האוניברסיטה הפתוחה
The Open University of Israel

סימולטור למחקר תנועת רכב אוטונומי-חנייה אוטונומית
CARLA

פרוייקט גמר – קורס 22997
תואר שני M.Sc במדעי המחשב

מגישה: זוהר ואנונו חדד
מנחה: לאוניד ברנבוים
דצמבר 2023

תוכן עניינים

הקדמה

תחום הרכב האוטונומי הינו תחום חדשני הלוקח חלק מרכזי בתעשייה ובאקדמיה.
תחום רחב, בעל אתגרים רבים בתהליכי הפיתוח והמחקר שלו, החל מיצור וחקירת סנסורים שונים עד לבניית תמונה מציאותית. זאת בעזרת שימוש באלגוריתמים רבים לחיזוי והבנה של מסלול הנסיעה ועד לניהול תנועת הרכב.
בפרויקט זה נציג סימולטור גרפי של מציאה וחניית הרכב בצורה אוטונומית.
הסימולטור מאפשר ממשק נוח להתמצאות הרכב במרחב וכתיבת אלגוריתם למציאה וחניית הרכב בצורה אוטונומית כפי שכתוב למעלה.

הסימולטור פותח בשפת python על גבי ממשק גרפי Carla.
ממשק זה הוא בתצורה של שרת – לקוח עם אפשרות לשליטה ברכב שנמצא במרחב (ימינה/שמאלה, מהירות ועוד...) או עם אפשרות שרת בלבד שבו הרכב עולה בנקודה מסוימת במרחב וחונה בחנייה הפנויה הקרובה ביותר.

מכונית אוטונומית

מכונית אוטונומית או מכונית רובוטית (Autonomous car) היא מכונית המנווטת ומתגברת על מכשולים ללא התערבות פעילה של נהג אנושי ובלא צורך בתכנון מיוחד של הכביש וסביבתו.
מכונית אוטונומית יכולה להגיע לכל מקום שמכונית סטנדרטית מגיעה אליו ולעשות כל מה שנהג אנושי מנוסה עושה.
האגודה של מהנדסי רכב (SAE) מגדירה כיום 6 רמות של אוטומציה של נהיגה, החל מרמה 0 (ידנית מלאה) לרמה 5 (אוטונומית מלאה).

רמות אוטונומיות

ארגון הרכב הבינלאומי, SAE International הגדיר 6 רמות שונות של נהיגה אוטונומית לכלי הרכב :

רמה 0 : ללא אוטומציה בנהיגה. ברמה זו הנהג מחויב בכל פעולות הנהיגה. רמה זו הייתה האפשרות היחידה ברוב הגדול של מכוניות המאה ה-20.

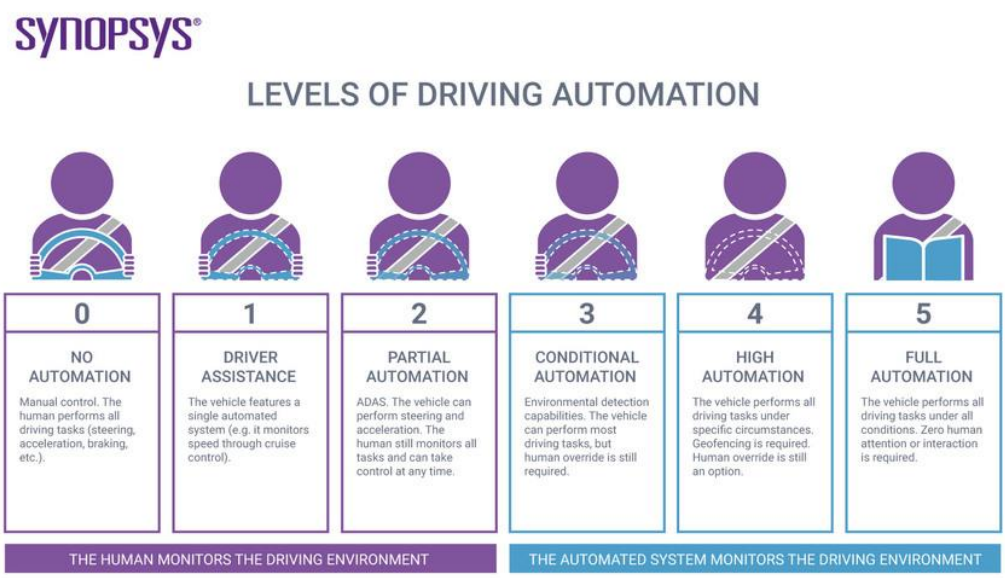
רמה 1 : סיוע לנהג. ברמה זו קיימת ברכב מערכת סיוע נקודתי לנהג כמו התראה בעת סטייה מהנתיב. כלי רכב ברמה זו יצאו לשוק החל מסביבות שנת 2010.

רמה 2 : אוטומציה חלקית. מכונית שבה פעילות שתי מערכות בטיחות ואוטונומיה או יותר, אשר מסוגלות לטפל בו זמנית הן בהיגוי הרכב והן בבלימה. ברמה זו עדיין חייב הנהג לבחון את תנאי הכביש ולהגיב בהתאם. כלי רכב ברמה זו יצאו לשוק החל מסביבות שנת 2016.

רמה 3 : אוטומציה מותנית. ברמה זו המערכות של המכונית יהיו בשלות מספיק כדי לקחת פיקוד על הנהיגה בתנאים מסוימים. בתנאים אלה הנהג לא יידרש לפקח על תנאי הדרך, אך יידרש להיות זמין לנהיגה במקרי חירום.

רמה 4 : אוטומציה ברמה גבוהה. (דומה לרמה 3) אפשר לבצע נהיגה אוטונומית מלאה בתנאים מסוימים. ברמה 4 בשונה מרמה 3, אין צורך בנהג זמין למצבי חירום. נכון לתחילת שנת 2022, אף חברה לא קרובה לשיווק מסחרי של מכוניות כאלו, וברמה זו ישנן מכוניות ניסוי בלבד.

רמה 5 : אוטומציה מלאה. ברמה זו הרכב מסוגל להחליף את הנהג בכל תנאי דרך ואינו זקוק לנהג. נכון ליוני 2021, יש מספר קטן של מכוניות שהורשו לנסוע, ללא נהג ביטחון, בתוך עיר. גם מכוניות אלו מוגבלות רק לאזור מוגדר היטב.



יתרונות וחסרונות

יתרונות	**חסרונות**
שיפור פקקים	יקר
מניעת תאונות	לא ניתן להסתמך בתנאי מזג אוויר קשים
נסיעה בטיחותית יותר	חיבוריות לרשת – פגיעויות סייבר
ניצול זמן ואנרגיה של הנהג	

האתגרים

ישנם אתגרים שונים בהגעה למצב שימוש ברכב אוטונומי:

- חיישנים- הרכבים האוטונומיים צריכים שורה של חיישנים מסוגים שונים המסוגלים להבחין במציאות שסביב הרכב. במעבדה קל לעבוד ולתמרן עם החיישנים הללו אבל במזג אוויר מורכב, כבישים עמוסים, מנהרות, גשרים, רכבים לא שגרתיים ועוד.. יכולים כולם לפגוע ביכולת החיישנים לפענח את המציאות.
- בינה מלאכותית- אלגוריתמים המבוססים על בינה מלאכותית לעיבוד מידע המגיע מהחיישנים לקבלת החלטות על הצעדים העתידי של הרכב. קיימת הסכמה רחבה שבעתיד המכונות יוכלו לזהות ולהחליט טוב יותר מהנהג הממוצע, אך בהווה אין בדבר זה ביטחון. כיום לא קיים יכולת של מכונה ללמוד ולקבל את ההחלטות על הכביש בצורה נכונה בנוסף, אין הסכמה בין חברות וגופים בתעשייה לסטנדרטיזציה של מחשבים אוטונומיים- מה מידת החופש שיקבלו, איך יבדקו וכמה יתוקפו. עד שלא תיווצר תכנית מסודרת חברות רכב לא ירצו לסכן את הנהגים שלהם.
- מציאות- כאשר רכב אוטונומי יעלה על הכביש אנחנו נצפה ממנו שימשיך ללמוד ולהשתפר, ממש כמו בן אדם. אנחנו לא יכולים להבטיח שהלמידה בעולם האמיתי תבוצע

בכיוון הנכון ותביא לבטיחות גבוהה יותר. התעשייה צריכה למצוא את הדרך להבטיח שהרכבים יהיו שומרי חוק ובטוחים.

- רגולציה- לא קיים סטנדרט בינלאומי ברור לפונקציות שמבוצעות ע״י מחשב- כמו בקרת שיוט, בלימה אוטומטית או שמירה על נתיבים. מצב זה חושף חברות לתביעות על הולכת שולל וחושף נהגים לפער מול חברות הביטוח. בלי תקנות שיעזרו להסדיר את פעילות הרכבים האוטונומיים יהיה קשה להביא אותם לשימוש נרחב.
- אחראיות לתאונות- לא הוגדר עד היום מי אחראי לתאונות שנגרמות ע״י רכב אוטונומי, היצרן או הנוסע האנושי? לפי תכניות הרכבים נראה כי למכונית אוטונומית מלאה ברמה 5 לא יהיה כלל לוח מחוונים או הגה, כך שלנוסע אנושי לא תהיה אפילו אפשרות להשתלט על הרכב במקרה חירום.

סימולטור CARLA

CARLA פותחה עבור בניית סימולציות ואימות של מערכות נהיגה אוטונומיות.
בנוסף לקוד פתוח, CARLA מספקת נכסים דיגיטליים פתוחים (פריסות עירוניות, בניינים, כלי רכב ועוד...) שנוצרו למטרה זו וניתנים לשימוש חופשי.
פלטפורמת הסימולציה תומכת במפרט גמיש של חבילות חיישנים, תנאי סביבה, שליטה מלאה בכל השחקנים הסטטיים והדינמיים, יצירת מפות ועוד.

