The Andrew and Erna Viterbi Faculty of ELECTRICAL & COMPUTER ENGINEERING

Control Robotics and Machine Learning Laboratory

TECHNION Israel Institute of Technology

דו"ח סיכום פרויקט: ב'

תכנון בעיית ריבוי סוכנים פיזיקליים בסימולציה

Designing a multi-agent physical simulation problem

מבצעים:

סער שטרן Saar Stern

ימית פיסמן Yamit Pisman

מנחה: איל טייטלר Ayal Taitler

סמסטר רישום: אביב תשפ"ג

תאריך הגשה: ינואר, 2024

P 7310-2-23

תודות

אנו מודים מקרב לב לאיל טייטלר על העזרה והתמיכה לאורך כל הדרך. הפרויקט היווה לנו ניסיון ראשון בתחום של בעיות תכנון, רובוטיקה ו-AI. למדנו המון מאיל, את שלבי הפיתוח של מערכת תכנון, ועקרונות של התחום, והכי חשוב את דרך המחשבה בעת הופעה של בעיות ופתירתן.

נרצה גם להודות למעבדת CRML וקובי כוחי על תמיכה והכוונה לאורך הדרך.

תוכן עניינים

רשימת איורים

תקציר

תחום התכנון וסימולציה נחוץ במגוון תחומים כגון רחפנים, כלי טיס וכדומה. בעיות אלו כוללות בתוכן תכנון משימות לכל סוכן, שילוב משימות משותפות לסוכנים, סנכרון זמנים ומיקום בין סוכנים שונים. בנוסף לאלו, ישנם אילוצים נוספים שיש להתחשב בהם, כמו אילוצים פיזיקליים של הסוכן או הסביבה.

בפרויקט זה בחרנו להתמקד בבעיית ניווט של משלוח חבילות באמצעות רחפנים, כיוון שכיום זו בעיה נפוצה ותחום שמתפתח בהאצה. הפתרון שלנו הינו סכמת פעולה שבליבתה אלגוריתם הנקרא Scotty, אותו אנו עוטפים בבלוקים שאנחנו כתבנו, ולבסוף מבצעים סימולציה בסביבת Unity.

בבעיה שלנו ישנם שני רחפנים המבצעים איסוף חבילות משני מחסנים שונים, משלוח של חמש חבילות לארבעה בתים שונים עם אילוץ על מסירת חבילות בחלון זמנים ספציפי. בנוסף, בסימולציה שלנו יש התחשבות בזמני האצה, האטה וסיבוב.

Abstract

The realm of planning and simulation is essential across diverse fields, including applications in drones and aircraft. It involves addressing challenges related to individual agent tasks, coordinating tasks among different agents, and achieving temporal and spatial synchronization. Crucial considerations also extend to the physical limitations of agents and the environment, adding layers of complexity to the planning process.

This project specifically focuses on the navigation challenges associated with drone-delivered packages, a rapidly evolving field. Our solution is an operation scheme, at the core of which lies an algorithm called Scotty. We encapsulate Scotty within blocks that we've written, and ultimately perform a simulation in the Unity environment.

In this problem scenario, two drones are assigned the task of picking up packages from distinct warehouses and delivering five packages to four different houses, all within specified time windows. The planning process considers acceleration, deceleration, rotation times, and adheres to delivery constraints. These aspects collectively mirror the real-world challenges inherent in drone-based package delivery systems.

1. תיאור הבעיה

נציג את הרקע התיאורטי הרלוונטי להבנת הבעיה והפתרון שאליו הגענו.

1.1 תכנון טמפורלי

תכנון טמפורלי הוא סוג מסוים של בעיית תכנון בו יש גם יש חשיבות לזמן ביצוע הפעולות ולא רק לסדר שלהן.

כאשר אנחנו חושבים על משימה ממושכת שצריכה להתבצע, יש לה 3 חלקים:

- זמן התחלה
- משך הביצוע
- זמן סיום

תנאים לביצוע פעולה ניתן לבדוק בתחילה, בסוף ותוך כדי הפעולה, כמו כן גם התוצאים (effects) יכולים לקרות בתחילה, סוף או תוך כדי הפעולה (אם התוצא הוא השפעה רציפה). אך סנכרון מושלם אינו אפשרי. כאשר נרצה שפעולות יתבצעו בסנכרון, לאחר תחילת פעולה אחת, פעולה אחרת יכולה להתחיל ϵ זמן אחריה.

ניתן להיעזר בהגדרה של STN – Simple Temporal Network , בה קיימת קבוצה של n משתנים (נקודות זמן) $\{t_i\}^n{}_{i=1}$, וקבוצה של אילוצים. אילוץ הוא בעל המבנה הבא: $< I, t_1, t_2, u >$

כאשר t_1, t_2 הם זמני הסיום וההתחלה בהתאמה, ו- I, u הם מספרים. המשמעות היא: $I \leq t_2 - t_1 \leq u$.

הפתרון עבור STN הוא השמה למשתנים אשר מספקת את האילוצים.. בדיקת תקינות ההשמה עבור *n* נקודות זמן היא בסדר גודל של $O(n^3)$. דרך אחת לפתרון בעיות תכנון טמפורלי היא התעלמות מהזמנים(משכי הזמן), מציאת רצף פעולות באמצעות מתכנן קלאסי ולאחר מכן קביעת לוח הזמנים עבור הפעולות בהתאם למשכי הזמן שלהן. כאשר יש פעולות שצריכות להתבצע במקביל, ניתן לבצע חיפוש בצורת ניסוי וטעיה באמצעות פיצול לפעולות *start* ו-*end* לכל פעולה, ובכל שלב לבדוק את התקינות.

המודל בו אנו נשתמש לתיאור הבעיה הטמפורלית הוא המודל המוצג ב *[1]*:

בעיית התכנון הטמפורלי תוגדר ע"י ה *tuple* הבא: $\langle P, X, A, I, G, C, J \rangle$:

P – סט של פרדיקטים (דגלים) המגדירים את מרחב המצב הבדיד של הבעיה.

X – סט של משתנים ממשיים, המגדירים את מרחב המצב הרציף של הבעיה, בנוסף אליו, גם נשמור את $\dot{X}$ הנגזרות של כל המשתנים הרציפים.

A – סט של פעולות היברידיות, הכוונה היא שלכל פעולה a תוגדר ע"י זמני הסוף וההתחלה שלה, עבור הסוף וההתחלה קיימים תנאים בדידים ורציפים, והתוצא של הפעולה (בסוף ובהתחלה) הם בדידים, אך קיימים אפקטים רציפים שיכולים לפעול לכל אורך הפעולה, וגם שמורה(אילוץ) שצריכה להתקיים לכל אורך הפעולה.

I – מצב התחלה, שהוא השמה למשתנים הבדידים והרציפים.

G – קבוצת מצבי הסיום, זוהי בעצם השמה חלקית למשתנים הבדידים, ואילוצים למשתנים הרציפים שצריכים להתקיים בסוף התוכנית.

C – הוא ה-*tuple* $\langle U, U_C \rangle$, כש- U הוא וקטור משתני הבקרה, ו- U_C הוא האילוצים עליהם.

J – פונקציית המטרה.

הפתרון הוא תוכנית היברידית שכוללת תזמון של כל הפעולות על ציר הזמן, ותיאור המסלול ומשתני הבקרה של המערכת (הגדרה פורמלית במאמר [1].)

תכנית היברידית תקינה היא תוכנית שמגיעה ממצב התחלה לקבוצת מצבי הסיום, ומקיימת את כל האילוצים על משתני הבקרה ומשתני מרחב המצב.

מודל זה ממומש ע"י שפת *PDDL* (עם הרחבות מסוימות שנוכל לפרט עליהן בהמשך)

1.2 שפת PDDL

PDDL – Planning Domain Definition Language – שפת תכנות קלאסית לתכנון משימות, שהיא בעצם גם מודל עבור הבעיה.

מרכיבי בעיית ה*PDDL*:

- *Objects* – אובייקטים המעניינים אותנו.
- *Predicates* – מאפיינים של אובייקטים המעניינים אותנו, יכולים להיות במצב *true* או *false*.
- *Functions* – פונקציות רציפות של העולם.
- *Initial state* – המצב ההתחלתי של העולם בו אנו נמצאים.
- *Goal specification* – דברים שנרצה שיהיו במצב *true*.
- *Action / operators* – פעולות שדרכן ניתן לשנות את העולם.

ב*PDDL* הבעיה מוגדרת באמצעות 2 קבצים:

- קובץ *Domain* עבור הפעולות וה-*Predicates*.
- קובץ *Problem* עבור האובייקטים, מצבים התחלתיים, ומטרות הבעיה.

בפרט נפרט כעת על *Action* אשר כוללת:

- משך הפעולה, ע"י גבול עליון ותחתון למשך הפעולה וכך ה *Planner* יכול לקבוע כמה זמן תימשך הפעולה.
- תנאים לפעולה, חלק מהתנאים צריכים להתקיים בתחילת הפעולה (*at start*), חלק בסוף הפעולה *(at end)* וחלק במשך כל הפעולה *(over all)*.

