ספר פרוייקט

**יפתח קסורלה**

**מערכת הגנה על גבולות מסוכנים**

Table of Contents

[מבוא 2](#_Toc199882795)

[יזום 2](#_Toc199882796)

[מי הלקוח של המערכת הזו? 2](#_Toc199882797)

[מה המטרות של המערכת? 2](#_Toc199882798)

[אילו בעיות הפרויקט פותר ומה היתרונות שלו? 3](#_Toc199882799)

[**הבעיות שהפרויקט הזה מנסה לפתור:** 3](#_Toc199882800)

[**מה היתרונות והחיסכון של המערכת:** 3](#_Toc199882801)

[**אילו שירותים המערכת תיתן?** 3](#_Toc199882802)

[אפיון 7](#_Toc199882803)

[תיאור תחום הידע 9](#_Toc199882804)

[תיאור הארכיטקטורה של המערכת המוצעת 12](#_Toc199882805)

[תיאור זרימת המידע במערכת 17](#_Toc199882806)

[תיאור האלגוריתמים המרכזיים בפרויקט 19](#_Toc199882807)

[מציאת מיקום הרובוט: ניסוח הבעיה ופתרונות אפשריים 19](#_Toc199882808)

[בחינת פתרונות קיימים ונימוקי פסילתם 19](#_Toc199882809)

[הפתרון הנבחר: זיהוי סמני דרך באמצעות רווחים במסלול 19](#_Toc199882810)

[איתור איומים (חפצים): ניסוח הבעיה ופתרונות אפשריים 20](#_Toc199882811)

[בחינת פתרונות קיימים ונימוקי הבחירה בסונר 20](#_Toc199882812)

[הפתרון הנבחר: זיהוי אובייקטים וקביעת זווית פגיעה אופטימלית באמצעות סונר ומנוע סרבו 21](#_Toc199882813)

[שליטה על הרובה: ניסוח הבעיה והפתרון שנבחר 22](#_Toc199882814)

[הפתרון הנבחר: בקרת זרם באמצעות דרייבר ויצירת PWM מוגבל בזמן 22](#_Toc199882815)

[סביבת הפיתוח 24](#_Toc199882816)

[סיבות לבחירת PyCharm ויתרונותיה על פני סביבות אחרות: 24](#_Toc199882817)

[סביבת פיתוח: Arduino IDE 24](#_Toc199882818)

[סיבות לבחירת Arduino IDE ויתרונותיה על פני סביבות אחרות: 25](#_Toc199882819)

[תיאור פרוטוקול התקשורת 26](#_Toc199882820)

[סוגי הודעות 27](#_Toc199882821)

[תיאור מסכים בפרויקט 30](#_Toc199882822)

[תיאור מבני הנתונים: 33](#_Toc199882823)

[טיפול בחולשות במערכת 34](#_Toc199882824)

[מימוש הפרויקט 35](#_Toc199882825)

[שימוש בספריות חיצוניות במערכת 35](#_Toc199882826)

[תיעוד 38](#_Toc199882827)

[בדיקות 50](#_Toc199882828)

[מדריך למשתמש 54](#_Toc199882829)

[עץ קבצים 54](#_Toc199882830)

[רפלקציה 58](#_Toc199882831)

[ביבליוגרפיה 59](#_Toc199882832)

[נספחים 60](#_Toc199882833)

[server 60](#_Toc199882834)

[Client 1 75](#_Toc199882835)

[Client2 86](#_Toc199882836)

[client3 95](#_Toc199882837)

[Protocol 105](#_Toc199882838)

[Aes 108](#_Toc199882839)

[RSA 110](#_Toc199882840)

[CryptoBase 112](#_Toc199882841)

[Arduino1 113](#_Toc199882842)

[Arduino2 117](#_Toc199882843)

# **מבוא**

## יזום

הפרויקט שבחרתי הוא פיתוח של **מערכת הגנה חכמה לגבולות**. תדמיינו מערכת שיודעת לשמור על גבולות חשובים (כמו גבולות מדינה או מתחמים מסוכנים) בצורה יעילה ובטוחה. המערכת הזו תדע לזהות סכנות – כלומר, אנשים או חפצים שלא אמורים להיות באזור – לאורך קטע גבול שאנחנו מגדירים.

כשהמערכת מזהה איום, היא תעביר לחדר בקרה מידע חשוב כגון: **איפה בדיוק נמצא האיום הזה**. בחדר הבקרה, המפעיל יוכל להחליט מה לעשות. אם הוא יאשר, המערכת תשלח מיד **רובוט מיוחד עם נשק**, שיטפל באיום בעצמו וינטרל אותו באופן אוטומטי.

בחרתי בפרויקט הזה מכמה סיבות: קודם כל, אני חושב שלמערכת כזו יש פוטנציאל אדיר **לשמור על גבולות מסוכנים**, כמו אלה שיש לישראל. היום, כדי לשמור על גבולות, צריך המון מצלמות והמון אנשים. המערכת שלי יכולה להחליף את כל זה במכשיר אחד, או בכמה בודדים. זה יכול לחסוך המון כסף וגם להגן טוב יותר. שנית, אני מאוד אוהב לעבוד עם דברים שאפשר לגעת בהם – חומרה, ולא רק תוכנה במחשב. אני רוצה לראות את התוצאות של העבודה שלי "חיות", פועלות בשטח, וזה מה שהפרויקט הזה מאפשר.

האתגרים שאני צופה בפרויקט הזה הם: לגרום למערכת לזהות דברים בדיוק רב, עבודה מרובה עם חומרה כמו חותים מתנתקים או בעיות עם שבבים כאלה או אחרים, חיבור רובה חיצוני (לא שייך לארדואינו) לאחד מן הרובוטים וסנכרון בין שתי הרובוטים

**מי הלקוח של המערכת הזו?**

המערכת הזו מיועדת בעיקר ל**גופים גדולים ומדינות** – לא לאנשים פרטיים או לעסקים קטנים. היא מתאימה למי שצריך לשמור על גבולות ארוכים או על שטחים גדולים וחשובים (כמו בסיסים צבאיים, נמלים גדולים, שדות תעופה, או מפעלים גדולים).

הלקוחות האלה רוצים לרוב להחליף מערכות ישנות יותר, או לעבור מפתרון שמסתמך בעיקר על אנשים (כמו שומרים) לפתרון טכנולוגי חכם ויעיל יותר. המשתמשים במערכת יהיו אנשי ביטחון, אנשים בחדרי בקרה, ומפקדים, שיסתמכו על המערכת כדי לקבל החלטות חשובות בזמן אמת ולטפל באירועים ביטחוניים.

**מה המטרות של המערכת?**

המטרות העיקריות שאני רוצה להשיג עם המערכת הזו הן:

1. **לזהות במדויק סכנות:** המערכת תצטרך לזהות כל דבר שלא אמור להיות באזור המוגן, בין אם זה אדם או חפץ. חשוב שהיא תהיה מדויקת מאוד כדי לא להפעיל סתם אזעקות שווא.
2. **להעביר את מיקום האיום במהירות לחדר הבקרה:** ברגע שיש זיהוי, המערכת תשלח לחדר הבקרה את המיקום המדויק של הסכנה.
3. **לאפשר למפעיל להחליט מהר:** בחדר הבקרה, המפעיל יקבל הודעה ששואלת אותו אם הוא רוצה שהמערכת תטפל באיום. הוא יוכל לאשר את הפעולה או לדחות אותה.
4. **לשלוח רובוט שינטרל את האיום:** אם המפעיל יאשר, המערכת תשלח מיד רובוט מיוחד. הרובוט הזה ינוע במהירות למיקום האיום, וינטרל אותו באופן אוטומטי (למשל, על ידי ירי).

המטרות האלה הן החזון הגדול שלי לפרויקט. בשלב הזה, אתרכז בבניית הבסיס הטכנולוגי שיאפשר להשיג אותן, תוך מחשבה על איך לשפר ולהרחיב את המערכת בעתיד.

**אילו בעיות הפרויקט פותר ומה היתרונות שלו?**

**הבעיות שהפרויקט הזה מנסה לפתור:**

הבעיה העיקרית שהפרויקט שלי מנסה לפתור היא איך לשמור על שטחים ארוכים או מתחמים גדולים **במינימום אמצעים** (מעט מצלמות, מעט אנשים, מעט חיישנים). היום, אבטחה כזו דורשת המון ציוד יקר והמון אנשים. זה גורם לכמה בעיות:

* **יקר מאוד:** להקים ולתחזק מערכות אבטחה כאלה עולה המון כסף.
* **לא מכסה הכל:** גם אם יש המון ציוד, עדיין יכולים להיווצר "שטחים מתים" או נקודות חלשות שאפשר לחדור דרכן.
* **תלוי באנשים:** כשמסתמכים על אנשים רבים, יש סיכון לטעויות, עייפות, וצריך הרבה כוח אדם.
* **תגובה איטית:** אם אדם מזהה איום דרך מצלמה, לוקח זמן עד שכוחות תגובה מגיעים לשטח.

**מה היתרונות והחיסכון של המערכת:**

המערכת שלי, "מערכת הגנה חכמה לגבולות", תביא איתה המון יתרונות וחיסכון:

* **חיסכון בכסף:** מכיוון שהיא יכולה לכסות שטח גדול עם מכשיר אחד (או כמה בודדים), זה יחסוך המון כסף על קניית ציוד, התקנה ותחזוקה.
* **יותר יעילה:** המערכת תנטר את השטח כל הזמן ובאופן אוטומטי, מה שיאפשר לזהות סכנות הרבה יותר מהר מאשר עם מעקב אנושי.
* **בטיחות לכוח אדם:** במקום לשלוח שומרים לאזורים מסוכנים, המערכת תאפשר לטפל בסכנות מרחוק באמצעות הרובוט, ובכך תגן על חיי אדם.
* **מדויקת ואמינה:** היא תשתמש בתוכנה חכמה כדי לזהות סכנות, מה שיפחית טעויות ויהפוך את המערכת לאמינה יותר.
* **קבלת החלטות מהירה ותגובה מיידית:** המערכת תדע לזהות את מיקום האיום ולבקש אישור לשלוח רובוט כמעט מיד, מה שיקצר דרמטית את זמן התגובה לאירועים.
* **קלה להרחבה:** אם נרצה לכסות שטח גדול יותר בעתיד, יהיה קל להוסיף עוד יחידות למערכת.
* **מרתיעה:** עצם הידיעה שיש מערכת אוטונומית שיכולה לזהות ולטפל בסכנות, עשויה להרתיע מראש אנשים מלהיכנס לאזור.

**אילו שירותים המערכת תיתן?**

המערכת תספק את השירותים העיקריים הבאים:

1. **מעקב תמידי ואוטומטי:** המערכת תסרוק את האזור כל הזמן כדי למצוא תנועה או נוכחות חריגה.
2. **מצא את המיקום המדויק:** היא תאתר את המיקום המדויק של הסכנה בשטח.
3. **ממשק שליטה נוח בחדר הבקרה:** בחדר הבקרה תהיה תוכנה קלה לשימוש, שתראה את מצב הגבול ואת מיקומי האיומים, ותאפשר למפעיל לאשר או לבטל פעולות.
4. **אישור והפעלת רובוט:** המערכת תאפשר למפעיל לאשר במהירות את שליחת הרובוט, ולאחר מכן הרובוט ייצא לדרך באופן אוטומטי.
5. **תגובה אוטומטית מיידית:** לאחר קבלת אישור, הרובוט יישלח אוטומטית למיקום האיום ויפעל לנטרולו (למשל, יירה בו) ללא צורך בהתערבות נוספת של אדם.

**פתרונות קיימים**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | מערכות מכ"ם | מערכות אלקטרו-אופטיות | מערכות חישה | מערכות מודיעין | מערכות הגנה פיזיות |
| עקרון פעולה | מכ"ם משדר פולסים של גלי רדיו. כאשר פולס פוגע באובייקט, חלק מהאנרגיה שלו מוחזרת בחזרה למכ"ם. על ידי מדידת הזמן שלוקח לפולס לחזור, המכ"ם יכול לחשב את המרחק לאובייקט. בנוסף, על ידי ניתוח השינוי בתדר של הפולס המוחזר (אפקט דופלר), המכ"ם יכול לקבוע את מהירות האובייקט וכיוונו. | מערכות אלו פועלות על ידי לכידת אור הנפלט או מוחזר מאובייקטים. מצלמות רגילות קולטות אור, ויוצרות תמונה וידאו. מערכות אינפרא אדום קולטות קרינה תרמית (חום) הנפלטת מאובייקטים, ומאפשרות זיהוי גם בתנאי חשיכה. | חיישנים סיסמיים מזהים רעידות בקרקע שנוצרות על ידי תנועה של אנשים או כלי רכב. חיישני תנועה מזהים שינויים בשדה אלקטרומגנטי או אקוסטי בסביבה, שיכולים להיגרם על ידי תנועה של אנשים או בעלי חיים. | מערכות מודיעין פועלות על ידי איסוף מידע ממקורות שונים, כמו מודיעין אנושי (סוכנים, מקורות מידע), מודיעין אותות (יירוט שיחות טלפון, הודעות רדיו), מודיעין חזותי (צילומי לוויין, תצפיות), ועוד. המידע שנאסף מנותח על ידי מומחים כדי לזהות איומים פוטנציאליים ולנתח כוונות של גורמים עוינים. | מערכות אלו נועדו ליצור מחסום פיזי בפני חדירה. גדרות וחומות נועדו לעכב או למנוע מעבר של אנשים וכלי רכב. תעלות ומכשולים אחרים נועדו להקשות על חדירה, תוך שימוש מרכזי בכוח אדם (שומרים). |
| יתרונות | זיהוי אובייקטים ממרחק רב, יכולת פעולה בכל תנאי מזג אוויר, כיסוי שטח נרחב. | זיהוי פרטים ויזואליים, יכולת פעולה בתנאי חשיכה (אינפרא אדום), עלות נמוכה יחסית למכ"ם. | זיהוי פעילות חריגה, עלות נמוכה יחסית. | זיהוי איומים פוטנציאליים, ניתוח כוונות של גורמים עוינים. | הרתעה, עיכוב חדירה ולא כולל הסתמכות על טכנולוגיה (יכול להתפרש גם כיתרון וגם כחיסרון) |
| חסרונות | עלות גבוהה, דורש תחזוקה שוטפת, רגישות להפרעות אלקטרוניות. | טווח מוגבל, תלוי בתנאי מזג אוויר, דורש עיבוד תמונה מורכב. | רגישות לרעשים סביבתיים, טווח מוגבל. | תלוי באיכות המידע, עלות גבוהה. | עלות גבוהה, לא יעיל נגד חדירה אווירית. |
| מחיר | מיליוני דולרים למערכות מתקדמות | אלפי עד מאות אלפי דולרים, תלוי באיכות המצלמות והמערכות. | מאות עד אלפי דולרים לחיישן. | משתנה מאוד, תלוי בהיקף המערכת והיכולות. | משתנה מאוד, תלוי באורך ובסוג המכשול ובכמות המאבטחים |
| כוח אנושי | צוות טכנאים גדול לתפעול ותחזוקה, מנתחי נתונים לזיהוי איומים. | צוות תצפיתנים לניטור וזיהוי, טכנאים לתחזוקה. | צוות טכנאים לתחזוקה, מנתחי נתונים לזיהוי פעילות חריגה. | צוות מודיעין מיומן לאיסוף וניתוח מידע | צוות תחזוקה ואבטחה |
| **כמות רכיבים**  **אלקטרוניים** | תלוי בדגם ובגודל המערכת, אך לרוב מדובר בעשרות עד מאות רכיבים אלקטרוניים. | תלוי במערכת, אך לרוב מדובר בעשרות רכיבים. | תלוי בסוג החיישן ובמערכת, אך לרוב מדובר בעשרות רכיבים. | תלוי במערכת, אך לרוב מדובר במערכות מחשוב ותקשורת מורכבות |  |
| **פריסה אופטימלית** | תלויה בטופוגרפיה ובאופי האיומים, אך לרוב נפרסים מכ"מים במרחקים של כמה קילומטרים זה מזה. | תלויה בצרכים ובאופי השטח, אך לרוב מדובר במצלמות כל כמה מאות מטרים. | תלויה בטופוגרפיה ובאופי האיומים, אך לרוב נפרסים חיישנים במרחקים של כמה עשרות מטרים זה מזה. |  | בערך מאבטח על 100 מטר גדר |

המערכת שלי לא ממציאה כלום חדש, היא לוקחת שילוב של כמה מערכות קיימות, לדוגמא תצפיתניות האחראיות על בקרה, שימוש של מצלמות וקולטני תנועה (סונר) ורובוטים המסוגלים לזוז. והמערכת משלבת אותם יחד לפתרון שמנסה לשמור על עלויות נמוכות, תחזוקה מינימלית באמצעות הורדת כמות הרכיבים של המערכת היכולים להרס, פריסה גדולה כמה שיותר תוך ניסיון ליצור שמירה מרבית במצבים שונים.

