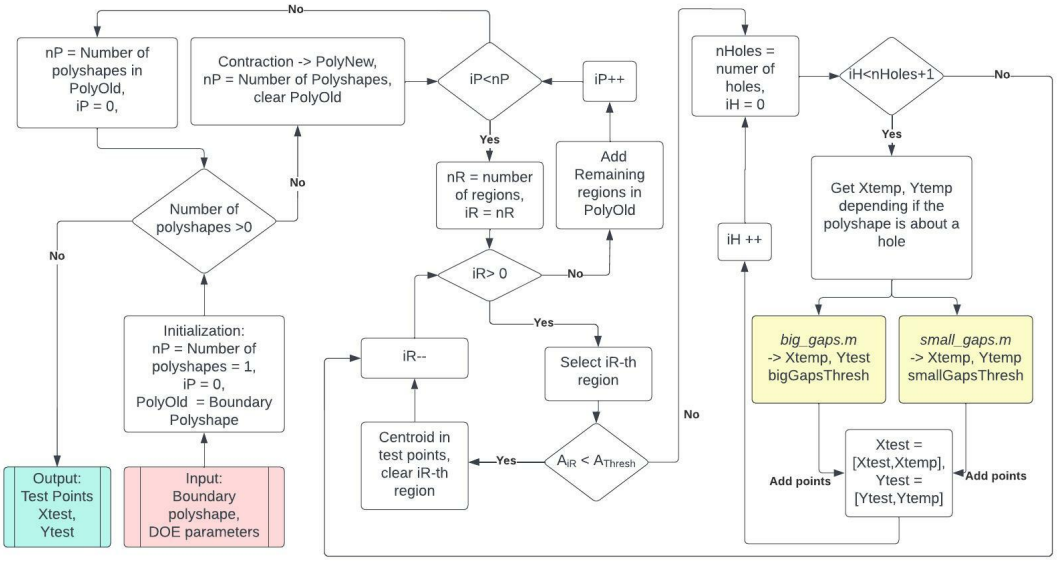
Σχήμα 20: Λογικό διάγραμμα συνάρτησης *distance.m*.



Σχήμα 21: Παράδειγμα υπολογισμού απόστασης από όρια χωρίου.

7.3.6 DOE.m

Η συνάρτηση αυτή βρίσκει αρχικά σημεία για τον υπολογισμό των τροχιών του συστήματος με τελικό σκοπό τον υπολογισμό κόστους. Η διαδικασία χοντρικά περιγράφτηκε στην ενότητα 5.3.1. Εδώ, παρουσιάζεται ένα αναλυτικό flowchart στο σχήμα 22.



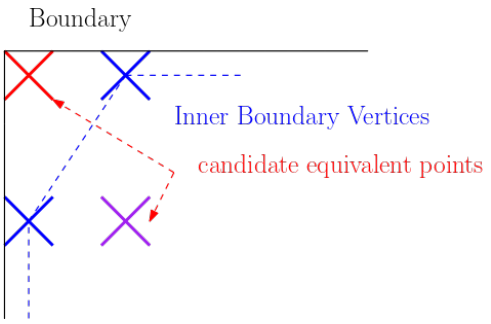
Σχήμα 22: Λογικό διάγραμμα του προγράμματος *DOE.m*

7.3.7 small_gaps.m

Αυτός το πρόγραμμα καλείται στην συνάρτηση *DOE.m* για να εκφυλίσει κοντινές κορυφές σε μια ισοδύναμη. Διαβάζει δυο διανύσματα με τις θέσεις των κορυφών των εσωτερικών πολυγώνων κατά τον άξονα x και y αντίστοιχα, **τα οποία είναι διατεταγμένα σε σειρά**, και ένα περιθώριο $thresh$, το οποίο καθορίζει την ελάχιστη απόσταση των κοντινών κορυφών.