- השפעות/תוצאות הפעולה, גם כאן הן יכולות להיות בתחילה וסוף הפעולה, וגם לאורך כל הפעולה *(increase, decrease)*.

ניתן דוגמה לפעולה שמכילה את כל אלו:

```
(:durative-action drone1-deliver-house1-package4-at-slot1
    :duration (and (>= ?duration 0.1) (<= ?duration 200))
    :condition (and
        (at start  (drone1-delivery-ongoing))
        (at end  (drone1-delivery-ongoing))
        (at start (not (drone1-busy)))
        (at end   (drone1-busy))
        (at end   (inside (house1-region (drone1-x) (drone1-y) )))
        (at start (drone1-has-package4-at-slot1))
        (at start (house1-needs-package4))
        (over all (>= (drone1-battery) 0))
    )
    :effect (and
        (at start (drone1-busy))
        (at end (not (drone1-busy)))
        (at end (not (drone1-has-package4-at-slot1)))
        (at end (not (drone1-slot1-full)))
        (at end (not (house1-needs-package4)))
        (at end (house1-got-package4))
        (increase (drone1-x) (* (vx-drone1) #t))
        (increase (drone1-y) (* (vy-drone1) #t))
        (decrease (drone1-battery) (* 1.0 (norm (energy-drone1)) #t))
    )
)
```

1.3 בעיית תכנון כחיפוש בגרף

על בעיית התכנון נהוג להתבונן כבעיית חיפוש בגרף, כשכל נקודה במרחב המצב הינה קודקוד בגרף, פעולה הינה קשת המחברת בין 2 קודקודים לפי פונקציית המעבר f, או במקרה של SAS^+ הקשת תחבר למצב שהינו המצב המקורי בנוסף לאפקטים, כמובן שניתן לעבור על הקשת (כלומר לבצע את הפעולה) רק אם עומדים בתנאי ההתחלה. לכל קשת יש גם את מחיר הפעולה (משקל הקשת)

מצב ההתחלה הינו גם כן קודקוד בגרף, ומצבי הסוף הינם אוסף של מצבים בגרף.

כעת בעיית התכנון הפכה לבעיה "פשוטה" של חיפוש מסלול בגרף לה יש כמובן אלגוריתמים סגורים לפתרון כמו אלגוריתם דייקסטרה, אך אלגוריתם זה פועל ב $\mathcal{O}(|V|\log|V| + |E|)$, דבר שלא ישים עבור רוב הבעיות שנרצה לפתור בהן מרחב המצב הינו עצום.

לכן, פותחו אלגוריתמים מהירים יותר המשתמשים ביוריסטיקות (לדוגמה A^*).

בבואנו לפתור בעיה מורכבת הכוללת גם אילוצי זמן, דרושה טכניקה אחרת מרק חיפוש בגרף.

1.4 ScottyActivity planner (מכאן והלאה Scotty)

Scotty הינו Planner זמני בדיד – רציף מתקדם, הוא ממנף את הביצועים המוכחים של heuristic forward search עבור התכנון, והוא מסוגל להתמודד עם אילוצים מורכבים של רובוטים הכוללים דינמיקה לא-לינארית. בעצם זהו אלגוריתם היודע להתמודד עם בעיות מהסוג שאנו באים לפתור בפרויקט זה.

scotty עובד ב2 שלבים, בשלב הראשון ימצא פתרון לפי אלגוריתם חיפוש בגרף, שאינו מתחשב בפונקציות ומשתנים רציפים. לאחר מכן תיפתר בעיית אופטימיזציה שתתחשב במשתנים רציפים אלו.

Scotty תומך בשני אלגוריתמי חיפוש בגרף. הראשון EHC – Enforced Hill-Climbing, שנעשה בו שימוש נרחב במתכננים. EHC הינו אלגוריתם חמדן, לכן לא בהכרח יספק את הפתרון האופטימלי, ואף יתכן שלא ימצא פתרון כלל למרות שקיים אחד כזה. האלגוריתם השני, A* נפוץ מאוד ב- Planners, במידה וקיים פתרון הוא ימצא אותו, ואף ימצא את הפתרון הטוב ביותר, אך חסרונו הוא שאלגוריתם זה איטי הרבה יותר. בפרויקט שלנו, רק אלגוריתם החיפוש בגרף A^* הצליח להתמודד עם האילוצים הזמניים שהגדרנו.

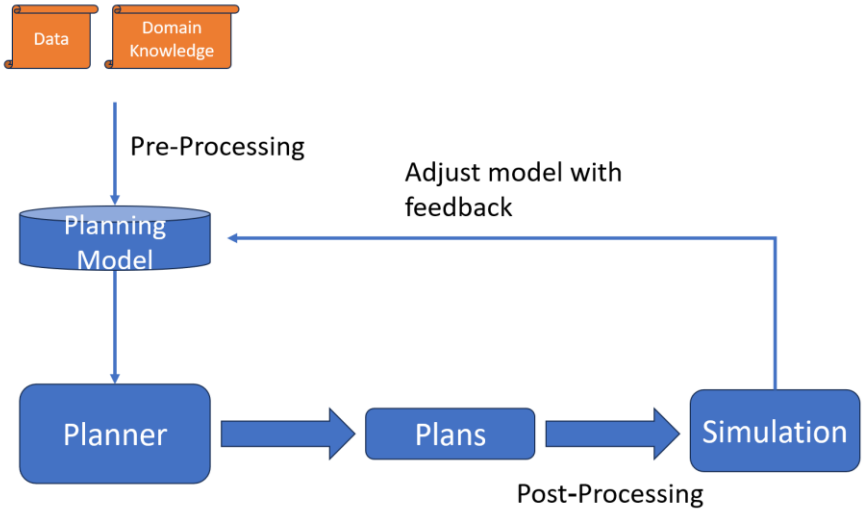
הבעיה ב- scotty היא שלהכניס אילוצים פיסיקליים כמו תאוצה יכול לסבך מאד את הגדרת הבעיה ולהאט את מציאת הפתרון, ולכן ננהג בגישה אחרת, בה נקבל מ scotty את התוכנית "הכללית" של היכן כל אובייקט צריך להיות ובאיזו נקודת זמן, ונדאג לשאר באופן חיצוני.

הערה: Gurobi Optimizer היא חבילת תכונה לאופטימיזציה מתמטית ש-Scotty משתמש בה. הגרסה שתומכת ב-Scotty הינה Gurobi Optimizer 7.

1.5 סכמת פתרון לבעיה תכנון פיזיקלית

כפי שהזכרנו, בבואנו לפתור בעיה פיזיקלית לעיתים נזניח את הפרטים ה"קטנים" ונתמקד בתכנון השלבים הכלליים של הבעיה. לדוגמה בבעיה של רחפן שצריך לדוור משלוחים תוך הרבה אילוצי זמן ומקום, נרצה לקבל מה- planner רק את המיקום שהרחפן צריך להיות בו בכל נק' זמן חשובה, ואיזו פעולה הוא אמור לבצע כמו "אספת חבילה", "חלוקת חבילה", "טעינה\תדלוק אווירי", "אספת חבילה עם רחפן אחר". לאחר מכן הטייס/בקר/תוכנת מחשב תבצע את הצעדים ההכרחיים (תאוצה ובקרה) להשגת המטרות שה Planner קבע. לעיתים לא ניתן לבצע באופן פיזיקלי את התוכנית שה- planner פלט, ואז נדרש משוב בו משנים את מודל הבעיה (דרישות פחות מחמירות) כדי להבטיח שניתן יהיה לממש את הפתרון באופן פיזיקלי.

ניתן לראות את סכמת הפעולה הכללית באיור הבא:



איור 1. סכמת הפעולה הכללית לפתרון בעיית תכנון פיזיקלית.

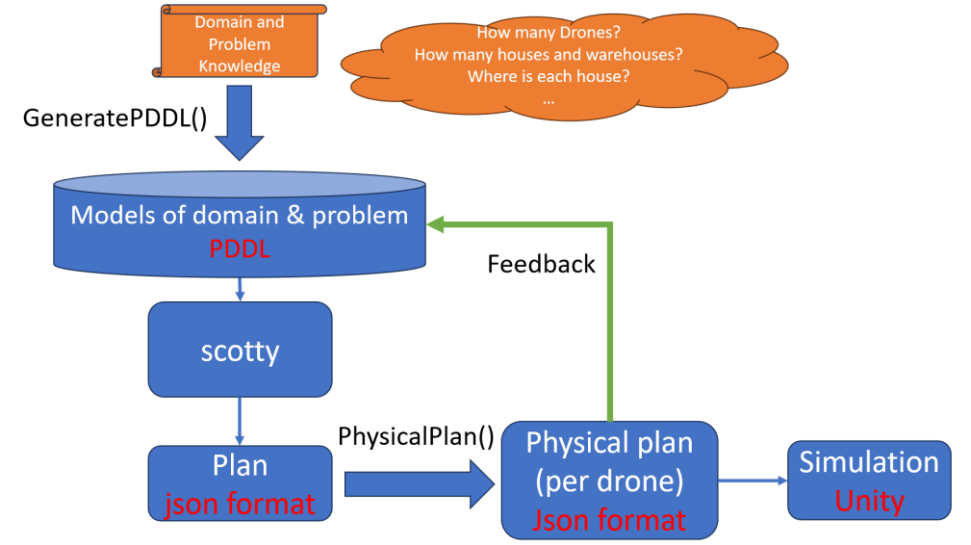
1.6 UNITY

UNITY הוא מנוע גרפי חוצה פלטפורמות המשמש לפיתוח משחקי וידיאו למחשב, קונסולות, טלפונים חכמים ואתרי אינטרנט.