הפרוייקט הולך להשתמש במגבן רחב של טכנולוגיות:

1. Ultrasonic sensor – חיישן שבאמצעות גלי קול מסוגל למדוד מרחק מחפצים
2. Arduino uno – בקר ידוע המשומש בתעשייה, קל ונוח לשימוש ובעל מודולים רבים שיכולים להתחבר אליו
3. 4 dc motors connected to tires – מנועים שמטרתם להזיז את הרובוטים שלי ממקום למקום
4. Transistor – באמצעותו ניתן להוציא זרם גדול יותר מזה הנכנס לו. משומש ע״י המנועים והרובה
5. Hc-05 Bluetooth module – שבבי בלוטוס מוכרים ומומלצים בתעשייה. מאוד נוחים וקלים לשימוש.
6. Line tracking module – באמצעות אור מסוגל להבחין בין צבעים כהים לבהירים ובאמצעות ממימוש יעיל ליצור מעקב אחרי קווים.
7. רובה נרף – רובה שנלקח מצעצוע ילדים על שלט רחוק שהלחמתי וחיברתי לרובוט לשם מימוש של נשק
8. הקוד ייכתב בarduinoIDE בשביל הרובוטים וpython לשרת והלקוחות

הפרוייקט יתמקד בעיקר בתחומים הבאים: 1) תקשורת על בלוטוס לשם מימוש תקשורת מרחוק עם הרובוטים. 2) תקשורת מוצפנת באמצעות tls בין השררת ללקוחות. 3) עבודה צמודת חומרה כגון עבודה על הסונר או על המעקב אחרי קווים. 4) gui נוח וקל למשתמש. 5) database הכולל פרטי התחברות של המשתמשים (כאשר הסיסמאות נשמרות כhash).

## אפיון

נגדיר שלושה דברים

1. הרובוט3 – רובוט מבוסס ארדואינו עם יכולת בלוטוס, סונר, גלגלים ויכולת מעקב אחרי קווים.
2. הרובוט4 – רובוט מבוסס ארדואינו עם יכולת בלוטוס, סונר, גלגלים ,יכולת מעקב אחרי קווים ורובה.
3. השרת – שרת ראשי בפייטון האחראי לחבר בין המצלמה לרובוט

הפרוייקט יעבוד בצורה הבאה, הרובוט3 ייסע במסלול קבוע מראש לאורך הגבול פעם בכמה זמן ישנם עצירות בהם הוא עוצר וסורק את הסביבה באמצעות הסונר ומנוע סרבו. כאשר הרובוט מוצא אובייקט כולשהו הוא שולח הודעה לשרת המרכזי (באמצעות לקוח מקשר) שעוקב אחרי המיקום של הרובוט. השרת שולח הודעה למשתמש שיושב על לקוח נפרד ולמשתמש היכולת לשלוח את רובוט4. הרובוט4 יצא מהחנייה שלו ויתקוף את המטרה ובסיום יחזור למקומו ויחקה למציאת מטרה נוספת

**פירוט הבדיקות**

חיבור לבלוטוס – שליחת הודעה לבלוטוס ואם הוא מחזיר תשובה החיבור פעיל

עבודה של הסונר – הצבת חפץ מול הרובוט ואם נשלחה אזהרה לשרת הסונר עובד

תקשורת מוצפנת – לשלוח הודעות בין הלקוחות לשרת ובאמצעות כלים חיצוניים לנסות לקרוא את החבילות ולראות אם הם מוצפנות

**לוח זמנים**

|  |  |  |
| --- | --- | --- |
| **שלב בפיתוח** | **טווח תאריכים רצוי** | **טווח תאריכים בפועל** |
| **הצעת פרוייקט** | **1.12-20.12** | **1.12-20.12** |
| **מבוא** | **21-31.12** | **21-31.12** |
| **בלוטוס** | **1-7.1** | **1.1-1.2** |
| **עבודה על רובוט ראשון** | **8-31.1** | **2.2-26.2** |
| **קוד לרובוט שני** | **1.2-1.4** | **27.2-1.4** |
| **לקוחות שמתחברים לרובוטים** | **1-14.4** | **2.4-24.4** |
| **שרת** | **15-21.4** | **25.4-10.5** |
| **ספר פרוייקט** | **22.4-22.5** | **11.5-1.6** |
| **רפלקציה** | **23.5-1.6** | **2.6-4.6** |

**ניהול הסיכונים בפרוייקט והדרכים להתמודד איתם**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **מספר סיכון** | **פירוט הסיכון** | **ההסתברות שיקרה** | **מידת השפעה** | **תגובה מתוכננת לסיכון** | **תגובה בפועל** |
| 1 | בעיית התחברות לבלוטוס | גבוהה מאוד | גבוהה | לנסות להשתמש בתוכנות חיצוניות כדי לחבר את הרובוט למחשב | לאחר ניסיונות רבים החלפת הmodule במודל שונה שאמור לעבוד |
| 2 | נטייה קלה של הגלגלים לכיוון מסוים בזמן נסיעה | גבוהה מאוד | בינונית | לשחק עם כיוון הפניה עד לתוצאה הנדרשת |  |
| 3 | קריסה של השרת המרכזי | בינונית | בינונית | ניהול שגיאות יעיל ושימוש בtry exept |  |
| 4 | מעקב לא מדוייק אחרי קווים | בינונית | גבוהה | שינוי במסלול | עבודה מרובה על זוויות והדגשת ניגודים של צבעים ושינוי הטייפ |
| 5 | חוסר הצלחה בעט חיבור הרובה | גבוהה מאוד | בינונית | בדיקת המעגל ניסיון לתיקון מגעים | הוספת נגד בשל דרישת זרם נמוכה של הרובה |
| 6 | בעיות מגעים | גבוהה | נמוכה | תיקון המגעים באמצעות הלחמות |  |

## תיאור תחום הידע

שם היכולת: התחברות למערכת

מהות: התחברות של אחד מן המשתמשים המוכרים לחדר בקרה השולט ברובוט

אוסף יכולות נדרשות:

1. בסיס נתונים
2. ממשק משתמש עם מסך התחברות
3. בדיקת תקינות של המשתמש עם בסיס הנתונים
4. הצפנת tls בין המחשב לבין הסרבר
5. פענוח ההצפנה

שם היכולת: ממשק משתמש

מהות: חדר בקרה המשמש להפעלה ידנית של הפרוייקט, מאפשר להתחיל את הפרוייקט ונותן את האפשרות לתקוף חפצים חשודים.

אוסף יכולות נדרשות:

1. ממימוש gui
2. תקשורת בין השרת לבין השרת המממש את ממשק המשתמש
3. הצפנה במהלך התקשורת
4. התייחסות בשרת לשליטה באמצעות המשתמש

שם היכולת: הצפנה

מהות: יצרית תקשורת בטוחה, שלא ניתנת לפיענוח.

אוסף יכולות נדרשות:

1. תקשורת RSA בין המשתתפים כאשר כל אחד יוצר מפתח בצורה לוקאלית אצלו.
2. יצירת מפתח AES בצורה לוקאלית אצל אחד מן המשתתפים בתקשורת
3. מימוש תקשורת AES בין המשתתפים.
4. תקשורת בסיסית לא מוצפנת
5. פרוטוקול המאפשר קליטה נכונה של המידע

שם היכולת: תקשורת Bluetooth בין הלקוחות לבין הרובוטים

מהות: העברת מידע בזמן אמת בין הרובוטים לבין השרת ללא צורך בחיבור פיזי (כבל)

אוסף יכולות נדרשות:

1. מציאת שבב בלוטוס מתאים
2. חיבור השבב לרובוטים בצורה נכונה שמאפשרת שימוש
3. יצירת פורט במחשבים לשם יצירת שיחה עם הרובוטים
4. מימוש תקשורת על Serial (החבילה שבה אני משתמש לשם תקשורת בלוטוס)

שם היכולת: נסיעה ישר/ימינה/שמאלה/אחורה

מהות: מאפשר לי לשלוט ברובוטים ולגרום להם לזוז לאן שאני צריך

אוסף יכולות נדרשות:

1. חיבור מנועים וגלגלים לרובוטים באמצעות טרנזיסטור כדי להעלות את הזרם שארדוינו יכול להוציא
2. מימוש נסיעה קדימה באמצעות הזזת כל הגלגלים קדימה
3. מימוש נסיעה אחורה באמצעות הזזת כל הגלגלים אחורה
4. מימוש נסיעה ימינה או שמאלה באמצעות הזזה של חצי מן הגלגלים קדימה וחצי מן הגלגלים אחורה (תלוי בכיוון שאני רוצה לפנות)

שם היכולת: מעקב אחרי קווים

מהות: יצירת מסלו נסיעה קבוע לכל אחד מן הרובוטים המדמה את הגבול, או השטח עליו אתה מנסה להגן

אוסף יכולות נדרשות:

1. נסיעה קדימה שמאלה וימינה
2. שימוש בשבב שמוציא אור וקולט אור בחזרה וכך יכול להודיע האם הוא נצא על משהו שחור או לבן.
3. פונקציה מורכבת המנתחת את המידע שמקבלים מן השבב ושומרת על כיוון הנסיעה
4. יצירת משטח כמה שיותר סטרילי כדי שלא יקלו לכלוכים או דברים דומים בעת הקריאה מן השבב
5. יצירת מסלול מאיזולירבנד שחור

שם היכולת: מציאת מיקום הרובוט

מהות: כאשר הרובוט הראשון מוצא איום הוא צריך לשלוח למיקומו את הרובוט השני ולכן הוא צריך לדעת את מיקומו

אוסף יכולות נדרשות:

1. מעקב אחרי קווים
2. תקשורת בלוטוס
3. פונקציה היודעת מתי הקו הופך למקוטע
4. נסיעה ישר קדימה ושמאלה

שם היכולת: שימוש במנוע סרבו

מהות: מנוע שעליו יושב הסונר וכדי שהרובוט יוכל לסרוק לצדדים מבלי להזיז את גלגליו ניתן להשתמש במנוע סרבו

אוסף יכולות נדרשות:

1. תקשורת בלוטוס
2. יכולת שליטה ביציאות של הרדואינו (זמן ומתח)

שם יכולת: סונר

מהות: מציאת איומים

אוסף יכולות נדרשות:

1. מדידת הזמן בין הוצאת הצליל לקבלת הצליל
2. המרת הזמן שחזר לידת מרחק

שם יכולת: מציאת זווית למרחק מינמלי

מהות: הסונר יכול לראות את אותו החפץ ממגבן זוויות שונות ואנו רוצים רק את הישרה (הקרובה ביותר) כדי לכוון את הרובה של הרובוט השני

אוסף יכולות נדרשות:

1. שימוש בסרבו לשינוי הזווית
2. מעקב אחרי הזווית
3. מדידת מרחקים באמצעות הסונר
4. מציאת המרחק המינימלי עם הזווית

שם יכולת: יריה

מהות: נטרול איומים פוטנציאלים בצורה אוטומתית

אוסף יכולות נדרשות:

1. חיבור רובה לרובוט באמצעות טרנזיסטור
2. יצירת זרם pwm
3. שימוש בזרם לזמן קבוע

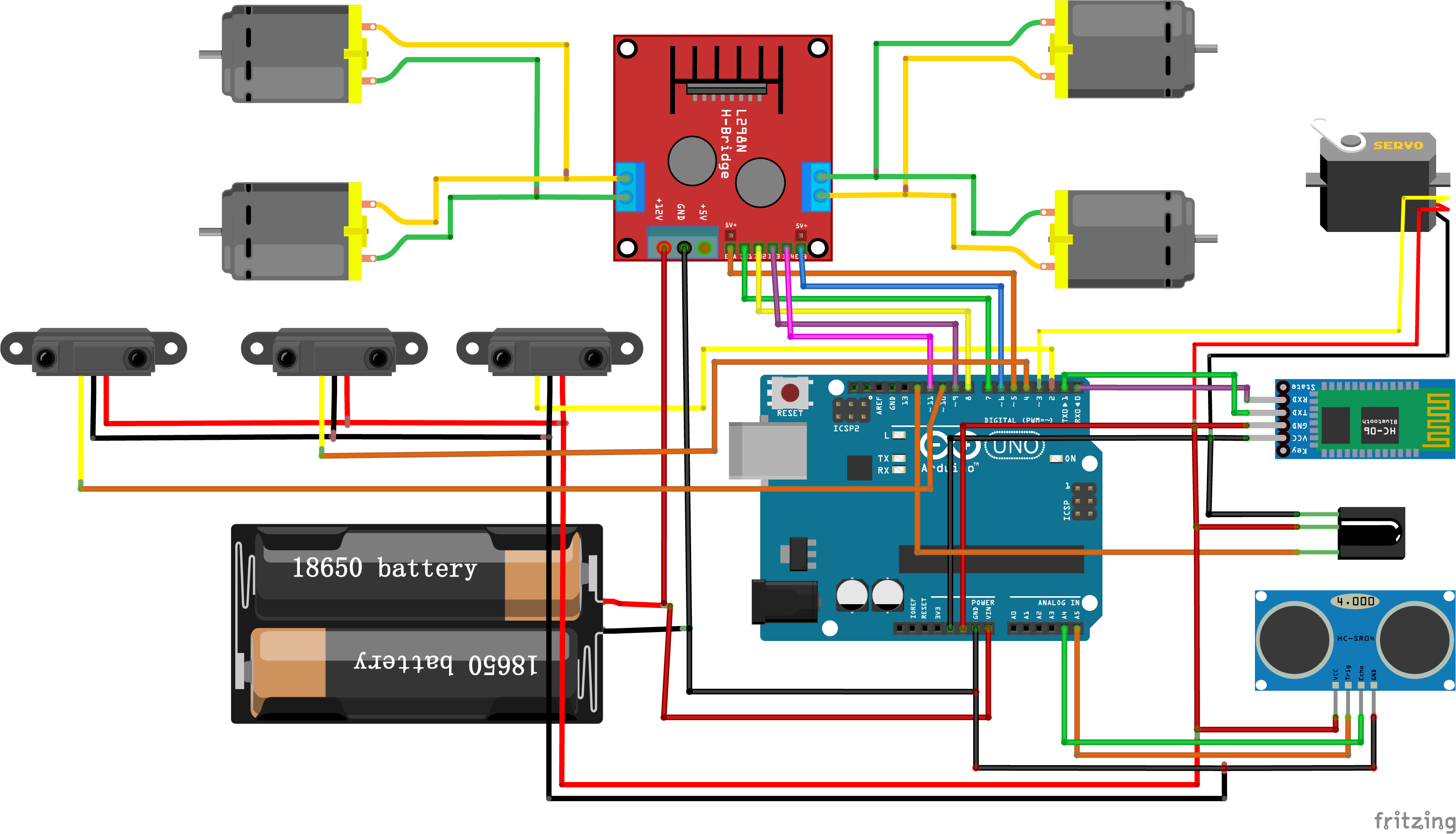
# **תיאור הארכיטקטורה של המערכת המוצעת**

הפרוייקט שלי מורכב משישה חלקים מרכזיים.

1. רובוט סורק
2. רובוט יורה
3. לקוח של רובוט סורק
4. לקוח של רובוט יורה
5. שרת מרכזי (עם database)
6. לקוח עם user interface

הסבר על כל אחד מן הרכיבים

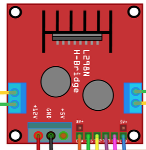
רובוט סורק



**HC-SR04** הוא חיישן מרחק על-קולי שמשתמש בגלי קול כדי למדוד כמה רחוק נמצא עצם ממנו. הוא שולח גל קול, מחכה שיחזור, ומחשב את המרחק לפי הזמן שלקח לגל לחזור. החיישן משמש בפרויקטים פשוטים שצריכים למדוד מרחק, כמו ברובוטיקה או במערכות חניה.

A blue device with black circles

AI-generated content may be incorrect.



ה-L298 Dual H-Bridge Motor Driver הוא רכיב לשליטה על מנועי DC ומנועי צעד. הוא משמש בפרויקטים של רובוטיקה כדי לשלוט על כיוון ומהירות תנועה. למעשה, הוא מאפשר לחבר מנועים הדורשים זרם גבוה יותר ממה שבקרים פשוטים יכולים לספק.

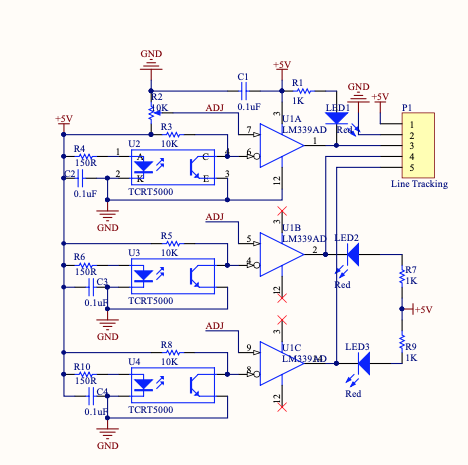
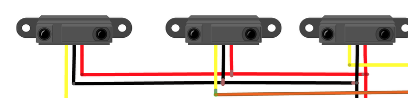
HC-05 הוא **מודול בלוטות'** שמאפשר תקשורת אלחוטית בין מכשירים, כמו ארדואינו לסמארטפון. מטרתו לחבר פרויקטים אלקטרוניים באופן אלחוטי. הוא משמש למשל לשליטה מרחוק על רובוטים או העברת נתונים בין מכשירים בלי כבלים.

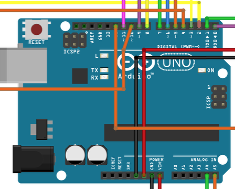


מנוע סרוו הוא מנוע חשמלי קטן ומדויק שמאפשר שליטה מדויקת על הזווית שלו. מטרתו העיקרית היא להזיז דברים למיקום ספציפי ולשמור עליו. הוא משמש ברובוטיקה (למשל, כדי להזיז זרועות רובוטיות) ובפרויקטים שדורשים תנועה מבוקרת ומדויקת.