Ύστερα, γίνονται τα εξής βήματα:

- Δημιουργεί 2 διανύσματα $Xtest$, $Ytest$ στα οποία σε πρώτη φάση αποθηκεύονται όλα τα σημεία υπό εξέταση.
- Βρίσκει την απόσταση κάθε σημείου από το επόμενο σημείο και ελέγχει αν είναι μικρότερη από το επιθυμητό περιθώριο.
- Δημιουργείται ένας πίνακας με δείκτες σε όσα σημεία είναι κοντά με το επόμενο.
- Κάθε σημείο που η απόστασή του από το επόμενο είναι μικρότερη του περιθωρίου, αντικαθίσταται με ένα ισοδύναμο σημείο. Αν τα δύο σημεία έχουν συντεταγμένες $\boldsymbol{x}_a = (x_a, y_a)$ και $\boldsymbol{x}_b = (x_b, y_b)$, τότε το ισοδύναμο σημείο έχει συντεταγμένες $\boldsymbol{x}_{equiv} = (x_a, y_b)$ ή $\boldsymbol{x}_{equiv} = (x_b, y_a)$. Επιλέγεται έτσι ώστε να είναι μέσα στον χώρο εργασίας. Τα ισοδύναμα σημεία φαίνονται στο σχήμα 23. Το επόμενο σημείο παραμένει στην λίστα.



Σχήμα 23: Τρόπος εκφυλισμού σημείων από την συνάρτηση *small_gaps.m*

- Όλα τα στοιχεία στα διανύσματα $Xtest$, $Ytest$ μετατοπίζονται μια θέση αριστερά.
- Διώχνονται από τα διανύσματα $Xtest$, $Ytest$ τα δεύτερα σημεία, με βάση τους δείκτες, που προηγουμένως έδειχναν στα αρχικά

7.3.8 big_gaps.m

Αυτό το πρόγραμμα καλείται στην συνάρτηση *DOE.m* για να εμπλουτίσει με σημεία κορυφές σε μεγάλη απόσταση. Διαβάζει δυο διανύσματα με τις θέσεις των κορυφών των εσωτερικών πολυγώνων κατά τον άξονα x και y αντίστοιχα, **τα οποία είναι διατεταγμένα σε σειρά**, και ένα περιθώριο $thresh$, το οποίο καθορίζει την μέγιστη απόσταση των μακρινών κορυφών. Ύστερα, γίνονται τα εξής βήματα:

- Βρίσκεται η διανυσματική απόσταση μακρινών σημείων $\boldsymbol{d} = (dx, dy)$.
- Βρίσκεται ο αριθμός των ευθύγραμμων κομματιών με τον οποίο θα διαιρεθεί η μακρινή απόσταση από τη σχέση:

$$SM = \lceil \frac{||d||}{thresh} \rceil$$

- Βρίσκονται τα έξτρα σημεία που πρέπει να προστεθούν στα σημεία εκκίνησης.