בחרנו בסביבה זו להרצת הסימולציה שלנו כיוון שהיא נוחה לעבודה ולביצוע שינויים, וגם ניתן ללמוד אותה במהירות. השתמשנו בגרסה 2022.3.9f1.

2. השיטה

ראשית, נציג את המימוש שלנו עבור סכמת הפתרון:



איור 2.סכמת הפעולה הספציפית לפתרון בעיית תכנון פיזיקלית.

כעת נפרט על כל חלק בה:

2.1 הבעיה

הבעיה שבחרנו היא בעיית משלוח חבילות ממחסנים לבתים ע"י רחפנים.

ניתן לבחור:

- כמה מחסנים ובתים יהיו, והיכן ימוקמו.

- באיזה מחסן כל חבילה נמצאת, ולאיזה בית היא מיועדת.

- את כמות הרחפנים ומיקומם ההתחלתי והסופי, את המהירות והתאוצה המקסימלית שלהם, ואת הסוללה ההתחלתית שלהם, ומנגנון הבזבוז שלה, וכמה חבילות רחפן יכול להרים ביחד.

- קטעי זמן שבהם בית מסוים יכול לקבל משלוחים.

2.2 מידול ע"י קבצי PDDL

בשביל המעבר מהבעיה לקבצי PDDL כתבנו את הפונקציה GeneratePDDL ב Matlab, הפונקציה מקבלת 2 קבצי קונפיגורציה עבור הdomain ועבור הProblem, וכך ניתן לשמור כמות גדולה של קונפיגורציות שונות.

בפונקציה עצמה, נעשה שימוש חוזר, שכפול ושינוי של קבצי טקסט בהם מוגדרים הפרדיקטים הבסיסיים, הפעולות הבסיסיות והפונקציות הבסיסיות של התוכנית. כך, נוצרים שני קבצי PDDL, קובץ Problem וקובץ Domain.

בעצם, שלב זה מחליף את השימוש בפרמטרים בשפת PDDL. כאשר ישנם מספר אובייקטים מאותו הסוג, ניתן לכתוב את הפעולות בצורה גנרית עם פרמטרים. לאחר מכן בעצם הפעולות משוכפלות לפי כמות האובייקטים. במקרה שלנו זה לא עבד ו- Scotty לא הצליח להתמודד עם הקבצים האלו, ולכן כתבנו סקריפט שמבצע את השכפולים והשינויים בעצמו.

כעת נסביר ביתר פירוט איך נבנו קבצי הPDDL:

<u>Problem</u>

- Init - בחלק זה מגדירים את כל הפרדיקטים שיקבלו '1', ואת ערכי ההתחלה של הפונקציות.

  `(first-time)` – דגל התחלה לתוכנית
  `(= (drone1-x) 640.0), (= (drone1-y) 640.0)` – ערכי מיקום התחלתי עבור הרחפן
  `(= (drone1-battery) 4200)` – ערך סוללה התחלתי עבור הרחפן.
  `(house1-needs-package1)` – מגדירים כך איזה בית צריך איזה חבילה
  `(package1-at-warehouse1)` – מגדירים כך באיזה מחסן תהיה כל חבילה.

- Goal – בחלק זה מגדירים את כל הפרדיקטים שמגדירים את המצב הסופי של התוכנית

  `(drone1-complete-delivery)` – דרישה לבצע בסוף פעולת סיום לאחריה הרחפן לא יוכל יותר לעוף.
  `(house1-got-package1)` – דרישה שחבילה מסוימת הגיעה ליעדה.

- Metric – בחלק זה מגדירים את המטריקות עליהם תתבצע אופטימיזציה.

```
(:metric minimize (+
                    (* -0.1 (drone1-battery))
                    (* -0.1 (drone2-battery))
                    (* 20 (total-time) )
                    )
)
```

עוד על המטריקה בנספח.

<u>Domain</u>

- Predicates – כלל הדגלים שישמשו אותנו לבדיקת תנאי התחלה וסיום של הפעולות

  `(first-time)` – דגל התחלה לתוכנית
  `(TL0start), (TL1start), (TL0), (TL1)` – דגלים עבור אינטרוול הזמן, המשך בנספח.

(drone1-delivery-ongoing) – דגל המציין כי הרחפן התחיל את תוכנית המשלוחים שלו.
(drone1-complete-delivery) – דגל המציין כי הרחפן סיים את תוכנית המשלוחים שלו.
(drone1-busy) – דגל המציין כי הרחפן עסוק, כלומר באמצע action.
(drone1-slot1-full) – דגל המציין כי slot מסוים של הרחפן תפוס.
(drone1-has-package1-at-slot1) – דגל המציין כי לרחפן מסוים יש חבילה מסוימת בslot מסוים.

(package1-at-warehouse1) – דגל המציין אם חבילה מסוימת נמצאת במחסן מסוים.

(house1-needs-package1) – דגל המציין איזה חבילה מיועדת לאיזה בית.
(house1-got-package1) – דגל המראה שבית מסוים קיבל בהצלחה חבילה מסוימת.

- Functions – משתנים שיתקבלו כפונקציה של הcontrol-variables, ושל התוכנית עצמה (סדר פעולות, זמנים).

(drone1-x), (drone1-y) – מיקום של הרחפן.
(drone1-battery) – כמות סוללה שנשארה לרחפן.

- Scotty – Control variable בוחר שאת הערכים של הפרמטרים, אנחנו מספקים לו טווח ערכים רצוי, ניתן לראות למטה כי הגדרנו מהירות בציר x ו-y. וגם את האנרגיה כלומר הנורמל של המהירות, כולם עם הגבלה ל-13. (שרירותי לדוגמה זו)

```
(:control-variable vx-drone1
    :bounds (and (>= ?value -13.0) (<= ?value 13.0))
)
(:control-variable vy-drone1
    :bounds (and (>= ?value -13.0) (<= ?value 13.0))
)
(:control-variable-vector energy-drone1
    :control-variables ((vx-drone1) (vy-drone1))
    :max-norm 13
)
```

- Region – הגדרת אזורים, מכילים את המיקום והגודל שלהם, ניתן פה לראות את הגדרת המפה כמרובע עם קצה שמאלי-תחתון ב(0,0), גובה 1280 ורוחב 1280 . בצורה זו גם הגדרנו את הבתים, המחסנים, והמנחת של הרחפנים.

```
(:region earth
    :parameters (?x ?y)
    :condition (and (in-rect (?x ?y) :corner (0 0) :width 1280 :height 1280))
)
```

- Durative-actions - כלל הפעולות הזמניות שיש לנו בתוכנית, Scotty צריך לבחור את הסדר שלהן, ולבצע אופטימיזציה על אורכן.

להלן הפעולה הראשית של התחלת המשלוחים, מתחילה רק פעם אחת, דואגת להפעיל את חלונות הזמנים.

```
; start the delivery, this action starts the plan
(:durative-action start-delivery
    :duration (and (>= ?duration 0.1) (<= ?duration 0.1))
    :condition (and
        (at start (first-time))
        (at end (drone1-busy))
```

```
            (at end (drone2-busy))
            (at end (not (TL0start)))
            (at end (not (TL1start)))
        )
        :effect (and
            (at start (not (first-time)))
            (at start (TL0start))
            (at start (TL1start))
        )
    )
```