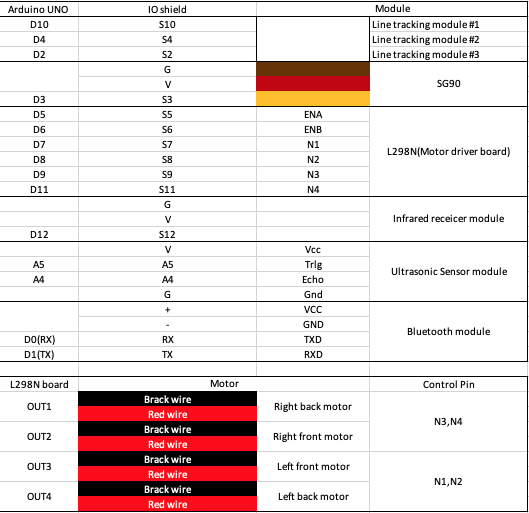
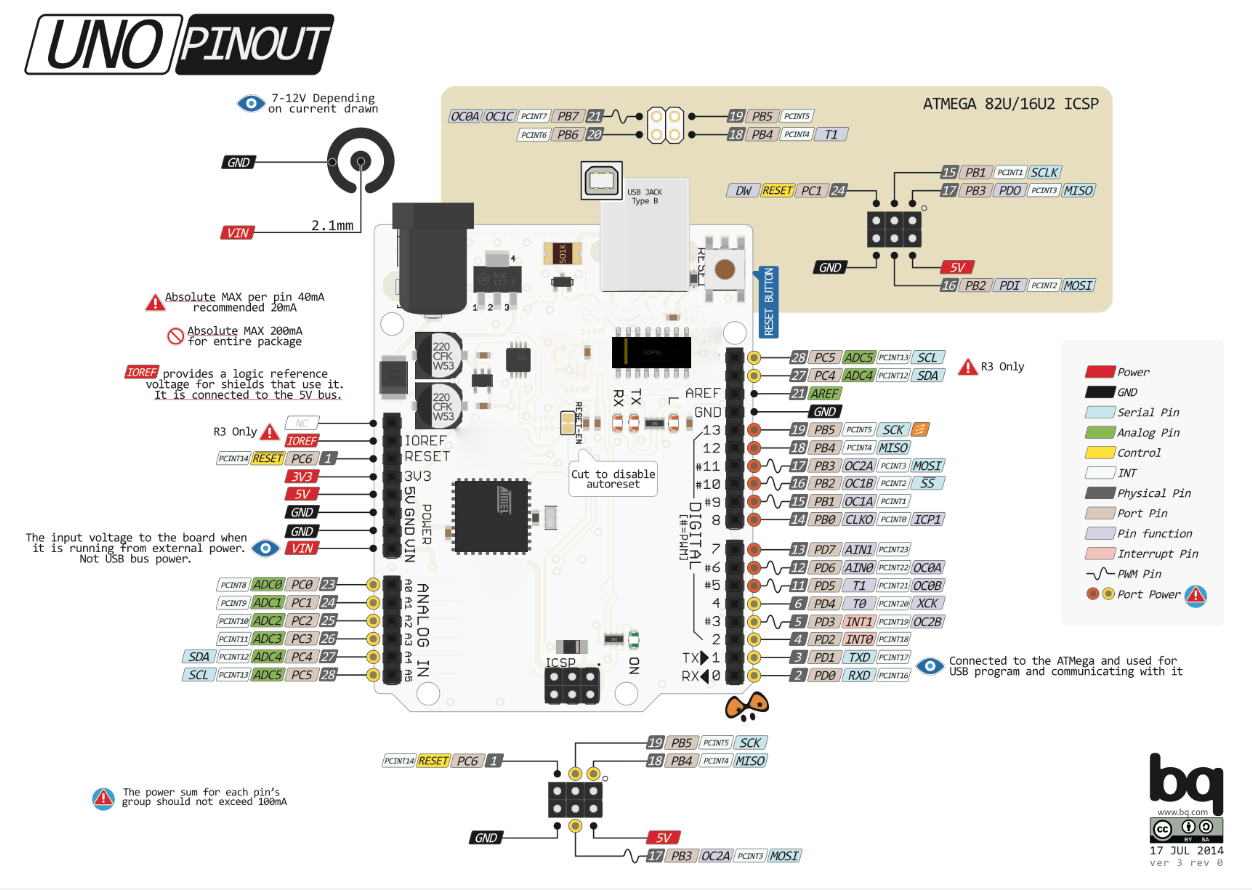


מודול מעקב קו הוא חיישן המזהה קווים כהים על רקע בהיר (או להפך) באמצעות נורות אינפרא-אדום. מטרתו העיקרית היא לאפשר לרובוט או לרכב אוטונומי לעקוב אחר מסלול מוגדר. הוא משמש רבות ברובוטיקה, למשל, בבניית רובוטים עוקבי קו או רכבים אוטונומיים זעירים.





ארדואינו אונו הוא לוח פיתוח אלקטרוני קטן ופתוח (קוד פתוח), שקל מאוד לתכנת אותו. מטרתו העיקרית היא לאפשר לכל אחד לבנות פרויקטים אלקטרוניים אינטראקטיביים. הוא משמש לבניית רובוטים פשוטים, מערכות בית חכם, גאדג'טים ועוד מגוון רחב של פרויקטים יצירתיים.



רובוט תוקף

בגלל שעיקרי הרובוט דומים לא אסביר שוב על החלקים, משום שגם החלקים שהשתנו נשארים עם שימוש דומה.

הדבר העיקרי ששונה ברובוט התוקף זו היא המודולציה שהוספתי של הרובה. עשיתי את זה באמצעות drv8833 מאוד דומה לחלק האדום שנמצא ברובוט הראשון משום שגם מטרתו היא לספק יותר זרם משנכנס אליו למנועים או דברים דומים.

הרובה עצמו נלקח מצעצוע ילדים העובד על שלט רחוק (slide shooter) שאני ניתקתי ממנו את הרובה שהיה לו וחיברתי לדריבר. הדריבר מחובר לפין 11 (שאמור להיות מחובר לservo2) שאין בו שימוש.

משתמש של רובוט1 ורובוט2

מטרתו היחידה היא להעביר הודעות בין הרובוטים לשרת המרכזי

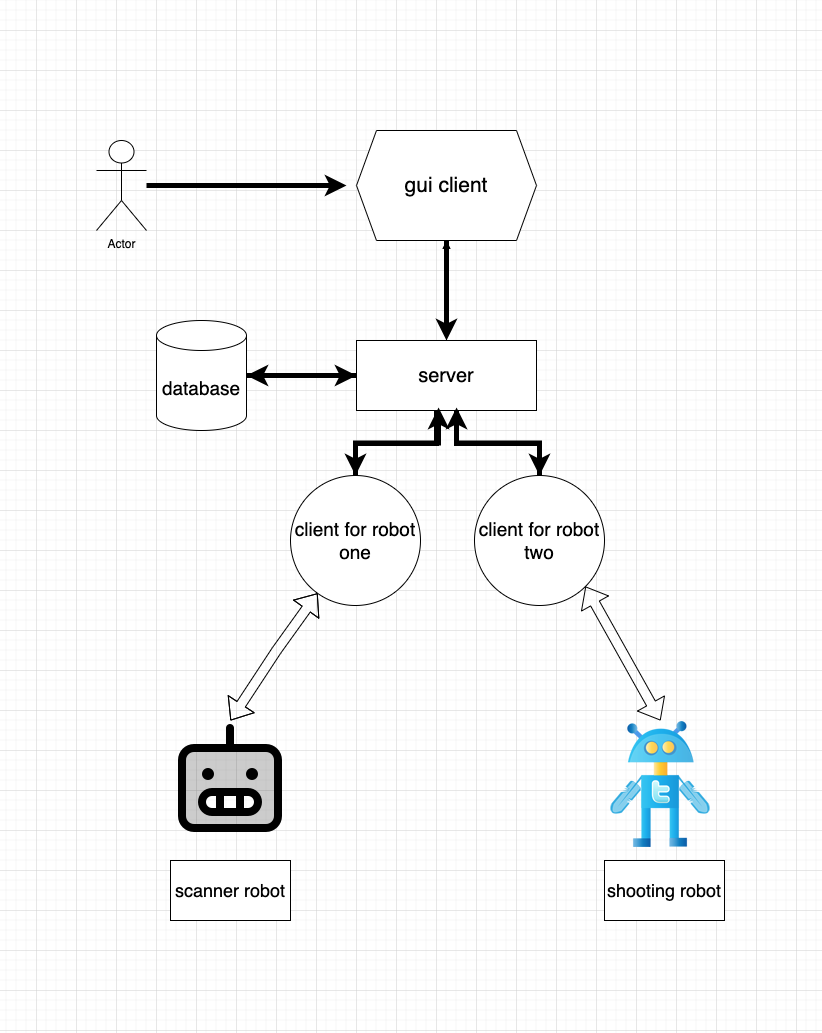
שרת

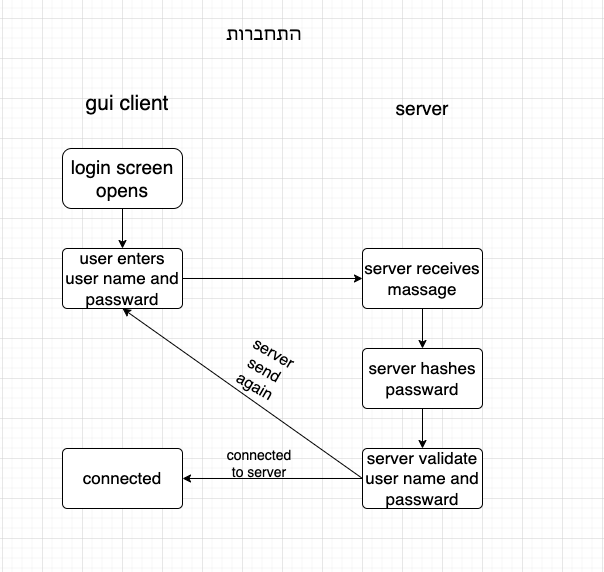
מטרתו לנהל את כל המרכיבים השונים ולתזמן את כולם

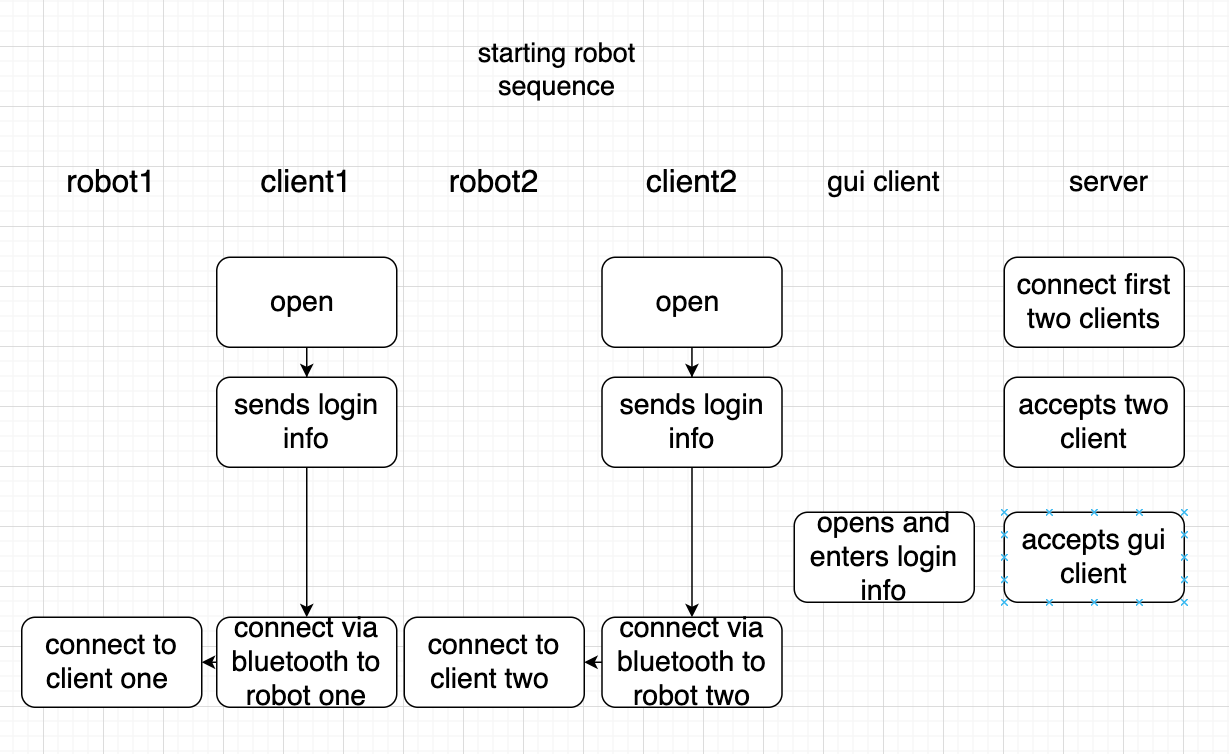
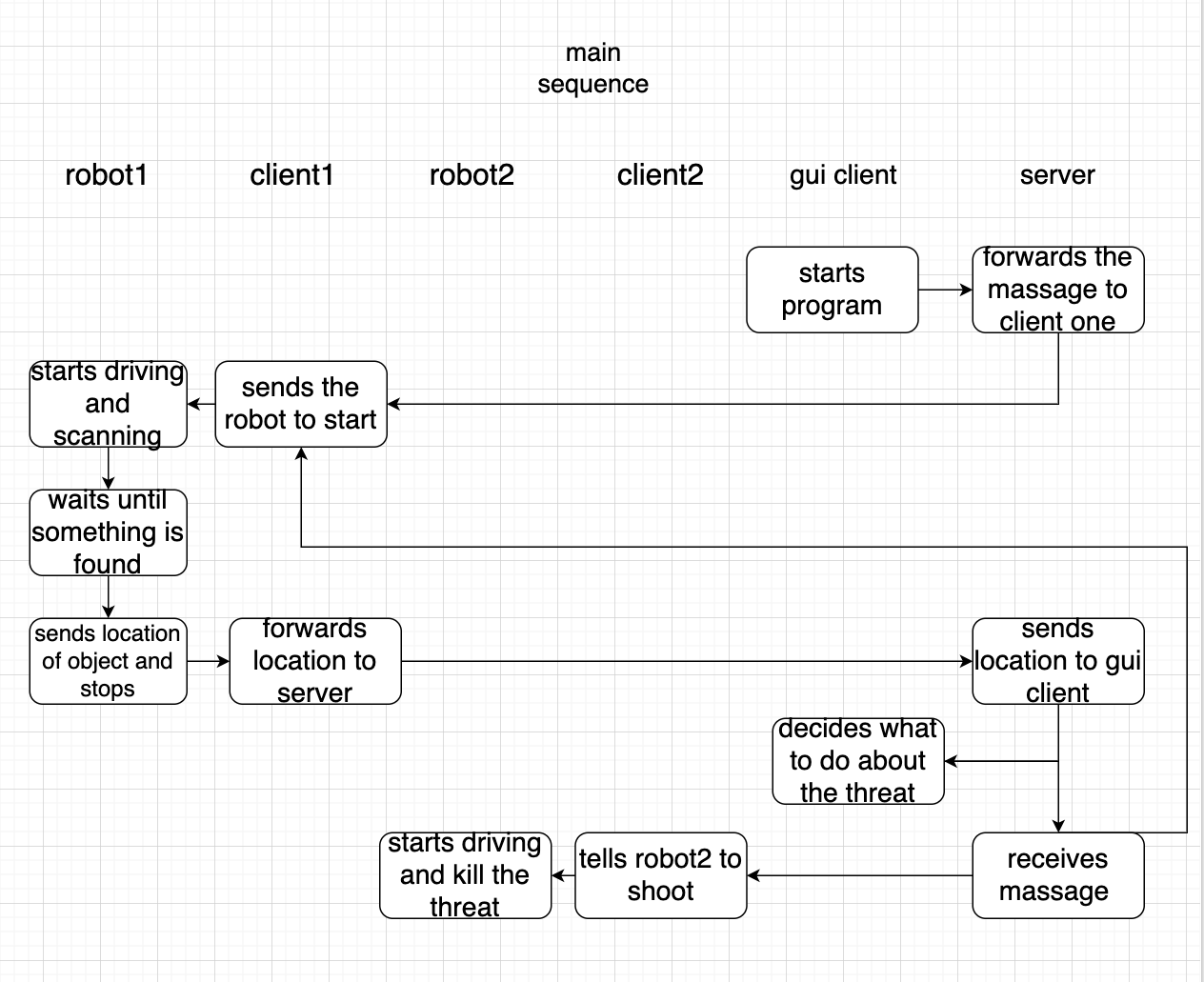
משתמש gui

מאפשר למשתמשים לנווט בפרוייקט בקלות ונוחות

## תיאור זרימת המידע במערכת







## תיאור האלגוריתמים המרכזיים בפרויקט

## מציאת מיקום הרובוט: ניסוח הבעיה ופתרונות אפשריים

האתגר המרכזי שדרש פתרון אלגוריתמי בפרויקט זה נגע ליכולת של הרובוט הנייד לקבוע את מיקומו העצמי לאורך המסלול. **הצורך במיקום מדויק נובע מהאינטראקציה עם רובוט נוסף:** כאשר הרובוט הראשון מזהה חפצים (מטרות), הוא נדרש להעביר את נתוני המיקום לרובוט שני, האחראי על טיפול במטרות אלו (לדוגמה, ירי). ללא ידיעת מיקום הרובוט המזהה, העברת המידע ותיאום הפעולה הופכים בלתי אפשריים.

