

# הערכת עומק רחפן מתמונה בודדת - סקירה טכנית מקיפה

## 1. הגדרת המשימה - זיהוי עומק רחפן בתמונה מונוקולרית

**תיאור כללי:** המשימה היא לבצע לוקליזציה תלת-ממדית של רחפן באמצעות מצלמה יחידה, בדגש על קביעת מרחקו של הרחפן מהמצלמה (הציר Z). במילים אחרות, על סמך תמונה בודדת שבה מופיע הרחפן, יש לזהות היכן הרחפן נמצא במרחב ביחס למצלמה - **בעיקר את המרחק (עומק) שלו**. Unlike stereo or multi-camera systems, a single image doesn't provide direct geometric triangulation, so depth must be inferred from visual cues<sup>1</sup>. לשם כך משלבים בדרך כלל שתי משימות: (1) **זיהוי אובייקט** - למשל שימוש ברשת כמו YOLO לאיתור ותיבת תיחום של הרחפן בתמונה, ו-(2) **הערכת עומק מונוקולרית** - שימוש ברשת נפרדת לחיזוי מרחק או מפת עומק מתוך תמונה בודדת. בחיבור של שני אלה ניתן לקבל את מיקום הרחפן בתלת-ממד. הפרויקט המדובר עובד על נתונים סינתטיים בלבד (מתוך Unreal Engine), כך שלרשותנו **אמת-מידה מושלמת** למיקום הרחפן בכל תמונה, מה שמאפשר אימון ובדיקה יעילים ללא מדידות בעולם האמיתי<sup>2</sup>. התוצאה הסופית המצופה היא מערכת שמקבלת תמונה (ממסדת וידאו או סטילס), מזהה את הרחפן בתוכה, ומעריכה את מרחקו מהמצלמה (כלומר את קואורדינטת ה-Z שלו ביחס למצלמה). בשלב מתקדם יותר, ניתן גם להסיק את המיקום המלא (X,Y,Z) ולבצע עיקוב בזמן אמת. בפועל, הפיפליין המלא יכול: **Object Detection (YOLO) → Depth Estimation → 3D Reconstruction** (מ-2D ל-3D) **Tracking (קלמן)**<sup>3</sup>. בפרויקט שלכם, ההתמקדות הראשונית היא בחלק העומק (Z) בלבד, ולכן בשלב זה היעד הוא לקבל עבור כל תמונה את מרחק הרחפן. חשוב להבין שזהו אתגר לא טריוויאלי - בהיעדר מידע סטריאוסקופי, המערכת צריכה להסיק עומק מרמזים ויזואליים (גודל מופעם של אובייקטים, פרספקטיבה, טשטוש אטמוספרי וכו') בדומה לראיית עומק בעין אחת<sup>1</sup>.

## 2. תחומי ידע וטכנולוגיות רלוונטיות

הפרויקט משלב מספר תחומי למידה עמוקה וראייה ממוחשבת. חלק מהנושאים ייתכן שאתם כבר מכירים, ואחרים דורשים העמקה נוספת. להלן המרכזיים שבהם, יחד עם הסבר קצר:

- רשתות להערכת עומק מונוקולרי:** מדובר ברשתות נוירוניות החוזות מרחק מכל פיקסל בתמונה בודדת. רשת עומק מונוקולרית מסיקה עומק יחסי - כלומר, היא יודעת אילו חלקים בתמונה קרובים או רחוקים יותר זה מזה, אך ללא קנה מידה מוחלט ללא כיול נוסף<sup>4</sup> <sup>5</sup>. יש להבין את בעיית **עמימות הסקאלה (Scale Ambiguity)**: סצנות שונות יכולות להיראות זהות למרות מרחקים מוחלטים שונים, ולכן רשת ללא מידע נוסף לא יודעת אם הרחפן במציאות במרחק 5 מ' או 50 מ' - היא רק יודעת למשל שאובייקט א' רחוק פי 2 מאובייקט ב'<sup>4</sup>. ישנם מודלים מודרניים מבוססי Transformer שהוכשרו על מיליוני תמונות כדי ללמוד רמזי עומק (כגון **ZoeDepth**, **MiDaS**, **Depth Anything** - ראו פירוט בהמשך)<sup>5</sup>. אם תחום זה חדש לכם, תצטרכו ללמוד על ארכיטקטורות נפוצות (Encoder-Decoder עם ResNet או Vision Transformer), הבדל בין עומק יחסי לעומק מטרי, וכיצד לאמן או לכייל מודלים כאלה.

- זיהוי אובייקטים (Object Detection) - YOLO:** כדי להתאם את חישוב העומק לרחפן בלבד, יש לזהות אותו בתמונה. סביר שתשתמשו במשפחה של מודלי YOLO (You Only Look Once) לזיהוי מהיר של הרחפן ומתן תיבת תיחום סביבו. ייתכן וכבר עבדתם עם YOLO או רשת דומה; אם לא, כדאי להכיר מושגים כמו Bounding Box, IOU, והאופן שבו רשתות כאלה מאומנות. בפרויקט הנוכחי, YOLO (למשל גרסת YOLOv5/YOLOv8) תשמש כמודול לזיהוי מיקום הרחפן בפיקסלים<sup>3</sup>. הזיהוי מבוסס תמונה בודדת, ומספק את מרכז הרחפן ותיבת התיחום, המשמשים בהמשך לחישוב המיקום המרחבי.

- **גיאוטרית מצלמה והמרת קואורדינטות:** נושא חשוב במיוחד הוא להבין כיצד לעבור מתמונה דו-ממדית לקואורדינטות תלת-ממד. לשם כך דרושה היכרות עם **פרמטרים פנימיים של המצלמה (Camera Intrinsics)** – אורך מוקד (בפיקסלים), נקודת מרכז, וכד'. המרה של פיקסל למרחק דורשת את הפרמטרים הללו ואת עומק ה-Z המשוער. למשל, ברגע שידועים את Z (מרחק מהמצלמה) ואת מיקום הפיקסל של הרחפן  $x_{\text{pixel}}$ , אפשר לחשב את  $Y$ ,  $X$  המישוריים באמצעות:  $X = (x_{\text{pixel}} - c_x) \cdot Z / f_x$  וכן  $Y = (y_{\text{pixel}} - c_y) \cdot Z / f_y$  <sup>6</sup>. לכן, יש לוודא שמכירים את מודל המצלמה (מודל ה-Pinhole) ולוודא שפרמטרי המצלמה הווירטואלית ב-Unreal תואמים לחישובים (עוד על כך בחלק 5). היכרות עם המושגים הללו חיונית כדי להפוך עומק מחושב (Z) לקואורדינטה מטריית בעלת משמעות. כמו כן, תצטרכו להבין את ההבדל בין מערכת קואורדינטות **עולם** למערכת **מצלמה** – המיקום האמיתי של הרחפן צריך להיות מיוצג במערכת הצירים של המצלמה לצורך לימוד ובדיקה <sup>7</sup>. אם הקונספטים האלה חדשים, מומלץ לרענן את הידע באופטיקה גיאומטרית בסיסית.