דוגמא לפעולת משלוח חבילה, בדוגמא זו אנחנו מציגים את משלוח החבילה שמוגבל להתבצע בחלון זמן. זו פעולה של משלוח חבילה 5 לבית 3. על מנת שהפעולה תתחיל, הרחפן צריך להחזיק בחבילה 5, ובית 3 צריך את חבילה 5. בנוסף, ניתן לראות את בדיקת התנאים של החלונות (חלון אחד "דולק", ואחד "מכובה"). לאורך כל הפעולה הסוללה מתבזבזת וישנו אילוץ שבסוף הפעולה הרחפן צריך להיות באזור הבית.

```
    (:durative-action drone1-deliver-house3-package5-at-slot1
        :duration (and (>= ?duration 0.1) (<= ?duration 200))
        :condition (and
            (at start  (drone1-delivery-ongoing))
            (at end  (drone1-delivery-ongoing))
            (at start (not (drone1-busy)))
            (at end   (drone1-busy))
            (at end (TL0))
            (at end (not (TL1)))
            (at end   (inside (house3-region (drone1-x) (drone1-y) )))
            (at start (drone1-has-package5-at-slot1))
            (at start (house3-needs-package5))
            (over all (>= (drone1-battery) 0))
        )
        :effect (and
            (at start (drone1-busy))
            (at end (not (drone1-busy)))
            (at end (not (drone1-has-package5-at-slot1)))
            (at end (not (drone1-slot1-full)))
            (at end (not (house3-needs-package5)))
            (at end (house3-got-package5))
            (increase (drone1-x) (* (vx-drone1) #t))
            (increase (drone1-y) (* (vy-drone1) #t))
            (decrease (drone1-battery) (* 1.0 (norm (energy-drone1)) #t))
        )
    )
```

דוגמא לפעולת איסוף חבילה. ניתן לראות שלאורך כל הפעולה הסוללה מתבזבזת וישנו אילוץ שבסוף הפעולה הרחפן צריך להיות באזור המחסן.

```
    (:durative-action drone1-pickup-package1-from-warehouse1-into-slot1
        :duration (and (>= ?duration 0.1) (<= ?duration 200))
        :condition (and
            (at start  (drone1-delivery-ongoing))
            (at end  (drone1-delivery-ongoing))
            (at start  (not (drone1-busy)))
            (at end    (drone1-busy))
            (at end    (inside (warehouse1-region (drone1-x) (drone1-y) )))
            (at start  (package1-at-warehouse1))
            (at start  (not (drone1-slot1-full)))
            (over all (>= (drone1-battery) 0))
        )
        :effect (and
            (at start  (drone1-busy))
```

```
            (at end    (not (drone1-busy)))
            (at end    (drone1-has-package1-at-slot1))
            (at end    (not(package1-at-warehouse1)))
            (at end    (drone1-slot1-full))
            (increase (drone1-x) (* (vx-drone1) #t))
            (increase (drone1-y) (* (vy-drone1) #t))
            (decrease (drone1-battery) (* 1.0 (norm (energy-drone1)) #t))
        )
    )
```

דוגמא לפעולת סיום של רחפן. ניתן לראות שלאורך כל הפעולה הסוללה מתבזבזת וישנו אילוץ שבסוף הפעולה הרחפן צריך להיות באזור מנחת הרחפנים.

```
    (:durative-action drone1-finish-delivery
        :duration (and (>= ?duration 0.1) (<= ?duration 200))
        :condition (and
            (at end (inside (helipad-region (drone1-x) (drone1-y))))
            (at start  (drone1-delivery-ongoing))
            (over all (>= (drone1-battery) 0))
        )
        :effect (and
            (at start (not (drone1-delivery-ongoing)))
            (at end (drone1-complete-delivery))
            (increase (drone1-x) (* (vx-drone1) #t))
            (increase (drone1-y) (* (vy-drone1) #t))
            (decrease (drone1-battery) (* 1.0 (norm (energy-drone1)) #t))
        )
    )
```

דוגמא ל2 פעולות המייצגות חלונות זמן. בזמן הביצוע, מורם הדגל TL0\TL1 המייצג שחלון הזמנים פעיל. בעצם, timed Literal0 הינו חלון הזמנים הגדול, ו-timed literal1 הינו חלון הזמנים הקטן. ניתן לראות שהקטן מתחיל מיד לאחר הגדול. החלון הגדול אורכו 180 שניות, החלון הקטן אורכו 150, וכך אנו מקבלים חלון באורך 30 שניות בין 150 ל180 שניות.

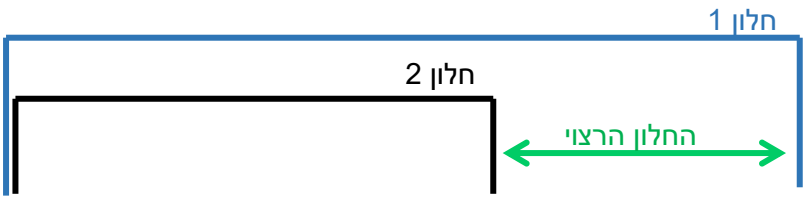
```
    (:durative-action timedliteral0
        :duration (and (>= ?duration 180) (<= ?duration 180))
        :condition (and
                (at start (TL0start))
                (at start (not (TL1start)))
        )

        :effect (and
            (at start (drone1-delivery-ongoing))
            (at start (drone2-delivery-ongoing))
            (at start (not (TL0start)))
            (at start (TL0))
            (at end (not (TL0)))
        )
    )

    (:durative-action timedliteral1
        :duration (and (>= ?duration 150) (<= ?duration 150))
        :condition (and
                (at start (TL1start))
        )
        :effect (and
            (at start (not (TL1start)))
            (at start (TL1))
            (at end (not (TL1)))
        )
    )
```

הסבר מרחיב על חלונות הזמן:

בגרסת PDDL 2.2 ישנה אפשרות להתנות דברים בזמן מסוים לדוגמא ((at time(predicate)).

לצערנו, Scotty אינו תומך בגרסה זו, והיינו צריכים להשתמש בשיטות אחרות ע"מ לבצע פעולות בחלון זמנים שנרצה.

בשיטה בה השתמשנו, בעצם מתבצע שימוש בשני חלונות זמן שמתחילים האחד אחרי השני. חלון זמן 1 הינו ארוך, וחלון זמן 2 קצר יותר, ונמצא בתוכו. וככה אנחנו יכולים להגדיר את חלון הזמן ששניהם יוצרים יחד:



איור 3. דיאגרמת אילוץ זמני במנגנון TIL חלקי

כך, חלון 1 מתחיל עם תחילת התוכנית, חלון 2 מתחיל ϵ לאחר מכן, שניהם מדליקים *predicates* בזמן שהם פועלים. ועל הפעולה שאותה נרצה לבצע בחלון זמנים מסוים, נרצה שבתנאי ההתחלה חלון 1 עם *predicate* דולק, ואילו חלון 2 עם *predicate* מכובה.

2.3 הרצה עם Scotty

הרצת Scotty עם קבצי ה-Problem וה-Domain שנוצרו, תוך בחירת אלגוריתם הפתרון הרצוי, לקבלת קובץ json המתאר את כל משתני המערכת בכל time stamp שScotty מייצר.

משתני המערכת הם:

- מיקום וכמה סוללה נשארה לכל רחפן.
- מהירות הרחפן.

הערה: נשים לב כי הפתרון ש- Scotty מייצר אינו פיזיקלי, והמעבר בין מהירויות שונות הוא מידי.

2.4 בדיקת היתכנות פיזיקלית ותכנון המסלול הפיזיקלי

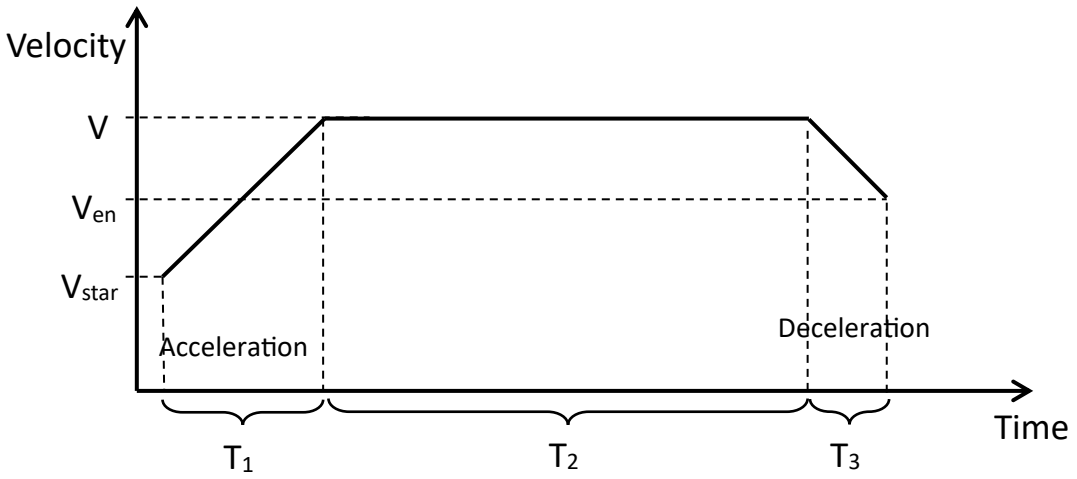
לאחר קבלת קובץ הJSON שScotty יצר, המטרה שלנו היא לבדוק היתכנות פיזיקלית שמתחשבת בצורה טובה יותר בתנועת הרחפנים, פירוק מהירות הרחפנים לפי הצירים עבור תנועה חלקה יותר, והתחשבות בתאוצה והאטה שלהם. במידה וקיימת היתכנות, ליצור קובץ JSON חדש המתחשב בנ"ל עבור כל אחד מהרחפנים. לשם כך בנינו את הפונקציה PhysicalPlan().

