Control Robotics and Machine Learning Laboratory

הטכניון - מכון טכנולוגי לישראל
TECHNION - ISRAEL INSTITUTE OF TECHNOLOGY

הפקולטה להנדסת חשמל
המעבדה לבקרה רובוטיקה ולמידה חישובית

דוח פרויקט : פרויקט א'

AI Algorithm Training simulator

מגישים:

Alon Kwart Yonathan Bettan

מנחה:

Eden Sasson

סמסטר: אביב

שנה: 2018

תוכן עניינים:

1. מטרת הפרויקט:

מטרת הפרויקט הינה יצירת סביבת עבודה וירטואלית לרכב אוטונומי לטובת אתגר ה- FSD בכדי שיהיה ניתן לזרז את מהירות הלימוד, לספק סביבה בטוחה וזולה לאימון מבלי לחשוש מנזקים פוטנציאליים על הרכב וללא צורך ברכב עצמו.

מכיוון שקיימות סביבות מוכנות שבהן נוכל להשתמש כמו Microsoft AirSim נתמקד בבחירת פרמטרים פיסיקליים המתאימים למודל ולסביבה בה נצפה להיות. לאחר מכן נרצה להשוות בין האותות הנוצרים בחיישנים הווירטואליים לבין האותות הנוצרים ע"י החיישנים האמתיים. לבסוף נרצה לקרב ככל הניתן בין האותות הווירטואליים לאמתיים שכן כל הפרמטרים החופשיים במערכת שלנו, כגון פרמטרים של רשתות נוירונים, פרמטרי כיול, Hyperparameters של אלגוריתמים בעיבוד וניתוח תמונות וכ״ו, יאותחלו בהתאם.

בנוסף עיצובן של רשתות נוירונים מלאכותיות עם זיכרון, הלומדות באופן איכותי, מושפע מאוד מהדיוק של המנוע הפיזיקלי המושרה בגרפיקה אשר נקלט בחיישנים הווירטואליים לכן נרצה לבחור את הפרמטרים הפיזיקליים של AirSim כך שנקבל מודל קרוב למציאות ככל שניתן.

אחרי שלב ניתוח האותות בסביבה הווירטואלית נקבל תחושה לגבי אותות שהם מידע חשוב שאותו יש לשקלל בפקודות הבקרה כך שנוכל לאמן רשת בשיטת RL לנווט על סמך וידאו בפרויקט ההמשך וכן להעריך את הזמן הדרוש כדי להשיג את הביצועים הנדרשים.

לסביבה הווירטואלית יתרונות וחסרונות.

יתרונות:

- אין מגבלות פיזיקליות כגון צריכת דלק או בלאי של הרכב.
- ניתן להריץ את האימון בקצב מהיר יותר מאשר באימון האמתי ע״י האצת הסימולטור.
- זול, אין צורך ברכב אמתי או בתחזוקה של הרכב במידה וקיים או אנשי צוות אשר יתפעלו את הרכב.

חסרונות:

- ברוב המקרים המודל הפיזיקלי של הסימולטור יכיל סטייה מסוימת מהמציאות.
- סביבה וירטואלית נכתבת בידי מהנדסים ולכן יתכנו שגיאות אנוש או באגים לא צפויים.

בתכנון שלנו יבוא לידיי ביטוי בעיקר יתרונות העלות ומהירות האימון שכן בתחילת האימון הרכב מתנגש באופן תמידי ולומד מחיזוקים שהינם התנגשויות עד אשר הוא לומד לנסוע ללא התנגשויות, לא נרצה לאמן רשת על רכב אמתי שכן לא נרצה לגרום להתנגשויות מכוונת על רכב אמתי לצורך למידה.
מנגד החיסרון הבולט ביותר הנובע משימוש בסימולטור הוא, כאמור, הסטייה הקיימת מהפיזיקה האמתית של הרכב.
מטרת פרויקט זה היא למזער סטייה זו ככל הניתן.

לבסוף שימוש בסימולטור מאפשר לנו "חיים קלים" מבחינה בירוקרטית שכן אין צורך בנהג שתפקידו להיות נוכח ברכב למקרה של יציאה משליטה, אין צורך ברכב, אין צורך בניהול הבירוקרטיה של תיאום מול אנשי צוות אחרים על זמן אימון, תחזוקה שוטפת וכ"ו

2. <u>מבוא:</u>

התכנה AirSim של Microsoft רצה מעל מנוע פיזיקלי בשם Unreal Engine כפי שיפורט בפרק 5.