- CLARA מאפשרת גמישות בעזרת שימוש בארכיטקטורה של server & multi-client.
- מכילה API למשתמשים לשלוט באספקטים שונים כמו : סוג הרכב, שליטה ברכב, סוג חיישן ומיקומו, עומס תעבורה, תנאי מזג אוויר ועוד.

סוגי אובייקטים

שחקנים (Actor) ב-CARLA הם האלמנטים שמבצעים פעולות בתוך הסימולציה, והם יכולים להשפיע על שחקנים אחרים.

בממשק CARLA אובייקטים אינם נמחקים עצמאית מהשרת. לכן, נדרש ממפתח הסימולטור לשמור את רשימת השחקנים במשתנה/ מערך מוגדר שאותו יש לנקות ע"י destroy().

ישנם שחקנים שונים ב-CARLA : כלי רכב (הכוללים הולכי רגל), הצופה וגם חיישנים, תמרורים, רמזורים.

- כלי רכב :

אחת מקבוצות השחקנים החשובות ביותר ב-CARLA. אלה כוללים כל סוג של רכב ממכוניות ועד משאיות, אופנועים, טנדרים, אופניים וגם רכבים רשמיים כגון ניידות משטרה. קבוצה רחבה של שחקנים אלה מסופקת ב carla.BlueprintLibrary כדי להקל על דרישות שונות. ניתן לשלוט ברכבים באופן ידני או להגדיר למצב טייס אוטומטי שיבוצע בצד הלקוח על ידי מנהל התנועה.

- הצופה :

הצופה הוא מבט לתוך הסימולציה. כברירת מחדל, הצופה נפתח בחלון כאשר אתה מפעיל את שרת CARLA, הצופה עוזר לדמיין את הסימולציה שלך. באמצעות הצופה אפשר להכיר את המפה, ולראות את התוצאה של כל שינוי שמתבצע כמו הוספת רכבים, שינוי מזג האוויר ועוד.. המשתמש יכול "להטיס" את הצופה מסביב לעולם באמצעות העכבר כדי לשלוט בגובה וזוויות התצוגה ע"י לוח המקשים והעכבר.

  - לוח ומקשים :
    - Q - זז כלפי מעלה (לכיוון הקצה העליון של החלון)
    - E - זז כלפי מטה (לכיוון הקצה התחתון של החלון)
    - W - לנוע קדימה
    - S - לנוע אחורה
    - A - זז שמאלה
    - D - זז ימינה
  - עכבר :
    - קליק שמאלי וגרירת העכבר בחלון הצופה למעלה ולמטה כדי לשלוט בגובה
    - קליק שמאלי וגרירת העכבר בחלון הצופה שמאלה וימין כדי לשלוט זווית אופק

- חיישנים:

חיישנים הם שחקנים (Actors) שמייצרים זרם של נתונים, רוב החיישנים יחוברו לרכב כדי לאסוף מידע על סביבתו. חיישנים מאזינים לנתונים, כאשר הנתונים מתקבלים הם קוראים לפונקציה ספציפית שהוגדרה מראש.

סוגי חיישנים:

- מצלמות

המצלמה מצלמת את העולם מנקודת המבט שלה ומאפשרת התחלת האזנה וקבלת נתונים בכל שלב בסימולציה.

Sensor	Output	Overview
Depth	carla.Image	Renders the depth of the elements in the field of view in a gray-scale map.
RGB	carla.Image	Provides clear vision of the surroundings. Looks like a normal photo of the scene.
Optical Flow	carla.Image	Renders the motion of every pixel from the camera.
Semantic segmentation	carla.Image	Renders elements in the field of view with a specific color according to their tags.
Instance segmentation	carla.Image	Renders elements in the field of view with a specific color according to their tags and a unique object ID.
DVS	carla.DVSEventArray	Measures changes of brightness intensity asynchronously as an event stream.

- גלאים

התחלת האזנה וקבלת נתונים יכולה להתבצע רק כאשר האובייקט שהם מחוברים אליו רושם אירוע ספציפי, למשל פגיעה ברכב יעלה collision event.

Sensor	Output	Overview
Collision	carla.CollisionEvent	Retrieves collisions between its parent and other actors.
Lane invasion	carla.LaneInvasionEvent	Registers when its parent crosses a lane marking.
Obstacle	carla.ObstacleDetectionEvent	Detects possible obstacles ahead of its parent.

- אחר

פונקציות שונות כגון ניווט, מדידת מאפיינים פיזיים ומפות נקודות דו-ממדיות/תלת-ממדיות של הסצנה המאפשרת האזנה וקבלת נתונים בכל שלב סימולציה.

Sensor	Output	Overview
GNSS	carla.GNSSMeasurement	Retrieves the geolocation of the sensor.
IMU	carla.IMUMeasurement	Comprises an accelerometer, a gyroscope, and a compass.
LIDAR	carla.LidarMeasurement	A rotating LIDAR. Generates a 4D point cloud with coordinates and intensity per point to model the surroundings.
Radar	carla.RadarMeasurement	2D point map modelling elements in sight and their movement regarding the sensor.
RSS	carla.RssResponse	Modifies the controller applied to a vehicle according to safety checks. This sensor works in a different manner than the rest, and there is specific RSS documentation for it.
Semantic LIDAR	carla.SemanticLidarMeasurement	A rotating LIDAR. Generates a 3D point cloud with extra information regarding instance and semantic segmentation.

האלגוריתם ומימוש- חנייה אוטונומית

<u>שלבי פעולה:</u>

כיום מספר המכוניות רק הולך וגדל, איתו מצוקת החנייה, במקרה כזה חנייה אוטומטית היא חלק הכרחי בחיינו.

מסיבה מרכזית זו בחרתי לחקור את האלגוריתם של חנייה אוטונומית.

מערכת החנייה האוטומטית מורכבת מזיהוי מיקום חנייה, תכנון נתיבים וביצוע חנייה:

1. זיהוי מקום חנייה- נשתמש בסנסורים של הרכב האוטונומי לקבלת מידע במערכת ועיבוד המידע- חנייה במקביל או במאונך עבור רכב מסוג private.
2. תכנון נתיב- התכנון מבוסס על הדינמיקה והאילוצים של הרכב לתכנון נתיב גיאומטריה אפשרי מראש.
3. ביצוע חנייה- לפי תכנון הנתיב נשלוט ברכב (מהירות, כיוון וזוויות) כדי ללכת בנתיב שתוכנן ולבצע את החנייה.

במימוש האלגוריתם בעולם CARLA נאפשר 2 אופציות לסימולציה:

- תצורה של שרת - הרכב עולה בנקודה מסוימת במרחב וחונה בחנייה הפנויה הקרובה ביותר. זאת בעזרת אלגוריתם שסורק כל פעם את החנייה הקרובה ביותר (במקביל או בניצב) עד למציאת החנייה הפנויה. אחרי מציאת החנייה הפנויה יש מעבר ישיר בין מציאת החנייה לשלב ביצוע החנייה על ידי התקדמות קדימה וחנייה ברוורס, לפי סוג החנייה.
- תצורה של שרת - לקוח, הנותן למשתמש אפשרות לשליטה ברכב שנמצא במרחב (ימינה/שמאלה, מהירות ועוד...)

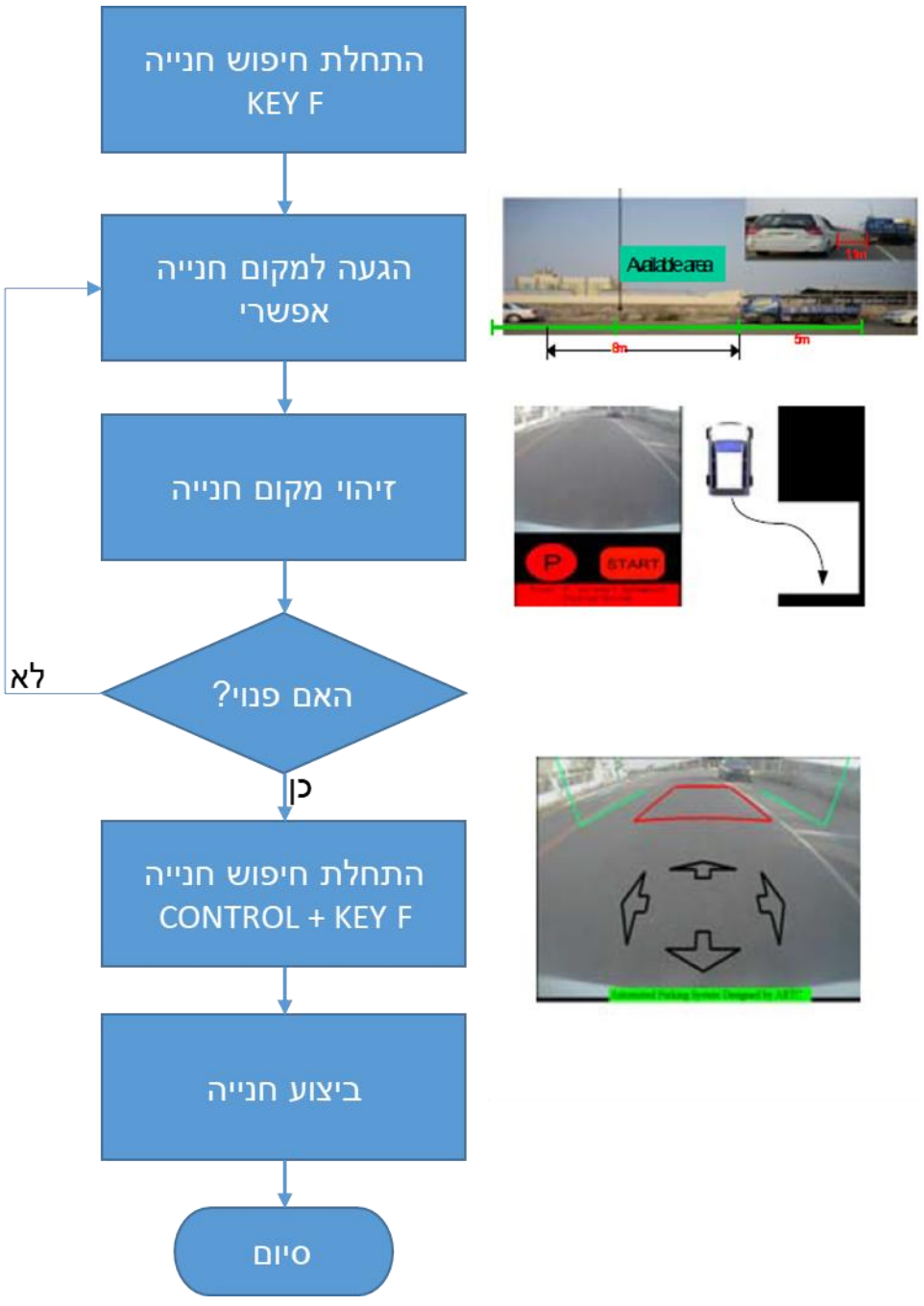
במימוש האלגוריתם בעולם CARLA נתחיל בנהיגה עצמאית של הרכב ע״י משתמש וכאשר המשתמש יגיע לאזור חנייה מוגדר (תיחום החנייה ע״י קווים לבנים על הכביש) והוא רוצה לבצע חנייה אוטונומית הוא ילחץ על KEY F ותוצג אינדיקציה למשתמש על התחלת חיפוש חנייה.