- **מעקב וחלונת קלמן (Kalman Filter):** מאחר שמדובר ברחפן שעשוי לנוע, סביר שהמערכת שלכם תידרש לעקוב אחריו על פני רצף פריימים. מסנן קלמן הוא כלי קלאסי למיזוג הערכות רועשות לאורך זמן ולקבלת מסלול חלק. גם אם בשלב ראשון תעבדו על תמונות בודדות, בשלב המשך (או עבור וידאו) כדאי להכיר את השימוש ב-Kalman לצורך **החלקת נתוני המיקום והעומק**. בפרויקט הדוגמה, אחרי שחישבו בכל פריים את המיקום, החילו פילטר קלמן כדי לסנן רעשים (ריצוד של תיבת הזיהוי, אי-יציבות בחיזוי העומק) וגם כדי לחזות את מיקום הרחפן בפריים הבא במקרה שהזיהוי מתפספס <sup>8</sup>. אם טרם עבדתם עם Kalman, מומלץ ללמוד את מודל המצבים, מטריצות התהליך והמידה, וכיצד להזין אלגוריתם קלמן בפיתוח (יש לכך ספריות מוכנות – ראו סעיף כלים).

- **עבודה עם דאטה סינתטי (Unreal Engine):** עבודה עם נתונים מסונתזים מציבה הזדמנויות ואתגרים. **היתרונות:** אפשר לקבל אמת-מידה מושלמת לזיהוי ומרחק (Unreal יכול לספק את מיקום האובייקט המדויק בכל פריים) <sup>2</sup>, וכן לשלוט במגוון תנאי הסצנה באופן שיטתי. בפרויקט לדוגמה, ניצלו זאת ליצירת וריאציות רבות: מזג אוויר שונה, תנאי תאורה (בוקר/ערב), רקעים מגוונים, ומרחקי רחפן שונים כדי שהמודל ילמד לטפל בכל המצבים <sup>9</sup> <sup>10</sup>. **האתגרים:** יש לשים לב לפער האפשרי בין עולם סינתטי לעולם האמיתי (Domain Gap) – טקסטורות וגרפיקה ממוחשבת עלולות להיות "נקיות" מדי, כך שאם מתכננים בהמשך להפעיל את המודל על וידאו אמיתי, ייתכן ויידרש Fine-tuning או אוגמנטציה חזקה כדי שהמודל יתמודד עם רעשים אמיתיים. בנוסף, יש לוודא שפורמט הנתונים התפעולי (למשל, קבצי תמונה + קובצי JSON של קואורדינטות) מוגדר היטב כדי שתהליך האימון יהיה חלק. נקודה חשובה נוספת היא סנכרון נתונים – מכיוון שב-Unreal ניתן להפיק גם וידאו וגם לוג של מיקום הרחפן, חייבים לוודא שידועים להתאים כל פריים לתווית (זמן אמת) הנכונה <sup>11</sup>. בפרט, **המיקום שישמש כתווית לעומק חייב להיות באותה מערכת צירים של המצלמה** – כלומר, אם Unreal נותן מיקום בעולם, יש לתרגם אותו לקואורדינטות יחסיות למצלמה <sup>7</sup> (עוד פירוט בסעיף 5). אם נושא זה חדש, מומלץ ללמוד על תהליכי **Domain Randomization** בסימולציות (שינוי פרמטרים אקראי כדי שהמודל לא יתאים עצמו מדי לפרטים ספציפיים), ועל שיטות ייצוא נתונים מ-Unreal (למשל שימוש ב-UnrealCV, או כתיבת סקריפט באנג'ין שמצלם ושומר זוגות של תמונה+נתוני מיקום).

- **אוגמנטציה נתונים (Data Augmentation):** זהו נושא מוכר בראיית מכונה, וסביר שכבר השתמשתם בו, אך שווה להדגיש אותו בהקשר שלנו. מלבד הגיוון הסינתטי שניתן להשיג מתוך המנוע (כאמור – תנאי תאורה, זוויות טיסה, מרחקים וכו'), אפשר ומומלץ גם לבצע אוגמנטציה קלאסית על התמונות הסינתטיות בעת האימון. למשל: הוספת רעש, טשטוש קל, שינויי בהירות/ניגודיות, היפוך תמונה וכדומה. מטרת האוגמנטציה היא לשפר את יכולת ההכללה (generalization) של הרשת כך שלא תלמד פרטים ספציפיים מדי בסצנות הסינתטיות. במיוחד אם תכננתם בסוף לבדוק את האלגוריתם על וידאו אמיתי, אוגמנטציה תעזור לגשר על הפער. יש כלים נוחים בפיתוח לביצוע אוגמנטציות (כמו Albumentations, או המודול torchvision.transforms), ואפשר לשלב זאת בצינור הטעינה של הדאטה.

בנוסף לנושאים העיקריים הללו, תשתמשו ודאי בידע כללי בלמידה עמוקה (אימון מודלים, פונקציות הפסד – למשל MSE עבור גרסיית מרחק, אופטימיזציה, וכו'). ייתכן ותיישמו **למידת מעבר (Transfer Learning)** – למשל, שימוש במודל עומק טרום-מאומן והתאמתו לדאטה שלכם, דבר שדורש הבנה איך לטעון מודל מאומן ולהמשיך לאמן אותו על דאטה חדש.

לסיכום, הרשימה לעיל מרכזת את עיקר התחומים שבהם כדאי לוודא שיש לכם ידע מספק, ולזהות אילו דורשים למידה נוספת לפני או במקביל לביצוע הפרויקט.

### 3. כלים, ספריות וסביבת פיתוח מומלצים

פרויקט מסוג זה מחייב שימוש בערכת כלים מתאימה לתחזוקת קוד, אימון רשתות ועיבוד נתונים. להלן הכלים המרכזיים שתדקקו להם (רבים מהם ייתכן וכבר בידיכם):

- **שפת תכנות וסביבה:** כמעט כל המחקר והפיתוח ייעשו ב-Python, שהיא השפה הנפוצה ביותר ללמידה עמוקה וראייה ממוחשבת. מומלץ לעבוד בסביבת פיתוח שתומכת בהרצת קוד אינטראקטיבית, כגון Jupyter Notebook/Lab (לניסויים מהירים) או סביבות IDE כמו PyCharm ו-VS Code. ודאו שיש לכם גישה לחומרה גרפית – **GPU** – לצורך אימון והרצת המודלים. מודלי עומק מודרניים כבדים יחסית; למשל, במימוש המלא דווח על קצב ~25-30FPS על GPU צרכני (בדומה ל-RTX 3060) <sup>12</sup> <sup>13</sup>. אם אין בידכם GPU מקומי חזק, ניתן להשתמש בשירותים כמו Google Colab או Kaggle Notebooks המספקים GPU בענן לניסויים.