2.1. מנוע פיזיקלי

תפקידו לדמות התנהגות פיסיקלית ניוטונית של אובייקטים במרחב-זמן ולהגדיר את הדינמיקה והאינטראקציה בניהם. דוגמה פשוטה לכך הינה כדור המתנגש בקיר, קיים תיאור פיסיקלי באמצעות מודל של הכדור, הקיר והאינטראקציה ביניהם .

מנוע פיזיקלי זה מיועד לדמות את הפיזיקה של הרכב בתוך הסימולטור כל שנוכל לקבל ממנו מידע על מצב הרכב בנקודות זמן מסוימות . המנוע, כאמור, מספק לנו מהירות והילוך נוכחי של הרכב, מיקום, כיוון, מהירות לינארית וזוויתית ותאוצה לינארית וזוויתית כפי שניתן לראות באיור #1

- `getCarState` : This retrieves the state information including speed, current gear and 6 kinematics quantities: position, orientation, linear and angular velocity, linear and angular acceleration. All quantities are in NED coordinate system, SI units in world frame except for angular velocity and accelerations which are in body frame.

איור #1

המנוע הפיזיקלי המוזכר לעיל הינו מובנה בתוך Unreal Engine ומשתמש ב- PhysX של NIVIDIA כפי שמתואר במבוא באיור #2

The physics engine

In the real world, objects are governed by the laws of physics. Objects collide and are set in motion according to Newton's laws of motion. Attraction between objects also obeys the law of gravity and Einstein's theory of general relativity. In the game world, for objects to react similarly to real life, it has to have the same system built through programming. Unreal physics engine makes use of PhysX engine developed by NVIDIA to perform calculations for life-like physical interactions such as collision and fluid dynamics. The presence of this advanced physics engine in place allows us to concentrate on making the game instead of spending time making objects interact with the game world correctly.

איור #2

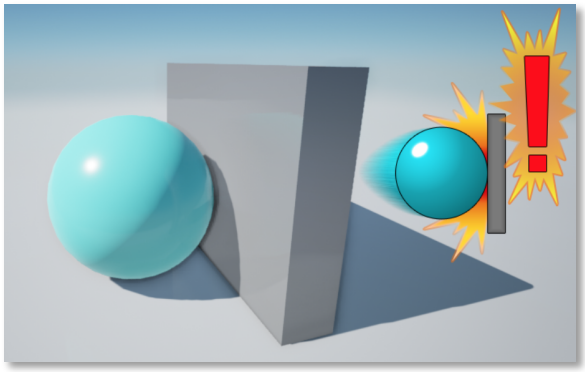
2.2. סביבה גרפית

נועדה לתת ממשק גרפי למנוע פיסיקלי וכן מבחר של אובייקטים והגדרות שנועדו ליצור סביבות עשירות ובקלות יחסית.

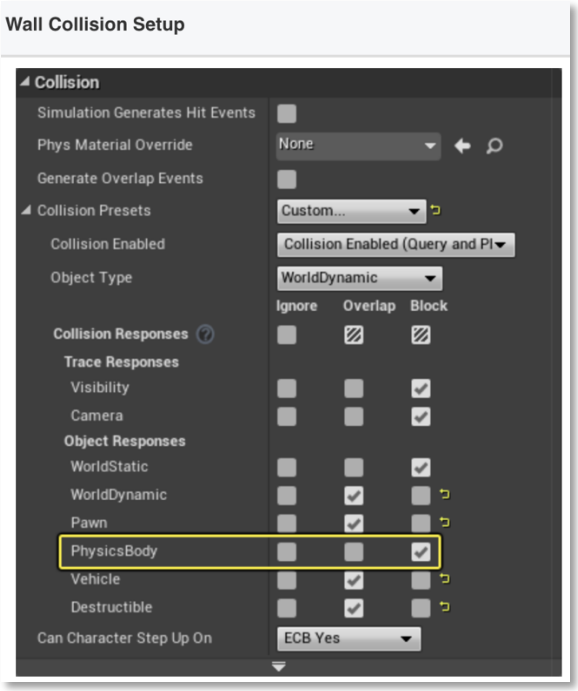
המנוע של Unreal מאפשר לנו להגדיר התנהגויות שונות בעת התנגשות של 2 אובייקטים.
כל אובייקט אשר יכול לבצע collision מקבל Object Type וסדרה של תגובות אשר מגדירות איך הוא מתנהג עבור התנגשות בכל אחד מה- Object Types הנוספים. ההתנהגויות המרכזיות הינן Block ו- Overlap/ignore.