## בחינת פתרונות קיימים ונימוקי פסילתם

בשלב תכנון הפרויקט, נבחנו מספר פתרונות סטנדרטיים למדידת מיקום (לוקליזציה) של רובוטים, אך אלה נמצאו כלא מתאימים למאפייני הפרויקט ולמגבלותיו:

* **מערכות GPS (Global Positioning System):** בעודן יעילות ביותר למיקום בשטח פתוח, מערכות GPS אינן רלוונטיות לסביבה פנימית ומוגדרת כשלנו, שבה הרובוט נע במסלול מוגדר מראש. הדיוק הנדרש עבור נקודות ספציפיות לאורך המסלול, בשילוב עם העובדה שהרובוט פועל בחלל סגור, הפכו את פתרון ה-GPS ללא ישים.
* **פתרונות לוקליזציה מבוססי חיישנים מתקדמים (לדוגמה: LiDAR, מצלמות עומק, סלאם - SLAM):** קיימים בשוק פתרונות רבים המשתמשים בחיישנים מורכבים ליצירת מפה של הסביבה ולמיקום הרובוט בתוכה (Simultaneous Localization and Mapping). עם זאת, פתרונות אלו נחשבים למסובכים יחסית ליישום, דורשים כוח עיבוד משמעותי, ובמקרים רבים כרוכים בעלויות גבוהות. בהתחשב באופי הפרויקט, שבו הרובוט נע על מסלול ידוע וקבוע, מורכבות יתר זו נראתה מיותרת ולא פרופורציונלית לצורך. המטרה הייתה למצוא פתרון פשוט, אמין וקל ליישום.

## הפתרון הנבחר: זיהוי סמני דרך באמצעות רווחים במסלול

לאור האתגרים והמגבלות של הפתרונות הקיימים, פותח ויושם בפרויקט פתרון ייחודי ויצירתי, המנצל את מאפייני המסלול של הרובוט:

* **מה זה?** שיטה לקביעת מיקום הרובוט על ידי זיהוי רווחים קטנים ומוגדרים מראש במסלול השחור, שבו הרובוט עוקב באמצעות חיישן אינפרא-אדום. כל רווח מייצג נקודת ציון ידועה.
* **מה המטרה שלו?** לאפשר לרובוט לדעת היכן הוא נמצא לאורך המסלול בנקודות קריטיות, מבלי להזדקק למערכות מיקום חיצוניות או מורכבות. המיקום נקבע על ידי ספירת הרווחים שזוהו מאז תחילת המסלול.

האלגוריתם מתבסס על **שינויים מכוונים במסלול התנועה של הרובוט**. הרובוט מתוכנן לנוע לאורך קו שחור באמצעות חיישן אינפרא-אדום העוקב אחר שינויים בצבע. הפתרון המוצע הוא ליצור **רווחים קטנים, ידועים ומתוכננים מראש בקו השחור**. רווחים אלו משמשים כסמני דרך. כאשר הרובוט מגיע לרווח כזה, חיישן האינפרא-אדום יזהה לפתע שאין קו שחור מתחתיו (או שקיים שינוי בצבע הרקע), ובכך יסמן את הגעתו לנקודת ציון.

**היבטים ונימוקים לפתרון זה:**

* **פשטות וקלות יישום:** האלגוריתם דורש מינימום משאבי חומרה ותוכנה. הוא מסתמך על חיישן קיים (חיישן מעקב קו) ועל לוגיקה פשוטה של ספירה וזיהוי שינוי.
* **דיוק בנקודות מוגדרות:** הפתרון אינו מספק מיקום רציף לאורך כל המסלול, אלא מיקום מדויק בנקודות עניין ספציפיות שהוגדרו מראש באמצעות הרווחים. זה עונה בדיוק על הדרישה - הרובוט צריך לדעת את מיקומו רק בנקודות מסוימות שבהן נדרשת קריאה לרובוט היורה.
* **מינימום הפרעה לתנועה:** הרווחים תוכננו להיות קטנים מספיק כך שהם לא יפריעו לרצף תנועת הרובוט. הרובוט ימשיך להתקדם בצורה חלקה גם כאשר הוא חוצה רווח, אך יוכל לזהות אותו לצורך עדכון מיקום.
* **עלות-תועלת:** זהו פתרון חסכוני מאוד בהשוואה למערכות לוקליזציה אחרות, הן מבחינת רכיבים והן מבחינת זמן פיתוח.
* **שלילת פתרונות אחרים:** הפתרון הפשוט והאמין הזה מייתר את הצורך בפתרונות מורכבים ויקרים כמו GPS (לא רלוונטי לסביבה פנימית), או מערכות לוקליזציה מתקדמות (מורכבות יתר על המידה עבור בעיה ספציפית זו).

## איתור איומים (חפצים): ניסוח הבעיה ופתרונות אפשריים

הדרישה המרכזית השנייה בפרויקט הייתה הקמת מערכת יעילה לאיתור חפצים בסביבת הרובוט. **המטרה העיקרית של איתור זה היא זיהוי מטרות הדורשות טיפול על ידי הרובוט השני** (לדוגמה, ירי). הרובוט הראשון נע וסורק את סביבתו, וכאשר הוא מזהה חפץ, עליו לאסוף נתונים רלוונטיים אודותיו לצורך העברה והפעלת הרובוט השני.

## בחינת פתרונות קיימים ונימוקי הבחירה בסונר

קיימות מגוון טכנולוגיות וחיישנים לאיתור חפצים וסריקת סביבה, כל אחת עם יתרונות וחסרונות:

* **מצלמות (רגילות, עומק, אינפרא-אדום):** מצלמות רגילות יכולות לספק מידע חזותי עשיר, אך דורשות אלגוריתמים מורכבים של עיבוד תמונה לזיהוי אובייקטים, העלולים להיות יקרים חישובית ורגישים לתנאי תאורה. מצלמות עומק (כמו RealSense או Kinect) מספקות מידע תלת-ממדי, אך יקרות יחסית. מצלמות אינפרא-אדום יעילות בתנאי חושך או תאורה מוגבלת, אך גם הן דורשות עיבוד תמונה מורכב ועלותן גבוהה יותר מחלופות פשוטות.
* **חיישני LiDAR:** חיישני LiDAR (Light Detection and Ranging) מספקים מפות עומק מדויקות מאוד של הסביבה באמצעות לייזר, והם פתרון מצוין לזיהוי חפצים וניווט. עם זאת, חיישני LiDAR יקרים באופן משמעותי ומורכבים לשילוב בפרויקטים בקנה מידה קטן או בעלי תקציב מוגבל.
* **חיישני סונר (אולטרה-סאונד):** חיישני סונר פועלים על עיקרון של פליטת גלי קול בתדר גבוה וחישוב המרחק על סמך זמן חזרת ההד. הם פשוטים, זולים, קלים לתפעול וצריכת האנרגיה שלהם נמוכה.

**נימוק הבחירה בסונר:** הבחירה בחיישן סונר כאמצעי העיקרי לאיתור חפצים נבעה משילוב של מספר גורמים קריטיים, בהתחשב באופי הפרויקט והמגבלות התקציביות:

1. **מחיר:** חיישני סונר זולים משמעותית בהשוואה לחיישני מצלמה מתקדמים או LiDAR, מה שהתאים לתקציב הפרויקט.
2. **קלות שימוש ויישום:** הסונר מספק נתוני מרחק פשוטים וישירים, מה שמפשט את תהליך איסוף הנתונים והעיבוד הראשוני שלהם, ודורש פחות משאבי מחשוב.
3. **התאמה למטרה:** למרות שסונר פחות מדויק ממצלמות עומק או LiDAR, הוא מספק מספיק מידע כדי לזהות נוכחות של חפץ ולמדוד מרחק בסיסי אליו, וזה עמד בדרישות הליבה של זיהוי המטרות לצורך העברת מידע לרובוט היורה.

## הפתרון הנבחר: זיהוי אובייקטים וקביעת זווית פגיעה אופטימלית באמצעות סונר ומנוע סרבו

האלגוריתם שפותח לאיתור איומים מתבסס על שילוב של חיישן הסונר עם מנוע סרבו, המאפשר סריקה זוויתית של הסביבה. מטרתו העיקרית היא לא רק לזהות נוכחות של חפץ, אלא גם **לקבוע את הזווית האופטימלית (הישרה ביותר) ביחס לחפץ זה**, כדי להבטיח כיוון מדויק של הרובה של הרובוט השני.

* **מה זה?** שימוש בסונר המורכב על מנוע סרבו לביצוע סריקה זוויתית. עבור כל חפץ שזוהה, האלגוריתם מזהה את הזווית שבה המרחק הנמדד לסונר הוא מינימלי, וזהו גם הנתון שיישלח לרובוט השני.
* **מה המטרה שלו?** למצוא את המרחק הקצר ביותר לחפץ ואת הזווית שבה הוא נמדד, כדי לייצג את "מרכז" החפץ מנקודת מבטו של הרובוט הראשון, ובכך לאפשר כיוון מדויק של יחידת הירי.

**תיאור פעולת האלגוריתם:**

1. **סריקה זוויתית:** מנוע הסרבו מניע את חיישן הסונר בטווח זוויות מוגדר (לדוגמה, מ-0 מעלות ל-180 מעלות), בקפיצות זווית קבועות (לדוגמה, כל 5 מעלות).
2. **איסוף נתונים:** בכל זווית, חיישן הסונר מודד את המרחק לעצם הקרוב ביותר. נתון זה נרשם יחד עם הזווית שבה הוא נמדד.
3. **זיהוי חפץ:** האלגוריתם מזהה "חפץ" כאשר קיימת סדרה של מדידות מרחק קצרות ועקביות בטווח זוויות מסוים.
4. **מציאת זווית המינימום:** עבור כל חפץ שזוהה, האלגוריתם מחפש את הזווית הספציפית בטווח המדידות של אותו חפץ שבה נמדד המרחק הקצר ביותר. נקודת המינימום הזו מייצגת את הזווית שבה הסונר "רואה" את החפץ בצורה הישרה ביותר, כלומר, אל מרכזו או הקצה הקרוב ביותר אליו.
5. **שליחת נתונים:** המרחק המינימלי וזווית המינימום (ביחס לרובוט הראשון) נשלחים לרובוט השני כדי שישתמש בהם לכינון הרובה שלו.

**נימוקי הבחירה ויתרונות האלגוריתם:**

* **דיוק בכיוון:** על ידי מציאת נקודת המינימום, האלגוריתם מבטיח שהרובוט השני יכוון אל הנקודה הקרובה ביותר בחפץ, דבר שמגביר את הסיכוי לפגיעה מדויקת.
* **פשטות ומינימליזם:** האלגוריתם פשוט יחסית ליישום מבחינת קוד ודרישות חומרה (סונר וסרבו). הוא אינו דורש עיבוד תמונה מורכב או אלגוריתמים של SLAM.
* **התאמה למטרה:** הפתרון עונה ישירות על הצורך של הרובוט השני במיקום יחסי (זווית ומרחק) של המטרה, מבלי לדרוש מפה מורכבת של הסביבה.
* **התגברות על מגבלות סונר:** בעוד שלסונר יש שדה ראייה רחב יחסית ("קונוס"), תנועת הסרבו ומציאת המינימום מאפשרות לצמצם את אי הוודאות ולקבל אינדיקציה טובה לזווית "הישרה" ביותר.

## שליטה על הרובה: ניסוח הבעיה והפתרון שנבחר

החלק האחרון בפרויקט התמקד בשליטה אפקטיבית על מנגנון הירי (הרובה) של הרובוט השני. **המטרה הייתה להבטיח שהרובה יפעל בצורה מבוקרת, עם אספקת זרם מספקת ומשך ירי מוגבל בזמן**, כדי למנוע בזבוז תחמושת או התחממות יתר, וזאת בהתבסס על פקודת הירי המתקבלת מהרובוט הראשון.

## הפתרון הנבחר: בקרת זרם באמצעות דרייבר ויצירת PWM מוגבל בזמן

הפתרון שנבחר לשליטה על הרובה הוא שילוב של חומרה ותוכנה: שימוש ב**דרייבר זרם** ייעודי בשילוב עם **אות PWM (Pulse Width Modulation) מבוקר-זמן**.

* **מה זה?** יישום תוכנתי של PWM (שינוי רוחב פולס) כדי לשלוט על משך הפעלת מנוע הירי (או מנגנון ירי אחר) ברובה, כאשר ה-PWM מופעל באמצעות דרייבר המגביר את הזרם.
* **מה המטרה שלו?** לספק לרובה את הזרם הנדרש לפעולתו בצורה בטוחה ויעילה, ולוודא שהירי מתבצע למשך זמן קבוע וקצר בלבד, כדי לחסוך באנרגיה ולמנוע נזק.

**תיאור פעולת האלגוריתם:**

1. **הגברת זרם באמצעות דרייבר:** מנגנוני ירי (כמו מנועים או סולנואידים) דורשים בדרך כלל זרם גבוה יותר מזה שבקר המיקרו (Microcontroller) יכול לספק ישירות. לכן, השלב הראשון היה חיבור **דרייבר (Driver)** ייעודי בין בקר המיקרו לרובה. הדרייבר משמש כמגבר זרם, ומאפשר לבקר המיקרו לשלוט במנגנון בעל דרישות זרם גבוהות, תוך הגנה על הבקר עצמו מפני עומס יתר.
2. **יצירת אות PWM:** שליטה על פעולת הרובה אינה דורשת הפעלה רציפה, אלא הפעלה נקודתית ומבוקרת. לצורך כך, האלגוריתם יוצר **אות PWM**. אות זה מייצר פולסים דיגיטליים (הדלקה וכיבוי מהירים) שרוחבם (משך הזמן שהם "דולקים") קובע את "עוצמת" האות הממוצעת. במקרה זה, ה-PWM משמש להפעלה/כיבוי של הזרם לרובה, בתיאום עם קבלת פקודת ירי מהרובוט הראשון.
3. **הגבלת זמן הפעולה (ירי מוגבל):** זהו היבט קריטי של האלגוריתם. במקום לאפשר לרובה לפעול כל עוד אות ה-PWM פעיל, האלגוריתם **מגביל את משך הפעלת ה-PWM (וכתוצאה מכך את משך הירי) לפרק זמן קצר וקבוע מראש**. לדוגמה, כאשר מתקבלת פקודת ירי, ה-PWM מופעל למשך 100 מילישניות בלבד, ולאחר מכן הוא מפסיק באופן אוטומטי.

**נימוקי הבחירה ויתרונות האלגוריתם:**

* **בקרת זרם ואמינות:** הדרייבר מבטיח שהרובה מקבל את הזרם הנדרש לפעולתו היעילה, תוך הגנה על מעגלי הבקרה העדינים של הרובוט.
* **יעילות אנרגטית:** הגבלת זמן הפעולה באמצעות PWM קצר מונעת בזבוז אנרגיה ומאריכה את חיי הסוללה של הרובוט.
* **מניעת נזקים:** הפעלה מוגבלת בזמן מונעת התחממות יתר של מנוע הירי או רכיבים אחרים, ובכך מגנה על רכיבי החומרה.
* **שליטה מדויקת:** האלגוריתם מאפשר "ירייה" יחידה ומבוקרת בכל פעם שמתקבלת פקודה, במקום ירי רציף או בלתי מבוקר.
* **פשטות יישום:** פתרון זה פשוט יחסית ליישום ברמת הקוד של בקר המיקרו, ואינו דורש חומרה מורכבת מעבר לדרייבר בסיסי.

לסיכום, האלגוריתם לשליטה על הרובה מספק פתרון אמין, בטוח ויעיל להפעלת מנגנון הירי, על ידי שילוב נכון של חומרת דרייבר עם בקרת PWM מוגבלת בזמן, ובכך משלים את שרשרת הפעולות של איתור וטיפול במטרות.

## סביבת הפיתוח

**PyCharm היא סביבת פיתוח משולבת (IDE - Integrated Development Environment) מובילה, המתוכננת במיוחד עבור שפת התכנות Python.** בפרויקט זה, PyCharm שימשה כפלטפורמת הפיתוח המרכזית עבור לוגיקת השרת וכלל רכיבי המשתמש, הודות למגוון רחב של תכונות המייעלות את תהליך העבודה.

### סיבות לבחירת PyCharm ויתרונותיה על פני סביבות אחרות:

הבחירה ב-PyCharm נבעה ממספר יתרונות מהותיים שהיא מציעה למפתחי Python, במיוחד בפרויקטים הכוללים גם צד שרת וגם ממשקי משתמש:

1. **נוחות שימוש ופרודוקטיביות גבוהה:**
   * **ממשק אינטואיטיבי:** PyCharm מציעה ממשק משתמש נקי וקל לניווט, המאפשר התחלה מהירה ומיקוד בקוד עצמו.
   * **השלמה אוטומטית חכמה (IntelliSense/Code Completion):** אחד היתרונות הבולטים הוא יכולת ההשלמה האוטומטית המתקדמת שלה, שמזהה קוד, מודולים, פונקציות ופרמטרים באופן חכם, וחוסכת זמן רב בכתיבת קוד ומניעת שגיאות הקלדה.
   * **ניווט מהיר בקוד:** יכולות כמו קפיצה מהירה להגדרת פונקציה/מחלקה, מציאת שימושים ועוד, מקלות מאוד על הבנה ושינוי של קוד קיים, דבר חיוני בפרויקטים מורכבים.
2. **כלי ניפוי באגים (Debugging) עוצמתיים:**
   * PyCharm מציעה דיבאגר מובנה ויעיל במיוחד. ניתן להגדיר נקודות עצירה (Breakpoints), לעקוב אחר ערכי משתנים בזמן ריצה, לעבור על הקוד שורה אחר שורה (Step-by-step execution), ולהעריך ביטויים בזמן אמת. תכונות אלו קריטיות לאיתור ותיקון שגיאות בקוד השרת ובקוד המשתמש בצורה מהירה ויעילה.
3. **שילוב עם מערכות שליטה בגרסאות (VCS):**
   * PyCharm מציעה אינטגרציה חלקה עם מערכות כמו Git, מה שמאפשר לבצע פעולות כמו Commit, Push, Pull, Branching ועוד, ישירות מתוך ה-IDE. זהו יתרון עצום בעבודה על פרויקטים, במיוחד כאשר עובדים בצוות.
4. **מגוון רכב של ספריות**
   * ל-PyCharm קהילה גדולה ותומכת, וזמינים לה מגוון רחב של תוספים המרחיבים את יכולותיה ומאפשרים התאמה אישית לצרכים ספציפיים.