באמצעות קובץ הJSON שנפלט מScotty קיבלנו את תגיות הזמן והמיקום עבור כל פעולה של כל אחד מהרחפנים $\{x_i, y_i, t_i\}_{i=1}^{N}$. פתרון לאילוצים האלה על הרחפן הוא וקטור תאוצה $\boldsymbol{a}(t)$.

מכיוון שקיימים לעיתים אינסוף פתרונות כאלה, בחרנו להכניס עוד אילוץ על פונקציית התאוצה, והוא שבכל מקטע שהרחפן עובר בין מחסן\בית וכו' , התאוצה תחולק (בכל ציר בנפרד) ל-3 חלקים:

- הראשון – האצה / האטה
- השני – תנועה במהירות קבועה V
- השלישי – האצה / האטה

כפי שניתן לראות באיור הבא:



איור 4. גרף המתאר את ההתחשבות שביצענו בהאצות והאטות

לאחר מכן נפתור משוואות תנועה פשוטות תוך אילוצים על הזמן והמיקום, וקיבלנו עבור כל מקטע של פעולה, את המהירות הרלוונטית עבורו לכל אחד מהצירים ותוך התחשבות בהאצה והאטה.

המשוואות שפתרנו בציר x : (זהות עבור ציר y)

$$T_1 = \frac{|V - V_{start}|}{a}, \qquad T_2 = \frac{|V - V_{end}|}{a}, \qquad T_3 = Total\ time - T_1 - T_2$$

מכאן:

$$X_1 = \int_0^{T_1} V_{start} + a \cdot sign(V - V_{start}) \cdot t \ \ dt$$

$$X_2 = T_3 \cdot V$$

$$X_3 = \int_0^{T_3} V + a \cdot sign(V_{end} - V) \cdot t \ \ dt$$

$$Total\ distance(V) = (X_1 + X_2 + X_3)(V)$$

פתרון המשוואה עבור V מתבצע ע"י *Matlab*, פתרונות שעבורם $T_3 < 0$ כמובן נפסלו.

הפתרון מתבצע ב2 איטרציות, באיטרציה הראשונה כל תנאי ההתחלה והסוף $V_{end}, V_{start} = 0$.

באיטרציה השנייה:

$$V_{end}(i) = 0.8 * V(i+1)$$

$$V_{start}(i+1) = V_{end}(i)$$

$$V_{start}(1) = 0, V_{end}(N) = 0$$

בחרנו בתנאי התחלה אלה כדי לקבל תנועה יותר חלקה בלי עצירות מוחלטות של הרחפן.

לבסוף, ביצענו בדיקה האם הפתרון החדש הינו אמיתי, כלומר האם המהירות המקסימלית שהתקבלה אפשרית (על פי מה שהוגדר מראש).

במידה וכן, נוצרו כעת קבצי JSON שניתן להשתמש בהם עבור השלב הבא, הקובץ מחולק לפי הפעולות שהרחפן נדרש לעשות, הנה דוגמה לפעולה הראשונה (אינדקס 0) שרחפן מס' 1 נדרש לעשות:

Field	Value
index	0
speedX	-2.3505
speedY	7.7604
direction	[-0.3041,0.9527]
accX	-4.7500
decX	4.7500
accY	4.7500
decY	-4.7500
start_time	0.0300
accX_time	0.5248
decX_time	67.1084
accY_time	1.6638
decY_time	64.2798
end_time	67.3681
step_action	'DRONE1-PICKUP-PACKAGE5-FROM-WAREHOUSE1-INTO-SLOT1'

איור 5. התכנית הפיזיקלית שנוצרה ע"י PhysicalPlan(): צעד מס' 1 עבור רחפן 1.

פירוט המשתנים:

speedX, speedY מציינים את המהירויות הקבועות שעל הרחפן להגיע אליהן במקטע T2.

direction – זה וקטור הכיוון בין נק' ההתחלה לנק' הסוף, לצורך דיבאג.

accX, accY – ערכי התאוצה\תאוטה במקטע T1.

decX, decY – ערכי התאוצה\תאוטה במקטע T3.

Start_time – זמן ההתחלה של כל המקטע.

AccX_time\AccY_time – זמני סוף מקטע T1 (לכל ציר זמנים אחרים)

DecX_time\DecY_time – זמני התחלת מקטע T3 (לכל ציר זמנים אחרים)

End_time – זמן הסוף של כל המקטע.

Step_action – תיאור הפעולה.

2.5 פידבק

אם לא הייתה היתכנות פיזיקלית לתוכנית ש- Scotty פלט (אם הרחפן נדרש להגיע למהירות שהיא מעל המהירות המקסימלית של הרחפן), ניתן לחזור אל קבצי ה-pddl ולשנות את הדרישות בהם כך שיהיו שונות מהדרישות הפיזיקליות.

לדוגמה: לדרוש מהירות מקסימלית נמוכה מהמהירות המקסימלית המקורית, ואז בתוכנית הפיזיקלית שכוללת תאוצה, עדיין לא נחרוג מהמהירות המקסימלית המקורית).

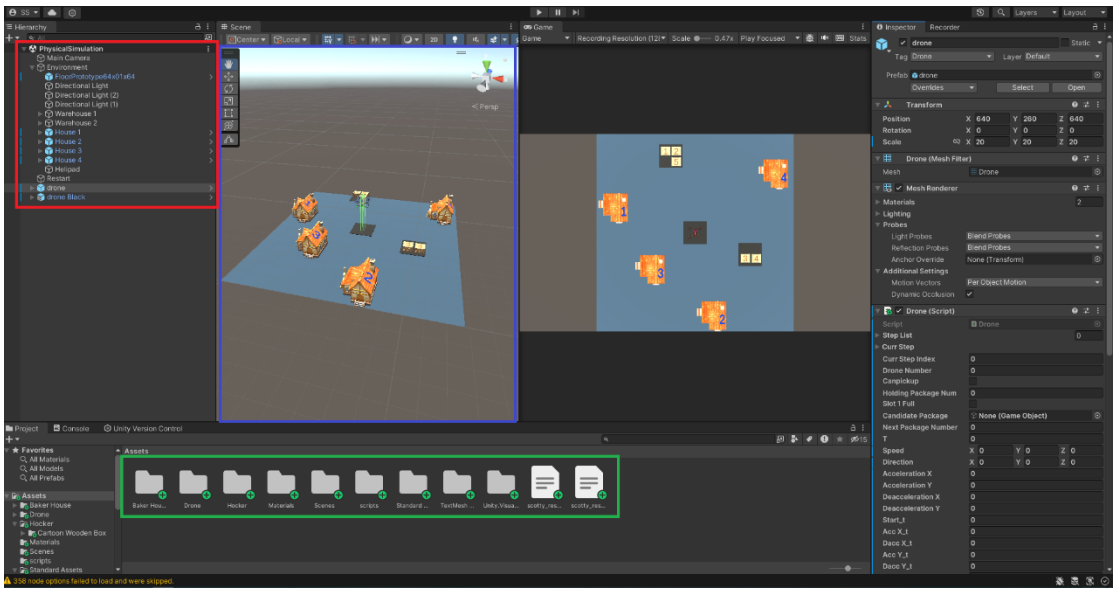
2.6 קיצוץ זמנים מתים (אופציונאלי)

לעיתים, התכנית שנקבל כוללת זמנים מתים, זה קורה בגלל חוסר-סימטריה בין הרחפנים: אחד הרחפנים לרוב עובד ללא הפסקה ולוקח לדוגמה 3 חבילות, ולרחפן השני נותרו רק 2 חבילות לקחת, והוא יכול "לנוח" בין המקטעים או לסיים מוקדם יותר (בהנחה שאין לו אילוצי זמן במקטעים עצמם, או שהוא סיים עם אילוצי זמן אלו). ניתן לנסות לקבוע מטריקה שאולי תמזער את זמנים אלו (ראו נספח) אך המחיר על כך יכול להיות זמן ריצה מאד ארוך, או אף גלישת זיכרון בעת ניסיון הפתרון, לכן הוספנו דגל לפונקציה PhysicalPlan() המאפשר קטיעה של זמנים מתים אלו, כמובן שהסקריפט בעצם משנה את סדר הפעולות בין הרחפנים, ויכול גם להרוס אילוצי זמן שהתקיימו לפני כן, ולכן דרושה וריפיקציה של התוכנית, לא כתבנו סקריפט וריפיקציה וביצענו אותה ידנית. נדגיש כי סקריפט זה אופציונלי וגם התוכנית המקורית עומדת בכל הדרישות.

2.7 סימולציה

הסימולציה שמומשה בUnity תשתמש בקבצי התוכנית הפיזיקלית שנוצרו כדי לבצע את תנועת הרחפנים וכך נוכל לקבל ויזואליזציה טובה של התוכנית.