7.3.9 Κώδικες

ControllerCostCalc.m:

```
%This code creates many starting points to calculate the controller cost
%from different starting points in order to create the cost surface.

close all;
clear all;
clc;

%% input
%Destination:
pd = [ 0.4;0.25];
%Initial Velocity
iVel = [0.5,0.5];

%For plot
uxTxt = ['$U_{x0}=\ ',num2str(iVel(1)), 'm/s$'];
uyTxt = ['$U_{y0}=\ ',num2str(iVel(2)), 'm/s$'];

%plot terrain with projecte vector field flag
TerrainPlot = 0;

%% Height Function
lamda = 10;
radius =1;
H = @( X,Y ) 0.5*sin(0.5*lamda*X)+0.3*sin(1.3*lamda*Y)+0.4*sin(0.7*lamda*(X+
    Y))+0.2*sin(0.8*lamda*X+0.9*lamda*Y) ;

%% AHPF CREATE WORKSPACE
% SET PLOT HANDLE
plot_1 = 1;
% CALL THE CONSTRUCTOR FOR THE WORKSPACE CLASS
workspace_1 = workspace_panels('./input_panels.txt');
% PLOT WORKSPACE
workspace_1.plot_ws(plot_1);
workspace_1.plot_sources(plot_1);
%% AHPF SETUP POLICY
% PLOT DESIRED POSITION
plot(pd(1),pd(2),'r*','MarkerSize',10,'LineWidth',1.5);
% INITIALIZE FIELD
robot = robot_harmonic(pd,workspace_1);
% CALCULATE INITIAL POLICY
robot.initial_policy;

%% Vector Field
Xx = 0:0.005:1;
Yy = 0:0.005:1;
[X,Y] = meshgrid(Xx,Yy);

% Artificial Potential Field -> get data from AHPFcode for plots
boundary = robot.workspace.workspace_polyshape;
[X2,Y2,dx_x,dx_y,dz_x,dz_y,dz_z] = potentialFIELD(robot) ;
data_size = size(X2,2);
height2 = zeros(1,data_size);
for ii = 1:data_size
    height2(ii) = H(X2(ii),Y2(ii));
```

```
54 end
55
56 %Terrain Surface
57 Height = zeros(size(X,1));
58 HeightFalse = zeros(size(X,1));
59 %maxnorm = 0;
60 for row = 1:size(X,1)
61     for cols = 1: size(X,2)
62         if isinterior(boundary,X(row,cols),Y(row,cols))
63             Height(row,cols) = H(X(row,cols),Y(row,cols));
64             HeightFalse(row,cols) = nan ;
65         else
66             Height(row,cols) = nan;
67             HeightFalse(row,cols) = H(X(row,cols),Y(row,cols));%for nicer
                   plot
68         end
69     end
70 end
71
72 %% plots
73 % 2D (projected) Vector Field
74 figure(31)
75 quiver(X2,Y2,dz_x,dz_y)
76 hold on
77 scatter(pd(1),pd(2),150,'r','Marker','*','LineWidth',1)
78 hold on
79 boundaryPlot = boundary.plot;
80 boundaryPlot.LineWidth = 2;
81 hold on
82 title('$Vector \ Field \ in \ Workspace$','Interpreter','latex','FontSize'
       ,20)
83 xlabel('$X$','Interpreter','latex','FontSize',20)
84 ylabel('$Y$','Interpreter','latex','FontSize',20)
85
86 if TerrainPlot == 1
87     figure(41)
88     offset = 4; %[m]
89     %vector field with height
90     quiver(X2,Y2,dz_x,dz_y,1)
91     hold on
92     scatter(pd(1),pd(2),150,'r','Marker','*','LineWidth',1)
93     hold on
94     scatter3(pd(1),pd(2),H(pd(1),pd(2))+offset,200,'red','Marker','*','
           LineWidth',2)
95     hold on
96     boundary.plot(LineWidth=2,FaceColor="none")
97     hold on
98     surf(X,Y,Height+offset, EdgeColor="none")
99     hold on
100    surf(X,Y,HeightFalse+offset, EdgeColor="none",FaceColor="black")
101    view( 42.35,35.2074)
102    hold on
103    title('$Workspace \ with \ Terrain$','Interpreter','latex','FontSize'
           ,20)
104    xlabel('$X$','Interpreter','latex','FontSize',20)
105    ylabel('$Y$','Interpreter','latex','FontSize',20)
106    zlabel('$Z$','Interpreter','latex','FontSize',20)
```

```
107
108 end
109 %% Cost Calculation
110
111 %design of experiments
112 [PSX,PSY] = DOE(boundary,0.01,0.01,0.05,0.015,0.05);
113 NP = length(PSX);
114
115 %function to calculate input force
116 inputF = @(t,y)( controller(y,robot) );
117
118 %mallocks
119 TOTAL_COSTa = zeros(1,NP);
120 TOTAL_COSTb = zeros(1,NP);
121 TOTAL_COSTc = zeros(1,NP);
122 TOTAL_COSTd = zeros(1,NP);
123 TOTAL_COSTe = zeros(1,NP);
124 tic
125
126 for ix = 1:NP
127     %feedback of the calculation progress
128     disp("iteation")
129     disp(ix)
130     disp("from")
131     disp(NP)
132     disp("~~~~~~~~~~")
133
134     ps = [PSX(ix);PSY(ix)];
135
136     %save the costs
137     if ~isinterior(boundary,ps(1),ps(2))
138         %extra check
139         TOTAL_COSTa(ix) = nan;
140         TOTAL_COSTb(ix) = nan;
141         TOTAL_COSTc(ix) = nan;
142         TOTAL_COSTd(ix) = nan;
143         TOTAL_COSTe(ix) = nan;
144     else
145      % simulation
146
147         final_fun = @(t,y) reach_targetFULL(t,y,pd);
148         options = odeset('Events',final_fun,'RelTol',1e-8,'AbsTol',1e-10  );
149         initial_cond = [ps(1); iVel(1); ps(2);iVel(2)];
150         tspan = 10; %[s]
151
152         %no gravity experiments
153         % [t,y]  = ode15s( @(t,y)( controller(y,robot) )  ,[0 tspan],
                initial_cond,options);
154
155         %gravity
156         [t,y]  = ode15s( @(t,y)( controller(y,robot) +  gravity2D(y))  ,[0
                tspan],initial_cond,options);
157
158         %% Cost
159         DC = 1; %distance coefficient
160         VC = 1; %velocity coefficient
161         IC = 1; %actuation coefficient
```

```
162
163         %cost_vectors
164         distDiff = sqrt(  (y(:,1)-pd(1) ).^2  + (y(:,3)-pd(2) ).^2    );
165         vel      = sqrt(  y(:,2).^2  + y(:,4).^2    );
166
167         %reinitialize input
168         input  = zeros(length(t),1);
169         for i=1:length(t)
170             temp     = inputF(t,y(i,:));
171             input(i,1) = sqrt( temp(2)^2+temp(4)^2 );
172         end
173
174         %costs
175         Cost_dist   = trapz(t, distDiff  );
176         Cost_vel    = trapz(t,vel  );
177         Cost_input  = trapz(t,input);
178
179         %Calculate different combinations
180         TOTAL_COSTa(ix) = DC*Cost_dist  + VC*Cost_vel + IC*Cost_input;
181         TOTAL_COSTb(ix) = 10*DC*Cost_dist  + VC*Cost_vel + IC*Cost_input;