בהתחלה העצם יבצע את האלגוריתם על מנת לחפש את מקום החנייה הכי קרוב אל הרכב. האלגוריתם יסרוק את הסביבה של הרכב ויבדוק האם החנייה פנויה או תפוסה.

אם החנייה לא פנויה- יחפש את מקום החנייה הקרוב הבא ויבדוק שוב אם החנייה פנויה או תפוסה (עד מציאת החנייה/ מעבר על כל החניות האפשריות), אם החנייה פנויה- הרכב עוצר ומחכה לפקודה הבאה.

בסוף תהליך זה העצם מכיל נתונים של מיקום החנייה וסוג החנייה (מקביל או ניצב) בנוסף תוצג הודעה מתאימה למשתמש- נמצאה או לא נמצאה חנייה.

אחרי מציאת החנייה ולחיצה על CTRL+F תתקבל אינדיקציה למשתמש על התחלת ביצוע החנייה. כעת הרכב מתקדם קדימה ומתחיל את החנייה ברוורס, בהתאם לסוג החנייה.


התחלת חיפוש חנייה
KEY F
הגעה למקום חנייה
אפשרי
זיהוי מקום חנייה
האם פנוי?
לא
כן
התחלת חיפוש חנייה
CONTROL + KEY F
ביצוע חנייה
סיום
Aadditional area
1.8m
8m
5m
P
START

שלבי האלגוריתם

1. זיהוי חנייה- סנסור, מיקום הסנסור, גודל החנייה.

- חיישן- חיישן מצלמה מסוג Semantic segmentation camera

מצלמת פילוח סמנטי, מצלמה זו מסווגת כל אובייקט הנראה על ידי הצגתו בצבע שונה בהתאם לתגיות שלה (למשל, הולכי רגל בצבע שונה מאשר כלי רכב). כשהסימולציה מתחילה, כל אלמנט בסצנה נוצר עם תג, תהליך זה קורה כששחקן נוצר.

בחרתי לקחת את האלמנטים : roadline & vehicles. זאת על מנת ←

1. לסווג את הקווים הלבנים
2. לבדוק שאין רכבים שמפריעים לחנייה
3. לדעת אם החנייה פנויה או תפוסה

Value	Tag	Converted color
0	Unlabeled	(0, 0, 0)
1	Building	(70, 70, 70)
2	Fence	(100, 40, 40)
3	Other	(55, 90, 80)
4	Pedestrian	(220, 20, 60)
5	Pole	(153, 153, 153)
6	RoadLine	(157, 234, 50)
7	Road	(128, 64, 128)
8	SideWalk	(244, 35, 232)
9	Vegetation	(107, 142, 35)
10	Vehicles	(0, 0, 142)
11	Wall	(102, 102, 156)
12	TrafficSign	(220, 220, 0)
13	Sky	(70, 130, 180)
14	Ground	(81, 0, 81)
15	Bridge	(150, 100, 100)
16	RailTrack	(230, 150, 140)
17	GuardRail	(180, 165, 180)
18	TrafficLight	(250, 170, 30)
19	Static	(110, 190, 160)
20	Dynamic	(170, 120, 50)

- <u>מיקום הסנסור-</u> מציאת החנייה תעשה בעזרת סנסור אחד : מצלמת מראה ימנית בזווית 38 מעלות . חשוב להדגיש שהנתונים המרוחקים יפריעו לתהליך מיצוי המידע ויגדילו את הזמן ואת עלות החישוב, לכן צריך למקם את הסנסור במיקום נכון ביחס לרכב על מנת לקבל תמונה מספיק רחבה אבל לא יותר מידי של מקום החנייה.

2. תכנון חנייה- חנייה במקביל או במאונך

- <u>חנייה במקביל</u>

האיור למטה מציג שרטוט של חנייה במקביל.

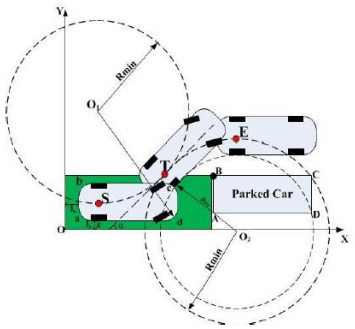
נקודות abcd מייצגות את המכונית האוטונומית ו-ABCD מייצגת את המכונית החונה "המכשול".

E = נקודת ההתחלה

T = סיבוב בציר שמותאם למכונית

S = נקודת הסיום

האלגוריתם מתכנן את נתיב הנסיעה בהינתן נקודה E (המקבילה לרכב החונה ABCD) תכנון הנסיעה אחורה נעשה על פי כיוון, זווית ומרחק עד ל T. בנקודה T נעשה שינוי הכיוון והזווית עד לנקודת הסיום – S.

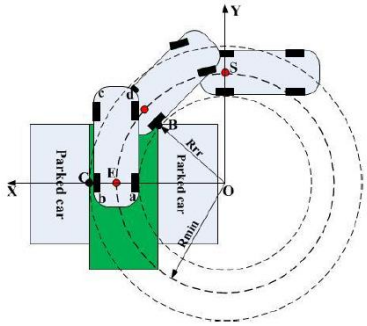


- <u>חנייה במאונך\ בניצב</u>

חנייה בניצב נעשית על פי השטח המצוי בהשוואה למקום החנייה.

האיור למטה מציג חנייה בניצב. הרכב הנתון נוסע אל מול החנייה בהתאמה לקווים המצויים.

לאחר שהתמקם בצורה מספקת יש נסיעה אחורה עד לטווח הבטיחות וסיום החנייה.



3. ביצוע החנייה-

לאחר שאושר כי החנייה תקינה ופנויה והמשתמש אישר את החנייה מתחילים בהכוונת הרכב– הזזת הגלגלים לימין או לשמאל ולאיזה כיוון נסיעה יש לבצע קדימה או אחורה. נמשיך לבצע לפי שלבי תכנון מסלול החנייה (זווית, כיוון ומהירות) עד לסיום התהליך - חניית הרכב בהצלחה.

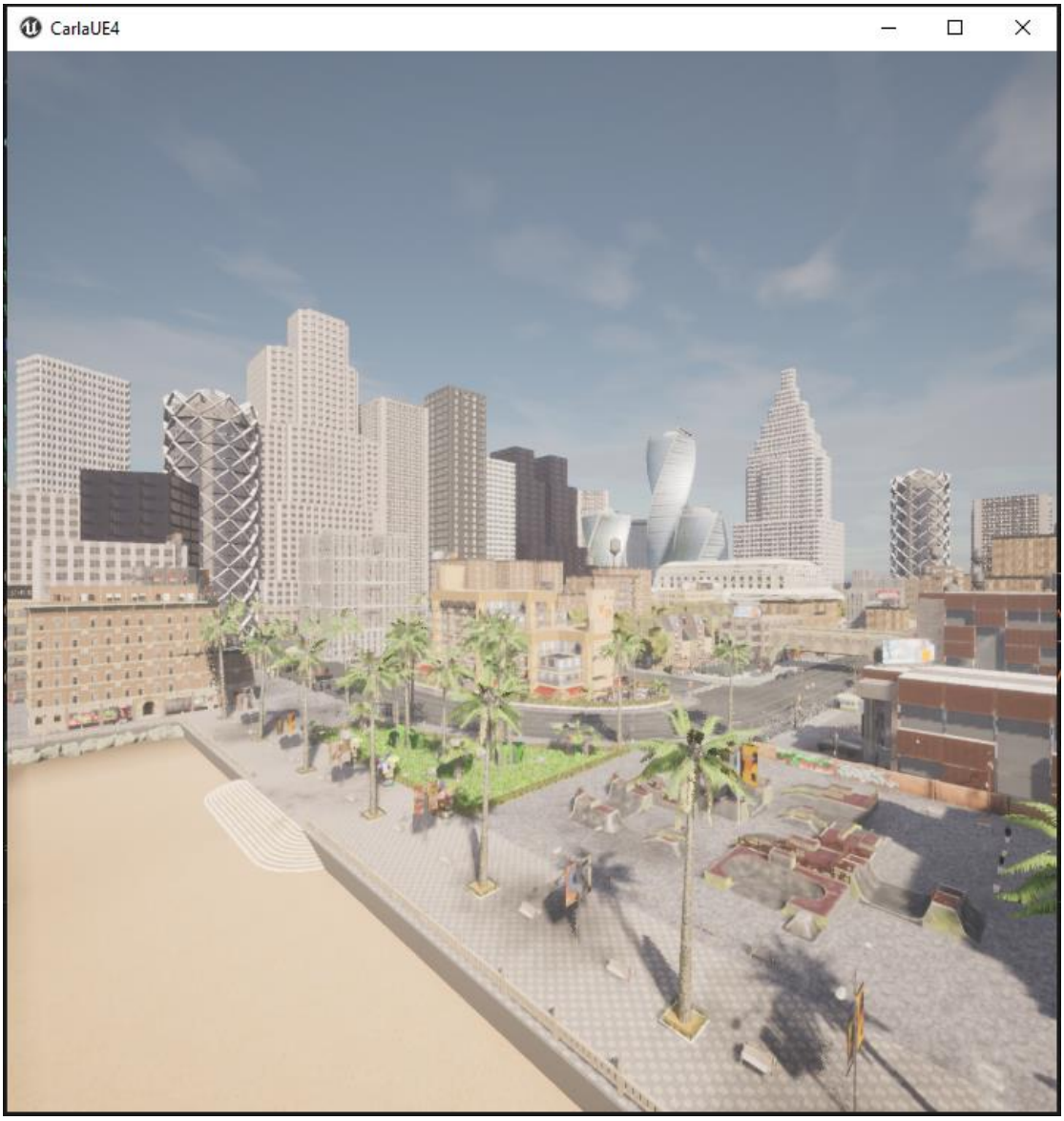
הסימולטור

גרסאות:

- CARLA 0.9.13
- Python 3.7

קונפיגורציות

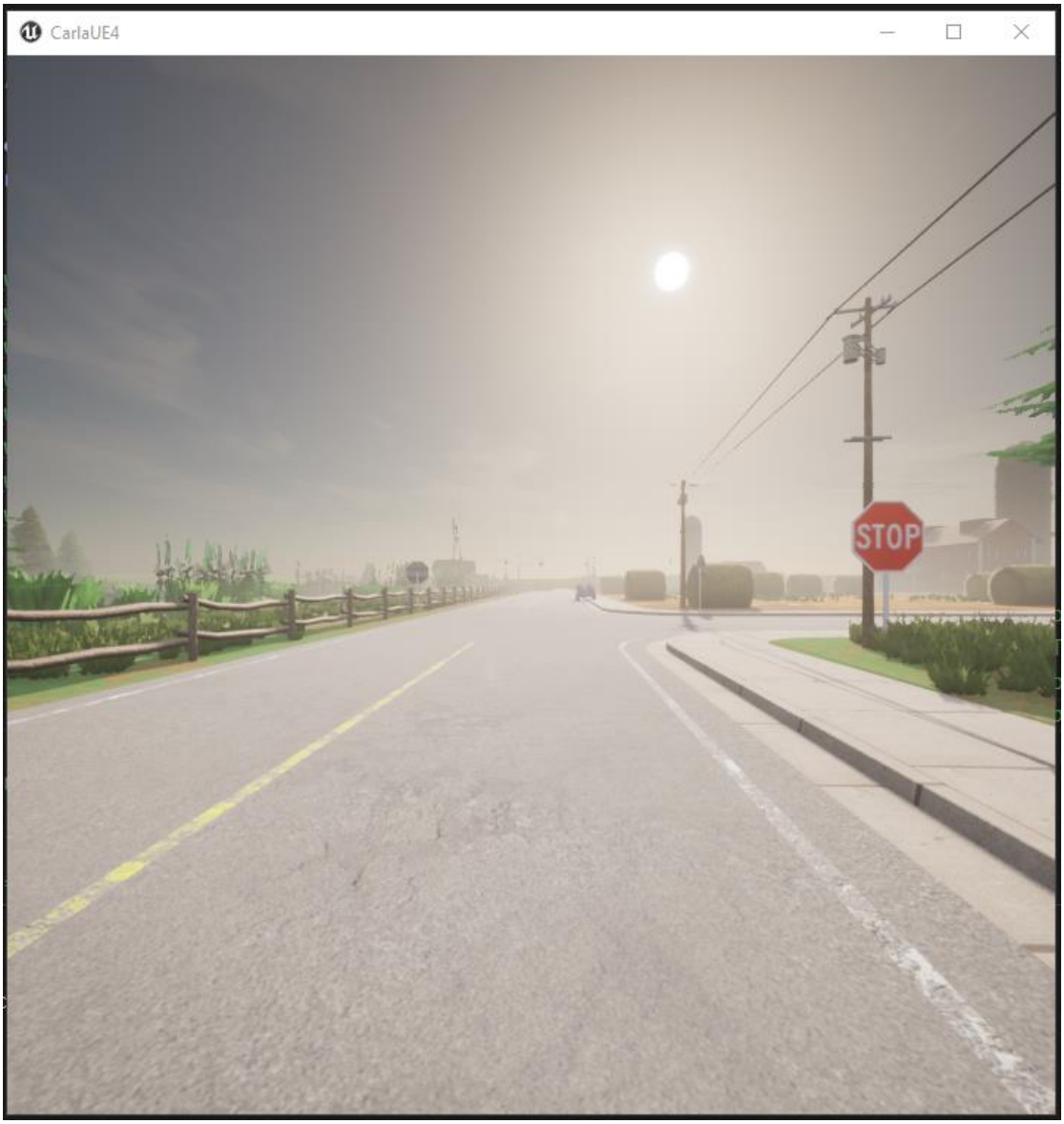
- שרת הסימולטור של CARLA עולה עם מפה דיפולטית "Town10HD_Opt".



רשימת המפות הקיימות:

- Town01, Town01_Opt
- Town02, Town02_Opt
- Town03
- Town06, Town06_Opt
- Town07, Town07_Opt
- Town10HD, Town10HD_Opt
- Town11

אחרי שעברתי על כל סוגי המפות הקיימות ומיפיתי את החניונים הקיימים בחרתי לפרויקט זה לעבוד על מפה Town07_Opt .



קוד להחלפת מפה:

change_map.py

```
client = carla.Client('127.0.0.1', 2000)
client.set_timeout(10.0)
client.load_world('Town07_Opt')
```

- עבור מפה זו מיפיתי את החניונים הרלוונטים, כשתי רשימות עבור חניון לחניות במקביל ועבור חניות בניצב.
  נמפה לפי מיקומים של x,y בעולם.

```
self.perpendicular_park_locations = []
max_spots = 18
spot_idx = 0
while spot_idx < max_spots:
   x = 29.5 + ((2.5 + 0.2)* spot_idx)
   park_pos = carla.Transform(carla.Location(x=-x, y=-70.5,  z=1),
carla.Rotation(yaw=90))
   self.perpendicular_park_locations.append(park_pos)
   spot_idx+=1

self.parallel_park_locations = []
max_spots = 10
spot_idx = 0
while spot_idx < max_spots:
  y = 53 - ((4.5 + 0.3)* spot_idx)
  park_pos = carla.Transform(carla.Location(x=-8, y=-y,  z=1),
carla.Rotation(yaw=90))
  self.parallel_park_locations.append(park_pos)
  spot_idx+=1
```

- הנחת הרכב בעולם:
  - לפי מבט הצופה-

```
spectator = self.world.get_spectator()
init_pos = spectator.get_transform()
```

  - נקודה קבועה בעולם-

```
init_pos= carla.Transform(carla.Location(x=-5, y=-60, z=1), carla.Rotation(yaw=90))
```

- גילוי חנייה (פנוי או תפוס): detect_parking_spot.py
  - הגדרת enum:

```
class ParkType(Enum):
    none = 0
    perpendicular = 1
    paralel = 2

class ParkSpaceType(Enum):
    none = 0
    free = 1
    occupaid = 2
```

  - גילוי חנייה בניצב

```
def get_perpendicular_park_spot_status(image, width, height):
    h = math.floor(height / 3) * 2

    parkStatusIn = get_perpendicular_park_status_in(image, width, h)
    while(parkStatusIn != ParkSpaceType.occupaid):
        h = h - 20
        parkStatusIn = get_perpendicular_park_status_in(image, width, h)
        if h < math.floor(height / 3):
            break
    return parkStatusIn

def get_perpendicular_park_status_in(image, width, h):
    helf_width = math.floor(width / 2)
    found_status = ParkSpaceType.none
    for w in range(helf_width, 0, -1):
        if all(image[h][w] == [142, 0, 0]):
            found_status = ParkSpaceType.occupaid
            break
        if all(image[h][w] == [50, 234, 157]):
            found_status = ParkSpaceType.free
            break
    if found_status == ParkSpaceType.free:
        found_status = ParkSpaceType.none
        for w in range(helf_width, width - 1, 1):
            if all(image[h][w] == [142, 0, 0]):
                found_status = ParkSpaceType.occupaid
                break
            if all(image[h][w] == [50, 234, 157]):
                found_status = ParkSpaceType.free
                break

    return found_status
```

עבור סוג חנייה זה- חנייה בניצב, נבצע עיבוד תמונה כדי לגלות האם החנייה תפוסה או פנויה בצורה זו:

נתחיל בגובה 2\3 מהתמונה ונרד כל פעם ב20 פיקסלים על ל1\3 גובה של התמונה, בכל שלב כזה נבצע תחילה סריקה של השורה מהאמצע לתחילת התמונה ונבדוק עבור כל פיקסל את ערך שלו, אם הערך הוא [142, 0, 0] זה אומר שמצאנו רכב בחנייה שלנו והחנייה תפוסה, במידה ומצאנו [50, 234, 157] זה אומר שמצאנו את הקו שמסמן את גבולות החנייה ולכן החנייה בצד זה פנויה, בכל מקרה אחר לא מצאנו חנייה. אם הצד השמאלי של החנייה פנוי נמשיך לבדוק את הצד הימני אחרת נצא. נבצע סריקה נוספת של השורה מהאמצע לסוף התמונה ונבצע את אותה הבדיקה. אם החנייה משני הכיוונים פנויה – מצאנו קווי חנייה ללא רכב באמצע נחזיר שהחנייה פנויה.