עבור Block ברגע זיהוי collision אם 2 האובייקטים המתנגשים מוגדרים כ- Block אזי הם יחסמו זה את זה, בנוסף ניתן להגדיר אופציה של שליחת Hit event לצורך טיפול המשך במקום אחר בקוד.
האופציה המרכזית השנייה הינה Overlap/ignore אשר מתנהגים בצורה כמעט זהה אחד לשני ומגדיר שהאובייקטים ימשיכו את תנועתם כאילו לא אירעה התנגשות, ההבדל בניהם נובע מכך שב- Overlap ניתן להגדיר שליחה של Overlap event בדומה ל- Hit event בשונה מ- Ignore שם נצפה להתעלמות מוחלטת בין 2 האובייקטים המתנגשים.

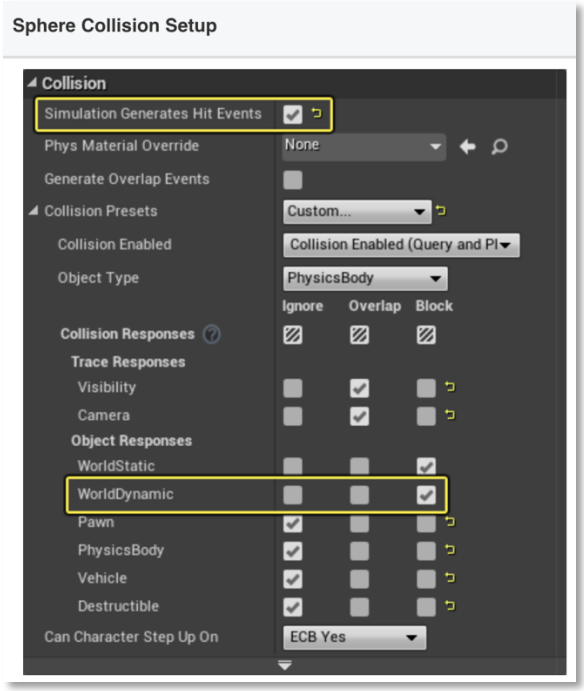
באיורים 5-3# ניתן לראות איך מגדירים Collision enabled עבור אובייקט של כדור שהינו מ- Object Type=PhisicsBody ואובייקט של קיר שהינו מ- Object Type=WorldDynamic וכן מגדירים עבור הכדור שישלח Hit event במקרה של collision.



איור 3#



איור #5

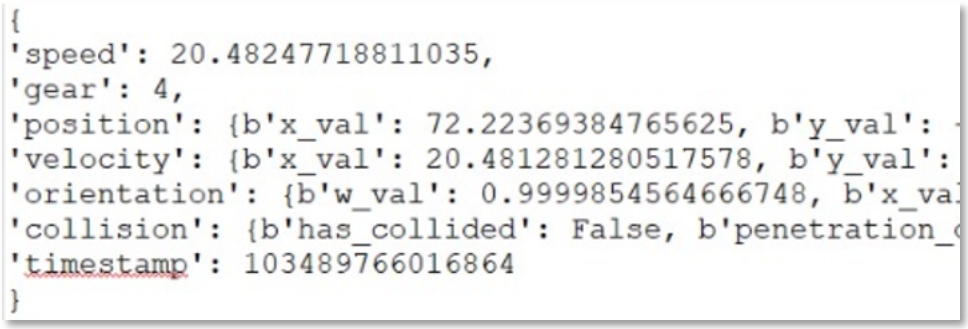


איור #4

2.3.הרכב

בנוסף להיותו אובייקט בפני עצמו בסימולטור אשר יכול להתנגש באובייקטים אחרים, לרכב יש 2 אלמנטים וירטואליים נוספים אשר מספקים לנו מידע אודות הנסיעה. הראשון הוא רכיב דמוי IMU והשני הוא מצלמת וידאו.