לסיכום לדעתי pycharm מאפשר לעבודה נוחה קלה ויעילה. לכן סביבת עבודה זו הייתה לי הבחירה האוטומטית לפיתוח השרת וכל סוגי הלקוחות.

## סביבת פיתוח: Arduino IDE

**ה-Arduino IDE (Integrated Development Environment) היא סביבת פיתוח קוד פתוח, קלה ואינטואיטיבית, המותאמת במיוחד לתכנות לוחות Arduino ומיקרו-בקרים תואמים.** היא נבחרה לפרויקט זה בזכות פשטותה, קלות השימוש בה, הקהילה הענפה ותמיכתה הרחבה ברכיבים ובספריות קוד.

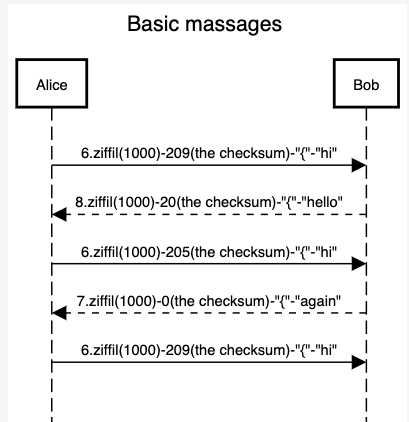
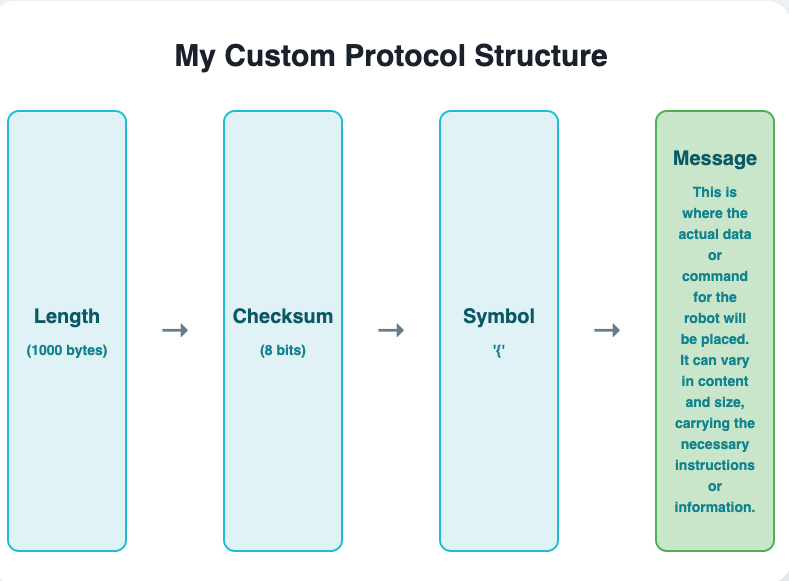
### סיבות לבחירת Arduino IDE ויתרונותיה על פני סביבות אחרות:

הבחירה ב-Arduino IDE נבעה ממספר יתרונות מרכזיים שהפכו אותה לכלי אידיאלי לפיתוח החלקים הפיזיים של הפרויקט, הכוללים אינטראקציה עם חיישנים, מנועים ורכיבים אלקטרוניים:

1. **נוחות שימוש ופשטות:**
   * **ממשק מינימליסטי וידידותי למתחילים:** ה-Arduino IDE ידועה בממשק הפשוט שלה, ללא עומס מיותר של תכונות, מה שהופך אותה לקלה מאוד ללמידה ולשימוש, גם למי שאינו מנוסה בפיתוח משובץ (Embedded Development).
   * **כתיבת קוד קריאה:** השפה המשמשת ב-Arduino (מבוססת על C++) מעוצבת להיות קריאה וברורה, ומפשטת את תהליך הכתיבה והבנת הקוד.
2. **מגוון עצום של קוד ומשאבים (Source Code):**
   * **קהילת קוד פתוח תוססת:** אחד היתרונות הגדולים ביותר של Arduino IDE הוא הקהילה העצומה והפעילה שלה. קיימים אינספור דוגמאות קוד (סקצ'ים), מדריכים ופורומים זמינים ברשת, המאפשרים לפתור בעיות במהירות ולשלב פונקציונליות מורכבת בקלות יחסית.
   * **שימוש חוזר בקוד:** היכולת למצוא ולהשתמש בקוד קיים לפונקציות נפוצות (כמו קריאה מחיישן סונר, שליטה במנוע סרבו, תקשורת סדרתית ועוד) חוסכת זמן פיתוח יקר.
3. **תמיכה רחבה ברכיבים וספריות מובנות:**
   * **מגוון רכיבים נתמכים:** ה-Arduino IDE, יחד עם פלטפורמת Arduino עצמה, תומכת במגוון אדיר של חיישנים, מנועים, מודולי תקשורת ורכיבים אלקטרוניים אחרים הזמינים בשוק.
   * **ספריות מובנות וצד שלישי:** עבור רוב הרכיבים, קיימות ספריות קוד מוכנות מראש. ספריות אלו מספקות פונקציות מופשטות וקלות לשימוש המאפשרות אינטראקציה עם הרכיב ללא צורך בהבנה מעמיקה של פרוטוקולי התקשורת או רישומי החומרה הפנימיים שלו. זה מפשט משמעותית את תהליך חיבור הרכיבים ושילובם בפרויקט.
4. **קלות העלאת קוד (Upload):**
   * תהליך ההידור (Compilation) וההעלאה של הקוד ללוח ה-Arduino הוא פשוט ומתבצע בלחיצת כפתור אחת, מה שמזרז את מחזור הפיתוח והבדיקה.

לעומת סביבות פיתוח מורכבות יותר המיועדות למיקרו-בקרים (כמו MPLAB IDE, Keil uVision), ה-Arduino IDE מציעה חוויה ידידותית ופחות תובענית מבחינת למידה ומשאבים. היא מאפשרת למקד את המאמץ בלוגיקה של הפרויקט ובאינטראקציה עם החומרה, במקום להתמחות בכלי פיתוח מורכבים, ובכך תורמת באופן משמעותי ליעילות ולמהירות הפיתוח.

## תיאור פרוטוקול התקשורת



## סוגי הודעות

**1. Authentication Message**

* **Message Name (Client to Server):** Client Authentication Request
  + **Sent from:** Client (Client 1 or Client 2)
  + **To:** Server
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    String in USERNAME!PASSWORD format, e.g.: "iftach!1234"
* **Message Name (Server to Client):** Server Authentication Response
  + **Sent from:** Server
  + **To:** Client (the client that requested authentication)
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "you are now connected"  
    or  
    "not existing user name ot password"

**2. "again" Message**

* **Message Name:** Resend Request ("again")
  + **Sent from:** Server (to Client 1 or Client 2) *or* Client (to Server)
  + **To:** The other party in the communication
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "again"
  + **Note:** In response to this message, the receiving side will resend the last message it sent (last\_massage).

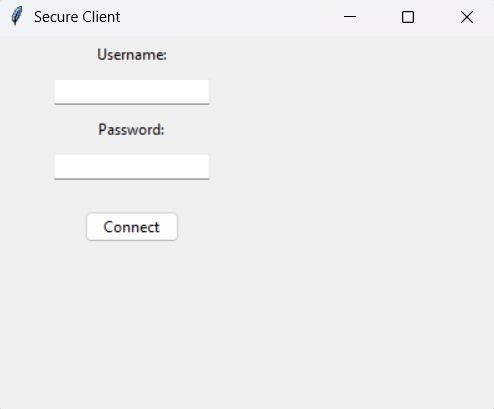
**3. "pic" Message and Image Transfer Related Messages**

* **a. Initial Trigger Message:**
  + **Message Name:** Image Transfer Trigger
  + **Sent from:** Server (to Client 1, after receiving "pic" from Client 2 and relaying it)
  + **To:** Client 1
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "pic"
* **b. Image Data Message (after Client 1 receives the trigger):**
  + **Message Name:** Encrypted Image Data
  + **Sent from:** Server
  + **To:** Client 1
  + **Structure on the network (after LENGTH\_HEADER):**  
    AES\_ENCRYPTED(IMAGE\_RAW\_BYTES)  
    Note: Here, the raw image data is encrypted directly, without the internal checksum and { prefix.
* **c. Image Checksum Message:**
  + **Message Name:** Encrypted Image Checksum
  + **Sent from:** Server
  + **To:** Client 1
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    ORIGINAL\_IMAGE\_CHECKSUM\_STRING (the checksum of the original IMAGE\_RAW\_BYTES).
* **d. Image Acknowledgment / Resend Request Message:**
  + **Message Name:** Image Acknowledgment
  + **Sent from:** Client 1
  + **To:** Server
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "ok"  
    or  
    "again"

**4. Regular Messages (General Purpose Messages)**

* **Example 1: Arduino data from Client 1 to Server**
  + **Message Name:** Client 1 Arduino Data
  + **Sent from:** Client 1
  + **To:** Server
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    String in VAL1@VAL2 format, e.g.: "97@98"
* **Example 2: Relaying part of Arduino data from Server to Client 2**
  + **Message Name:** Server Relays Part 1 of C1 Data to C2
  + **Sent from:** Server
  + **To:** Client 2
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    String VAL1, e.g.: "97"
* **Example 3: "DEFULT" message from Server to Client 1**
  + **Message Name:** Server Default Message
  + **Sent from:** Server
  + **To:** Client 1
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "DEFULT"
* **Example 4: "exit" command from Client to Server**
  + **Message Name:** Client Exit Command
  + **Sent from:** Client
  + **To:** Server
  + **ORIGINAL\_PAYLOAD\_STRING structure (before encryption):**  
    "exit" (or any other exit command defined on the client-side and interpreted accordingly on the server-side).

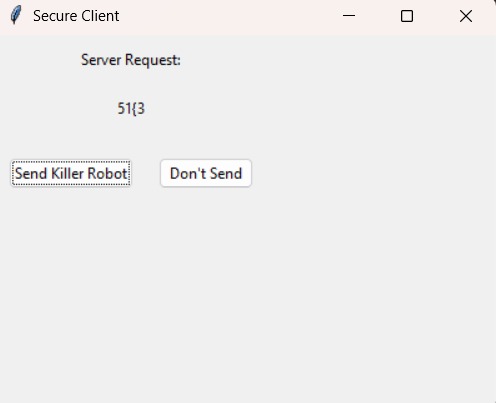
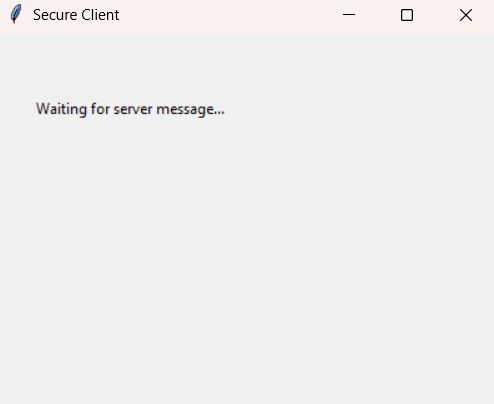
## תיאור מסכים בפרויקט



שם המסך: מסך התחברות

תפקידו: לאפשר למשתמשים קיימים במערכת להתחבר

פעולות שניתן לעשות בו: להכניס טקסט בתור שם משתמש וסיסמא ולהתחבר



שם המסך: מסך המתנה

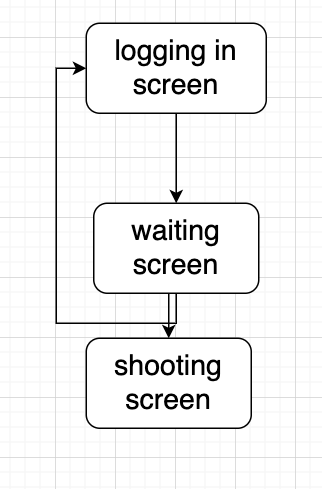
תפקידו: להציג את זמן הסריקה של הרובוט

פעולות שניתן לעשות בו: אין

שם המסך: מסך חיסול

תפקידו: לשלוח את הרובוט המחסל

פעולות שניתן לעשות בו: ללחוץ על הכפתורים ובאמצעותם לשלוח או לא לשלוח את הרובוט



## תיאור מבני הנתונים:

**1. מסד נתונים SQLite לאימות משתמשים**

* **שם המסד (קובץ:** DATABASE\_NAME (מוגדר כ-("server\_users.db")
* **מיקום:** מערכת הקבצים של השרת, באותה תיקייה שבה רץ קובץ השרת (או בנתיב אחר( אם DATABASE\_NAME היה מוגדר אחרת.
* **מטרה:** לאחסן באופן קבוע פרטי משתמשים (שם משתמש וסיסמה מגובבת) לצורך אימות (Authentication).
* **שם הטבלה:** users
  + **פירוט השדות בטבלה users:**
    1. **שם השדה:** id
       - **טיפוס:** INTEGER
       - **מגבלות/מאפיינים** PRIMARY KEY, AUTOINCREMENT :(מפתח ראשי, ערך נוצר אוטומטית ועולה עם כל רשומה חדשה(
       - **דוגמה לערכים אפשריים:** 1, 2, 3, ...
    2. **שם השדה:** username
       - **טיפוס** VARCHAR(50) :מחרוזת באורך מקסימלי של 50 תווים
       - **מגבלות/מאפיינים** UNIQUE, NOT NULL :ערך ייחודי, לא יכול להיות ריק.
       - **דוגמה לערכים אפשריים**"iftach", "gali", "yoav":
    3. **שם השדה:** password\_hash
       - **טיפוס:** VARCHAR(64) :מחרוזת באורך מקסימלי של 64 תווים, מתאים ל-SHA-256 הקסדצימלי
       - **מגבלות/מאפיינים:** NOT NULL :לא יכול להיות ריק
    4. **שם השדה:** created\_at
       - **טיפוס:** TIMESTAMP
       - **מגבלות/מאפיינים:** DEFAULT CURRENT\_TIMESTAMP :ערך ברירת מחדל הוא חותמת הזמן הנוכחית בעת יצירת הרשומה
       - **דוגמה לערכים אפשריים**"2024-10-27 10:30:00" :
* **מפתח ראשי (Primary Key) בטבלהusers :**
  + שדה  usernameהוא גם ייחודי (UNIQUE) ויכול לשמש כמפתח משני או אינדקס לחיפוש מהיר.

**.2 קובץ תמונה (File System)**

* **שם הקובץ (קבוע:** IMAGE\_FILENAME (מוגדר כ-('my\_pic.jpg')
* **מיקום:** מערכת הקבצים של השרת.
* **מטרה:** לאחסן את קובץ התמונה שהשרת יכול לשלוח ללקוחות.
* **מבנה:** קובץ בינארי.
  + **טיפוס:** רצף של בתים (bytes)
* **אין כאן "מפתח ראשי" במובן של מסד נתונים.**

## טיפול בחולשות במערכת

**MITMוהצפנות**

במהלך פיתוח המערכת, הושם דגש משמעותי על הבטחת סודיות ושלמות התקשורת, כמו גם על הגנה מפני גישה לא מורשית וניצול פרצות נפוצות. אחת החולשות המרכזיות שקיבלה התייחסות היא **Man-in-the-Middle (MITM)**. ההתמודדות עם איום זה מתבצעת באמצעות שילוב של הצפנה א-סימטרית (RSA) והצפנה סימטרית (AES). בשלב הראשוני של יצירת הקשר, השרת והקליינט מחליפים מפתח AES סימטרי באופן מאובטח באמצעות מפתחות RSA; השרת שולח את מפתח ה-RSA הציבורי שלו, והקליינט משתמש בו כדי להצפין מפתח AES ייחודי שהוא מייצר ושולח בחזרה לשרת. לאחר מכן, כל התקשורת בין הצדדים, כולל כל ההודעות המועברות, מוצפנת באמצעות מפתח ה-AES הסימטרי הזה. גישה זו מבטיחה שגם אם תוקף יצליח ליירט את התעבורה, הוא לא יוכל לקרוא את תוכן ההודעות המוצפנות ללא מפתח ה-AES, שנשאר סודי בין השרת והקליינט הספציפי.

**כניסה למערכת וhashing**

בנוסף להצפנת התקשורת, המערכת מטפלת באימות משתמשים בצורה מאובטחת. **בדיקת התקינות של שם משתמש וסיסמה** מתבצעת כאשר פרטי ההתחברות נשלחים מהקליינט לשרת רק לאחר שכבר כונן ערוץ תקשורת מוצפן AES. השרת, מצדו, מאמת את הפרטים הללו מול בסיס נתונים שבו הסיסמאות אינן נשמרות בצורתן הגלויה. לשם כך, המערכת משתמשת בטכניקת  Hashing(**גיבוב סיסמאות)**, תוך שימוש באלגוריתם SHA-256. במקום לשמור את הסיסמה המקורית, נשמרת בבסיס הנתונים רק תמצית (hash) חד-כיוונית שלה, מה שמקשה משמעותית על שחזור הסיסמה המקורית גם במקרה של חשיפת בסיס הנתונים.