182         TOTAL_COSTc(ix) = DC*Cost_dist  + VC*Cost_vel + 10*IC*Cost_input;
183         TOTAL_COSTe(ix) = IC*Cost_input; %unweighted actuation cost
184         TOTAL_COSTd(ix) = DC*Cost_dist  + 10*VC*Cost_vel + IC*Cost_input;
185
186         %plot
187         figure(31)
188         scatter(ps(1),ps(2),75,'b','Marker','*','LineWidth',1)
189         hold on
190         plot(y(:,1),y(:,3),'k','LineWidth',2)
191         hold on
192     end
193 end
194 toc
195
196
197 %% Interpolation procedures
198 Discretization = 250;
199 %interpolation points
200 Px = linspace( min( boundary.Vertices(:,1)' ),max( boundary.Vertices(:,1)' )
        ,Discretization);
201 Py = linspace( min( boundary.Vertices(:,2)' ),max( boundary.Vertices(:,2)' )
        ,Discretization);
202 [PPx,PPy] = meshgrid(Px,Py);
203
204 %interpolation
205 TCa = griddata(PSX,PSY,TOTAL_COSTa,PPx,PPy,'v4');
206 TCb = griddata(PSX,PSY,TOTAL_COSTb,PPx,PPy,'v4');
207 TCc = griddata(PSX,PSY,TOTAL_COSTc,PPx,PPy,'v4');
208 TCd = griddata(PSX,PSY,TOTAL_COSTd,PPx,PPy,'v4');
209 TCe = griddata(PSX,PSY,TOTAL_COSTe,PPx,PPy,'v4');
210
211 %nan in points out of bounds
212 for row = 1:size(PPx,1)
213     for cols = 1: size(PPy,2)
214         if ~isinterior(boundary,PPx(row,cols),PPy(row,cols))
215             TCa(row,cols) = nan;
216             TCb(row,cols) = nan;
```

```
217                 TCc(row,cols) = nan;
218                 TCd(row,cols) = nan;
219                 TCe(row,cols) = nan;
220             end
221         end
222 end
223
224
225 %% Cost plot
226
227 fig1 = figure(101);
228 %got it from matlab
229 annotation(fig1,'textbox',...
230     [0.424437499999995 0.455579438744204 (0.17322252194195)
        (0.086537191925333)],...
231     'String',{'$Initial \ Conditions:$',uxTxt,uyTxt},...
232     'Interpreter','latex',...
233     'HorizontalAlignment','center',...
234     'FontSize',18, 'FitBoxToText','on');
235
236 subplot(2,2,1)
237 boundaryPlot = boundary.plot;
238 boundaryPlot.LineWidth = 2;
239 hold on
240 surf(PPx,PPy,TCa,'FaceColor','interp','EdgeColor','none')
241 title(['$ Cost$',newline,'$a=1,\ b=1,\ c=1 $'],FontSize=20,Interpreter='
    latex')
242 xlabel('$X$','Interpreter','latex','FontSize',20)
243 ylabel('$Y$','Interpreter','latex','FontSize',20)
244 colorbar
245 view(2)
246 axis equal
247 xlim([-0.1 1.1])
248 ylim([-0.1 1.1])
249
250
251 subplot(2,2,2)
252 boundaryPlot = boundary.plot;
253 boundaryPlot.LineWidth = 2;
254 hold on
255 surf(PPx,PPy,TCb,'FaceColor','interp','EdgeColor','none')
256 title(['$ Cost$',newline,'$a=10,\ b=1,\ c=1 $'],FontSize=20,Interpreter='
    latex')
257 xlabel('$X$','Interpreter','latex','FontSize',20)
258 ylabel('$Y$','Interpreter','latex','FontSize',20)
259 colorbar
260 view(2)
261 axis equal
262 xlim([-0.1 1.1])
263 ylim([-0.1 1.1])
264
265
266 subplot(2,2,3)
267 boundaryPlot = boundary.plot;
268 boundaryPlot.LineWidth = 2;
269 hold on
270 surf(PPx,PPy,TCc,'FaceColor','interp','EdgeColor','none')
```

```
271 title(['$ Cost$',newline,'$a=1,\ b=1,\ c=10 $'],FontSize=20,Interpreter='
    latex')
272 xlabel('$X$','Interpreter','latex','FontSize',20)
273 ylabel('$Y$','Interpreter','latex','FontSize',20)
274 colorbar
275 view(2)
276 axis equal
277 xlim([-0.1 1.1])
278 ylim([-0.1 1.1])
279
280
281
282 subplot(2,2,4)
283 boundaryPlot = boundary.plot;
284 boundaryPlot.LineWidth = 2;
285 hold on
286 surf(PPx,PPy,TCd,'FaceColor','interp','EdgeColor','none')
287 title(['$ Cost$',newline,'$a=1,\ b=10,\ c=1 $'],FontSize=20,Interpreter='
    latex')
288 xlabel('$X$','Interpreter','latex','FontSize',20)
289 ylabel('$Y$','Interpreter','latex','FontSize',20)
290 colorbar
291 view(2)
292 axis equal
293 xlim([-0.1 1.1])
294 ylim([-0.1 1.1])
295
296 %% Contour plot
297 fig2 = figure(202);
298 annotation(fig2 ,'textbox',...
299 [0.424437499999995 0.455579438744204 0.17322252194195 0.086537191925333],...
300 'String',{'$Initial \ Conditions:$',uxTxt,uyTxt},...
301     'Interpreter','latex',...
302     'HorizontalAlignment','center',...
303     'FontSize',18, 'FitBoxToText','on');
304
305 subplot(2,2,1)
306 boundaryPlot = boundary.plot;
307 boundaryPlot.LineWidth = 2;
308 hold on
309 contour(PPx,PPy,TCa,50)
310 title(['$ Cost$',newline,'$a=1,\ b=1,\ c=1 $'],FontSize=20,Interpreter='
    latex')
311 xlabel('$X$','Interpreter','latex','FontSize',20)
312 ylabel('$Y$','Interpreter','latex','FontSize',20)
313 colorbar
314 view(2)
315 axis equal
316 xlim([-0.1 1.1])
317 ylim([-0.1 1.1])
318
319
320 subplot(2,2,2)
321 boundaryPlot = boundary.plot;
322 boundaryPlot.LineWidth = 2;
323 hold on
324 contour(PPx,PPy,TCb,50)
```

```
325 title(['$ Cost$',newline,'$a=10,\ b=1,\ c=1 $'],FontSize=20,Interpreter='
    latex')
326 xlabel('$X$','Interpreter','latex','FontSize',20)
327 ylabel('$Y$','Interpreter','latex','FontSize',20)
328 colorbar
329 view(2)
330 axis equal
331 xlim([-0.1 1.1])
332 ylim([-0.1 1.1])
333
334
335 subplot(2,2,3)
336 boundaryPlot = boundary.plot;
337 boundaryPlot.LineWidth = 2;
338 hold on
339 contour(PPx,PPy,TCc,50)
340 title(['$ Cost$',newline,'$a=1,\ b=1,\ c=10 $'],FontSize=20,Interpreter='
    latex')
341 xlabel('$X$','Interpreter','latex','FontSize',20)
342 ylabel('$Y$','Interpreter','latex','FontSize',20)
343 colorbar
344 view(2)
345 axis equal
346 xlim([-0.1 1.1])
347 ylim([-0.1 1.1])
348
349
350
351 subplot(2,2,4)
352 boundaryPlot = boundary.plot;
353 boundaryPlot.LineWidth = 2;
354 hold on
355 contour(PPx,PPy,TCd,50)
356 title(['$ Cost$',newline,'$a=1,\ b=10,\ c=1 $'],FontSize=20,Interpreter='
    latex')
357 xlabel('$X$','Interpreter','latex','FontSize',20)
358 ylabel('$Y$','Interpreter','latex','FontSize',20)
359 colorbar
360 view(2)
361 axis equal
362 xlim([-0.1 1.1])
363 ylim([-0.1 1.1])
```