הרכיב שתפקידו לדמות IMU מספק מידע כגון מהירות, מיקום, מידע אודות ההתנגשות ומידע נוסף כפי שניתן לראות באיור #6



איור #6

המצלמה הווירטואלית לעומת זאת מספקת רצף תמונות ומאפשרת מגוון פעולות על ה- stream המתקבל בצילום, לדוגמא ניתן לקבל את המידע כרצף של תמונות PNG או מערך של תמונות לא מכווצות בפורמט RGB ואף להמירן לתמונות בפורמט gray scale לצורך training של רשת כפי שניתן לראות באיורים 8-#7

Getting Images with More Flexibility

The `simGetImages` API which is slightly more complex to use than `simGetImage` API, for example, you can get left camera view, right camera view and depth image from left camera in a single API call. The `simGetImages` API also allows you to get uncompressed images as well as floating point single channel images (instead of 3 channel (RGB), each 8 bit).

Python

```
import airsim #pip install airsim

# for car use CarClient()
client = airsim.MultirotorClient()

responses = client.simGetImages([
    # png format
    airsim.ImageRequest(0, airsim.AirSimImageType.Scene),
    # uncompressed RGBA array bytes
    airsim.ImageRequest(1, airsim.AirSimImageType.Scene, False, False),
    # floating point uncompressed image
    airsim.ImageRequest(1, airsim.AirSimImageType.DepthPlanner, True)])

 # do something with response which contains image data, pose, timestamp etc
```

איור 7#

איור 8#

3. תכן ראשוני:

בתחילת דרכנו הבנו שעלינו למצוא סימולטור מתאים לצורך ביצוע המשימה, סימולטור כזה צריך היה להוות סביבה ווירטואלית אשר מדמה בצורה המדויקת ביותר את הסביבה הפיזיקלית בה אנחנו חיים על מנת שאלגוריתמים שלמדו על הסימולטור יתאימו למציאות אך בזמן סביר.
אחד המאפיינים היה שהסימולטור צריך להיות גמיש, קוד פתוח במידת האפשר והכי חשוב בעל API פשוט וגמיש ובנוסף שיהווה מודל פיזיקלי מדויק ככל הניתן.

לאחר חיפוש מצאנו כי ישנם 2 סימולטורים כנ"ל, Microsoft AirSim ו- rFactor2, אשר לכל אחד יתרונות וחסרונות משלו.

היתרון של AirtSim היה שהפרויקט הוא קוד פתוח ושיש לו API מתועד היטב לעומת זאת ניתנה פחות גמישות בעדכון הפרמטרים הממדלים את פיזיקת הרכב והסביבה, בשונה מ- rFactor2 אשר העניק גמישות גדולה במידול הרכב, ולכן מקשה על התאמה מדויקת של אימון לרכב בעל פרמטרים פיזיקליים מסוימים.

לאחר מחשבה מעמיקה ובעקבות העובדה שבזמנו התכנון היה ללמד את רכב הפורמולה של הטכניון בכדי שיוכל להתחרות בתחרות במקצה הרכב האוטונומי בחרנו לעבוד עם rFactor2 שכן המודל הפיזיקלי של הרכב עמד במרכז הבעיה.

3.1. rFactor

התחלנו לעבוד בעזרת rFactor2, כתבנו ממשק server-client ולמדנו את ה- APIs וחלק מה- plugins המוצעים לסימולטור זה.
באיור 9# ניתן לראות חלק ממשק server-client שמטרתו לקשר בין מכונה המריצה Linux, שכן שאר צוותי הפיתוח של הפורמולה האוטונומית הריצו Linux מכיוון שהריצו את האלגוריתמים על חומרה שנועדה להיות חלק מהרכב, לבין מכונה המריצה Windows שכן rFactor2 נכתב עבור Windows.

```python
host = '127.0.0.1'
port = 5000

mySocket = socket.socket()
mySocket.connect((host, port))

message = input(" -> ")

while message != 'q':
    mySocket.send(message.encode())
    data = mySocket.recv(1024).decode()

    print('Received from server: ' + data)

    message = input(" -> ")

mySocket.close()
```

איור 9#

rFactor הקנה גמישות מרבית במידול הבעיה הפיזיקלית אך לא היה קוד פתוח וה- API שלו הגיע מיותר מ- plugin אחד מה שיצר קצת פחות סדר בהבנתו.