- גילוי חנייה במקביל

```
def get_paralel_park_spot_status(image, width, height):
    h = math.floor(height / 3)
    parkStatusIn = get_paralel_park_status_in(image, width, h)
    while(parkStatusIn != ParkSpaceType.occupaid):
        h = h + 20
        if h >= math.floor(height):
            break
        parkStatusIn = get_paralel_park_status_in(image, width, h)
    if parkStatusIn != ParkSpaceType.occupaid:
        parkStatusIn = get_paralel_park_status_in_line(image, width, height)
    return parkStatusIn

def get_paralel_park_status_in(image, width, h):
    helf_width = math.floor(width / 2)
    found_status = ParkSpaceType.none
    for w in range(helf_width, 0, -1):
        if all(image[h][w] == [142, 0, 0]):
            found_status = ParkSpaceType.occupaid
            break
    if found_status == ParkSpaceType.none:
        found_status = ParkSpaceType.none
        for w in range(helf_width, width - 1, 1):
            if all(image[h][w] == [142, 0, 0]):
                found_status = ParkSpaceType.occupaid
                break
    return found_status

def get_paralel_park_status_in_line(image, width, height):
    helf_height = math.floor(height / 2)
    helf_width = math.floor(width / 2)
    found_status = ParkSpaceType.none
    for h in range(helf_height, height, 1):
        if all(image[h][helf_width] == [142, 0, 0]):
            found_status = ParkSpaceType.occupaid
```

```
            break
        if all(image[h][helf_width] == [50, 234, 157]):
            for w in range(0, width, 1):
                if all(image[h][w] == [142, 0, 0]):
                    found_status = ParkSpaceType.occupaid
                    break
            if found_status == ParkSpaceType.none:
                found_status = ParkSpaceType.free
                break
    return found_status
```

עבור סוג חנייה זה- חנייה במקביל, נבצע עיבוד תמונה כדי לגלות האם החנייה תפוסה או פנויה בצורה זו:

נתחיל בגובה 1\3 מהתמונה ונעלה כל פעם ב20 פיקסלים עד לגובה הכולל של התמונה, בכל שלב כזה נבצע תחילה סריקה של השורה מהאמצע לתחילת התמונה ונבדוק עבור כל פיקסל את ערך שלו, אם הערך הוא [142, 0, 0] זה אומר שמצאנו רכב בחנייה שלנו והחנייה תפוסה אחרת עדיין לא נמצאה חנייה. אם עבור הצד השמאלי של החנייה לא נמצא חנייה נמשיך לבדוק את הצד הימני אחרת נצא. נבצע סריקה נוספת של השורה מהאמצע לסוף התמונה ונבצע את אותה הבדיקה. אם החנייה משני הכיוונים לא תפוסה נמשיך לשלב הבא, בשלב זה מתבצעת סריקה של התמונה המתחילה בגובה 1\2 מהתמונה ונעלה כל פעם ב1 פיקסל עד לגובה הכולל של התמונה, נסתכל על אמצע רוחב התמונה בכל שורה ונבדוק את ערך הפיקסל שלו. אם הערך הוא [142, 0, 0] זה אומר שמצאנו רכב בחנייה שלנו והחנייה תפוסה, במידה ומצאנו [50, 234, 157] זה אומר שמצאנו את גובה הקו שמסמן את גבולות החנייה, נבדוק לאורך כל התמונה בגובה זה שאין לנו רכב שמפריע לחנייה. במקרה והכל תקין זה אומר שהחנייה אכן פנויה.

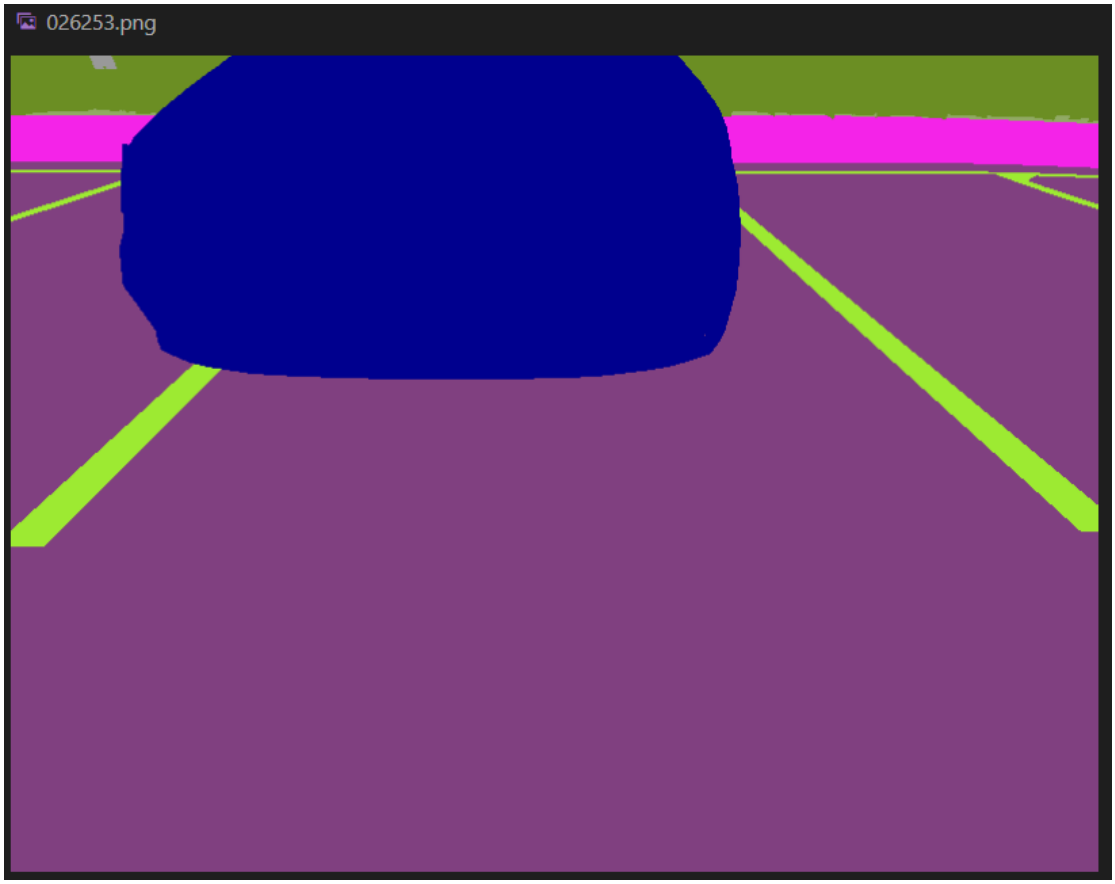
- גילוי חנייה כללי

```
def get_park_spot_status(park_type, image, width, height):
    if park_type == ParkType.perpendicular:
        return get_perpendicular_park_spot_status(image, width, height)
    elif park_type == ParkType.paralel:
        return get_paralel_park_spot_status(image, width, height)
    return ParkSpaceType.none
```

- הרצת כסקריפט עצמאי, לצורך בדיקות לפני הרצת הסימולציה. קריאת תמונה PNG, ובדיקה האם החנייה בניצב\ במקביל פנויה או תפוסה.

```
image = cv2.imread('026253.png')
height = image.shape[0]    # image height
width = image.shape[1]     # image width
status = get_perpendicular_park_spot_status(image, width, height)
print(status)
```

לדוגמא, עבור תמונה זו יודפס -> ParkSpaceType.occupaid



```
image = cv2.imread('024298.png')
height = image.shape[0]    # image height
width = image.shape[1]    # image width
status = get_paralel_park_spot_status(image, width, height)
print(status)
```

לדוגמא, עבור תמונה זו יודפס -> ParkSpaceType.free

- תהליך החנייה park.py

תהליך החנייה משתמש בתהליך גילוי החנייה שהגדרנו קודם לכן.

```
from detect_parking_spot import ParkSpaceType, ParkType, get_park_spot_status
```

```
class CarlaParkVehicle():
```

לצורך התהליכים הפנימיים הגדרתי משתנים במחלקה:

- search_parking_spot_enabled: Boolean

האם התחלנו תהליך חיפוש חנייה

- find_parking_spot_enabled: Boolean

האם מצאנו חנייה פנויה

- start_parking_enabled: Boolean

האם התחלנו תהליך חנייה

- parking_success: Boolean

האם הצלחנו לבצע את החנייה

- park_found_status: ParkSpaceType

האם החנייה פנויה\תפוסה\לא קיימת

הגדרתי מראש עבור החניות רשימת מיקומים לפי סוג מקבילי\ניצב:

- perpendicular_park_locations
- parallel_park_locations

מחלקת תהליך החנייה מרכזת את כל התהליך של חיפוש וביצוע החנייה עבור חנייה במקביל\בניצב. לשם כן הגדרתי 4 פונקציות מרכזיות שנשתמש בהן:

- התחלת חיפוש חנייה

```
  def start_serach_parking(self):
      self.search_parking_spot_enabled = True
      self.serach_parking()
```

עדכון משתנה האם התחלנו תהליך חיפוש חנייה לאמת והתחלת תהליך חיפוש חנייה.

```
def serach_parking(self, ignore_spots = []):
    pos, park_type = self.get_closest_parking_spot(ignore_spots)
    if pos is not None and park_type != ParkType.none:
        print("parking spot found")
        print(pos)
        if park_type == ParkType.perpendicular:
            self.move_to_front_perpendicular_parking_spot(pos)
```

```
        elif park_type == ParkType.paralel:
            self.move_to_front_paralel_parking_spot(pos)

        # check if parking spot is valid
        # in case valid- park
        # else return over this process- find the next near parking spot
        self.activate_sensors(park_type)
        while self.search_parking_spot_enabled == True and self.park_found_status
== ParkSpaceType.none:
            time.sleep(0.1)

        if self.search_parking_spot_enabled != True:
            self.park_found_status = ParkSpaceType.none
        elif self.park_found_status == ParkSpaceType.free:
            self.park_spot_data = pos, park_type
            self.vehicle_pos = self.vehicle.get_transform()
            self.find_parking_spot_enabled = True
        else:
            self.park_found_status = ParkSpaceType.none
            ignore_spots.append(pos)
            self.serach_parking(ignore_spots)
    else:
        print("parking spot not found")
        self.park_found_status = ParkSpaceType.none
        self.search_parking_spot_enabled = False
```

חיפוש חנייה קרובה אם נמצאה- נבצע נסיעה לכיוון החנייה, נפעיל את הסנסורים לקבלת מידע ונחכה לקבלת עדכון. במידה ומצאנו חנייה פנויה נעדכן משתנה האם מצאנו חנייה פנויה לאמת ונשמור את מיקום וסוג החנייה. במידה והחנייה תפוסה נחזור שוב על התהליך ונתעלם מהמיקום הנוכחי שנמצא. במידה ולא קיימת חנייה קרובה נעדכן את משתנה האם התחלנו תהליך חיפוש חנייה לשקר ובכך הסתיים תהליך חיפוש חנייה.