controller.m:

```
1 function [yt] = controller(input,robot)
2 %This function is the designed controllers
3
4 %Gains
5 k = 75;
6 kphi = 100;
7 Rd = 0.15; %Activation distance of bump function
8
9 x = input(1);x_t = input(2);
10 y = input(3);y_t = input(4);
```

```
11
12 %% Get Harmonic field
13 gradphi    = -input_next(robot,[x;y]); %gradient
14
15 %bump function
16 r = distance(robot.workspace.boundary{1},[x;y]); %to avoid holes
17
18 if r<Rd
19     sr = 1-exp(-(r./(r-Rd) ).^2 );
20 else
21     sr=1;
22 end
23
24 %% Get Inertial and gravitational forces
25 fcomp = -gravity2D(input);
26
27 %% Calculate Force
28 %with gravity
29  fin    = -kphi/(sr+1e-6)*gradphi - k* ( [x_t;y_t]-(-kphi*gradphi) ) +  fcomp
       ([2,4]);
30  yt = [0; fin(1); 0; fin(2)];
31
32 %no gravity tests
33 %fin    = -kphi/(sr+1e-6)*gradphi - k* ( [x_t;y_t]-(-kphi*gradphi) ) ;
34 %yt = [x_t; fin(1);    y_t; fin(2)];
```