**Sql injection an checksum**

חולשה קריטית נוספת, **SQL Injection**, מטופלת על ידי שימוש עקבי בשאילתות פרמטריות בעת כל גישה לבסיס הנתונים לצורך אימות. טכניקה זו מפרידה באופן ברור בין קוד ה-SQL לבין הנתונים המסופקים על ידי המשתמש, ובכך מונעת את האפשרות להזרקת פקודות SQL זדוניות שיכולות לסכן את שלמות הנתונים או לחשוף מידע רגיש. לבסוף, כדי להבטיח את שלמות ההודעות המועברות ברשת, הפרוטוקול האפליקטיבי כולל מנגנון **Checksum,** לפני הצפנת כל הודעה ושליחתה, מחושב לה ערך checksum. בצד המקבל, לאחר פענוח חלקי של מבנה ההודעה, ה-checksum מאומת. מנגנון זה, לצד הגנת השלמות שמספקת הצפנת AES במצב EAX, מוסיף שכבת ביטחון נוספת לכך שההודעות לא שונו או נפגמו במהלך העברתן.

# מימוש הפרויקט

## **שימוש בספריות חיצוניות במערכת**

**socket**

ספריית socket מהווה את עמוד השדרה לתקשורת הרשת במערכת. היא מאפשרת יצירת חיבורי TCP/IP אמינים בין השרת המרכזי לבין שלושת הקליינטים. בצד השרת, הספרייה משמשת להקמת נקודת האזנה לבקשות חיבור (server\_socket.bind, server\_socket.listen), לקבלת חיבורים נכנסים מקליינטים (server\_socket.accept), ולשליחה וקבלה של נתונים מוצפנים דרך החיבורים שנוצרו (conn.sendall, conn.recv). בצד הקליינטים, היא מאפשרת יצירת חיבור יוצא אל השרת (my\_socket.connect) וניהול דו-כיווני של התקשורת עמו.

**sqlite3**

לצורך ניהול אימות משתמשים בצד השרת, נעשה שימוש נרחב בספריית .sqlite3ספרייה זו מאפשרת יצירה, גישה ועדכון של בסיס נתונים SQLite מקומי, קל משקל ומוטמע בקובץ. לדוגמה, השרת משתמש בה בפונקציות כמו  init\_database לאתחול מבנה טבלת המשתמשים, add\_user להוספת משתמשים חדשים לבסיס הנתונים (כאשר סיסמאותיהם מגובבות), ו-authenticate\_user כדי לאמת פרטי התחברות שנשלחו מהקליינט מול הנתונים השמורים.

**hashlib**

אבטחת הסיסמאות במערכת, ובפרט בצד השרת, מתבצעת בעזרת פונקציונליות שמספקת ספריית hashlib. ספציפית, השרת משתמש בפונקציית הגיבוב SHA-256 מספרייה זו בתוך הפונקציה hash\_password. תפקידה הוא ליצור תמצית (hash) חד-כיוונית של סיסמת המשתמש לפני שמירתה בבסיס הנתונים. תהליך זה מבטיח שהסיסמאות אינן נשמרות בצורתן הגלויה, ובכך מגן עליהן במקרה של חשיפה לא מורשית של בסיס הנתונים.

**logging**

לתיעוד ומעקב אחר פעולות המערכת, זרימת התוכנית, ושגיאות פוטנציאליות, נעשה שימוש מקיף בספריית logging . הן השרת והן כל אחד מהקליינטים מגדירים לוגרים ייעודיים אשר כותבים הודעות מפורטות לקבצי לוג נפרדים כגון:(tls\_server.log, tls\_client.log, gui\_client.log) דוגמא לכך היא שהשרת רושם אירועים כמו קבלת חיבור חדש מקליינט (logging.info('Connected by: %s', addr)) או שגיאות הקשורות לתקשורת הרשת.

**serial (pyserial)**

התקשורת עם רכיבי החומרה החיצוניים (ארדואינו) בקליינטים 1 ו-2 מתבצעת באמצעות הספרייה  serialהמוכרת גם בשם החבילה .pyserial ספרייה זו מספקת את הכלים לפתיחת פורט תקשורת טורי (Serial Port), שליחת נתונים (המשמשים כפקודות) אל מכשיר הארדואינו, וקבלת נתונים (כמו קריאות סנסורים) ממנו. לדוגמה, קליינט 1 משתמש בה בתוך הפונקציה  connect\_to\_robo\_v3 כדי ליצור את החיבור הטורי הראשוני, ובפונקציה  receive\_massage\_v3כדי לקרוא את הנתונים הנשלחים מהארדואינו.

**time**

ספריית time משמשת במערכת בעיקר להוספת השהיות (delays) קצרות בקוד, במקומות בהם יש צורך להמתין זמן מוגדר. לדוגמה, בפונקציה connect\_to\_robo\_v3 בקליינטים 1 ו-2, נעשה שימוש ב-time.sleep(2). השהיה זו נועדה לאפשר לחיבור הטורי עם הארדואינו להתייצב באופן מלא לאחר פתיחת הפורט ולפני שמתבצעת תקשורת נוספת.

**tkinter  וכל הרכבותיה**

עבור ממשק המשתמש הגרפי (GUI) האינטראקטיבי בקליינט 3, נעשה שימוש נרחב בספריית  tkinter ובהרחבותיה. tkinter עצמה מספקת את הבסיס ליצירת חלונות (self.root = tk.Tk()) ואלמנטים גרפיים. הרחבת  tkinter.ttk משמשת לרכיבים ויזואליים בעלי מראה מודרני יותר, כגון כפתורים (ttk.Button), תוויות (ttk.Label), ושדות קלט .(ttk.Entry) הרחבת tkinter.messagebox מאפשרת הצגת תיבות דו-שיח סטנדרטיות למשתמש, כגון הודעות שגיאה או התראות אינפורמטיביות. כל אלו משולבים יחד במחלקה  ClientGUI ליצירת מסכי ההתחברות, המסך הראשי ומסך ההחלטות.

**threading**

כדי לשמור על היענות (responsiveness) של ממשק המשתמש בקליינט 3, במיוחד בזמן ביצוע פעולות רשת שעשויות להיות ארוכות או חוסמות, נעשה שימוש בספריית  .threadingספרייה זו מאפשרת הרצת משימות מסוימות בתהליכונים (threads) נפרדים, המקבילים לתהליכון הראשי של ה-GUI לדוגמה, תהליך ההתחברות לשרת (\_connect\_to\_server) וקבלת הודעות שוטפות מהשרת (\_receive\_messages) מורצים כל אחד בתהליכון משלו, ובכך מונעים מהממשק הגרפי "לקפוא" ולהפסיק להגיב לפעולות המשתמש.

**queue**

להעברת נתונים והודעות בצורה בטוחה ויעילה בין התהליכונים השונים הפועלים בקליינט 3 (בעיקר בין תהליכון הרשת לתהליכון הראשי של ה-GUI משמשת הספרייה .(queue המחלקה ClientGUI משתמשת באובייקט תור (self.message\_queue) כאמצעי תקשורת. תהליכון הרשת יכול להכניס הודעות (כגון נתונים שהתקבלו מהשרת או הודעות שגיאה) לתור זה, והתהליכון הראשי של ה-GUI, באמצעות הפונקציה  process\_queue, בודק את התור באופן קבוע ומעבד את ההודעות שבו, למשל על ידי עדכון התצוגה למשתמש.

**Crypto (PyCryptodome)**

הליבה הקריפטוגרפית של המערכת, האחראית על הצפנה ופענוח, נשענת באופן מלא על הפונקציונליות העשירה שמספקת הספרייה Crypto  הידועה כ .PyCryptodome-ספרייה זו מספקת את הכלים ליצירת זוגות מפתחות RSA א-סימטריים (ציבורי ופרטי), להצפנה ופענוח של נתונים באמצעות מפתחות אלו (למשל, בפונקציות MyRSA.encrypt\_with\_key להצפנת מפתח ה-AES עם המפתח הציבורי של השרת, ו-MyRSA.decrypt לפענוח). בנוסף, היא משמשת ליצירת מפתחות AES סימטריים אקראיים, ולהצפנה ופענוח של כלל התקשורת האפליקטיבית באמצעות מפתחות AES אלו (למשל, בפונקציותMyAES.encrypt (MyAES.decrypt תוך שימוש במצב הפעלה מאובטח (EAX) המספק גם אימות ושלמות.

**base64**

ספריית  base64 משמשת במערכת לקידוד ופענוח של נתונים בפורמט .Base64 פורמט זה מאפשר ייצוג של נתונים בינאריים באמצעות רצף של תווים הניתנים להדפסה (ASCII). השימוש העיקרי בספרייה זו מופיע בקטעי הדגמה או בדיקה עצמית הנמצאים תחת התנאי  if \_\_name\_\_ == '\_\_main\_\_': בקבצים aes.py ו-rsa.py. קטעים אלו, המיועדים לבדיקה מקומית של פונקציונליות ההצפנה והפענוח, משתמשים ב-Base64 להצגה נוחה של הנתונים המוצפנים. שימוש זה אינו מהווה חלק מרכזי מהלוגיקה של התקשורת הראשית בין השרת לקליינטים.

## תיעוד

**File: server.py**

קובץ זה מהווה את "המוח" המרכזי של המערכת. הוא מאזין לחיבורים משלושת הקליינטים, מנהל את תהליך האימות שלהם מול בסיס נתונים, ומתווך את התקשורת ביניהם. השרת אחראי על החלפת מפתחות הצפנה עם כל קליינט, ולאחר מכן מצפין ומפענח את כל ההודעות באמצעות. AES הוא מקבל נתוני סריקה מקליינט 1 (הרובוט הסורק), מעביר מידע רלוונטי לקליינט 3 (ממשק המשתמש) לצורך קבלת החלטה, ומקבל מקליינט 3 הוראה האם להפעיל את רובוט הירי. אם ההוראה היא "לירות", השרת מעביר את נתוני המטרה הדרושים (שמקורם בקליינט 1) לקליינט 2 (הרובוט היורה) כדי שיוכל לבצע את הירי.

**Module-Level Functions (not within a class):**

* **hash\_password**
  + **Input argument(s):** password (password, can be a number or string)
  + **Output argument(s) / Action:** The function takes a password, converts it to a string, encodes it to bytes, computes its SHA-256 hash, and returns the hash as a hexadecimal string.
* **init\_database**
  + **Input argument(s):** None.
  + **Output argument(s) / Action:** The function connects to the SQLite database (creating it if it doesn't exist), and creates a users table if it doesn't exist, with columns for ID, username, password hash, and creation timestamp. It commits the changes and closes the connection. Does not return a value.
* **add\_user**
  + **Input argument(s):** username (string, the username), password (password, can be a number or string)
  + **Output argument(s) / Action:** The function connects to the database, hashes the password, and attempts to insert a new user record into the users table with the username and password hash. It returns True if the insertion was successful or if the user already existed and no change was made (due to INSERT OR IGNORE). It returns False if an sqlite3.IntegrityError occurred (though less likely with OR IGNORE for duplicate usernames). Closes the connection.
* **populate\_initial\_users**
  + **Input argument(s):** None.
  + **Output argument(s) / Action:** The function iterates over a dictionary of initial users and passwords and calls the add\_user function for each to populate the database. Does not return a value.
* **authenticate\_user**
  + **Input argument(s):** username (string, the username), password (password, can be a number or string)
  + **Output argument(s) / Action:** The function connects to the database, hashes the password, and checks if a user exists with the given username and password hash. It returns True if the user exists and is authenticated, and False otherwise. Closes the connection.
* **send\_image**
  + **Input argument(s):** conn (socket object for the client connection), aes\_protocol ( MyAES object for that client)
  + **Output argument(s) / Action:** The function reads an image file (IMAGE\_FILENAME) from disk, calculates its checksum, encrypts the image data and the checksum (separately) using the provided AES protocol, and sends them to the client via conn. It then waits for an "ok" confirmation from the client, and resends if the confirmation is not valid or if the client-side checksum did not match. Logs success or failure. Does not return a value. Modifies global variables related to resending (though their use in this specific function is not entirely clear).
* **starting\_sql**
  + **Input argument(s):** rsa\_protocol (server's MyRSA object), conn (socket object for the client connection)
  + **Output argument(s) / Action:** The function sends the server's public RSA key to the client. It then receives an AES key from the client (encrypted with the server's public RSA key), decrypts it, and creates a new MyAES object with the received AES key. Returns the new MyAES object.
* **receive\_server\_messages**
  + **Input argument(s):** conn (socket object for the client connection), aes\_protocol ( MyAES object for that client)
  + **Output argument(s) / Action:** The function receives an encrypted message from the client, decrypts it, separates the checksum from the message, and checks the checksum's validity. If the message is "again", it sets the global variable did\_send to True to trigger a resend of the previous message. If the checksum is invalid, it sends "again" to the client. If it's the first communication (IS\_FIRST is True), it tries to parse the message as a username and password, authenticates them against the database, and sends an appropriate response to the client. If authentication is successful three times (num1 == 3), it sets IS\_FIRST to False. If it's not the first communication and the message is valid, it prints the message and logs it. Returns True if a valid non-control message was received (after the initial phase), and False otherwise. Modifies many global variables (send\_again, did\_send, last\_massage, msg, IS\_FIRST, num1).
* **send\_server\_msg**
  + **Input argument(s):** conn (socket object for the client connection), aes\_protocol ( MyAES object for that client)
  + **Output argument(s) / Action:** The function checks if a resend is needed (did\_send). If not, it checks if the current message (msg) is the same as the last sent message (last\_massage), and if so, changes msg to "DEFULT". It then stores the current msg in last\_massage, encrypts and sends msg to the client. If msg is "pic", it calls send\_image. Does not return a value. Modifies global variables (msg, last\_massage, did\_send).
* **main**
  + **Input argument(s):** None.
  + **Output argument(s) / Action:** This is the main function of the server. It initializes the database and populates initial users. It creates a server socket, listens for connections, and accepts three client connections (conn, conn1, conn2) sequentially. For each client, it performs the starting\_sql process (key exchange), receives an initial authentication message, and sends a response. After all clients are connected, it enters a main loop that manages communication: receives a message from client 1, relays part of it to client 3, receives a decision from client 3, and if the decision is "kill", it sends commands to client 2, waits for a response, and then enters an infinite loop (bug). If the decision is not "kill", it sends messages to client 1. The main loop continues until the global variable msg equals "by by". Closes connections and the server socket on completion or in case of an error. Does not return a value. Modifies many global variables.

**File: client1.py**

קובץ זה מנהל את התקשורת עבור הרובוט הראשון, שתפקידו לסרוק את הסביבה באמצעות חיישן סונאר. הקליינט מתחבר לשרת, עובר תהליך אימות ומבצע החלפת מפתחות הצפנה. לאחר מכן, הוא מתקשר עם רכיב הארדואינו המחובר לרובוט כדי לקבל ממנו את קריאות הסונאר. כאשר הרובוט מזהה אובייקט או שינוי בסביבה (על פי נתוני הסונאר), קליינט זה שולח את המידע הרלוונטי (למשל, מרחק או זווית) לשרת בצורה מוצפנת. הוא גם מקבל מהשרת הוראות או משוב (למשל, החלטה שהתקבלה על ידי המפעיל).

**Module-Level Functions (not within a class):**

* **connect\_to\_robo\_v3**
  + **Input argument(s):** None (uses global variables arduino\_port, baud\_rate).
  + **Output argument(s) / Action:** The function creates and returns a serial connection object to the Arduino based on the defined port and baud rate, after a short delay. Sends a byte 'a' to the Arduino.
* **send\_robo\_v3**
  + **Input argument(s):** arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** The function sends the byte 'a' to the Arduino via the serial connection. Does not return a value.
* **receive\_massage\_v3**
  + **Input argument(s):** arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** The function reads two bytes from the Arduino's serial connection. It returns a string composed of the numeric (ASCII) value of the first byte, an '@' character, and the numeric value of the second byte.
* **receive\_image**
  + **Input argument(s):** my\_socket (socket object for the server connection), aes\_protocol ( MyAES object).
  + **Output argument(s) / Action:** The function receives encrypted image data and an encrypted checksum from the server. It checks the checksum, and if invalid, requests a resend. After successful reception, it sends "ok" to the server and saves the image data to a local file (IMAGE\_FILENAME). Logs events. Does not return a value.
* **receive\_client\_messages**
  + **Input argument(s):** conn (socket object for the server connection), aes\_protocol ( MyAES object).
  + **Output argument(s) / Action:** The function receives an encrypted message from the server, decrypts it, and checks the checksum. If the message is "again", it sets the global variable did\_send to True to enable a resend of the previous message. If the checksum is invalid, it sends "again" to the server. If the message is "pic", it calls receive\_image. Otherwise, it prints the message and logs it. Does not return a value. Modifies global variables (send\_again, did\_send, last\_massage, msg).
* **send\_client\_message**
  + **Input argument(s):** conn (socket object for the server connection), aes\_protocol ( MyAES object), arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** If did\_send is True, it resets it. Otherwise, it reads a message from the Arduino using receive\_massage\_v3, stores it as the last sent message (last\_massage and msg), encrypts it, and sends it to the server. Does not return a value. Modifies global variables (msg, last\_massage, did\_send).
* **starting\_sql\_client**
  + **Input argument(s):** my\_socket (socket object for the server connection).
  + **Output argument(s) / Action:** The function receives the server's public RSA key. It creates a new AES key, encrypts it using the server's public RSA key, and sends it to the server. It then sends hardcoded login credentials (iftach!1234) encrypted to the server and receives a response. Returns the created MyAES object.
* **run\_client**
  + **Input argument(s):** None.
  + **Output argument(s) / Action:** The main function for client 1. It connects to the server, performs the starting\_sql\_client process (key exchange and initial authentication), and receives a confirmation message. It then connects to the Arduino. Enters a main loop where it sends a message to the server (read from Arduino), receives a response from the server, and sends an 'a' command to the Arduino again. The loop continues until the message received from the server is "by by". Closes the socket on completion or in case of an error. Does not return a value. Modifies global variables.