- קבלת מיקום החנייה הקרובה ביותר למיקום הרכב

```
def get_closest_parking_spot(self, ignore_spots):
    curr_vehicle_loc = self.vehicle.get_location()

    # Initialize variables to track the closest parking position and its distance
    closest_perpendicular_parking_position = None
    closest_parallel_parking_position = None
    min_distance_perpendicular = float('inf')  # Initialize to positive infinity
    min_distance_parallel = float('inf')  # Initialize to positive infinity

    # Calculate the distance from the vehicle to each parking position and find the
closest one
    for parking_position in self.perpendicular_park_locations:
        if self.search_parking_spot_enabled == False:
            closest_perpendicular_parking_position = None
```

```
        elif parking_position not in ignore_spots:
            distance = self.calculate_distance(curr_vehicle_loc,
parking_position.location)
            if distance < min_distance_perpendicular and distance > 4 and distance < 15:
                min_distance_perpendicular = distance
                closest_perpendicular_parking_position = parking_position

    for parking_position in self.parallel_park_locations:
        if self.search_parking_spot_enabled == False:
            closest_parallel_parking_position = None
        elif parking_position not in ignore_spots:
            distance = self.calculate_distance(curr_vehicle_loc,
parking_position.location)
            if distance < min_distance_parallel and distance > 5 and distance < 10:
                min_distance_parallel = distance
                closest_parallel_parking_position = parking_position

    if closest_perpendicular_parking_position or closest_parallel_parking_position:
        if min_distance_perpendicular < min_distance_parallel:
            return closest_perpendicular_parking_position , ParkType.perpendicular
        else:
            return closest_parallel_parking_position, ParkType.paralel
    else:
        return None, ParkType.none

def calculate_distance(self, location1, location2):
    return math.sqrt((location1.x - location2.x)**2 + (location1.y - location2.y)**2)
```

בדיקת מיקום הרכב מול רשימת מיקומי החניות תוך התחשבות במינימום ומקסימום אפשרי לביצוע החנייה. מחזיר את מיקום החנייה וסוג החנייה.

- קבלת נתוני סנסור תמונה RGB

```
def process_camera_data_rgb(self, image, sensor_data):
    image.save_to_disk('{}/{}/{:06d}.png'.format(TEMP_OUTPUT_PATH,
sensor_data, image.frame))
    self.deactivate_sensor_by_name(sensor_data)
```

לוקח את הפריים הראשון שמתקבל ושומר בתיקיה. מבטל האזנה לסנסור.

- קבלת נתוני סנסור תמונה SS

```
def process_camera_data_ss(self, image, sensor_data, park_type):
    image.convert(cc.CityScapesPalette)
    array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
    array = np.reshape(array, (image.height, image.width, 4))
    rgb_array = array[:, :, :3]
    image.save_to_disk('{}/{}/{:06d}.png'.format(TEMP_OUTPUT_PATH,
sensor_data, image.frame))
    print('frame - ', image.frame)
```

```
        self.park_found_status = get_park_spot_status(park_type, rgb_array,
image.width, image.height)
        print('park_found_status - ', self.park_found_status)
        self.deactivate_sensor_by_name(sensor_data)
```

לוקח את הפריים הראשון שמתקבל, ממיר את הצבעים שלו ויוצר מערך חדש בדומה לקריאת תמונה מקובץ (אותו מבנה), שומר את התמונה בתיקיה ומבצע בדיקה על התמונה האם התמונה היא של חנייה פנויה או תפוסה לפי סוג החנייה מקבילי או ניצב. בסוף מבטל האזנה לסנסור.

- עצירת חיפוש חנייה

```
def stop_serach_parking(self):
    self.search_parking_spot_enabled = False
    self.find_parking_spot_enabled = False
    self.deactivate_sensors()
```

נעדכן את המשתנים של חיפוש ומציאת חנייה לשקר ומבטל האזנה לסנסורים.

- התחלת תהליך חנייה

```
def start_parking(self):
    if self.find_parking_spot_enabled == True and self.park_spot_data is not None:
        self.start_parking_enabled = True
        pos, park_type = self.park_spot_data
        if park_type == ParkType.perpendicular:
            self.move_to_near_perpendicular_parking_spot(pos)
            self.park_perpendicular(pos)
        elif park_type == ParkType.paralel:
            self.move_to_near_paralel_parking_spot(pos)
            self.park_paralel(pos)
```

לאחר שביצענו חיפוש חנייה ומצאנו חנייה פנויה שמרנו את נתוני החנייה. בשלב זה ניתן להתחיל בתהליך החנייה, נעדכן משתנה האם התחלנו תהליך חנייה לאמת ולפי סוג החנייה ומיקום החנייה ששמרנו קודם לכן נזיז את הרכב למיקום המתאים לתחילת ביצוע החנייה ואז התחלת תהליך החנייה עצמו.

- חנייה במקביל

```
def park_paralel(self, park_pos):
    # Save the current stdout for later use
    original_stdout = sys.stdout
    with open('output-park-paralel.log', 'w') as file:
        # Redirect stdout to the file
        sys.stdout = file
        print('park_pos.location')
        print(park_pos.location)
        print('back right')
        steer=0.8 #right
        while self.start_parking_enabled == True:
            location = self.vehicle.get_location()
```

```
            if (location.y < park_pos.location.y +  0.5 and location.y >
park_pos.location.y -  0.5 ) and (location.x < park_pos.location.x +  0.05 and location.x
> park_pos.location.x -  0.05 ):
                print('park success')
                self.parking_success = True
                break
            if (location.x < park_pos.location.x + 0.1 and location.x > park_pos.location.x
- 0.1 ) and (abs(self.vehicle.get_transform().rotation.yaw) < 91 and
abs(self.vehicle.get_transform().rotation.yaw) > 89):
                steer=0.0
                print('dir')
            # elif location.x < park_pos.location.x + 2.35:
            elif location.x < park_pos.location.x + 2.5 and location.y <
park_pos.location.y + 2 :
                steer=-0.75 #left
                print('back left')

            print(self.vehicle.get_transform())
            self.vehicle.apply_control(
            carla.VehicleControl(throttle=0.35, steer=steer, brake=0.0, reverse=True))
            time.sleep(0.05)
        print('brake')
        self.vehicle.apply_control(carla.VehicleControl(throttle=0.0, brake=1.0))
        time.sleep(0.1)
        # Restore the original stdout
        sys.stdout = original_stdout
```

לפי מיקום הרכב הנמצא במצב התחלתי כמו בתמונה, נבצע נסיעה אחורה ואת הגלגלים נפנה ימינה לאחר שנגיע לנקודת השבירה נפנה את הגלגלים שמאלה וניישר אותם אחרי שנתיישר עד להגעה למיקום החנייה ונעצור.

apply_control – שליטה ותפעול הרכב

steer=0.8 - גלגלים ימינה

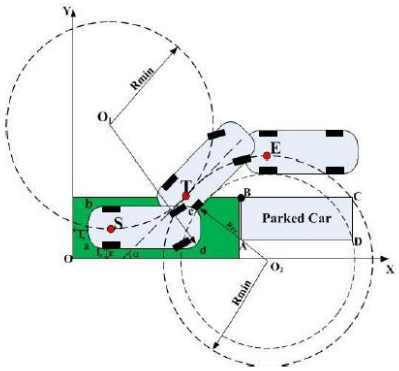
steer=-0.75 - גלגלים שמאלה

steer=0.0- יישור הגלגלים

throttle=0.35- מהירות נסיעה, מינימום מהירות נסיעה כדי שהרכב יצליח לעלות על המדרכה.

reverse=True – נסיעה אחורה, רוורס.

brake=1.0- עצירה של הרכב



- חנייה בניצב

```
def park_perpendicular(self, park_pos):
    # Save the current stdout for later use
    original_stdout = sys.stdout
    with open('output-park_perpendicular.log', 'w') as file:
        # Redirect stdout to the file
        sys.stdout = file
        print('park_pos.location')
        print(park_pos.location)
        print('back right')
        steer=1.0
        while self.start_parking_enabled == True:
            location = self.vehicle.get_location()
            if location.y < park_pos.location.y + 0.05 and location.y >
park_pos.location.y - 0.05 :
                print('park success')
                self.parking_success = True
                break
            if self.vehicle.get_location().x > park_pos.location.x and
abs(self.vehicle.get_transform().rotation.yaw) <= park_pos.rotation.yaw:
                steer = 0.0
                print('dir')
            print(self.vehicle.get_transform())
            self.vehicle.apply_control(
                carla.VehicleControl(throttle=0.3, steer=steer, brake=0.0, reverse=True))
            time.sleep(0.05)
        print('brake')
        self.vehicle.apply_control(carla.VehicleControl(throttle=0.0, brake=1.0))
        time.sleep(0.1)
        # Restore the original stdout
        sys.stdout = original_stdout
```

לפי מיקום הרכב הנמצא במצב התחלתי כמו בתמונה, נבצע נסיעה אחורה ואת הגלגלים נפנה ימינה לאחר שנגיע לנקודת השבירה ניישר את הגלגלים ונמשיך נסיעה אחורה עד להגעה למיקום החנייה ונעצור.

apply_control – שליטה ותפעול הרכב

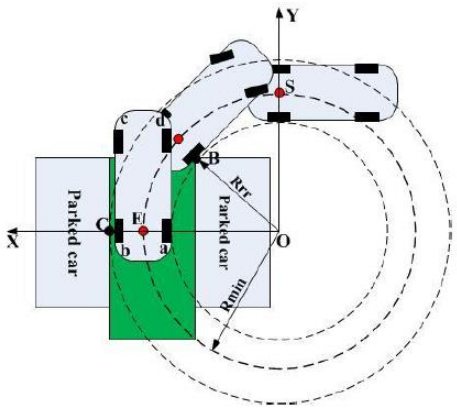
steer=1.0 - גלגלים ימינה

steer=0.0- יישור הגלגלים

throttle=0.3- מהירות נסיעה

reverse=True – נסיעה אחורה, רוורס.

brake=1.0- עצירה של הרכב



- עצירת תהליך חנייה

```
def stop_parking(self):
    self.start_parking_enabled = False
```

נעדכן משתנה האם התחלנו תהליך חנייה לשקר ובכך נעצור את תהליך החנייה.