- **ספריות למידה עמוקה (Deep Learning):** הפרויקט צפוי להתבצע ב-PyTorch (בהתחשב שרוב דוגמאות הקוד והמודלים המוכנים נמצאים שם). PyTorch יספק את התשתית לבניית רשתות, טעינת דאטה (באמצעות `torch.utils.data.Dataset`), אימון לולאת למידה וכו'. במידת הצורך ניתן לשקול גם שימוש ב-PyTorch Lightning כדי לפשט קוד אימון, אך זה לא חובה. ספריית torchvision תסייע עם מודלים חזקים קיימים (ResNet, וכו' – אם תרצו להשתמש ב-Transfer Learning) ועם טרנספורמציות לתמונות. אם משהו בצוות מגיע מרקע של TensorFlow/Keras, ניתן תאורטית לממש שם, אך מרבית הכלים הקיימים לנושא (מודלי YOLO, מודלי עומק) זמינים בנוחות בפייתון/PyTorch.

- **כלי זיהוי אובייקטים (YOLO):** לשלב זיהוי הרחפן בתמונה, תוכלו להשתמש במימושים מוכנים של YOLO. ספריית **ultralytics** (ניתנת להתקנה דרך pip) מספקת ממשק נוח לדגמי YOLOv5 ו-YOLOv8 בפייתון, כולל אפשרות אימון על דאטה שלכם. תצטרכו לאסוף את התמונות הסינתטיות ולסמן להן תיבות תיחום של הרחפן כנתוני אימון ל-YOLO (ניתן להוציא מ-Unreal באופן אוטומטי את תיבות התיחום על בסיס מודל התלת-ממד). לאחר אימון מודל כזה, תוכלו להשתמש בו כדי לקבל עבור כל תמונה את מיקום הרחפן בפיקסלים (נניח כ-`(x_center, y_center)` או תיבת תיחום בצורה `[x_min, y_min, x_max, y_max]`). יש גם גרסאות קוד פתוח אחרות כמו darknet (C++), אך בפייתון התהליך יהיה פשוט יותר.

- **מודלים להערכת עומק:** כמתואר בסעיף 4, מומלץ להתחיל משימוש במודל עומק מוכן. לדוגמה, **MiDaS** של Intel או **ZoeDepth** של Intel-ISL. את MiDaS v2.1/v3 אפשר לטעון ישירות דרך PyTorch Hub ללא צורך לשכפל ריפו: לדוגמה, הטענת המודל מתבצעת כך: `midas = torch.hub.load("intel-isl/MiDaS", "MiDaS")` <sup>14</sup>. גם את הטרנספורמציות הדרושות (שינוי גודל ונרמול) אפשר לטעון: `midas_transforms = torch.hub.load("intel-isl/MiDaS", "transforms")` <sup>15</sup>. באופן דומה, קיים ריפוזיטורי עבור ZoeDepth (ב-GitHub תחת intel-isl/ZoeDepth) ואפילו אינטגרציה ל-HuggingFace Transformers, המאפשרים טעינה קלה של הדגם. בשלב ראשון, אין הכרח לבצע אימון מחדש של מודל העומק – אפשר להשתמש במודל מאומן וליישם **כיול סקאלה** כדי לקבל עומק מוחלט (ראו סעיף 5.3 במאמר המצורף) <sup>16</sup>. במידה ותרצו **לאמן מודל עומק בעצמכם** על הדאטה הסינתטי, תדקקו לספריות הנ"ל בכל מקרה, ובנוסף אולי לכלי מעקב ניסויים (TensorBoard למשל) כדי לצפות בתוצאות.

- **ניהול וקדם-עיבוד של נתונים:** ספריות כמו **OpenCV** ו-Pillow (PIL) שימושיות לטעינת תמונות, המרת פורמטים, ועיבוד בסיסי (חיתוך תמונה לפי bounding box, שינוי גודל, וכו'). OpenCV גם מספק פונקציות לציון תיבות תיחום על תמונה – שימושי להדמיה ולבדיקה. אם יש לכם פורמט ייצוא מ-Unreal (למשל קובצי JSON עם מיקום רחפן לכל פריים), תוכלו להשתמש בספריית **json** הפייתונית או Pandas לקריאת קבצים אלה, ולחיבור בין הרשומות לתמונות.

בניית מערך הנתונים לאימון תעשה כנראה באופן ייעודי: אפשר לכתוב Dataset שפשוט קורא לכל אינדקס את התמונה ואת הערך \$Z\$ המתאים מה-JSON. ודאו שסידור הקבצים (תיקיות, שמות) מובנה והגייוני כדי למנוע שגיאות. כמו כן, אם אתם מפיקים **מפות עומק אמיתיות** מ-Unreal (לכל פיקסל), קבצים אלו יכולים להיות בפורמט תמונת עומק או EXR – צריך לוודא שאופן הטעינה שלהם נכון (למשל OpenCV יודע לקרוא TIFF/PNG של 16-ביט, ו-Numpy יכולה לקרוא EXR דרך imageio וכו').

- **ספריות לאוגמנטציה:** כאמור, Albumentations היא ספרייה חזקה ופשוטה ליישום אוגמנטציות על התמונות בזמן הטעינה. היא משתלבת יפה עם PyTorch (מחזירה תמונת Numpy שאפשר להפוך לטנזור). גם torchvision.transforms יכולה להספיק לדברים בסיסיים (היפוכים, חיתוך אקראי, שינוי בהירות/ניגוד). אוגמנטציות אפשר להגדיר בתוך מחלקת ה-Dataset או בשלב יצירת ה-batch.

- **כלים למסנן קלמן ולמעקב:** את שלב הקלמן אפשר לממש "מאפס" עם Numpy, אך קל יותר להשתמש בספרייה כמו **pykalman** או **FilterPy** בפיתון, המספקות מימושי Kalman Filter ו-EKF. מסנן קלמן דורש להגדיר וקטור מצב, מטריצות תהודה ורעש וכו' – בפרויקט הדוגמה הגדירו את המצב כ-6 מימדים  $[X, Y, Z, V_x, V_y, V_z]$  ושילבו את חישובי  $X, Y, Z$  שציינו קודם <sup>6</sup>. ספריות אלה יחסכו לכם קצת זמן בניהול האלגברה. לחלופין, יש מימושי Kalman ב-OpenCV (אם עובדים ב-C++), אבל נניח שתעבדו בפיתון.