**File: client2.py**

קובץ זה שולט ברובוט השני, בעל היכולת לירות חצי נרף. בדומה לקליינט הראשון, הוא מתחבר לשרת, מאמת את זהותו ומחליף מפתחות הצפנה. הקליינט נמצא במצב המתנה עד שהוא מקבל מהשרת פקודת ירי ונתונים ספציפיים על המטרה (שמקורם בסריקה של רובוט 1). עם קבלת הפקודה, קליינט 2 מתקשר עם רכיב הארדואינו שלו כדי להפעיל את מנגנון הירי ולכוון אותו אל המטרה שזוהתה. לאחר ביצוע הירי (או ניסיון ירי), שולח אישור או דיווח סטטוס בחזרה לשרת.

**Module-Level Functions (not within a class):**

* **connect\_to\_robo\_v3**
  + **Input argument(s):** None (uses global variables arduino\_port, baud\_rate).
  + **Output argument(s) / Action:** The function creates and returns a serial connection object to the Arduino, after a short delay.
* **send\_robo\_v3**
  + **Input argument(s):** arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** The function sends the byte 'a' to the Arduino via the serial connection. Does not return a value. (Note: The use of this function is not clear in client 2's main loop logic).
* **receive\_massage\_v3**
  + **Input argument(s):** arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** The function reads a single byte from the Arduino's serial connection and waits until a byte is received. Does not return the read byte. (Note: This function differs from its counterpart in client 1; it only reads and waits, not processing the value).
* **receive\_image**
  + **Input argument(s):** my\_socket (socket object for the server connection), aes\_protocol ( MyAES object).
  + **Output argument(s) / Action:** Same as the receive\_image function in client 1.
* **receive\_client\_messages**
  + **Input argument(s):** conn (socket object for the server connection), aes\_protocol ( MyAES object).
  + **Output argument(s) / Action:** Same as the receive\_client\_messages function in client 1.
* **send\_client\_message**
  + **Input argument(s):** conn (socket object for the server connection), aes\_protocol ( MyAES object), arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** If did\_send is True, it resets it. Otherwise, it reads (and ignores) a message from the Arduino using receive\_massage\_v3 (which essentially just waits for a byte), sets the message to be sent as a hardcoded 'h', stores it as the last sent message (last\_massage and msg), encrypts it, and sends it to the server. Does not return a value. Modifies global variables.
* **starting\_sql\_client**
  + **Input argument(s):** my\_socket (socket object for the server connection).
  + **Output argument(s) / Action:** Same as the starting\_sql\_client function in client 1.
* **send\_stop\_num\_to\_arduino**
  + **Input argument(s):** num (string representing a number), arduino (serial connection object to Arduino).
  + **Output argument(s) / Action:** The function converts the string num to an integer and sends it as a single byte to the Arduino. Does not return a value.
* **run\_client**
  + **Input argument(s):** None.
  + **Output argument(s) / Action:** The main function for client 2. Connects to the server, performs starting\_sql\_client. Connects to the Arduino. Enters a main loop where it: receives a message from the server, sends the received number to the Arduino. Receives another message from the server, sends the received number to the Arduino. Then sends a hardcoded 'h' message to the server. The loop continues until the hardcoded sent message ('h') equals "by by" (which will not happen). Closes the socket on completion or in case of an error. Does not return a value.

**File: client3.py**

קובץ זה מספק את ממשק השליטה האנושי למערכת. הוא מתחבר לשרת, עובר אימות והחלפת מפתחות. לאחר שקליינט 1 (הרובוט הסורק) מדווח לשרת על זיהוי אובייקט, השרת מעביר מידע זה לקליינט 3. המשתמש האנושי רואה את המידע שהתקבל (למשל, נתוני המטרה) דרך הממשק הגרפי ומתבקש לקבל החלטה: האם להורות לרובוט השני לירות ("kill") או לא ("no"). החלטת המשתמש נשלחת בחזרה לשרת, אשר פועל בהתאם.

**Methods in Class: ClientGUI**

* **\_\_init\_\_ (constructor)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** Initializes the main Tkinter window, initializes network components (socket, aes\_protocol, message\_queue), creates all frames by calling internal helper methods (\_create\_frames, \_setup\_login\_screen, etc.), shows the initial login screen, and starts a recurring process (process\_queue) to check the message queue.
* **show\_frame**
  + **Input argument(s):** self, name (string, name of the frame to show).
  + **Output argument(s) / Action:** The function brings the frame whose name is given to the front of the user interface. Does not return a value.
* **\_attempt\_login (internal method, called from button)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function reads the username and password entered in the login screen fields and runs the \_connect\_to\_server method in a separate thread to avoid blocking the GUI. Does not return a value.
* **\_connect\_to\_server (internal method, runs in a thread)**
  + **Input argument(s):** self, username (string), password (string).
  + **Output argument(s) / Action:** The function attempts to connect to the server. Performs RSA and AES key exchange (receives public RSA key, sends encrypted AES key). Sends the login credentials (username and password) encrypted to the server. Receives a response from the server. If connection and authentication are successful ("connected" in response), it stores the AES protocol and puts a message in the queue to show the main screen. If it fails, it puts an error message in the queue. In case of a general network error, it also puts an error message in the queue. Does not return a value directly (updates the queue).
* **\_send\_start (internal method, called from button)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function sends an encrypted "start" message to the server, shows the wait screen (WaitScreen), and runs the \_receive\_messages method in a separate thread. Does not return a value.
* **\_receive\_messages (internal method, runs in a thread)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function enters a loop (which effectively runs once and then exits) to receive an encrypted message from the server. If the message is received successfully, it puts it in the queue for display to the user. If an error occurs, it puts an error message in the queue. Does not return a value directly.
* **\_send\_decision (internal method, called from button)**
  + **Input argument(s):** self, decision (string, "kill" or "no").
  + **Output argument(s) / Action:** The function sends the encrypted decision to the server, shows the wait screen, and re-runs the \_receive\_messages method in a separate thread to wait for the next message from the server. Does not return a value.
* **process\_queue (method, called cyclically by after)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function checks the message queue (self.message\_queue). If there's a message, it retrieves it and performs an action based on the message type: show the main screen, display a message to the user (transition to decision screen), or show an error message and close the connection. Schedules itself to run again using self.root.after. Does not return a value.
* **\_display\_message (internal method)**
  + **Input argument(s):** self, message (string, the message received from the server).
  + **Output argument(s) / Action:** The function shows the decision screen (DecisionScreen), updates the label on the screen with the received message, and schedules via self.root.after to remain on this screen for a defined time (MSG\_DISPLAY\_TIME). (Note: The call to show\_frame with DecisionScreen at the end of the after seems redundant if already on that screen). Does not return a value.
* **run**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function runs the main Tkinter event loop (self.root.mainloop()), which activates the user interface and waits for events. Does not return a value until the window is closed.

**File: protocol.py**

קובץ זה מגדיר את הכללים והפורמט של ההודעות המועברות בין השרת לקליינטים. הוא כולל פונקציות לשליחה וקבלה של הודעות, תוך טיפול באריזת ההודעה (למשל, הוספת אורך ההודעה ו-checksum לבדיקת תקינות), והוא גם משמש כממשק לפעולות ההצפנה והפענוח. כל התקשורת בין הרכיבים משתמשת בפונקציות המוגדרות בקובץ זה כדי להבטיח שההודעות נשלחות, מתקבלות, מוצפנות ומפוענחות בצורה עקבית ותקינה.

**Module-Level Functions (not within a class):**

* **calculate\_checksum**
  + **Input argument(s):** data (bytes, the data to calculate the checksum for).
  + **Output argument(s) / Action:** The function calculates a simple checksum by summing the byte values of the data (with an AND & 0xFF operation to keep the value within a single byte range). Returns the checksum as a string.
* **encrypt\_and\_send\_msg**
  + **Input argument(s):** msg (bytes, the original message to send), comm (socket object), enc\_dec (object with an encrypt method, e.g., MyAES or MyRSA).
  + **Output argument(s) / Action:** The function calculates a checksum for the original message, decodes the message to a string (assuming it was encoded), concatenates the checksum, the '{' character, and the message. Encrypts the combined string using enc\_dec.encrypt, and then calls send\_msg to send the encrypted message. Does not return a value.
* **send\_msg**
  + **Input argument(s):** msg (bytes, the message to send, usually already encrypted), comm (socket object).
  + **Output argument(s) / Action:** The function calculates the length of the message, creates a length string of fixed size (LEN\_LEN) with leading zeros, and prepends it to the message. Sends the total message (length + content) via the socket. Does not return a value.
* **recv\_encrypted\_msg**
  + **Input argument(s):** comm (socket object), dec (object with a decrypt method, e.g., MyAES or MyRSA).
  + **Output argument(s) / Action:** The function calls recv\_msg to receive a message from the socket. If a message was received (not empty), it decrypts it using dec.decrypt. Returns the decrypted message (as bytes).
* **recv\_msg**
  + **Input argument(s):** comm (socket object).
  + **Output argument(s) / Action:** The function first reads the message length string from the socket (of LEN\_LEN size). If the read fails, it sets the length to 0. It then reads the number of bytes specified by the length from the socket. If this read fails, it returns empty bytes. Otherwise, returns the content of the read message (as bytes).
* **is\_checksum**
  + **Input argument(s):** chk (string, the received checksum), data (the data to check, can be a string or bytes).
  + **Output argument(s) / Action:** If data is a string, it encodes it to bytes. The function calculates a checksum for data using calculate\_checksum and compares it to the received chk. Returns True if they are identical, and False otherwise.

**File: aes.py**

קובץ זה מכיל את המחלקה  ,MyAESהאחראית על כל פעולות ההצפנה והפענוח הסימטריות במערכת באמצעות אלגוריתם .AES לאחר החלפת מפתח AES ראשונית, כל ההודעות בין השרת לקליינטים מוצפנות ומפוענחות באמצעות מופעים של מחלקה זו. היא כוללת פונקציות ליצירת מפתח, הצפנת נתונים ופענוח נתונים מוצפנים.

**Methods in Class: MyAES (inherits from CryptoBase - not provided, but assumed to have abstract encrypt/decrypt methods)**

* **\_\_init\_\_ (constructor)**
  + **Input argument(s):** self, key (bytes, optional, AES key. If not provided, a random key of KEY\_SIZE is generated).
  + **Output argument(s) / Action:** Initializes the MyAES object. If a key is provided, it checks that its length is valid (16, 24, or 32 bytes). If no key is provided, it generates a new random key. Stores the key in the instance variable self.key.
* **export\_key**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function returns the internal AES key of the object (self.key) as bytes.
* **encrypt**
  + **Input argument(s):** self, text (bytes, the text to encrypt).
  + **Output argument(s) / Action:** The function encrypts text using the object's AES key (self.key) employing AES EAX mode. This mode generates a Nonce (a random once-used value) and an authentication tag. The function returns the Nonce concatenated with the encrypted text (ciphertext).
* **decrypt**
  + **Input argument(s):** self, cipher\_text (bytes, the encrypted text, including the Nonce at the beginning).
  + **Output argument(s) / Action:** The function separates the Nonce (first 16 bytes) from the cipher\_text. It decrypts the remainder of the cipher\_text using the object's AES key (self.key) and the separated Nonce, employing AES EAX mode. Returns the decrypted text (as bytes). (Note: EAX mode also performs authentication using the tag, but this code doesn't explicitly handle the tag in decryption – the library does this).

**File: rsa.py**

קובץ זה מכיל את המחלקה ,MyRSA המממשת הצפנה א-סימטרית באמצעות אלגוריתם .RSA השימוש העיקרי בה הוא בשלב הראשוני של יצירת הקשר בין השרת לקליינט, לצורך החלפה מאובטחת של מפתח ה-AES הסימטרי. השרת משתמש במפתח RSA כדי לאפשר לקליינט לשלוח אליו את מפתח ה-AES בצורה מוצפנת, מבלי שגורם שלישי יוכל ליירט אותו. המחלקה כוללת פונקציות ליצירת זוג מפתחות ,RSA הצפנה עם מפתח ציבורי ופענוח עם מפתח פרטי.

**Methods in Class: MyRSA (inherits from CryptoBase)**

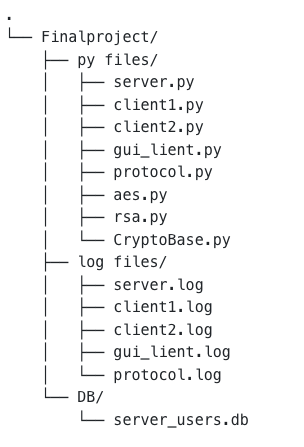
* **\_\_init\_\_ (constructor)**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** Initializes the MyRSA object by generating a new RSA key pair (public and private) of length KEY\_LEN (2048 bits). Stores the key object in the instance variable self.key.
* **encrypt\_with\_key (static method)**
  + **Input argument(s):** message (bytes, the message to encrypt), p\_key (bytes, the public RSA key to encrypt with).
  + **Output argument(s) / Action:** The function imports p\_key as an RSA key object. It encrypts message using this public key employing the PKCS1\_OAEP padding scheme. Returns the encrypted message (as bytes).
* **encrypt**
  + **Input argument(s):** self, message (bytes, the message to encrypt).
  + **Output argument(s) / Action:** The function encrypts message using the public key of the current object (self.key.publickey()) employing the PKCS1\_OAEP padding scheme. Returns the encrypted message (as bytes).
* **decrypt**
  + **Input argument(s):** self, cipher\_text (bytes, the encrypted text).
  + **Output argument(s) / Action:** The function decrypts cipher\_text using the private key of the current object (self.key) employing the PKCS1\_OAEP padding scheme. Returns the decrypted message (as bytes).
* **export\_public\_key**
  + **Input argument(s):** self.
  + **Output argument(s) / Action:** The function exports and returns the public key of the current object (self.key.publickey()) as bytes in a standard format.