בתקופה זו המכשול העיקרי שלנו היה חוסר גמישות בגישה ל- API מסודר אשר הקשה לבנות סביבה מתאימה מתוך הסימולטור שיתאים לסביבת האימון האמתית של הרכב, שכן כאמור זה היה אחד החסרונות של rFactor2.

באיור 10# ניתן לראות חלק מסימולטור המקלדת אותו כתבנו על מנת להתגבר על המכשול הנ"ל בכדי שנוכל להקליט נהיגה מראש במקום לשלוח פקודות API לנהיגה.

```python
def AltTab():
    """Press Alt+Tab and hold Alt key for 2 seconds
    in order to see the overlay.
    """
    PressKey(VK_MENU)    # Alt
    PressKey(VK_TAB)     # Tab
    ReleaseKey(VK_TAB)   # Tab~
    time.sleep(2)
    ReleaseKey(VK_MENU)  # Alt~
```

איור 10#

לאחר מספר שבועות הגיע לאוזנינו הבשורה המפתיעה על כך שהפרויקט אינו מתכנס והוחלט שהשנה הטכניון לא יגיש מועמדות למקצה האוטונומי.

בשורה זו שינתה לחלוטין את הבעיה שהיה עלינו לפתור, כלומר לפתע לא היינו זקוקים יותר לסביבת אימון ספציפית מאוד אלא נפתחה בפנינו האפשרות להשתמש בסימולטור גמיש יותר על חשבון בניית סביבה לרכב כללי ולאו דווקא מדגם מסוים.

באותו יום קיבלנו החלטה להעביר את הפרויקט לסימולטור השני שהיה לרשותנו, AirSim של חברת Microsoft.

3.2. AirSim

החל מנקודה זאת קיבלנו גמישות שלא הייתה קיימת קודם ו- API מאוד ברור ומסודר ומרגע זה הפרויקט התחיל לרוץ.

הורדנו סביבה מוכנה של AirSim בשם Neighborhood אשר מכילה מפה של עיר והמגיעה עם רכב מובנה, כיוון שבשלב זה המודל הפיזיקלי של הרכב כבר לא היה מטרתנו החלטנו להשאיר את הרכב כפי שהוא ולא לצלול לתוך המודל הפיזיקלי שלו.

כפי שציינו במבוא, AirSim מספק API מאוד נוח לשימוש ובין היתר מתודת getCarState() אשר מספקת נתונים כגון מיקום, מהירות וקטורית, סטאטוס התנגשות ועוד, כפי שניתן לראות באיור #11, כאשר חלק מהנתונים הנ"ל בעצם ממדלים IMU וירטואלי הממוקם על הרכב.

```
client.reset()

# go forward
car_controls.throttle = 1.0
car_controls.steering = 1
client.setCarControls(car_controls)

imagequeue = []

while True:
    car_state = client.getCarState()
    file.write(str(car_state.speed) + ",")
    file.write(",".join([str(car_state.velocity[x]) for x in car_state.velocity] + \
                        [str(car_state.position[x]) for x in car_state.position] + \
                        [str(car_state.orientation[k]) for k in car_state.orientation]) )
```

איור #11

4. תיאור חמרה:

בפרויקט זה אין חמרה מיוחדת, שכן מדובר בסימולטור. אנחנו עבדנו על מכונה ביתית עם הנתונים הבאים:

OS	Windows 10
GPU	NIVIDIA GeForce GTX 1080 Ti
CPU	Inter Core i7-4770 3.40 GHz 4 Cores 8 Logical Proccessors
RAM	16GB

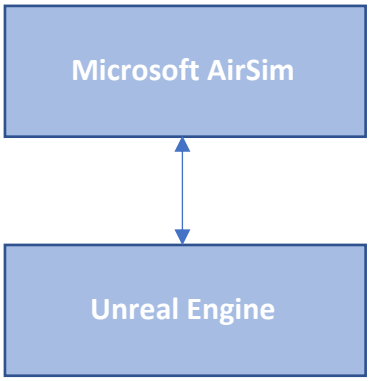
5. תיאור תכנה:

5.1. סביבת עבודה:

התכנה מורכבת מ-2 חלקים עיקריים, החלק הראשון נקרא Unreal Engine והינו המנוע עצמו אשר מטרתו לפתח משחקים, סביבות עבודה שלמות כגון עיר לצורך אימון רשת אוטונומית, עיר חכמה וכ"ו

החלק השני של התכנה נקרא Microsoft AirSim והינו מעטפת אשר רצה מעל Unreal Engine ומספקת מודל של רכב (וגם של רחפן אך זה לא רלוונטי עבור הפרויקט שלנו) ו- API לשליטה על הרכב וקבלת Data מהסימולטור כגון תמונות ממצלמת הרכב, מהירות הרכב, זווית הגה וכ"ו

איור 12# ממחיש את המבנה של התכנה שלנו.