distance.m:

```
1  function R = distance(boundary,P)
2  %this function calculates the distance from the boundary of the workspace
3
4  %initialization
5  R = 1;
6  %flags
7  RLp = 0; %right-left flag -1=left,1=right
8  UDp = 0; %up-down flag -1=down,1=up
9
10 for i = 1:size(boundary,2)
11 %     x=boundary(1,i);
12 %     y=boundary(2,i);
13
14     %check relative position
15     if P(1)<boundary(1,i)
16         RL = -1;
17     else
18         RL =1;
19     end
20     %-------------------------------
21     if P(2)<boundary(2,i)
22         UD = -1;
23     else
24         UD =1;
25     end
26     %choose right metric
27     if UD*UDp == -1 || RL*RLp ==-1
```

```
28          if boundary(1,i)-Xp == 0
29              r = abs( P(1)-Xp);
30          elseif (boundary(2,i)-Yp)==0
31              r = abs( P(2)-Yp) ;
32          else
33              %exclude l=inf,0
34              lamda = (boundary(2,i)-Yp)/(boundary(1,i)-Xp);
35              x_common = (P(1)/lamda + P(2) + lamda*Xp -Yp )*lamda/(lamda^2+1)
                    ;
36              y_common = lamda(x_common-Xp)+Yp;
37              r = norm([x_common;y_common]-P);
38          end
39      else
40          r = norm(P-boundary(:,i) );
41      end
42      UDp = UD; RLp =RL;
43      %-------------------------------
44
45      %keep current nearest point
46      if r <R
47          R =r;
48      end
49      %save the position of edge. (at i=0, we wont need, so no need for
50      %initialization outside of loop
51      Xp = boundary(1,i);
52      Yp = boundary(2,i);
53
54  end
```