- **סביבת Unreal Engine:** למרות שזה לא בדיוק "ספרייה" בפיתון, כלי הליבה להפקת הדאטה הוא מנוע Unreal Engine 5 (או 4). ודאו שיש לכם גישה לפרויקט Unreal שבו יש סצנה עם רחפן ומצלמה וירטואלית. כדאי להשתמש בתוסף **Python ל-Unreal** המאפשר להריץ סקריפטים שמצלמים תמונות ושומרים מידע. לחלופין, ניתן להריץ את הסימולציה ולשמור וידאו + קובץ לוג (כפי שתואר במאמר). ייתכן ושימוש בחבילת **AirSim** של Microsoft (סימולטור רחפנים מבוסס Unreal) יכול לספק קיצור דרך, שכן הוא מוכן לצלם תמונות ממצלמה על רחפן ולתת גם מידע מיקום. כך או כך, כלי Unreal הוא חלק בלתי נפרד מניהול הדאטה הסינתטי. בנוסף, תזדקקו לכלי עריכת נתונים – למשל, אם תגלו פיקסולים חריגים במפות העומק או צורך להתאים את פורמט הקואורדינטות, ייתכן שתכתבו סקריפט לתיקון/סינון הדאטה.

**סיכום כלי הפיתוח:** השילוב של Python עם PyTorch יתן מענה לרוב צרכי המימוש (אימון מודלים, הרצת רשתות מאומנות בזמן אמת, וכד'). Unreal Engine משמש ליצירת הדאטה. ספריות עזר כמו FilterPy, Albumentations, OpenCV וכו' ישלימו את התמונה לעיבוד נתונים ומעקב. הקפידו לעבוד בסביבה שיש בה מספיק זיכרון GPU (מודלי עומק גדולים צורכים ~2GB ומעלה לשרת אחד, ו-YOLO קטן עוד כמה מאות מגה), ובצעו בדיקות לכל רכיב בנפרד לפני שמחברים אותם לצינור שלם.

## 4. ארכיטקטורת רשת עומק מומלצת להתחלה

ישנם מודלים רבים להערכת עומק ממבט יחיד, אך נרצה לבחור בפתרון **פשוט יחסית ללמידה וליישום** – כזה שיהיה מספיק טוב להוכחת ההיתכנות הראשונית. להלן מספר אפשרויות מובילות, והמלצתנו:

- **MiDaS (Mixed Depth Estimation):** זהו מודל חד-עיני שפותח ע"י אינטל, ונחשב *baseline* חזק למשימות עומק מונוקולרי <sup>17</sup>. יתרונו הוא שיש דגמים מאומנים זמינים ("off-the-shelf") שניתן פשוט להוריד ולהריץ, ללא צורך באימון יקר. MiDaS שולב במערכי נתונים רבים כדי להיות רובסטי (מיועד להערכת עומק יחסי כללי). הגרסאות החדשות מבוססות על Transformer (דגם DPT) בתצורת Encoder-Decoder <sup>18</sup> – ארכיטקטורה מודרנית שנותנת תוצאות מדויקות בהרבה מרשתות CNN מסורתיות. היתרון הגדול: קל **מאוד להשתמש** במודל הזה – טעינה דרך PyTorch Hub, וקבלת מפות עומק איכותיות ישירות מתמונות קלט <sup>14</sup> <sup>19</sup>. החיסרון: הוא נותן עומק בסקאלה יחסית שרירותית. בפרויקט שלכם, זה לא בהכרח בעיה – כפי שנפרט בסעיף 5, ניתן לבצע **כיול** כדי להמיר עומק יחסי למטרים <sup>16</sup>. MiDaS תהיה בחירה מצוינת להתחלה: תלמדו איך להשתמש בו, אולי תבצעו Fine-tune קל על הנתונים הסינתטיים שלכם (אם תרצו לשפר ביצועים), ותקבלו baseline של עומק. אפילו בפעולה ללא אימון נוסף,

MiDaS מספקת עומק יחסי המבוסס על רמזי פרספקטיבה, גודל, טקסטורה וכו' שהיא למדה ממיליוני תמונות <sup>5</sup> - מה שיכול להספיק כדי לספק סדר גודל נכון של מרחק לרחפן.

**ZoeDepth:** מודל חדש (2023) מבית intel-isl שנועד להתמודד עם בעיית הסקאלה ע"י שילוב בין למידת עומק יחסי ועומק מטרי. **ZoeDepth** משתמש באדריכלות של MiDaS כבסיס, אך מוסיף לה ראש נוסף הנקרא *Metric Bins* **Module** המאפשר למודל להפיק עומק בסקאלה אבסולוטית (מטרים) לאחר שעבר **כיוול על דאטה מטרי** <sup>20</sup>. האימון של Zoe כלל שלב פרה-טריינינג על דאטה יחסי (כמו MiDaS) ואז Fine-tuning על ערכי עומק מוחלט (ממאגרים כמו KITTI, NYU). בפועל, ZoeDepth מסוגל לתת הערכת מרחק ישירה במטרים ללא כיוול נוסף, בתנאי שהנתונים דומים לדאטה שבו הוא אומן. השימוש ב-Zoe יכול לחסוך לכם את שלב כיוול הסקאלה, אך יש לשים לב: אם תנאי הסצנה שלכם (נופים או טווחי מרחק) שונים מאוד מהדאטה שעליו Zoe אומן, עדיין ייתכן צורך לכייל או לאמן אותו מחדש על הדאטה הסינתטי. ZoeDepth זמין בקוד פתוח (ראו GitHub לעיל) ודרך ספריית HuggingFace Transformers, כך שההרצה שלו די נוחה. הוא מעט **כבד יותר** מ-MiDaS (עקב תוספת הראש וכנראה דגם backbone גדול), אבל עדיין רץ בזמן אמת (עד ~10-15 FPS על GPU בינוני לפי דיווחים). למידה שלו מעט מורכבת יותר ממודל סטנדרטי כי צריך להבין את עניין ה-bins המטריים, אך תיעוד המודל והמאמר קיימים. Zoe מומלץ אם אתם שואפים לדיוק גבוה יותר בעומקים המוחלטים כבר מההתחלה, או אם תרצו לנסות גישה עדכנית שמשיגה תוצאות state-of-the-art. אפשר להתחיל עם MiDaS ואחר כך לעבור ל-ZoeDepth כניסוי מתקדם. שימו לב ש-Zoe כבר לא מפותח אקטיבית ע"י אינטל (לפי ה-GitHub), אך זה לא קריטי לשימושכם.

**"Depth Anything" / "Depth (Everything)":** מודל עומק בסגנון "מודל יסוד" (Foundation Model) שפורסם לאחרונה, המאומן על כמות דאטה אדירה - לפי האתר, **1.5 מיליון תמונות עם תוויות עומק ועוד 62 מיליון תמונות ללא תוויות** <sup>21</sup>. הרעיון שהוא יכול לשמש כמודל עומק כללי וחזק מאוד כמעט לכל סצנה. היתרון: אמור לספק איכות ורובסטיות מירביות בהערכת עומק, אולי טובות אף יותר מ-MiDaS/Zoe. החיסרון: מודל כבד (בהתאם לכמות הנתונים), פחות נפוץ בשימוש בקהילה נכון להיום, ודאי רץ לאט יותר. כמו כן, ייתכן ש-"Depth Anything" נועד בעיקר להפקת **מפת עומק מלאה** ולא לכיוול מדויק של אובייקט בודד. אם תרצו, אפשר להתנסות גם בו - הוא זמין (depth-anything github), אך להערכה ראשונית ייתכן וזה overkill.