- ניקוי

איפוס המשתנים וניקוי השחקנים : הסנסורים, הרכבים החונים במידה וקיימים.

- לוגים לקובץ

בתהליך החנייה היו הרבה שלבים וכדי לוודא את הכיוון והמיקום של הרכב ביחס לחנייה עשיתי הדפסות רבות. הדפסות אלה לconsole היו כבדות והיו נעלמות בסגירת התהליך.
כדי לייעל תהליך זה, הייתי צריכה את האפשרות להדפיס לקובץ.

```
import sys

# Save the current stdout for later use
original_stdout = sys.stdout

# Open a file in write mode
with open('output.log', 'w') as file:
    # Redirect stdout to the file
    sys.stdout = file

    # Now all print statements will be written to the file
    print("Hello, this will be written to the file.")

    # Restore the original stdout
    sys.stdout = original_stdout

# This will be printed to the console
print("This will be printed to the console.")
```

בדוגמא זו מוצג איך מתבצע תהליך ההדפסה לקובץ ע"י דריסה של משתנה sys.stdout.

- חנייה שרת

הרצת סקריפט זה עובד על צד שרת בלבד, ניתן להריץ מספר פעמים את אותו הסקריפט כך שעל גבי השרת יוצרו מספר רכבים שחונים אחד אחרי השני לפי המקום הכי קרוב אליהם.

park_server.py

```
class CarlaServerParkVehicle():
```

- אתחול

בעת אתחול המחלקה נוצר חיבור מול שרת CARLA, לאחר מכן ניצור רכב במיקום מסוים בעולם ונאתחל את מחלקת תהליך החנייה CarlaParkVehicle.

- הרצה

בשלב זה נריץ במחלקת תהליך החנייה את תהליך חיפוש החנייה, נמתין עד לקבלת תשובה – האם קיים מקום חנייה קרוב פנויה. לאחר קבלת התשובה נריץ במחלקת תהליך החנייה את ביצוע החנייה.

- ניקוי

נבצע ניקוי השחקנים, במקרה שלנו את הרכב שייצרנו בשלב האתחול ונבצע ניקוי עבור מחלקת תהליך החנייה.

- חנייה שרת-לקוח

כחלק מתהליך חניית שרת לקוח, יצאתי מדוגמא Open Source של CARLA הנקרא manual_control שמספקת נהיגה ידנית בעולם. לתהליך זה שילבתי את יכולות החנייה ע״י הפניה למחלקת תהליך החנייה, הפנייה זו מתבצעת ע״י יצירת תהליך נפרד thread על מנת לא לגרום לתקיעת הלקוח.

park_manual_control.py

- נגביל את אפשרות סוגי הרכבים ליצירת רכב מסוג אחד : 'vehicle.audi.a2'.
- הוספת מידע לתפריט תמיכה במקשים :

F : toggle find parking spot

CTRL + F : toggle start parking

class KeyboardControl

- האזנה למקשי מקלדת :

```
def parse_events(self, client, world, clock, sync_mode):
…
elif event.key == K_f and not pygame.key.get_mods() & KMOD_CTRL:
                if self._autopilot_enabled:
                    world.hud.notification('Can not search for parking spot while autopilot mode in on')
                    return
                if world.park_manager.search_parking_spot_enabled == False:
                    world.search_park_thread = SingleThread(target=world.park_manager.start_serach_parking, clear_target=world.park_manager.stop_serach_parking)
                    world.search_park_thread.start()
                    world.hud.notification('Start searching for parking spot ')
                elif world.search_park_thread is not None and world.park_manager.search_parking_spot_enabled == True:
                    world.search_park_thread.stop()
                    world.search_park_thread = None
                    world.hud.notification('Stop searching for parking spot ')
            elif event.key == K_f and pygame.key.get_mods() & KMOD_CTRL:
                if world.park_manager.search_parking_spot_enabled == True and world.park_manager.find_parking_spot_enabled == True and world.park_manager.start_parking_enabled == False:
```

```
                    world.park_thread =
SingleThread(target=world.park_manager.start_parking,
clear_target=world.park_manager.stop_parking)
                    world.park_thread.start()
                    world.hud.notification('Start park proccess')
                elif world.park_thread is not None and
world.park_manager.start_parking_enabled == True:
                    world.park_thread.stop()
                    world.park_thread = None
                    world.hud.notification('Stop park proccess')
                else:
                    world.hud.notification('Park is not allowed')
```

בעת לחיצה על מקש F : התחלת חיפוש חנייה

במידה והרכב במצב נסיעה אוטומטי AutoPilot לא ניתן להתחיל לחפש חנייה, יוצג הודעה מתאימה למשתמש.

במידה ולא התחלנו כבר תהליך חיפוש חנייה נאתחל את תהליך חיפוש חנייה search_park_thread ונריץ.

- בעקבות התחלת חיפוש, הקוד על הצגת הודעה על מציאה או אי מציאת מקום חנייה מופיע בפונקצית world.render

בעת לחיצה נוספת על מקש F : ביטול תהליך חיפוש חנייה

במידה והתחלנו כבר תהליך חיפוש חנייה והתהליך כבר מאותחל search_park_thread נעצור את התהליך ונאפס את המשתנה של התהליך.

בעת לחיצה על מקש CTRL + F : התחלת ביצוע חנייה

במידה וביצענו כבר את תהליך חיפוש החנייה והוא הסתיים ונמצאה חנייה פנויה נאתחל את תהליך ביצוע החנייה park_thread ונריץ.

בעת לחיצה נוספת על מקש CTRL + F : ביטול תהליך ביצוע חנייה

במידה והתחלנו כבר תהליך ביצוע החנייה והתהליך כבר מאותחל park_thread נעצור את התהליך ונאפס את המשתנה של התהליך.

בכל אחד מהתהליכים למעלה יוצג הודעה מתאימה למשתמש.

- מניעת נסיעה תוך כדי אחד מתהליכי החנייה

```
if not self._autopilot_enabled and not world.search_park_thread and not  world.park_thread:
    ...
    if self._control.throttle > 0 and world.park_manager.find_parking_spot_enabled == True:
        world.park_manager.clear_park_res()
```

במקרה בו הורץ אחד מתהליכי החנייה: חיפוש חנייה או ביצוע חנייה, המערכת שולטת ברכב ומנהגת אותו בהתאם לצרכיה. במקרה זה לא נרצה הפרעות מבחוץ לדוגמא שהמשתמש יזיז לכיוון אחר את הרכב תוך כדי התהליך. לכן במצב זה כמו במצב נהיגה אוטומטי AutoPilot יש למנוע מהמשתמש את אפשרות השליטה ברכב. אפשר בכל שלב לעצור את התהליכים ולהמשיך נסיעה ידנית של המשתמש.
במקרה שהתחלנו תהליך חיפוש והוא מסתיים בהצלחה המשתמש נדרש ללחוץ על כפתור נוסף כדי לאשר ביצוע חנייה, במידה והמשתמש יזיז את הרכב, כל תוצאות החנייה שנמצאו יאותחלו- במקרה זה המשתמש יצטרך להתחיל מחדש את תהליך החיפוש והחנייה.

class World

- איתחול

בעת איתחול המחלקה נוסיף משתנה park_manager עבור מחלקת תהליך החנייה CarlaParkVehicle.

- restart

```
def restart(self):
        ...
    self.park_manager = CarlaParkVehicle(self.world,  self.player)
    self.search_park_thread = None
    self.park_thread = None
```

בשלב זה ניצור רכב במיקום מסויים בעולם player. נאתחל את מחלקת תהליך החנייה CarlaParkVehicle, ו 2 משתנים נוספים עבור תהליך חיפוש החנייה search_park_thread ותהליך ביצוע החנייה park_thread.

- render

```
def render(self, display):
    if self.search_park_thread is not None and self.search_park_thread.stopped():
        self.hud.notification('Parking spot %s' % ('found' if
self.park_manager.find_parking_spot_enabled else 'not found'))
        self.search_park_thread = None

    if self.park_thread is not None and self.park_thread.stopped() and
self.park_manager.parking_success == True:
        self.hud.notification('Park success')
        self.park_manager.clear_park_res()
        self.park_thread = None

    self.camera_manager.render(display)
    self.hud.render(display)
```

כדי להמשיך לעדכן את התצוגה של הלקוח ולהציג התראות בהתאם לתהליכי החנייה הרצים במקביל, נוסיף בחלק זה בדיקה האם התהליך הסתיים ונציג התראה מתאימה.