DOE.m:

```
1   function [Xtest,Ytest] = DOE(polyB,Athresh,in1,in2,smallGapsRad,bigEdgeDist)
2   % This function takes as input the polyshape object of the boundary, an
3   % area threshold to stop the contraction of the workspace, in1, in2 which
4   % are the offset distances (the first one is for finer contraction, and the
5   % second is for coarser contraction), and thresholds for the functions
6   % small_gaps.m and big_gaps.m
7
8   %% for debugging purposes
9   % Athresh = 0.05;
10  % in1 = 0.01;
11  % in2 = 0.1;
12  % polyG  = boundary;
13
14  %% initialization
15  Xtest = [];
16  Ytest = [];
17  PolyOld = polyB; %the old collective polyshape
18  counter = 1;
19
20  %% start plots ------------------------------
21  figure(404)
22  polyB.plot(FaceAlpha=1)
23  hold on
24  leg_text     = {             }; %legend_text
```

```
leg_text(1) = {'Workspace'};
%-------------------------------------------

%% contractions until there is no area left
while (size(PolyOld,2)~=0)

    %the new collective polyshape
    if counter < 4
        polyNew = polybuffer(PolyOld,-in1,'JointType','square');
    else
        polyNew = polybuffer(PolyOld,-in2,'JointType','square');
    end
    PolyOld = [];
    nP      = size(polyNew,1);

    %for plot ----------------------------------------
    polyNew.plot(FaceAlpha=1)
    hold on
    leg_text(counter+1) = {['$Contraction \ n.',num2str(counter),'$']};
    %-------------------------------------------------

    %points of interest
    for iP = 1:nP %for each polyshape
        % maybe contraction will divide the polygon into two polygons

        %take the regions
        nR  = polyNew(iP).NumRegions;
        Reg = polyNew(iP).regions;

        %check if small
        Indeces = Reg.area < Athresh;

        %Select each region in this polyshape (new regions may be
        %created in the current polybuffer process
        for iR=nR:-1:1 %from nr to 1, for each region

            %check if it is a small region and add extra points accordingly
            if Indeces(iR) == 1
                % small regions -> we get only the centroid as a test point
                [x,y] = Reg(iR).centroid;
                Xtest = [Xtest,x];
                Ytest = [Ytest,y];
                Reg(iR) = []; %-> this polyshape region got as contracted as
                    possible
                %we do not need any more points, so we throw it
            else
                % Big regions -> we get all the boundary as tests points
                Xt = Reg(iR).Vertices(:,1);
                Yt = Reg(iR).Vertices(:,2);

                %treat holes seperately
                nan_index = isnan(Xt);          %common for x,y
                nan_loc   = find(nan_index);    %the locations of nans
                nHoles    = sum(nan_index);     %->the number of holes

                %Get Test Points
                for iH = 1:nHoles+1 %nHoles + perimeter
```

```
                    %get temp points depending if the vertices are about a
                    %hole:

                    if iH == 1 %surely iH will be 1 even with no holes
                        if nHoles == 0
                            %if there are no holes, we  do not worry
                            Xtemp = Xt;
                            Ytemp = Yt;
                        else
                            Xtemp = Xt(1:nan_loc(iH)-1);
                            Ytemp = Yt(1:nan_loc(iH)-1);
                        end
                    elseif iH ==nHoles+1
                        Xtemp = Xt(nan_loc(iH-1)+1:end);
                        Ytemp = Yt(nan_loc(iH-1)+1:end);
                    else
                        Xtemp = Xt(nan_loc(iH-1)+1:nan_loc(iH)-1);
                        Ytemp = Yt(nan_loc(iH-1)+1:nan_loc(iH)-1);
                    end

                    %we clear the close
                    [Xtemp,Ytemp] =small_gaps(Xtemp,Ytemp,smallGapsRad,polyB
                        .Vertices);
                    Xtest = [Xtest,Xtemp'];
                    Ytest = [Ytest,Ytemp'];

                    %we also check for big edges
                    [Xtemp,Ytemp] = big_gaps(Xtemp,Ytemp,bigEdgeDist);
                    Xtest = [Xtest,Xtemp'];
                    Ytest = [Ytest,Ytemp'];
                end
            end
        end

        PolyOld = [PolyOld;Reg];
    end

    counter = counter+1;
end


%% plot final
figure(404)

scatter(Xtest,Ytest,100,"black",Marker="x",LineWidth=2)
hold off
title('$ Starting \ Points $',FontSize=20,Interpreter='latex')
xlabel('$X$','Interpreter','latex','FontSize',20)
ylabel('$Y$','Interpreter','latex','FontSize',20)
legend(leg_text,Interpreter="latex",FontSize=15)
```

small_gaps.m:

```
function [Xtest,Ytest] = small_gaps(X,Y,thresh,boundary)
%This functions finds the vertices of the shrinked polygons created by
```

```
%DOE.m that are very close apart and replaces them with an equivalent point

n = length(X);
Xtest = X;
Ytest = Y;

%move each element to the left
Xc = circshift(X,-1);
Yc = circshift(Y,-1);

%calculate distances sequentially
dist = sqrt( (X-Xc).^2 + (Y-Yc).^2);
%indeces vertices that are closer than the specified threshold apart
ind  = dist<thresh;

%take an equivalent point
for i=1:n %check all points
    if ind(i) %if the have smalled distance from the next point
        %the equivalent point takes the position of the first of the pair
        %of close points

        Xtest(i) = X(i);
        Ytest(i) = Yc(i);
        if ~inpolygon(Xtest(i),Ytest(i),boundary(1,:),boundary(2,:))
            Xtest(i) = Xc(i);
            Ytest(i) = Y(i);
        end

    end
end

%to avoid having ind=end, we move all the test points to the left. So the
%indeces which point to the first point of the pair of close points, now,
%AFTER the circshift, point to the second point, which we want to throw out
Xtest = circshift(Xtest,-1);
Ytest = circshift(Ytest,-1);

%clear duplicates
Xtest(ind) =[];
Ytest(ind) =[];
```