**המלצה מעשית:** התחילו עם **MiDaS** או **ZoeDepth**. MiDaS יספק לכם את הדרך הפשוטה ביותר לקבל תוצר ולעבור את כל השרשרת הטכנית (טעינת תמונה → הרצת רשת → הפקת עומק לרחפן). ZoeDepth יכול להיות הצעד הבא אם זקוקים לשיפור בדיוק המטרי. שניהם קלים לשילוב בקוד (כמה שורות טעינה והרצה כפי שתואר) <sup>19</sup> <sup>22</sup>. בהמשך, לאחר שתקבלו תוצאות בסיסיות, תוכלו לבחון אם כדאי לאמן מודל מותאם אישית. למשל, אפשר **לאמן רשת קלה בעצמכם** שתעשה גרסיה של מרחק הרחפן מתמונה: לקחת מודל כגון ResNet-18 מוכן (TorchVision) ולשנות את השכבה האחרונה שלו להוציא נוירון יחיד, ולאמן על תמונות (או קרופים של הרחפן) עם תוויות \$\$\$ אמיתיות. זה אמנם "פשוט" אך דורש כמות דאטה רבה כדי להגיע לדיוק גבוה, בעוד שמודלים כמו MiDaS כבר למדו ממיליוני דוגמאות של רמזי עומק <sup>5</sup>. לכן, הגישה ההיברידית (זיהוי + מודל עומק מאומן + כיוול) היא קיצור דרך יעיל. לסיכום, כמודל עומק ראשוני - MiDaS (v3 עם DPT-large) מומלץ כפתרון שקל ללמוד אותו ולהתחיל להפיק תוצאות.

## 5. דגשים מיוחדים בעבודה עם דאטה סינתטי מ-Unreal Engine

כעת נתרכז בהיבטים ייחודיים לפרויקט כפי שהוגדר - שימוש ב-Unreal Engine להפקת הדאטה, ודרישות הנובעות מכך לצורך הערכת עומק מוצלחת:

**יישור מערכות הקואורדינטות ואמת המידה:** אחד היתרונות של Unreal Engine הוא היכולת לקבל **קואורדינטות אמת** של האובייקטים בסצנה <sup>2</sup>. ואכן, בקובצי הלוג/JSON שלכם כנראה יש לכל פריים את מיקום הרחפן. **חשוב לוודא שהמיקום הזה מומר למערכת המצלמה.** במילים אחרות, אם Unreal מספק מיקום ב-"World Space" (נניח קואורדינטות גלובליות במרחב המשחק), יש לבצע טרנספורמציה למערכת הצירים של המצלמה (Camera Space) כך שה-Z הוא למעשה המרחק מהמצלמה. במאמר המצורף מודגש: "המיקום חייב להיות במרחב המצלמה, לא במרחב העולם" <sup>23</sup>. הדרך לעשות זאת היא: לקחת את וקטור המיקום בעולם של הרחפן, לגרוע ממנו את וקטור המיקום

בעולם של המצלמה, ואז להחיל טרנספורמציה סיבוב הפוכה של המצלמה <sup>24</sup>. פעולה זו תניב וקטור  $(X, Y, Z)$  במרחב המצלמה, כאשר  $Z$  הוא העומק הישר קדימה. ודאו שתהליך זה מיושם נכונה כשאתם מכינים את תוויות ה-Z לאימון/בדיקה – אחרת תכניסו שגיאת כיוול משמעותית. הערה נוספת: כדאי לבצע את רישום הנתונים בצורה סינכרונית ומדויקת – לדוגמה, לרשום עבור כל פריים את מספר הפריים והמיקום, ולא להסתמך על timestamps שיכולים לגרום להסטה <sup>11</sup>.

**התאמת פרמטרי המצלמה הווירטואלית:** כדי שהערכת העומק והמיקום תהיה נכונה, המצלמה בסימולטור חייבת להתנהג כמו המצלמה האמיתית המשוערת. משמעות הדבר היא להגדיר נכון את הפרמטרים הפנימיים: אורך מוקד, גודל חיישן, רזולוציה, וכו'. במאמר מפורט כיצד הגדירו מצלמה ב-Unreal: לדוגמה, בחירת אורך מוקד (16–35 מ"מ), גודל חיישן ידוע, ורזולוציה  $1080 \times 1920$  <sup>25</sup>. מתוך אלה מחשבים את זווית הראייה (FOV) ומפיקים את מטריצת המצלמה  $K$  <sup>26</sup>. בעת חישוב מיקום ה- $X, Y$  של הרחפן מתוך העומק, אתם חייבים להשתמש באותם  $f_x, f_y, c_x, c_y$  של המצלמה הווירטואלית <sup>27</sup>. אם יהיה פער, תקבלו שגיאת קנה-מידה (למשל, העומק יכול להיות נכון אבל ה- $X, Y$  יצאו לא מדויקים, או להיפך). **לכן, יש לדאוג שפרמטרי המצלמה וירטואלית תועדו ומוכרים לכם**, וליישם אותם בכל חישוב (ובתהליך הכיול, אם יש). יתרה מזאת, אם אתם מתכננים להזין את פרמטרי המצלמה לרשת כלשהי (יש מודלים שיכולים לקבל FOV כקלט), וודאו שהדבר נעשה. לסיכום, מצלמת ה-Unreal צריכה להיות כיוול-פנימי "קרקע אמת" עבור הפרויקט – השקיעו זמן בהגדרתה בדיוק כפי שתצטרכו לבצע את החישובים. המקור מדגיש שכל חוסר-התאמה יוביל להטיות שיטתיות <sup>28</sup>.

**כיוול עומק והמרה לעומק מוחלט:** כפי שנזכר, רשתות כמו MiDaS מספקות עומק יחסי. מכיוון שברשותכם נתונים סינתטיים עם מרחק אמיתי, אפשר לנצלם כדי לכייל את הרשת. במאמר מציעים תהליך **כיוול סקאלה (Scale Calibration)**: להריץ את רשת העומק על מגוון דוגמאות סינתטיות (נניח פריימים שבהם הרחפן במרחקים ידועים 20m, 50m, 100m וכו'), לרשום עבור כל תמונה את ערך העומק שהרשת חוזה עבור הרחפן (למשל עומק ממוצע/מרכז תיבת התיחום), ולהשוות אותו לעומק האמיתי. מתוך סדרת זוגות (PredictedDepth, TrueDistance) ניתן להתאים פקטור  $s$  כך ש- $s \times \text{predicted} \approx \text{true}$  <sup>29</sup>. למעשה, מניחים שהרשת טועה בקנה-מידה קבוע (למשל מכפילה הכל ב-0.1), ואז  $s$  יהיה ההופכי של הטעות. את  $s$  מוצאים למשל ע"י רגרסיית מינימלי ריבועים על הנתונים <sup>29</sup>. לאחר מכן, כל פעם שהרשת חוזה עומק, נכפיל אותו ב- $s$  כדי לקבל מטרים. במאמר הראו שיש לשמור חלק מהנתונים לאימות – כדי לוודא שהכיוול תקף לטווח רחב <sup>30</sup>. תהליך זה פשוט לביצוע (ניתן לממש בכמה שורות עם NumPy/סקיפי) אך קריטי לדיוק מוחלט. **דגש:** אם הרשת שתשתמשו היא ZoeDepth או דומה שמראש מנסה לתת עומק מטרי, עדיין כדאי לבדוק אם יש סטייה וליישם פקטור תיקון אם צריך.