מצורף צילום מסך של סביבת העבודה שבנינו:



איור 6. סביבת העבודה בUNITY

בכחול – זהו המסך שבו רואים את הסצנה.

באדום – כאן ניתן את שם הסצנה PhysicalSimulation ואת האובייקטים שבהם השתמשנו:

- מצלמה ראשית
- Environment הכולל בתוכו את: הרצפה, התאורות, המחסנים (שתחתם יש את החבילות), את הבתים ואת המנחת.
- מקש Restart – המקש R משמש לאתחול מחדש של הסימולציה.
- Drone – זהו הרחפן הראשון
- Drone black- זהו הרחפן השני.

בירוק – כאן ניתן לראות את הקבצים שבהם השתמשנו: חוץ מהקבצים הסטנדרטים יש:

- תיקיית Drone הכוללת את האובייקטים של הרחפנים (הורדנו מunity asset store בחינם)
- תיקיית Baker house הכוללת את האובייקט של הבית (הורדנו מunity asset store בחינם)
- קבצי התוכנית הפיזיקלית לכל רחפן
- תיקיית scripts בה אנו שומרים את הסקריפטים בהם אנו משתמשים
  - JsonReader – סקריפט לקריאת התוכנית הפיזיקלית שמגיעה בפורמט Json, והפיכתה למשתנים או מערכי משתנים בשפת #C, תחת האובייקטים של הרחפנים יש אובייקט בשם JsonReader גם כן, שמשתמש בסקריפט זה (בנפרד לכל רחפן)

- Restart – הסקריפט של מקש האתחול R.
- Drone – סקריפט של פעילות הרחפן, בסקריפט זה ישנן 2 "מערכות"
  - מערכת התזוזה – בכל פעם שהזמן מתחילת הסימולציה עבר את זמן הסיום של הפעולה הנוכחית, הרחפן יעבור לבצע את הפעולה הבאה, לפי התוכנית הפיזיקלית.
  - מערכת הpickup and delivery – כשהרחפן נמצא מעל מחסן, הוא מחפש במחסן את החבילה הבאה שאותה הוא צריך, אם החבילה שם, הרחפן יאסוף אותה, ואז יוריד בבית הבא שיגיע אליו.

זאת הנקודה לציין כי ככל הנראה קיימים עוד אינספור שיטות להגיע לפתרון עבור בעיה זו, גם תוך שימוש באותם הכלים. פירוט נוסף על כיוונים בהם בחרנו להשתמש הן בשפת PDDL, המטריקה או פעולות אחרות, ניתן לקרוא בהרחבה בנספח.

2.8 תהליך הפיתוח – אפשרויות שנבדקו

בעת תהליך הפיתוח, בנינו את הפרויקט בצורה הדרגתית, בהתחלה פותח תרחיש קל ביותר של רחפן, חבילה, בית מחסן ופותחה סביבת הסימולציה ב Unity. לאחר מכן, יכולנו לשנות בכל פעם את קבצי ה-pddl, ולראות ויזואלית את השינויים, או את סטטיסטיקות הפתרון של scotty (זמן הפתרון, כמות הבעיות שנפתרו בדרך) והרבה פעמים בעצם קריסה של scotty עקב מחסור בזיכרון.

האפשרויות שנבדקו הם:

1) וריאציות של הבעיה

א. רחפן 1, חבילה 1, מחסן 1 ובית 1 – כאן התחלנו בעצם מהשלב הבסיסי ביותר בו יהיו לנו את כל היחידות הבסיסיות לעבוד איתן. בשלב זה בעיקר למדנו לכתוב קבצי PDDL, כיצד להריץ את הScotty, למדנו להשתמש בUnity ויצרנו את סביבת הסימולציה הראשונה.

ב. 2 רחפנים, 2 חבילות, 2 מחסנים ו-2 בתים – לאחר השלב הראשון, רצינו כמובן שהבעיה תהיה יותר מסובכת. בשלב זה הוספנו רחפן, חבילה ובית. יכולנו להבחין שלעומת השלב הקודם – זמן הריצה של Scotty ארוך יותר, וכמובן שהדבר היה הגיוני.

ג. 2 רחפנים, 4 חבילות, 2 מחסנים ו-4 בתים – גם כאן, לאחר השלב הקודם, רצינו להמשיך ולסבך את הבעיה. יכולנו להבחין שהזמן שלוקח לScotty לפתור אפילו בעיה כזו שנראית פשוטה, עולה בצורה משמעותית.

בעצם, בכל שלב במהלך בדיקת הפתרון, נבחרת פעולה לביצוע מתוך סט הפעולות הקיימות בקובץ הPDDL, וככל שיש לנו יותר רכיבים, כמות הפעולות האפשרויות גדלה באופן משמעותי.

לדוגמא: לרחפן 1 יש פעולה של איסוף ומשלוח עבור כל אחת מהחבילות, כנ"ל לרחפן 2, בסוף יש שימוש רק בחלק מהמשימות הנ"ל.

ד. 2 רחפנים, 5 חבילות, 2 מחסנים ו-4 בתים – כחלק מזה שרצינו להמשיך ולסבך את הבעיה, רצינו גם לשבור את הסימטריות שלה. אחת המטרות היא ביצוע התוכנית בזמן הקצר ביותר (תוך התחשבות באילוצים), ובשלב זה יש חשיבות תלות גדולה יותר לאופן חלוקת המשלוחים בין הרחפנים והסדר שלהם.

ה. 2 רחפנים, 4 חבילות, 2 מחסנים ו-4 בתים עם TIL – היה לנו חשוב להוסיף לבעיה אילוץ זמנים. כלומר, לאלץ פעולות להתבצע בזמן מסוים בתוכנית. ניתן להקביל זאת לעולם האמיתי, כאשר יש צורך במשלוח חבילה אצל הלקוח בחלון זמנים מסוים. בחרנו תחילה לעבוד רק עם 4 חבילות, בכדי להקל על ההרצות של Scotty, עד שנבין בצורה הטובה ביותר כיצד לממש את חלון הזמנים בקבצי הPDDL.

ו. 2 רחפנים, 5 חבילות, 2 מחסנים, ו-4 בתים עם TIL – השלב האחרון, המסובך ביותר שעבדנו עליו. כאן ביצענו בדיקות על שינוי חלונות הזמן והשפעתם על התוכנית. השתמשנו בחלון בסוף, בתחילה ובאמצע התוכנית, פירוט נוסף על כך – בתוצאות.

2) Fly action

כשהתחלנו את הפרויקט, עבדנו עם טיוטת קובץ pddl בה הייתה פעולת fly ראשית, ופעולות deliver\pickup נקודתיות שלא כללו תזוזה של הרחפנים.

הנה דוגמה לפעולת fly של drone1 :

```
(:durative-action fly-drone1
    :duration (and (>= ?duration 0.1) (<= ?duration 300.0))
    :condition (and
        (at start  (delivery-ongoing))
        (at end  (delivery-ongoing))
        (at start  (not (drone1-fly)))
        (at end    (drone1-fly))
        (over all (>= (drone1-battery) 0))
    )
    :effect (and
        (at start  (drone1-fly))
        (at end    (not (drone1-fly)))
        (increase (drone1-x) (* (vx-drone1) #t))
        (increase (drone1-y) (* (vy-drone1) #t))
        (decrease (drone1-battery) (* 1.0 (norm (energy-drone1)) #t))
    )
)
```

ודוגמה לפעולת deliver ללא תזוזה:

```
(:durative-action drone1-deliver-house1-package1-at-slot1
    :duration (and (>= ?duration 0.5) (<= ?duration 0.5))
    :condition (and
        (at start  (delivery-ongoing))
        (at end  (delivery-ongoing))
        (at start (not (drone1-busy)))
```

```
            (at end   (drone1-busy))
            (at start (inside (house1-region (drone1-x) (drone1-y) )))
            (at end   (inside (house1-region (drone1-x) (drone1-y) )))
            (at start (drone1-has-package1-at-slot1))
            (at start (house1-needs-package1))
        )
        :effect (and
            (at start (drone1-busy))
            (at end (not (drone1-busy)))
            (at end (not (drone1-has-package1-at-slot1)))
            (at end (not (drone1-slot1-full)))
            (at end (not (house1-needs-package1)))
            (at end (house1-got-package1))
        )
    )
```