big_gaps.m:

```
function [Xtest,Ytest] = big_gaps(X,Y,thresh)
%This function finds the big edges in the shrinked polygons created in the
%DOE.m function, and adds extra points along the edge in order to increase
%the sampling of the workspace

n = length(X);

Xc = circshift(X,-1);
Yc = circshift(Y,-1);

dist = sqrt( (X-Xc).^2 + (Y-Yc).^2);

```

```
Xtest = [];
Ytest = [];
for i=n:-1:1
    if dist(i)>thresh
        %add N=floor(dist/thresh) intermidiate points. So there are N+1
        %spaces
        SM = ceil(dist(i)/thresh);
        dx = Xc(i)-X(i);
        dy = Yc(i)-Y(i);

        %dx-dy for extra points
        DX = [ dx/(SM):dx/(SM):dx*(SM-1)/(SM)]' ;
        DY = [ dy/(SM):dy/(SM):dy*(SM-1)/(SM)]' ;

        if dx == 0
            DX = 0*DY; %DX had one element until that point, so not the same
                dimensions
        elseif dy ==0
            DY = 0*DX;
        end

        Xtest = [ Xtest;( X(i) + DX )]   ;
        Ytest = [ Ytest;( Y(i) + DY )]   ;


    end
end
```

potentialFIELD.m:

```
function [x,y,dx_x,dx_y,dz_x,dz_y,dz_z] =  potentialFIELD(obj)
%this function reads the vector field from AHPF code
%Also, it outputs normalized 3d vectors from plotting the field in 3D. But
%they are not used, as the plot wasn't nice.

NI = 100;
[x,y] = meshgrid( min(obj.workspace.boundary{1}(1,:)):...
    ((max(obj.workspace.boundary{1}(1,:)) - ...
    min(obj.workspace.boundary{1}(1,:)))/NI):...
    max(obj.workspace.boundary{1}(1,:)) ,...
    min(obj.workspace.boundary{1}(2,:)):...
    ((max(obj.workspace.boundary{1}(2,:)) - ...
    min(obj.workspace.boundary{1}(2,:)))/NI):...
    max(obj.workspace.boundary{1}(2,:)));

% scale = 0.8;
% v = ones(size(x));
% w = ones(size(x));


x = x(:);
y = y(:);
%checking for inside the workspace
[in] = isinterior(obj.workspace.workspace_polyshape,x,y);

```

```
x = x(in)';
y = y(in)';
utmp = obj.input_ini_ext([x;y]);
dx =  utmp;

%normalized
normdx = sqrt(dx(1,:).^2 + dx(2,:).^2);
% normu(normu==0) = 1;
dx_x = dx(1,:)./normdx;
dx_y = dx(2,:)./normdx;

%% 3D stuff
% normalized in 3d
% H = @(X,Y) 0.5*sin(0.5*X)+0.3*sin(1.3*Y)+0.4*sin(0.7*(X+Y))+0.2*sin(0.8*X
    +0.9*Y) ;
DH_DX =      0.25*cos(0.5*x) +     0.4*0.7*cos(0.7*(x+y)) +      0.2*0.8*cos
    (0.8*x+0.9*y) ;
DH_DY =      0.3*1.3*cos(1.3*y) +     0.4*0.7*cos(0.7*(x+y)) +      0.2*0.9*cos
    (0.8*x+0.9*y);
gradH = [DH_DX;DH_DY];

%% 3d quivers
vecs = [dx_x;dx_y;sum(gradH.*[dx_x;dx_y])];
normm  = sqrt( sum(vecs.^2) ) ;
temp = (vecs)./normm;

dz_x = temp(1,:);
dz_y = temp(2,:);
dz_z = temp(3,:);
```

reach_targetFULL.m:

```
function [value,isterminal,direction] = reach_targetFULL(t,y,Pd)

P = [y(1);y(3)];

if norm(Pd-P) < 1e-3 && ( norm([y(2);y(4)]) )<1e-3
    value = 0;
else
    value =1;
end
isterminal = 1;
direction= 0;
end
```