**חישוב עומק הרחפן מתוך מפת העומק / תיבת התיחום:** נשאלתם לגבי "הסקת עומק מתוך ה-bounding box על האובייקט בלבד". אכן, לאחר שיש לנו תיבת תיחום של הרחפן וזוהי המטרה העיקרית, אין טעם לקחת בחשבון את כל שאר חלקי התמונה עבור חיזוי העומק הסופי. יש כמה דרכים לעשות זאת:

**1. שימוש במפת עומק מלאה:** בגישה זו, נריץ את רשת העומק על כל התמונה ונקבל **מפת עומק דו-ממדית** (ערך עומק לכל פיקסל). לאחר מכן, נחתוך מתוך מפת העומק את האזור של תיבת התיחום של הרחפן, ונחשב מזה עומק מייצג – למשל, אפשר לקחת את **החציון או הממוצע** של ערכי העומק בתוך התיבה בתוך עומק הרחפן. החציון עשוי להיות עמיד לרעש/פערים (אם חלק מהתיבה כולל רקע או עצמים אחרים). לחלופין, אפשר לקחת את **ערך העומק בפיקסל המרכזי** של התיבה כהערכה (בהנחה שהמרכז נופל על הרחפן עצמו). כך עשה פחות או יותר המימוש במאמר: קיבלו עומק מנורמל לרחפן מהמודל, כיילו אותו, ואז השתמשו בו לחישוב המיקום <sup>31</sup>. שיטה זו קלה מאוד למימוש (מפת עומק כבר זמינה מהרשת).

**2. רשת עומק ממוקדת אובייקט:** כאן משלבים את התיבה כמידע קלט למודל. אפשרות אחת – **חיתוך (Crop) התמונה לאזור הרחפן** לפני העברת התמונה לרשת העומק. בכך הרשת תתמקד רק באובייקט. זה יכול לעזור אם הרקע מאוד מבלבל או מכיל עומקים שונים מאוד. עם זאת, איבוד הקונטקסט עלול דווקא לפגוע, כי חלק מרמזי העומק קשורים לרקע (למשל גודל הרחפן יחסית לסביבה, קו אופק וכו'). אפשרות אחרת היא להזין את התמונה כולה וגם לתת לרשת סוג של מסיכת תשומת לב – למשל, להוסיף ערוץ קלט נוסף המסמן את האזור המעניין (התיבה) באחד. יש עבודות שמשלבות כך מידע קשיח לתוך רשת

(בדומה ל-attention map). זה מתקדם יותר ודורש שינוי בארכיטקטורה, כך שלא נמליץ על כך לשלב התחלתי. ברמת התחלה, מומלץ דווקא להשתמש בגישה הראשונה: **קבלו את מפת העומק המלאה והפיקו ממנה את עומק הרחפן**. זה מהיר, לא דורש אימון נוסף, ומנצל את מלוא כוחו של מודל העומק הכללי. אם תגלו שהרשת מושפעת מאוד מפרטים שלא קשורים לרחפן, תמיד תוכלו לנסות לגזור את התיבה ולבצע **אימון רשת ייעודית** על קרופים של הרחפן. למשל, אימון רשת קטנה שתיקח תמונת קרופ של הרחפן ותלמד לרגרס באופן ישיר את  $Z$  (כפי שהצעתי לעיל כחלופה). אבל כדאי להשאיר זאת כאפשרות לשיפור מאוחר יותר, לאחר שהמערכת הבסיסית עובדת.

**3. שימוש בממדי הרחפן הידועים:** שיטה קלאסית לחישוב עומק אובייקט היא לנצל את גודל האובייקט בפיקסלים בהשוואה לגודלו הפיזי. אם ידוע לכם למשל שהרחפן הוא בקוטר 1 מטר, ותיבת התיחום שלו הנצפית היא 50 פיקסלים, אפשר בקירוב לחשב מרחק:  $Distance = \frac{Size_{real}}{Size_{pixels}}$  (לפי מודל המצלמה הפינולווי) <sup>32</sup>. ואכן, במאמר ציינו שיטה זו כבודקת **משנית** לאימות תוצאות הרשת <sup>33</sup>. בפועל, שיטה גיאומטרית כזו יכולה לתת לכם ערך מקורב מאוד, שתלוי בדיוק בתנוחת הרחפן (אם הוא לא ניצב בדיוק, התיבה היא לא בדיוק היטל פשוט) וכן מניח שהרחפן כולו נראה. לכן, ניתן להשתמש בזה כ-"Sanity check" – למשל, אם רשת העומק נתנה 100 מ' אך לפי חישוב גודל זה אמור להיות ~50 מ', תדעו שיש בעיה במקרה הזה. אולי משלבים את הערך הזה במסנן הקלמן בתור מדידה נוספת, או לפחות כניטור. אבל לא הייתי מסתמך על זה כפתרון יחיד, במיוחד אם אין לכם מודל תלת-ממדי מלא של הרחפן.

- **חלוקת טווחים ואיסוף נתונים:** עם נתונים סינתטיים, אתם **שולטים בפיזור הדוגמאות**, ולכן כדאי לוודא כיסוי טוב של כלל טווחי המרחק הרלוונטיים. המחקר המצורף מציין שכדאי להשקיע יותר דוגמאות בטווחים גדולים יותר, כי טעות העומק גדלה עם המרחק <sup>34</sup>. למשל, אם תחום העניין הוא 20-150 מטר, ודאו שיש מספיק פריימים ב-100-150m (כי שם הרשת תתקשה הכי הרבה). כמו כן, גוון תנוחות: הרחפן במרכז הפריים, בצדדים, בתנועה לכיוונים שונים – כדי שהרשת לא תלמד שהתמונה תמיד בנויה באופן אחיד. רצוי גם לגוון **רקעים:** שמים, בניינים, עצים <sup>10</sup>, שכן הרקע משפיע על ניגודיות וקונטרסט מול הרחפן. בניית **תוכנית טיסה** לייצור הנתונים היא רעיון טוב – למשל, במאמר השתמשו בתבניות כמו סריקה ישירה למצלמה, תנועה היקפית סביב המצלמה, גריד בזוויות שונות וכו' <sup>35</sup> – כל זאת כדי לכסות מצבים רבים. שימו לב גם לפרמטרים כמו מזג אוויר (ערפל מקשה על עומק – הוסיפו סצנות בערפל קל) <sup>10</sup>.