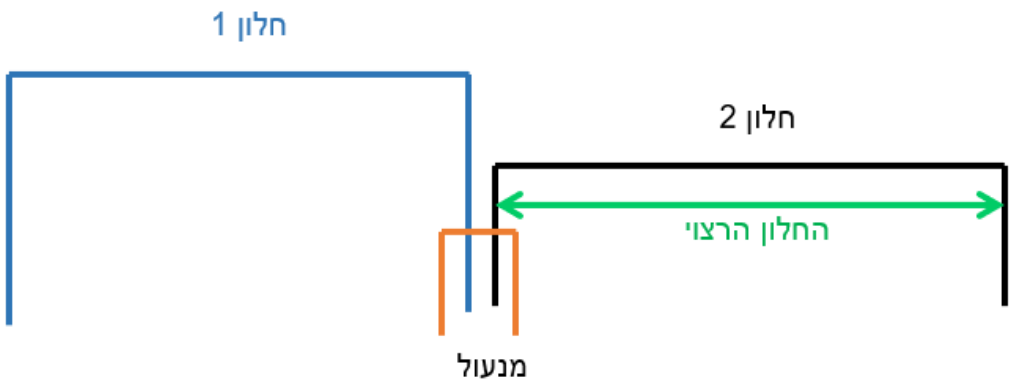
הבעיה התחילה כשרצינו להוסיף אינטרוולים של זמן בהן פעולות מסוימות יכולות לקרות (לדוגמה, בית 3 יכול לקבל חבילות רק באינטרוול $[150,180]$), ואז נתקלנו בבעיה שהחיפוש היה רחב מידי ו scotty קרס עם שגיאת Heap exhausted, לאחר המעבר לפעולות לוגיות שכוללות גם תזוזה לנקודה (כפי שרואים בסעיף 2.2 של הדו"ח) הבעיה נפתרה וזמן החיפוש אפילו התקצר, זה ככל הנראה נפתר בגלל שבכל פעם היה צריך לפתור בעיות אופטימיזציה קטנות (כי כל פעולה לוגית כללה גם תזוזה ותנאי על הבטרייה) במקום לפתור המון בעיות אופטימיזציה על כל המסלול שהיו גם יותר ארוכות, וגם לפעמים לא ברות-תרחיש, אך לא היה ניתן לפסול אותן מראש.

3) Time literal

מנגנון חלונות הזמן כפי שהצגנו אותו לא היה המנגנון הראשון שניסינו, קדמו לו 3 מנגנונים אחרים, כל אחד כלל בעיות אחרות ולכן היינו צריכים כל פעם לשפר את המנגנון:

א. מנגנון פונקציה - המנגנון הראשון אותו ניסינו, הוא יצירת פונקציה T בקובץ הdomain, במטרה שפונקציה זו תשמור בכל רגע כמה זמן עבר מתחילת הסימולציה, ניתן לבצע זאת ע"י יצירת action שמאולץ להתחיל בתחילת הסימולציה ולהסתיים בסופה, עם אפקט של increase #t. לאחר מכן, ביצענו התניה של פעולות מסוימות כמו משלוח בכך שהפונקציה T תהיה בין ערכים מסוימים (לדוגמה בין 150 ל180 שניות), מנגנון זה לא עבד טוב כי הכניס לבעיה מורכבות מאד גבוהה, פתרונות לא יכלו להיפסל בעת הרכבתם בבעיית החיפוש בגרף, אלא רק בסוף בעת פתרון הבעיה הרציפה.

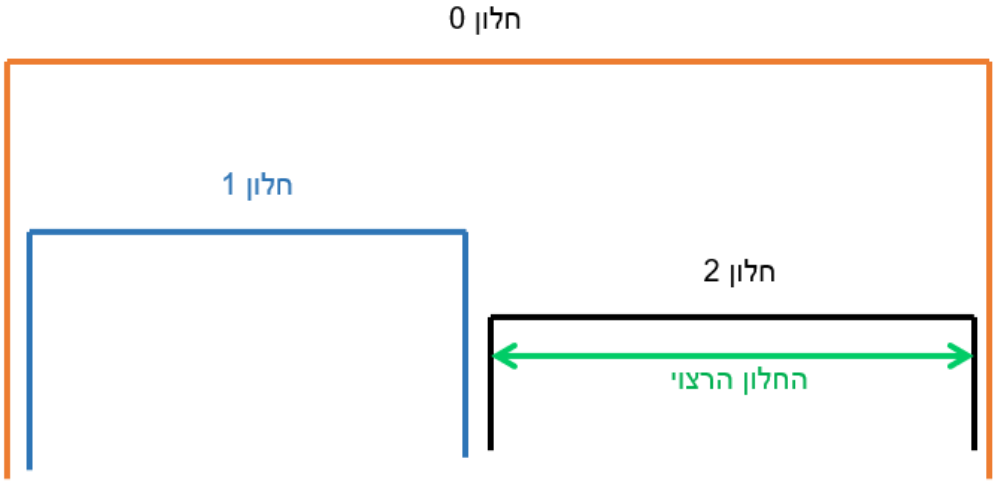
ב. מנגנון מנעול – זהו המנגנון השני אותו ניסינו, במנגנון זה, יש 2 פעולות של יצירת חלון (T0 ו T1 , כל אחד מרים את הדגל המתאים לאותו חלון) ופעולה שלישית של מנעול המקשרת בין החלונות, החלון הראשון מאולץ להתחיל בתחילת התוכנית, כפי שניתן לראות באיור הבא:



איור 7. דיאגרמת אילוץ זמני במנגנון מנעול

אבל מנגנון זה לא עבד טוב במיוחד, חשבנו שמבחינת סדר הפעולות מנגנון זה יאלץ את אילוץ הזמנים בצורה טובה, אבל ככל הנראה היה כשל בגלל סידור הזמנים.

ג. מנגנון TIL מלא – זה מנגנון הדומה למנגנון שבו השתמשנו בסוף, אך במקום להשתמש ב"דגל" שהוא T0& ~T1 , מגדירים חלון נוסף T2, שנדלק רק כאשר ה"דגל" דולק, ומשתמשים בדגל T2 עבור הפעולות עם האילוץ, כפי שניתן לראות באיור הבא:



איור 8. דיאגרמת אילוץ זמני במנגנון TIL מלא

הבעיה עם מנגנון זה, היא שהוא מוסיף מלאכותית עוד סיבוכיות לבעיה, בכך שהplanner צריך למצוא ש"כדאי" לו להפעיל את חלון 2 בזמן שחלון 0 דולק וחלון 1 לא. הפתרון שבו בחרנו (כפי שמפורט בפרקים הקודמים) הוא הסרת חלון 2.

4) מטריקות

כפי שהוזכר, יש 2 קבצי pddl : problem ו-domain. בקובץ הproblem יש חלק שבו מציינים מה המטריקה לפיה פותרים את בעיית האופטימיזציה.

הערה: scotty פותר את בעיית האופטימיזציה לפי מטריקה זו רק לאחר שכבר החליט ע"י חיפוש בגרף את כל הactions שיהיו, ולכן למטריקה זו אין השפעה על ה actions או הסדר שלהם כלל, אלא רק על ה control variables או על האם התוכנית פיזיבילית.

נזכיר שהמטריקה שאיתה עבדנו היא

```
(:metric minimize (+
                   (* -0.1 (drone1-battery))
                   (* -0.1 (drone2-battery))
                   (* 20 (total-time) )
                   )
)
```

אך בדקנו גם חלופות אחרות:

א. function חדש שסופר כמה זמן רחפן מסוים פעיל, ניתן לעשות זאת ע"י קביעת action שיפעל מתחילת תזוזת ה drone ויסתיים לאחר שהרחפן מסיים לגמרי, לדוגמה:

```
    (:durative-action drone1-timer
        :duration (and (>= ?duration 0.1) (<= ?duration 1000))
        :condition (and
            (at start  (drone1-delivery-ongoing))
            (at start (not (drone1-busy)))
            (at end  (drone1-complete-delivery))
        )
        :effect (and
            (at start (drone1-timer-on))
            (increase (drone1-time) (* 1.0 (const) #t))
        )
    )
```

כש- (drone1-timer-on) משמש לכך שה action יפעל כל זמן שdrone1 לא סיים, ו const הוא control-variable שהגדרנו אך בעצם חסמנו את ערכיו להיות 1 במדויק.

לאחר מכן ניתן להציע את המטריקה drone1-time+drone2-time, ניסינו ע"י כך לגרום לשינוי סדר הפעולות, אך כפי שרשמנו בהערה, הדבר בעצם לא אפשרי. יש לציין שדרך זו גם מאטה את הפתרון.

ב. ניתן לשלב את המטריקה של א' עם הסוללה שנשארה.

5) פעולת סיום משותפת או מפוצלת

בחרנו בין האפשרויות של פעולת "גש לנקודת הסיום" משותפת – כלומר 2 הרחפנים מקבלים את הפקודה ביחד (לרוב רחפן אחד סיים את המשלוח האחרון שלו ורק מחכה, והרחפן השני בדיוק סיים משלוח) ואז שניהם ינועו ביחד לנקודות הסיום.

לבין "גש לנקודת הסיום" מפוצלת – כל רחפן יכול לקבל לחוד את הפקודה ולגשת לנקודת הסיום.

מכיוון שהדבר לא הכביד או שינה מבחינת זמן הריצה של scotty, בחרנו באפשרות השנייה כי היא ריאליסטית יותר, אין סיבה שנרצה שרחפן יחכה באוויר ולא יחזור כמה שיותר מהר (כדי שיוכל אולי לקבל עוד משימות, וגם כי להישאר באוויר עולה בסוללה).