איור 12#

5.2. הסבר על התכנה:

לצורך הפרויקט יש שימוש ב-2 החלקים כאשר אנו משתמשים ב- API של AirSim על מנת לשלוט על הרכב בצורה אוטומטית ובסופו של דבר פקודות ה- API הנ"ל מתורגמות לפעולות על הרכב במנוע Unreal Engine.

השימוש העיקרי שלנו הינו במודול של AirSim אשר מעניק לנו תקשורת server-client לשליטה ברכב וקבלת מידע בזמן אמת. בפרויקט ההמשך נעשה שימוש במידע על מנת ללמד רשת לנהוג ברכב בצורה אוטונומית.

המחלקה העיקרית בה אנו משתמשים נקראת CarClient אשר יורשת מ- AirSimClientBase ונראה כיצד ניתן להשתמש בה.

5.2.1. Msgpackrpc

מודול זה הינו מודול שנכתב עבור python ומגדיר ממשק server-client, אנו נשתמש בו לצורך יצירת client עבור הסימולטור, שליחת פקודות נסיעה לרכב וקבלת מידע עבור סטאטוס הרכב בזמן נסיעה.

```
import msgpackrpc #install as admin: pip install msgpack-rpc-python
```

5.2.2. מבנה האובייקט

המחלקה CarClient מכילה שדה בודד בשם client אשר מכילה את ה- Client הנוצר באמצעות המודול msgpackrpc.

```
def __init__(self, ip, port):
    self.client = msgpackrpc.Client(msgpackrpc.Address(ip, port), timeout = 3600)
```

5.2.3. מתודות מצלמה

מתודות אלו משתמשות במתודות המובנות ב- AirSim על מנת להחזיר תמונה או רצף תמונות המצולמות ממצלמת הרכב בזמן אמת בפורמט compressed PNG המיוצג ע"י מערך של בתים.

```
# camera control
# simGetImage returns compressed png in array of bytes
# image_type uses one of the AirSimImageType members
def simGetImage(self, camera_id, image_type):
    # because this method returns std::vector<uint8>, msgpack decides to encode it as a string unfortunately.
    result = self.client.call('simGetImage', camera_id, image_type)
    if (result == "" or result == "\0"):
        return None
    return result

# camera control
# simGetImage returns compressed png in array of bytes
# image_type uses one of the AirSimImageType members
def simGetImages(self, requests):
    responses_raw = self.client.call('simGetImages', requests)
    return [ImageResponse.from_msgpack(response_raw) for response_raw in responses_raw]
```

להלן דוגמא לשימוש במתודה זו.

```
# get RGBA camera images from the car
responses = client.simGetImages([ImageRequest(1, AirSimImageType.Scene)])

# add image to queue
imagequeue.append(responses[0].image_data_uint8)
```

5.2.4. מתודות להגדרת סטאטוס נסיעה

מתודה בשם setCarControls המקבלת פרמטר מטיפוס CarControls המוגדר ב- AirSim ומכילה שדות כגון זווית הגה רצויה לנסיעה וגורמת לרכב לנסוע לפי פרמטרים שהוגדרו.

```
def setCarControls(self, controls):
    self.client.call('setCarControls', controls)
```

להלן דוגמא לשימוש במתודה זו.

```
car_controls = CarControls()

client.reset()

# go forward
car_controls.throttle = 1.0
car_controls.steering = 0
client.setCarControls(car_controls)
```

5.2.5. מתודות לקבלת סטאטוס הרכב

מתודה בשם getCarState המחזירה נתונים כגון מיקום הרכב, מהירות הרכב, זווית הגה וכדומה.