- **הערכת הביצועים על דאטה סינתטי:** יתרון של דאטה סינתטי הוא שיש לכם Ground Truth מדויק לכל פרט. הגדירו מראש מדדים לבדיקת הביצועים: למשל **שגיאת מרחק ממוצעת** (Mean Absolute Error) במטרים, ואולי **שגיאה יחסית (%)** עבור טווחי מרחק שונים <sup>36</sup> <sup>37</sup>. כך תוכלו לדעת עד כמה המערכת מדויקת בכל טווח. במאמר דיווחו, למשל, על שגיאה יחסית ~8-12% למרחקים 50-100m <sup>38</sup> <sup>39</sup>, שזה יכול לשמש לכם כאבן בוחן. כמובן, משום שהכול סינתטי, ייתכן שתגיעו אף לדיוקים טובים יותר, תלוי בכמה מגוון הדאטה. שמרו חלק מהנתונים כסט בדיקה שלא ראתה הרשת מעולם (למשל סצנות אחרות ב-Unreal או זוויות חדשות) כדי לוודא הכללה.

לסיכום חלק זה, עבודה עם Unreal נותנת לכם שליטה מלאה – נצלו זאת לקבוע את תנאי הניסוי באופן אופטימלי, אך בד בבד דאגו להתאים את הנתונים והמודלים בצורה עקבית (מערכות צירים, כיולים, פורמטים). תשומת לב לפרטים הללו בשלב ההכנה תחסוך כאבי ראש כשתריצו את המערכת המשולבת.

## 6. הצעה למימוש התחלתי – שילוב קוד לדוגמה

לאחר הסקירה התיאורטית, הנה הצעה לפרויקט התחלתי והוכחת יכולת, צעד-אחר-צעד, עם רכיבי הקוד העיקריים:

**שלב א': אימון/טעינת מודל זיהוי (YOLO) וקבלת Bounding Box** – אספו מספר תמונות סינתטיות וצרו קובץ אימון עבור YOLO (פורמט YOLO: לכל תמונה קובץ טקסט עם קטגוריה וקואורדינטות תיבה מנורמלות). אימנו מודל YOLO קטן (למשל YOLOv5s) על זה – או דלגו על אימון אם יש לכם דרך אחרת לקבל את תיבת הרחפן (אולי מנוע המשחק יכול לתת bounding box ישירות). לאחר שיש מודל, כתבו קוד Python שמשתמש בו כדי לקבל את תיבת התיחום מהתמונה. עם ספריית ultralytics זה נראה בערך כך:

```

from ultralytics import YOLO
model = YOLO('path/to/custom-yolo.pt') # טען מודל מאומן
results = model(image) # הרץ על תמונה בודדת
boxes = results[0].boxes.xyxy.cpu().numpy() # נניח מחזיר מערך [x1,y1,x2,y2]

```

קבלו את התיבה (או התיבה הרלוונטית אם איכשהו יש יותר).

**שלב ב': הרצת מודל עומק על התמונה המלאה** – נשתמש, למשל, ב-MiDaS הטרומ-מאומן. בעזרת PyTorch Hub אפשר לטעון את הדגם והטרנספורמציות בקלות <sup>19</sup> <sup>22</sup>:

```

import torch, cv2
midas = torch.hub.load("intel-isl/MiDaS", "MiDaS") # טוען מודל גדול (DPT-Large)
midas_transforms = torch.hub.load("intel-isl/MiDaS", "transforms")
transform = midas_transforms.default_transform # טרנספורמציה מתאימה לדגם הגדול
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
midas.to(device).eval()

img = cv2.imread("synthetic_frame.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # RGB-המודל מצפה ל
input_batch = transform(img_rgb).to(device)
with torch.no_grad():
    prediction = midas(input_batch)
    prediction = torch.nn.functional.interpolate(
        prediction.unsqueeze(1),
        size=img_rgb.shape[:2], # התאם בחזרה לגודל התמונה המקורי
        mode="bicubic",
        align_corners=False
    ).squeeze()
depth_map = prediction.cpu().numpy()

```

הקוד הנ"ל יפיק לנו מטריצת עומק `depth_map` בגודל התמונה. הערכים הם ביחס שרירותי (לא במטרים). אפשר להדפיס כמה ערכים כדי להתרשם מהסקאלה.

**שלב ג': חילוץ עומק הרחפן מתוך המפה** – נשתמש בתיבת התיחום שקיבלנו בשלב א'. נניח שהיא במשתנה `box = [x1, y1, x2, y2]` בפיקסלים (ברזולוציית התמונה המקורית). נחלץ את איזור העומק וניתן הערכה:

```

x1, y1, x2, y2 = map(int, box)
drone_region = depth_map[y1:y2, x1:x2]
drone_depth_pred = np.median(drone_region) # עומק יחסי חזוי של הרחפן
print("Predicted relative depth:", drone_depth_pred)

```



כך קיבלנו את עומק הרחפן כמו שחווה מודל MiDaS. כעת, בהינתן ערך זה, נוכל **לכיל** אותו. אם למשל יש לנו מה-Unreal את המרחק האמיתי של הרחפן בפריים הזה (נניח 50 מטר), נוכל לחשב פקטור כיל  $s = \frac{\text{true\_distance}}{\text{drone\_depth\_pred}}$ . כמובן, נעשה זאת על קבוצה רחבה של דוגמאות ונתאים  $s$  אופטימלי (למשל ממוצע או גרסיה). נוכל גם להעריך את טעות הרשת לפני כיל.

**שלב ד': כיל סקאלה ואימות** – אספו N דוגמאות (תמונות) עם רחפנים במרחקים ידועים שונים. הריצו את הקוד משלבים א-ג על כולן, וצרו רשימה של זוגות: [(pred1, true1), (pred2, true2), ...]. באמצעות NumPy אפשר בקלות לחשב את פקטור הכיל שממזער ריבועים:

```
import numpy as np
preds = np.array([pred1, pred2, ...])
trues = np.array([true1, true2, ...])
s_optimal = np.sum(preds * trues) / np.sum(preds**2)
```

זו נוסחת גרסיה ליניארית פשוטה למציאת  $s$  כך ש- $s \cdot \text{pred} \approx \text{true}$ . השתמשו ב-70% מהנתונים לזה (כמו שמציע המאמר <sup>30</sup>) וביתר כדי לוודא שהשגיאה indeed קטנה משמעותית אחרי כיל. לאחר שמצאתם  $s$ , תוכלו ליישם את התיקון בכל הרצה:  $\text{depth\_meters} = s_{\text{optimal}} * \text{drone\_depth\_pred}$ . כעת  $\text{depth\_meters}$  אמור להיות הערכת מרחק במטרים.