- ניקוי destroy

```
def destroy(self):
    if self.search_park_thread is not None:
        self.search_park_thread.stop()
    if self.park_thread is not None:
        self.park_thread.stop()
    if self.park_manager is not None:
        self.park_manager.destroy()

    if self.radar_sensor is not None:
        self.toggle_radar()
    sensors = [
        self.camera_manager.sensor,
        self.collision_sensor.sensor,
        self.lane_invasion_sensor.sensor,
        self.gnss_sensor.sensor,
        self.imu_sensor.sensor]
    for sensor in sensors:
        if sensor is not None:
            sensor.stop()
            sensor.destroy()
    if self.player is not None:
        self.player.destroy()
```

נבצע עצירת תהליכי החנייה במידה והם רצים בשלב זה thread.stop(), בנוסף נבצע ניקוי עבור מחלקת תהליך החנייה park_manager.

single_thread.py

```python
import threading

class SingleThread(threading.Thread):
    def __init__(self, target, clear_target=None, *args, **kwargs):
        super(SingleThread, self).__init__(target=target, args=args, kwargs=kwargs)
        self._stop_event = threading.Event()
        self._cleared_event = threading.Event()
        self._clear_target = clear_target

    def run(self):
        if self._target is not None:
            self._target(*self._args, **self._kwargs)
        self._cleared_event.set()
        self._stop_event.set()

    def clear(self):
        if self._clear_target is not None and not self._cleared_event.is_set():
            self._clear_target()
            self._cleared_event.set()

    def stop(self):
        self.clear()
        self._stop_event.set()

    def stopped(self):
        return self._stop_event.is_set()
```

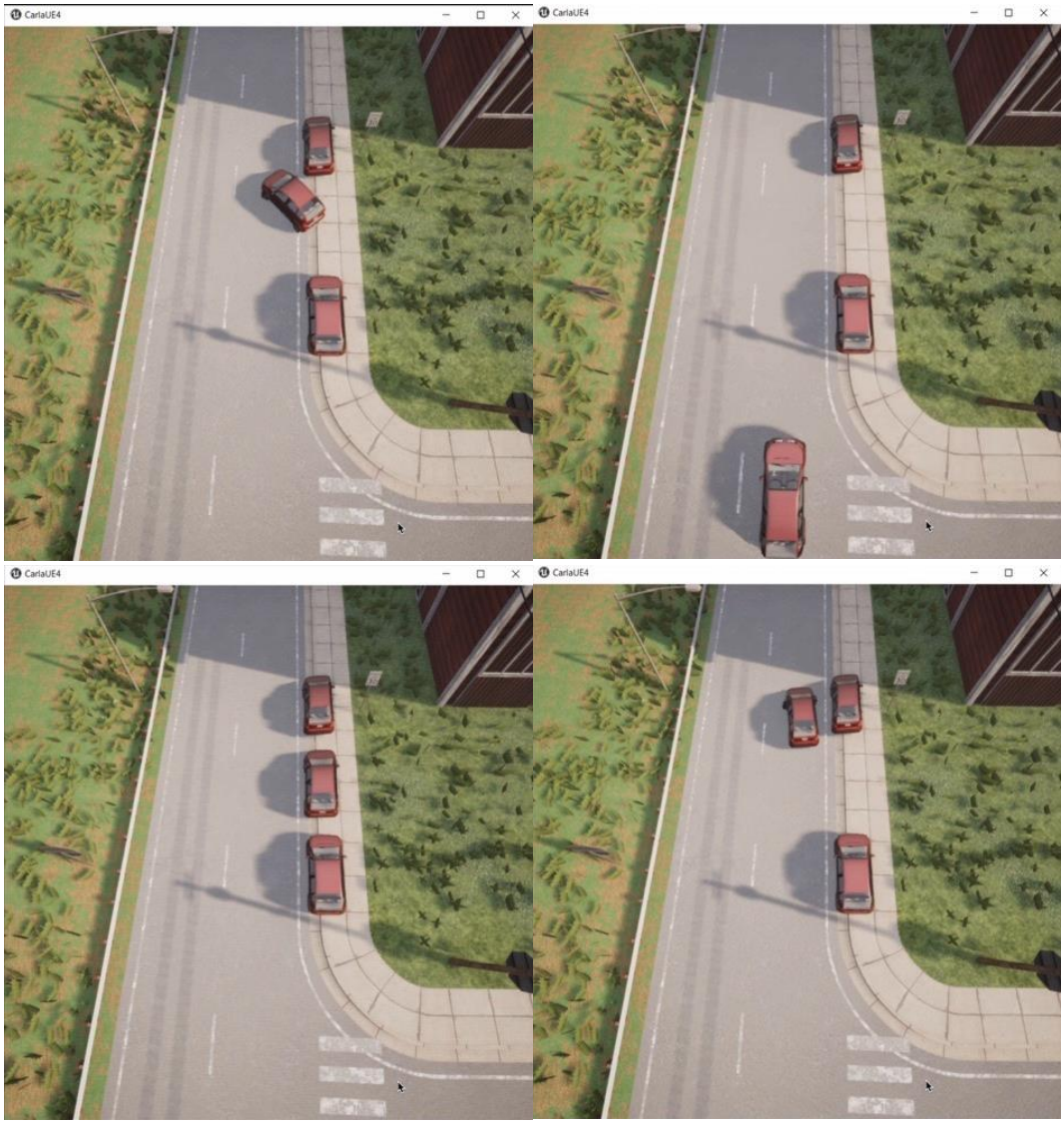
SingleThread - תהליך שעבורו מגדירים מראש פונקצית מטרה target ופונקצית ניקוי clear.

thread.start() מריץ את פונקצית המטרה.

thread.stop() מריץ את פונקצית הניקוי לפני עצירת התהליך.

thread.stopped() מחזיר האם התהליך הסתיים, אחרי סיום הרצת התהליך או עצירתו ידנית.

חנייה במקביל



חנייה בניצב


CarlaUE4
CarlaUE4
CarlaUE4
CarlaUE4


Start searching for parking spot

סיכום

פרויקט זה נועד להוות תשתית לחקירת אלגוריתם חנייה הן בניצב והן במקביל.
הסימולטור – CARLA מהווה תשתית גראפית להצגת העולם, מקרים ותרחישים שעל גביו נכתב קוד שמאפשר את הזזת העצמים השונים בעולם וכתיבת אלגוריתמים מתאימים.

פרויקט זה מהווה את החלק המעשי כבסיס לחלק התיאורטי של העבודה המסכמת.
תהליך הפיתוח היה מאתגר ומעניין ברמת הכרת תשתית גראפית חדשה (CARLA). תהליך ההתחברות והתממשקות לסימולציה היה מאתגר ברמת התכנות וברמת ארכיטקטורת התוכנה.

<u>תהליך הלמידה וכתיבת הפרויקט</u>
תחום הרכבים האוטונומיים הוא תחום חדשני, מעניין ומלא באתגרים.
נדרש ללמוד על עולם תוכן חדש, הכולל פיתוח בסביבה מורכבת והכרת חלקים תיאורטיים בתהליך החנייה והרכבים האוטונומיים, למשל: סוגי חנייה, זוויות חנייה וסנסורים שונים על מנת להגיע לתוצאה הרצויה של חנייה אוטונומית.

בפרויקט זה רציתי להתמקד בעולם התוכנה שמסקרן אותי ולהתפתח בו ולהעמיק יותר.
ברמה ההנדסית למדתי להתממשק לממשק גראפי חדש, השתמשתי בשפת תוכנה חדשה עבורי- python. התמקצעתי במידול הקוד בשימוש חוזר לכמה אופנים (כפי שמתואר בקטעי קוד), לדוגמה: שימוש בסקריפט עצמאי לבדיקות ולהפניה מקוד המערכת עצמו.

<u>אפשרויות להמשך פיתוח</u>
במהלך הפרויקט התמקדתי בכמה אופנים ספציפיים של עולם המכוניות האוטונומיות, אך ניתן להרחיב את הסימולטור לעוד מקרים:

- גילוי חנייה תוך כדי נסיעה ולא בתור נקודות המוגדרות מראש.
- גילוי סוג החנייה (מקבילי/ ניצב) באופן דינמי ללא קביעה מוקדמת.
- התייחסות לגודל הרכב: תמיכה במשאית / אופנוע ...
- מניעת התנגשויות תוך כדי נסיעה.

בנוסף ניתן להרחיב את הסימולטור לעוד תחומים לדוגמא תחום האלגוריתמים המבוזרים.
עבור אלגוריתמים מבוזרים ולמידת מכונה יש דרישה לקלט רב, לדוגמא אימון מודל לקבלת תמונה מסנסור וחישוב פלט האם החנייה פנויה או תפוסה, מקבילה או בניצב דורשת הרבה דוגמאות תמונה שיתויגו בהתאם. בפרויקט זה היינו צריכים קלטים רבים של תמונות עבור סוגי החניות, לא קיים מאגר תמונות כזה עבור CARLA ולייצר מאגר כזה צורך זמן רב ומשאבים כדי ליצור מאגר מידע גדול מספיק לאימות המידע ויצירת פלט תקין. בפרויקט יש שימוש במנוע גרפי CARLA, ממשק זה מאוד כבד גראפית (CPU,MEMORY,GPU) ומאוד קשה לחבר את התהליך לאימון מודל או ליצירת מאגר תמונות עצמאית.

ביבליוגרפיה

- אתרים:

http://carla.org/

https://carla.readthedocs.io/en/latest/

https://github.com/carla-simulator

https://pythonprogramming.net/introduction-self-driving-autonomous-cars-carla-python/

https://www.synopsys.com/automotive/what-is-autonomous-car.html

- מאמרים:

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio L´opez, and Vladlen Koltun, 2017. CARLA: An Open Urban Driving Simulator.

Bijun Lee, Yang Wei, I. Yuan Guo, 2017. AUTOMATIC PARKING OF SELF-DRIVING CAR BASED ON LIDAR.

Peizhi Zhang, Lu Xiong, Zhuoping Yu, Peiyuan Fang, Senwei Yan, Jie Yao and Yi Zhou, 2019. Reinforcement Learning-Based End-to-End Parking for Automatic Parking System.

Tsung-hua Hsu, Jing-Fu Liu, Pen-Ning Yu, Wang-Shuan Lee and Jia-Sing Hsu, 2008. Development of an Automatic Parking System for Vehicle.

Markus Heimberger ,Jonathan Horgan ,Ciaran Hughes ,John McDonald, Senthil Yogamani, 2021. Computer Vision in Automated Parking Systems: Design, Implementation and Challenges.