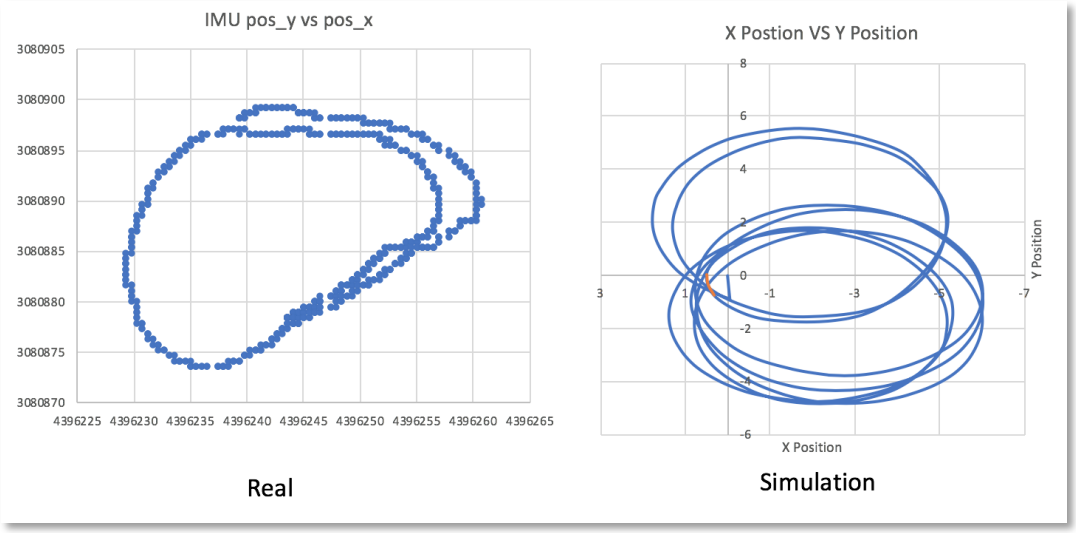
```
def getCarState(self):
    state_raw = self.client.call('getCarState')
    return CarState.from_msgpack(state_raw)
```

דוגמא לשימוש במתודה זו ניתן לראות בפרק 3 באיור 11#.

המידע אותו אנו מקבלי ב- ()getCarState הינו מידע המתקבל ע"י רכיב IMU ווירטואלי הממוקם על הרכב.

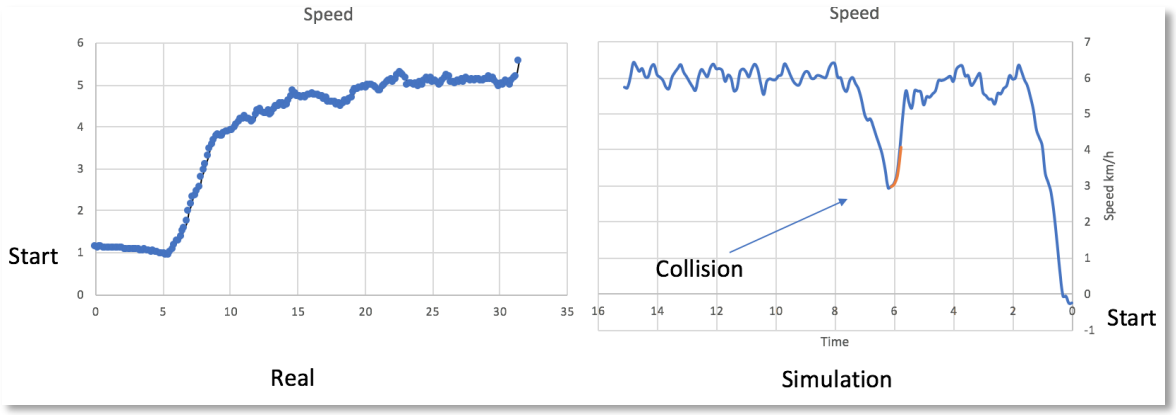
ביצענו דגימות של רכיב ה- IMU הווירטואלי בנסיעה מעגלית והשווינו לדגימות ה- IMU הממוקם על הרכב האמתי.

באיור 13# ניתן לראות כי מגמת רכיב המיקום דומה במציאות ובסימולטור ולכן ממחיש כי המנוע הפיזיקלי של הסימולטור אכן מדויק. הסיבה לכך שישנם 2 מעגלים מרכזיים בסימולטור היא שאירעה התנגשות במדרכה ולכן מסלול הרכב השתנה, ניתן לראות בכתום את המיקום המדויק בו אירעה ההתנגשות.