6) אילוצי Overall

בשפת PDDL ניתן לכתוב תנאים לביצוע פעולה בתצורת overall, כך שהתנאי ייבדק לאורך כל זמן ביצוע הפעולה. בתחילת הדרך, הוספנו הרבה תנאי Overall, מתוך מחשבה שבמהלך הפעולה לא נרצה שיקרו דברים לא רצויים. לצערנו, תנאים אלו מאוד מעמיסים על הריצה של scotty. נוכחנו לגלות שמספיקים תנאי at start ו-at end, שכן אין כל סיבה שתוך כדי הפעולה יתבצעו דברים חריגים, והאילוצים על ההתחלה והסוף מספיקים.

7) סוללה מתבזבזת בעת שהייה באוויר ללא תזוזה

התנהגות זו רצויה מכיוון שהיא ריאליסטית ומתאימה לבעיה אמיתית, אך כאן נתקלנו בקושי די גדול:

בחלופה הראשונה שבה יש action של fly, המימוש יחסית פשוט, וצריך להוסיף מחובר קבוע לסוללה שיורדת עם הזמן, לדוגמה:

```
(decrease (drone1-battery) (+

                              (* 1.0 (norm (energy-drone1)) #t)
                              (* 1.0 (const) #t)
                              )
)
```

אך נזכיר כי החלופה שבה יש action של fly, לא עבדה טוב באופן כללי ולא התאימה לקטעי זמן מיוחדים.

בחלופה השנייה שבה בחרנו, גם אם נוסיף את חלק קבוע לסוללה שיורדת, הנ"ל יפעל רק כאשר הרחפן זז, וכאשר הוא לא יזוז, שום action לא יפעל ולא נקבל הורדת סוללה.

ולכן ניתן בכל זאת להפעיל action שירוץ ברקע מתחילת תזוזת הרחפן עד סופו (בדומה לסעיף 4) ויוריד כל הזמן את הסוללה של הרחפן, לדוגמה:

```
(:durative-action drone2-constant-battery-drown
    :duration (and (>= ?duration 0.1) (<= ?duration 1000))
    :condition (and
        (at start  (drone2-delivery-ongoing))
        (at start (not (drone2-busy)))
        (at end  (drone2-complete-delivery))
    )
    :effect (and
        (at start (drone2-constant-battery-drown-on))
        (increase (drone2-time) (* 1.0 (const) #t))
    )
)
```

הוספת פעולה זו מכבידה על תהליך האופטימיזציה באופן די משמעותי ולכן בסוף לא בחרנו להשתמש בה.

3. תוצאות

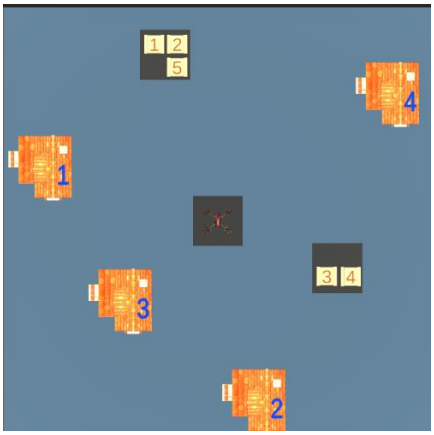
3.1 הבעיה שבחרנו

בחרנו בבעיה הבאה:

- מפה בגודל $1280x1280$.
- 2 רחפנים, עם סוללה של 4000, הסוללה יורדת בכל שנייה בגודל המהירות של הרחפן. נקודת ההתחלה והסוף של הרחפנים במנחת שנמצא ב[640,640] ברוחב [32,32].
  לרחפנים מהירות מקסימלית כוללת של $15[\frac{m}{sec}]$ ותאוצה של $4.75[\frac{m}{\sec^2}]$.
  הם יכולים בכל רגע נתון להחזיק חבילה אחת בלבד.
- 2 מחסנים, גם הם ברוחב [32,32] הממוקמים ב- [482,1138], [998,498]
- 4 בתים, גם הם ברוחב [32,32] הממוקמים ב-
  [108,798], [748,98], [348,398], [1148,1018].
- בית 3 פנוי לקבלת חבילות רק באינטרוול הזמן [150,180]. (עוד חלופות יתוארו בהמשך)
- 5 חבילות, עם התכונות הבאות :

חבילה	מחסן	בית מיועד
1	1	1
2	1	2
3	2	3
4	2	4
5	1	3

ממבט על כך נראית המפה ברגע $t = 0$:



איור 9. מבט מלמעלה על המפה ברגע ההתחלה

3.2 פתרון ראשון ולא פיזיקלי

את בעיה זו המרנו לקבצי pddl ונתנו ל- scotty לפתור, אך קיבלנו פתרון שאינו פיזיקלי שכן המהירות המקסימלית שאליה מגיע רחפן 1 היא $16.3[\frac{m}{\text{sec}}]$ והמהירות המקסימלית שאליה מגיע רחפן 2 היא $17.3[\frac{m}{sec}]$.

3.3 מיקום חלון הזמן

את מיקום חלון הזמן ב $[150,180]$ בחרנו בכוונה כך שחלון הזמן יהיה ב"אמצע" התוכנית כשהיא ללא אילוצי זמנים, וזאת כי בעת שבחנו חלון זמן בהתחלת התוכנית או בסופה, לא ניתן לדעת במדויק איזו יכולת יש ל-Planner להתמודד עם האילוץ, לדוגמה: אם האילוץ הוא בהתחלה\סוף אז הPlanner מתזמן את פעולות הdelivery המתאימות להתחלה\סוף התכנית, אך לא צריך להשהות פעולות אחרות. אך אם האילוץ הינו באמצע, ניתן לראות היטב כיצד ה- Planner צריך להשהות או להאט פעולות קודמות, אבל גם להתחשב בהמשך התוכנית.

3.4 שינוי קבצי ה-PDDL

הסיבה לכך שהפתרון אינו פיזיקלי היא שקובץ הpddl מרשה מהירות גבוהה מידי, ואז כדי לעמוד בדרישות התאוצה, נדרשת בפועל מהירות מקסימלית של הרחפן הגבוהה מהמהירות שאיתה scotty תכנן.

כדי לפתור את הבעיה, נחליף את המהירות המקסימלית המותרת לרחפן מ15 ל13, ונפתור שוב.

הפעם נקבל פתרון פיזיקלי:

איור 10. ניתן ללחוץ להצגת הסימולציה בYoutube.

4. סיכום ומסקנות

לסיכום, בפרויקט זה למדנו על בעיית התכנון למערכות AI, ניסחנו בעיית תכנון מרחבי-טמפורלי וכתבנו תוכנית שמייצרת קבצי PDDL עבור הבעיה, לאחר מכן השתמשנו באלגוריתם מתקדם כדי לפתור את הבעיה, ויצרנו מהתוכנית שקיבלנו פתרון פיזיקלי ע"י ניסוח ופתרון משוואות תנועה במחשב, לאחר מכן למדנו על המנוע ליצירת משחקים Unity וכתבנו בו סימולציה כדי להמחיש את הפתרון.

סך הכול יצרנו מערכת מלאה לפתרון בעיה וסימולציה פיזיקלית שלה, עם לולאת משוב אחת פשוטה.

היה מאד מעניין, והעבודה עם Unity הייתה מרעננת וויזואלית.

כיוונים להמשך:

- הכנסת התחשבות של החיכוך עם האוויר בעת חישובי המהירויות וההאצות.
- שילוב משימות עם אילוצי מקום וזמן עבור שני הרחפנים במקביל, למשל משלוח של חבילה אחת יחד.
- הכנסת אזורים אסורים למעבר – no fly zone.

5. רשימת מקורות

[1] Taitler, A. et al. (2019) *Minimum Time Validation for Hybrid Task Planning*, *icaps19.icaps-conference.org*. Available at: https://icaps19.icaps-conference.org/workshops/PlanRob/PlanRob_2019_submissions/PlanRob_2019_paper_1.pdf (Accessed: 04 December 2023).

[2] Fernandez-Gonzalez, E. Mixed Discrete-Continuous Planning with Complex Behaviors. In *The 26th International Conference on Automated Planning and Scheduling* (p. 48).

[3] *Temporal Planning as Heuristic Search*. (n.d.). Google Docs. https://docs.google.com/presentation/d/1N5gc__kmY3TE9cHqFg2zM8yT1zxehTc710nCRk6otxM/edit#slide=id.g221c7f3818_1_0

[4] *An Introduction to PDDL* *Introtopddl2.pdf (toronto.edu)*

[5] *Planning.wiki - the AI Planning & PDDL Wiki.* https://planning.wiki/

[6] *Matrice 600 - Product Information - DJI*. (n.d.). DJI Official. https://www.dji.com/global/matrice600/info