**שלב ה': המרה לקואורדינטות מרחביות (אופציונלי):** אם תרצו בשלב הזה גם להפיק את מיקום ה-X,Y של הרחפן, זה פשוט כשיש עומק מתוקן. יש להכניס את ערך  $Z$  ואת מרכז התיבה (או מרכז המסה של הפיקסלים של הרחפן) לנוסחאות:

$$X = \frac{(x_{\text{center}} - c_x) \cdot Z}{f_x}, \quad Y = \frac{(y_{\text{center}} - c_y) \cdot Z}{f_y}$$

כאשר  $(c_x, c_y)$  הם מרכז התמונה (מחצית הרוחב/גובה) ו- $(f_x, f_y)$  אורך המוקד בפיקסלים בכיוון אופקי/אנכי. אותם כדאי לחשב לפי הגדרות המצלמה שלכם (ראו חלק 5, "התאמת פרמטרי מצלמה"). כך תקבלו נקודה תלת-ממדית. אפשר להשוות זאת לנקודת ה-GT מ-Unreal כדי לוודא שהכל תקין.

**שלב ו': שילוב מסנן קלמן (לניתוח המשכיות בווידאו):** אם תעבדו על רצף תמונות (וידאו), מומלץ לעטוף את התוצאה של שלב ה' בפילטר קלמן כדי להחליק את המסלול. ניתן להשתמש בספריית FilterPy:

```
from filterpy.kalman import KalmanFilter
kf = KalmanFilter(dim_x=6, dim_z=3) # example: state [X, Y, Z, Vx, Vy, Vz],
measurement [X, Y, Z]
# בהתאם למודל התנועה (למשל תנועה קבועה + רעשים) F, H, Q, R ... הגדרת מטריצות
# אתחול מצב התחלתי ע"ס מדידה ראשונה
for measurement in measurements: # loop over frames
    pred_state = kf.predict()
    kf.update(measurement) # measurement could be [X_meas, Y_meas, Z_meas]
    filtered_state = kf.x
```

ברגע שהקלמן מכויל (ניתן להיעזר במדריכים או בקוד מוכן), הוא יקטין הבדלי מסגרת-למסגרת. במאמר צוין שהקלמן מסייע לנטרל ריצודים ואף לחזות בהיעדר זיהוי בפריים בודד <sup>8</sup>. בשלב ראשון אולי תבדקו את המסנן offline על נתוני מסלול שלמים.

**שלב ז': הרצת הכל ובדיקת הביצועים:** כעת כשכל הרכיבים קיימים – זיהוי YOLO, הערכת עומק, כיול, המרה ו-(אופציונלי) קלמן – שלבו אותם לתוכנית אחת שרצה על רצף פריימיים. בצעו השוואה מול ה-Ground Truth בכל שלב: לפני כיול ואחריו, לפני קלמן ואחריו. חישוב את ה-MAE, אחוזי שגיאה, וכד'. לדוגמה, ייתכן ותראו שבטווח 20-50m השגיאה המוחלטת היא  $\pm 2-4m$  (כ-5-8%), ובטווח 100-150m השגיאה  $\pm 12-27m$  (כ-12-18%)<sup>40</sup>. נתונים כאלה דומים למה שהושג במאמר, ומהווים אישור שהשיטה עובדת בתחום המצופה. אם תראו שגיאות גדולות בהרבה, חזרו לשלבים קודמים לבדוק היכן הפער – האם הזיהוי bbox לא מדויק? האם מודל העומק מתבלבל בתנאים מסוימים? האם הכיול לא מתאים לכל הטווח (יתכן שתצטרכו כיול מקטעים – piecewise – כפי שנרמז אם סטיית הסקאלה שונה בין קרוב לרחוק<sup>41</sup>).

**לסיכום**, לאחר השלמת השלבים לעיל, תהיה לכם מערכת התחלתית מתפקדת: היא תקבל תמונת Unreal ותחזיר מרחק רחפן. המסמך סקר את ההיבטים העיקריים – מהגדרת המשימה, דרך הנושאים הטכניים ללמידה, הכלים למימוש, המודלים המומלצים ועד לדגשים מיוחדים בסביבת Unreal. בהמשך, תוכלו להעמיק בכל רכיב: לשפר את רשת העומק (למשל Fine-tune על הסצנה הספציפית, או ניסיון מודלים אחרים), לשכלל את מודול הזיהוי (אולי מעקב אובייקט בין פריימיים), ולנסות את המערכת על וידאו אמיתי כדי לבחון הכללה. בהצלחה רבה בפרויקט הגמר שלכם! שתהיה למידה פוריה ומוצלחת.

### מקורות וקריאה נוספת:

Monocular Drone Detection and 3D Localization – Technical Paper (שצורף): המאמר הטכני המלא של הפרויקט הסיועני, כולל פירוט הארכיטקטורה, ניסויים ותובנות<sup>1 2 7</sup>. עוד. מומלץ לקרוא אותו לעומק כהשראה ישירה.  
- קוד ופרויקטים רלוונטיים בגיטהב: [intel-isl/MiDaS](https://github.com/intel-isl/MiDaS) – קוד ודגמי עומק, [isl-org/ZoeDepth](https://github.com/isl-org/ZoeDepth) – קוד [yolov5](https://github.com/ultralytics/yolov5) – קוד YOLOv5 לאימון והרצה.  
- בלוגים ומדריכים: הסבר על MiDaS והטמעתו<sup>17 42</sup>, בלוג Medium על ZoeDepth המסביר את הגישה המשלבת Relative+Metric<sup>43</sup>, ומאמרים על Monocular Depth Estimation (למשל מבוא ב-lightly.ai) למי שרוצה יותר רקע תאורטי.

תהנו מתהליך הפיתוח, ואל תשכחו לתעד את ההתקדמות והניסויים לצורך הדוח הסופי. בהצלחה!

1 2 3 4 5 6 7 8 9 10 11 12 13 16 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

Monocular Drone 3D Localization - Technical Paper (2).pdf<sup>41 40 39 38</sup>

[file:///file\\_0000000027b87207b355682aba141446](file:///file_0000000027b87207b355682aba141446)

GitHub - AbirKhan96/Intel-ISL-MiDaS<sup>22 19 15 14</sup>

<https://github.com/AbirKhan96/Intel-ISL-MiDaS>

| Monocular Depth Estimation using ZoeDepth : Our Experience | by Bhaskar Bose<sup>43 42 20 18 17</sup>

Medium

<https://medium.com/@bhaskarbose1998/monocular-depth-estimation-using-zoedepth-our-experience-42fa5974cb59>

Depth Anything<sup>21</sup>

<https://depth-everything.github.io/>