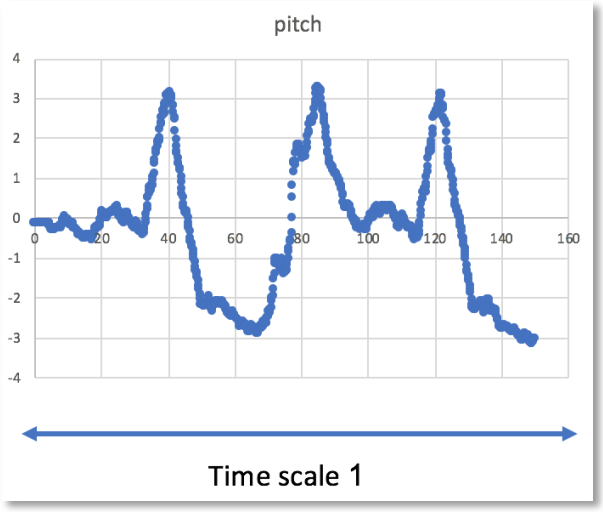
איור 13#

באיור 14# ניתן לראות כי מגמת רכיב המהירות בסימולטור ובמציאות אכן דומים באותו אופן ומאותן סיבות שהיה דמיון ברכיב המיקום. גם כאן ניתן לראות את מיקום ההתנגשות בכתום בגרף.



איור 14#

מידע חשוב נוסף שניתן לקבל דרך רכיב ה- IMU הינו ה- pitch אשר נותן לנו מידע בחלונות זמן קצרים, היכן שביצועי המצלמה בד"כ נפגמים ולכן לדוגמא פגיעה במדרכה תבלוט במידה המתקבל מה- IMU היכן שהמצלמה תתקשה לזהות אירוע מסוג זה. ניתן לראות זאת באיור 15#.



איור 15#

6. תוצאות ומסקנות:

נראה כי בסביבה זו נצפה ליצור אותות דומים לאותות שנצפו במציאות. האתגר הבא הוא להתמודד עם ממשק מכני שכן גם הוא מהווה "קופסה שחורה" שאותה יש לכלול במודל שלנו וכן מימוש של אלגוריתם בקרה באמצעות כלים ו- TRENDS של בינה מלאכותית.

נקודה חשובה נוספת הינה למצוא את האיזון בין סקר ספרות טרם תחילת העבודה לבין העבודה עצמה, שכן עבודת הכנה היא דבר חשוב אך במקרה הכללי רוב הלמידה נעשית תוך כדי העבודה וקשה מאוד לנבא את כל הקשיים בהם ניתקל מראש לכן אנו ממליצים לבצע סקר ספרות אך לא להתמהמה ולקפוץ למים מוקדם ככל האפשר.

בנוסף נרצה לעבוד עם כלים שנבנו למטרה אותה אנחנו רוצים להשיג עם עדיפות לקובץ BINARY מוכן כגון Microsoft AirSim over Unreal Engine שכן plugin זה נכתב במטרה לבנות סביבות סימולציה ככלל ואימון אלגוריתמים חכמים בפרט ולכן מספק API מאוד פשוט לשימוש, שכן זאת מטרתו.

7. ביבליוגרפיה:

1. https://store.steampowered.com/app/365960/rFactor_2/
2. https://github.com/Microsoft/AirSim.git
3. https://www.youtube.com/watch?v=aircAruvnKk
4. https://www.youtube.com/watch?v=IHZwWFHWa-w&t=1031s
5. https://www.youtube.com/watch?v=Ilg3gGewQ5U
6. http://neuralnetworksanddeeplearning.com/
7. http://krisbolton.com/a-practical-introduction-to-artificial-neural-networks-with-python/
8. https://docs.unrealengine.com/en-US/Engine/Physics/Collision/Overview

Bibliography:

AIRSIM development:

[1] Shah, Shital, et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles." *Field and service robotics*. Springer, Cham, 2018.

Use of Rfactor High-fidelity simulator for academic research on automotive:

[2] Advocaat, R. "Comparing and improving steering forces in a race car and race simulator to increase simulator fidelity." (2015).

[3] Miranda, Mateus R., et al. "Development of simulation interfaces for evaluation task with the use of physiological data and virtual reality applied to a vehicle simulator." *The Engineering Reality of Virtual Reality 2015*. Vol. 9392. International Society for Optics and Photonics, 2015.