## בדיקות

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **כיצד נפתרו – במידה והתגלו בעיות** | |  | | |  | | --- | | **מה היו תוצאות הבדיקה (צפוי/דוגמה)** | | |  | | --- | | **מה בוצע בפועל (פעולות שנבדקו)** | | |  |  | | --- | --- | | **מטרת הבדיקה** |  | |
| עבור קליינט 2: אם הייתה בעיית IP, היה צורך לשנות את HOST\_NAME בקוד של קליינט 2 ל-127.0.0.1 או לכתובת ה-IP הנכונה של השרת ברשת המקומית. | קליינט 1 ו-3 (עם IP מקומי: (התחברות ואימות מוצלחים. הודעת:  "you are now “connected התקבלה.  ולקוח שתים לא הצליח להתחבר. | הרצת השרת ושלושת הקליינטים. כל קליינט ניסה להתחבר לשרת ולשלוח פרטי אימות (שם משתמש וסיסמה קבועים בקוד). נבדק אם השרת מאמת את המשתמשים כראוי מול בסיס הנתונים ומחזיר הודעת "you are now connected" או הודעת שגיאה. | |  | | --- | | **אימות התחברות משתמשים לשרת** | |
| ללא בעיות שנצפו בבדיקה זו | התהליך עבר בהצלחה עבור כל הקליינטים. הודעות ראשוניות (אימות) והודעות נוספות עברו בצורה מוצפנת ופוענחו כראוי. | נבדק תהליך שליחת המפתח הציבורי של RSA מהשרת לכל קליינט, יצירת מפתח AES על ידי הקליינט, הצפנתו עם מפתח ה-RSA הציבורי ושליחתו לשרת. נבדק אם השרת מצליח לפענח את מפתח הAES-ולהשתמש בו לתקשורת מוצפנת. | |  | | --- | | **בדיקת החלפת מפתחות הצפנה** | |
| ללא בעיות שנצפו בבדיקה זו | הודעות עברו בהצלחה בין כל הרכיבים. תוכן ההודעות שהתקבל תאם לתוכן שנשלח (לאחר פענוח). | לאחר אימות והחלפת מפתחות, נשלחו הודעות מהקליינטים לשרת ומהשרת לקליינטים. לדוגמה: קליינט 1 שלח נתוני סונאר, קליינט 3 שלח פקודת "start" והחלטות "kill"/"no". השרת שלח הודעות אישור, נתונים לקליינט 3 ופקודות לקליינט 2. | |  | | --- | | **בדיקת שליחת וקבלת הודעות מוצפנות בין השרת לקליינטים** | |
| ללא בעיות שנצפו בבדיקה זו. המנגנון פעל כצפוי. | הודעות תקינות: התקבלו כראוי.  הודעות עם Checksum שגוי:  הצד המקבל שלח הודעת "again", והצד השולח שלח את ההודעה מחדש (בגלל .(did\_send = True התנהגות תקינה של מנגנון השליחה החוזרת. | נבדק אם הצד המקבל (שרת או קליינט) מזהה הודעה תקינה אם הChecksum-תואם, ומבקש שליחה חוזרת ("again") אם ה- Checksum אינו תואם (ניתן לדמות שיבוש על ידי שינוי הChecksum-ידנית לפני השליחה.( | |  | | --- | | **בדיקת תקינות הודעות באמצעות Checksum** | |
| ללא בעיות שנצפו בבדיקה זו. | התמונה נשלחה מהשרת, התקבלה בקליינט ונשמרה בהצלחה בקובץ received\_image.jpg. תהליך בדיקת ה-Checksum של התמונה פעל כראוי. | נשלחה מהשרת הודעת ) ."pic"ניתן לעשות זאת על ידי שינוי זמני של המשתנה הגלובלי msg בשרת) כדי לגרום לו להפעיל את פונקציית send\_image. נבדק אם הקליינט (למשל, קליינט 1) מקבל את התמונה ושומר אותה כקובץ. | |  | | --- | | **בדיקת תהליך שליחת תמונה מהשרת לקליינט** | |
| ללא בעיות שנצפו בבדיקה זו. | התהליך עבר כמצופה. קליינט 1 קיבל את ההודעה "no" או "DEFULT". קליינט 2 לא קיבל פקודות. | קליינט 1: שלח נתוני סונאר (למשל, "10@20") לשרת.  שרת: קיבל את הנתונים, העביר את החלק הראשון ("10") לקליינט 3. קליינט 3: המשתמש בחר "Don't Send" | **בדיקת לוגיקת העברת נתונים בין קליינט 1, שרת, קליינט 3 וקליינט 2** |
| היה צריך להוסיף תנאי עצירה ללולאה הסופית | עד שלב קבלת 'h' מקליינט 2: התהליך עבר כמצופה. קליינט 2 קיבל את הפקודות. לאחר קבלת 'h' מקליינט 2: השרת נתקע בלולאה אינסופית (while True: pass) והפסיק להגיב לכל הקליינטים. | קליינט 1: שלח נתוני סונאר  . שרת: קיבל את הנתונים, העביר את החלק הראשון לקליינט 3.  קליינט 3: המשתמש בחר "Send Killer Robot" ("kill"). הקליינט שלח "kill" לשרת. | **בדיקת לוגיקת העברת נתונים בין קליינט 1, שרת, קליינט 3 וקליינט 2** |
| הארדואינו תחילה לא הצליח להתחבר ואז לא הצליח ליצור פורט. בעיה זו טופלה באמצעו מחקר מעמיק על שבבי בלוטוס והחלפת השבב המקורי מספר פעמים | קליינט 1: תקשורת תקינה, נתונים התקבלו מהארדואינו.  קליינט 2: תקשורת תקינה, פקודות נשלחו לארדואינו, ואישור התקבל. | נבדק אם קליינט 1 מצליח לשלוח 'a' ולקבל שני בתים מהארדואינו שלו. נבדק אם קליינט 2 מצליח לשלוח את המספרים שקיבל מהשרת לארדואינו שלו ולקבל ממנו אישור | **בדיקת תקשורת עם ארדואינו** |
| ללא בעיות שנצפו בבדיקה זו | הממשק פעל כמצופה. מעבר בין מסכים תקין. כפתורים הגיבו. הודעות מהשרת הוצגו. החלטות נשלחו. טיימר פעל. | נבדקו כל המסכים: התחברות, מסך ראשי, מסך המתנה, מסך החלטה. נבדקה תגובתיות הכפתורים, הצגת הודעות מהשרת, ושליחת החלטות לשרת. נבדק טיימר הצגת ההודעה במסך ההחלטה. | **בדיקת ממשק משתמש** |
| ללא בעיות בבדיקה זו | סגירת קליינט: השרת זיהה את ניתוק הקשר וטיפל בו | נוסו תרחישי שגיאה כמו סגירת קליינט באמצע תקשורת, או ניסיון להתחבר לשרת כשהוא אינו פועל. נבדק אם המערכת מתמודדת עם שגיאות אלו בחן יחסי (למשל, סגירת חיבורים בצורה נקייה, | **בדיקת טיפול בשגיאות (כללי)** |
| צצו מגוון רכב של בעיות. תחילה הפניות היו חדות מדי. ואז ניגוד הצבעים עם הרצפה היה לא מספיק חד. ולבסוף הדבר שעשיתי ממנו את המסלול לא היה בצבע הנכון. לכן לבסוף תיקנתי כל בעיה באמצעות שינוי והתאמת המסלול | הרובוט נוסע בצורה רציפה ועוצר רק בנקודות הנכונות | בניתי מסלול לרובוט הכולל עצירות פניות וסיבובים לצורך בדיקת כל האספקטים של הפיצ׳ר | **מעקב אחרי קווים** |
| הרובה לא ירה, תחילה בשל מחסור בהתנגדות, ולכן הוספנו נגד. ולאחר מכן בשל חיבור חלקי (באמצעות כבל זכר ונקבה) לכן הלחמתי את הכבלים. | הרובוט יורה את הכדורים | הפעלת הפונקציה של הpwm לחמש שניות כדי לירות שתי כדורים | **יריה** |

# מדריך למשתמש

## עץ קבצים



**שלב 1: הכנת סביבת הפיתוח והתקנת תלויות**

**הכנות מקדימות והתקנת PyCharm   
\* ראשית, יש להוריד ולהתקין את סביבת הפיתוח המשולבת (IDE) PyCharm**

**התקנת ספריות Python נדרשות:**

* + הפרויקט עושה שימוש במספר ספריות Python חיצוניות כגון PySerial, PyCryptodome וכו'
  + בעת הרצת הקוד בפעם הראשונה, ייתכן ש-Python, ייתכן שתקבל הודעות שגיאה המצביעות על כך שספרייה מסוימת אינה קיימת.
  + במקרה כזה, יש להתקין את הספריות החסרות הספרייה החסרה. לרוב, ניתן לעשות זאת דרך מנהל החבילות של PyCharm

**שלב 2: הכנת החומרה, רובוטים וחיבורים**

1. **בדיקת הרכבת הרובוטים:**
   * ודא כי שני הרובוטים הפיזיים – הרובוט הסורק (המצויד בחיישן סונאר) והרובוט היורה (בעל מנגנון ירי חצי נרף) – מורכבים כהלכה. יש לוודא שכל הרכיבים, כולל מנועים, חיישנים, מנגנוני ירי, בקרי ארדואינו ומודולי בלוטות', מחוברים באופן תקין ובטוח.
2. **הפעלת הרובוטים:**
   * חבר כל אחד מהרובוטים למקור מתח מתאים (סוללה) והפעל אותם.
3. **שיוך מודולי בלוטות' וזיהוי מספרי פורטים:**
   * **דרישה קריטית:** כל רובוט דורש חיבור למחשב נפרד. כלומר, קובץ client1.py  המנהל את הרובוט הסורק ירוץ על מחשב אחד, וקובץ client2.py  המנהל את הרובוט היורה ירוץ על מחשב אחר.
   * בכל אחד מהמחשבים הייעודיים לרובוטים:
     + בצע תהליך "צימוד" (Pairing) של מודול הבלוטות' של הרובוט המתאים לאותו מחשב. יש לעקוב אחר הוראות מערכת ההפעלה לצימוד התקני בלוטות'.
     + לאחר השלמת הצימוד, יש לזהות את מספר הפורט הטורי (Serial Port) שאליו שויך חיבור הבלוטות' של הרובוט. ב-Windows, ניתן למצוא מידע זה ב"מנהל ההתקנים" (Device Manager), תחת "יציאות COM ו-LPT)".
     + **רשום בצד את מספר הפורט המדויק עבור כל רובוט במחשב המתאים לו.** מידע זה חיוני לשלב הבא.

**שלב 3: הגדרת תצורת הקוד**

**עדכון פרטי חיבור ב-  client1**

* + פתח את קובץ הקוד client1.py באמצעות סביבת הפיתוח PyCharm או עורך טקסט.
  + אתר את המשתנה הגלובלי  arduino\_port בתחילת הקובץ. שנה את ערכו למחרוזת המייצגת את מספר הפורט הטורי ששייכת לרובוט הסורק במחשב שעליו ירוץ קליינט זה (כפי שזיהית בשלב 2.3). לדוגמה arduino\_port = 'COM3':
  + אתר את המשתנה הגלובלי  HOST\_NAMEושנה את ערכו לכתובת ה-IP של המחשב שעליו ירוץ השרת (server.py). אם השרת והקליינט ירוצו על אותו מחשב (לצורכי בדיקה), ניתן להשתמש בכתובת '127.0.0.1'.
  + שמור את השינויים בקובץ.

1. **עדכון פרטי חיבור ב- client2**
   * פתח את קובץ הקוד client2.py.
   * אתר את המשתנה הגלובלי  arduino\_port ושנה את ערכו למחרוזת המייצגת את מספר הפורט הטורי ששייכת לרובוט היורה במחשב שעליו ירוץ קליינט זה. לדוגמה arduino\_port = 'COM4':
   * אתר את המשתנה הגלובלי  HOST\_NAME ושנה את ערכו לכתובת ה-IP של המחשב שעליו ירוץ השרת (זהה לכתובת שהוגדרה בקליינט 1.(
   * שמור את השינויים בקובץ.
2. **עדכון פרטי חיבור ב- client3**
   * פתח את קובץ הקוד client3.py.
   * אתר את המשתנה הגלובלי HOST\_NAME ושנה את ערכו לכתובת ה-IP של המחשב שעליו ירוץ השרת (זהה לכתובות שהוגדרו בקליינטים 1 ו-2.(
   * שמור את השינויים בקובץ.

**שלב 4: הרצת רכיבי המערכת (סדר קריטי(**

סדר ההרצה של רכיבי המערכת הינו קריטי לתפקודה התקין. יש להקפיד על הסדר המפורט להלן:

1. **הרצת השרת (server.py):**
   * במחשב המיועד לשרת, הרץ את קובץ server.py.
   * ודא שאין הודעות שגיאה, והמתן עד שתופיע בטרמינל או בחלון הפלט הודעה המציינת שהשרת החל להאזין לחיבורים
2. **הרצת client1.py**
   * במחשב המחובר לרובוט הסורק, לאחר שהשרת רץ ומאזין, הרץ את קובץ client1.py.
   * הקליינט ינסה להתחבר לשרת, לבצע החלפת מפתחות ולאמת את עצמו. עקוב אחר הפלט כדי לוודא שההתחברות והאימות הראשוני הצליחו.
3. **הרצת client2.py**
   * במחשב המחובר לרובוט היורה, לאחר שקליינט 1 התחבר בהצלחה לשרת, הרץ את קובץ client2.py.
   * גם קליינט זה ינסה להתחבר לשרת, לבצע החלפת מפתחות ולאמת את עצמו. עקוב אחר הפלט.
4. **הרצת client3.py (gui)**
   * לבסוף, לאחר ששני הקליינטים של הרובוטים (קליינט 1 וקליינט 2) התחברו בהצלחה לשרת, הרץ את קובץ  client3.py. קובץ זה יכול לרוץ על כל אחד מהמחשבים או על מחשב נוסף ברשת.
   * עם הרצתו, ייפתח חלון ממשק המשתמש הגרפי (GUI).

**שלב 5: התחברות ותפעול המערכת דרך ממשק המשתמש**

1. **התחברות למערכת:**
   * בחלון ממשק המשתמש (קליינט 3) שהופיע, תראה מסך התחברות.
   * בשדה "Username" הקלדiftach :
   * בשדה "Password” הקלד1234 :
   * לחץ על כפתור "Connect”
2. **תפעול המערכת:**
   * לאחר התחברות מוצלחת, יוצג המסך הראשי של ממשק המשתמש.
   * כעת, המערכת מוכנה לפעולה. יש לעקוב אחר ההנחיות והאפשרויות שיופיעו בממשק הגרפי. בדרך כלל, השלב הראשון יהיה להפעיל את תהליך הסריקה של רובוט 1 (על ידי לחיצה על כפתור "START” )
   * כאשר הרובוט הסורק (קליינט 1) ישלח נתונים על זיהוי אובייקט לשרת, השרת יעביר את המידע הרלוונטי לממשק המשתמש
   * בממשק המשתמש יוצגו לך נתוני המטרה, ותתבקש לקבל החלטה האם להורות לרובוט השני (קליינט 2) לירות על המטרה ("Send Killer Robot") או לא לירות
   * המשך לעקוב אחר ההוראות והאפשרויות שיוצגו בממשק להמשך תפעול המערכת.

# רפלקציה

כבר בהליך בחירת הפרוייקט, כאשר החלטתי שאני עושה את הפרוייקט אני הבנתי שהפרויקט שלי שונה בנוף. הפרוייקט בניגוד למרבית הפרויקטים מתעסק בעיקרו בחומרה ולא בסייבר. לכן מרבית הפרויקט היה פתירת בעיות חומרה ולמידה של נושאים חדשים בתחום הזה. בנוסף כמישהו שמעולם לא למד חומרה הבעיות שצצו במהלך הפרויקט היו שונות ומאתגרות בשבילי.

במכשול הראשון אני נתקלתי עוד לפני תחילת כתיבת הקוד, כאשר ניסיתי לחבר את הרובוט הראשון שלי למחשב. הרובוט, שמגיע בערכה שבה נמצא הבסיס וניתן להרכיב ולקנות חלקים מותאמים לצרכים שלך, הגיע עם שבב בלוטוס. כאשר ניסיתי לחבר את השבב ראיתי שהוא לא פותח port במחשב. תחילה ניסיתי לאתחל את המחשב ולראות עם זה עוזר. אחר כך ניסיתי להחליף מחשב (מmac לwindows) וכלום לא עזר. כאשר התחלתי לקרוא פורומים על הצ׳יפים האלה ראיתי שלהרבה מן האנשים קיימת אותה הבעיה. בסוף ניסינו, ע״פ המלצות בפורומים, להחליף שבבים עד שבסוף אחד עבד. כבר מן הבעיה הזאת אני התחלתי להבין למה אני נכנס. עוד לפני תחילת הפרוייקט נתקלתי בבעיה (שבכלל לא קשורה לתכנות) שלקחה לי בערך חודש וחצי לפתור.

המכשול השני שנתקלתי בו היה מכשול לוגי יותר. אני הייתי צריך למצוא דרך שבאמצעותה אני יכול לדעת מה המיקום של הרובוט הראשון כדי שאוכל לשלוח אליו את הרובוט השני. כמובן שהמחשבה הלוגית הראשונה שלי היה gps (כמו בטלפון) אך כאשר חקרתי על זה לעומק הבנתי שזה פתרון לא מעשי וכדי לממש אותו זה אהיה פרוייקט עצמאי. לכן הרעיון השני שלי היה סממנים על המסלול. חשבתי אולי ליצר סימן של + מטייפ שחור על המסלול ולראות מתי כל הקולטנים רואים את הצבע שחור, אך הפתרון עבד רק חלק מהזמן בשל קריאות לא מדויקות. לבסוף הגעתי לפתרון של רווחים קטנים במסלול. כך הרובוט הצליח לקרוא בזמן את המיקום שלו ועד שהוא קלט את העצירה הוא כבר חזר למסלול (בגלל המומנטום). אני מאוד גאה בפתרון הזה בשל היצירתיות שנדרשה ממני כדי להגיע אליו ובשל האופטימיזציה שלו עד למצב שהוא עובד בצורה קונסיסטית.

בעיה קטנה יותר שהיתה לי היא שהמצלמה לא עבדה. כאשר ניסיתי לחבר אותה היא פשוט לא נדלקה. ניסיתי למצוא את הבעיה אך לא מצאתי אותה. דבר זה יצר לי דילמה, אם להזמין מצלמה חדשה (לא בטוח שתספיק להגיע) או לשנות כיוון עם הרובוט השני. לכן החלטתי להפוך את הרובוט השני לתוקף ולהוסיף לו רובה, מה שמוביל אותנו לבעיה האחרונה והגדולה מכל.

חיבור הרובה בשבילי היה החלק הכי קשה ומספק של העבודה. תחילה הייתי צריך לקנות צעצוע על שלט רחוק עם רובה שאני יכול לפרק ולחבר לרובוט. כאשר קיבלתי את הצעצוע וניתרתי את הרובה (שלא מותאם לארדואינו) המחשבה הראשונה שלי הייתה פשוט לחבר את הפלוס לפלוס ואת המינוס למינוס ולראות אם זה עובד. אבל אחרי שיחה קצרה עם אבא שלי (שקצת מבין בתחום) הבנתי שאני לא בכיוון ומהלך כזה יכול לשרוף את הארדואינו בשל דרישת זרם גבוהה מדי של הרובה. לכן למדתי קצת חשמל (חוק אום, נגדים, זרם חילופין, זרם ישר, מגנטיות וכו...), וקניתים מד מתח וספק. תחילה מדדתי כמה V וכמה AMP הרובה צורך ובנוסף כמה V וכמה AMP הארדואינו מוציא. לאחר בדיקה הבנתי שאני צריך 3V ו1AMP בשביל הרובה כאשר הארדואינו מוציא 5V ו0.5AMP . כדי להוציא יותר זרם מהארדואינו חקרתי והבנתי שאני צריך לקנות driver/transistor לארדואינו, וליצור פונקצית pwm כדי להתאים את המטח. כאשר הגיע הטרנזיסטור למדתי להלחים, והלחמתי את הטרנזיסטור לארדואינו וניסיתי לחבר את הרובה. כאשר הרובה חובר והפעלתי את הpwm הרובה הוציא זמזום ולא זז. לכן ניסינו להוסיף את מד הזרם למעגל כדי לראות שהכל עובד, ופתאום הרובה התחיל לירות. לאחר מחקר קצר הבנתי שהבעיה נבאה ממחסור בהתנגדות. וכאשר הוספתי נגד ולאחר זמן רב, הרובה ירה.

אני חושב שכל הבעיות האלה מסכמות בדרך הטובה ביותר את הפרוייקט שלי ואת התהליך שעברתי. את השוני בין עבודה בסביבה סגורה לבין עבודה על חומרה. ואני באמת חושב ומאמין שהפרוייקט הזה זה אחד הבחירות הטובות שעשיתי. הוא היה מעניין בטירוף, מאוד רב גוני ומגוון, מאתגר ושונה. ואני מרגיש שכל שעה שהשקעתי בפרוייקט היתה שווה את זה.

# ביבליוגרפיה

Arduino forums

Perplexity

Google ai studio

# נספחים

## server

*"""  
Author: iftach kasorla  
Date: 3/6/25  
Description: a server that uses encryption/decryption and SQLite user authentication  
"""*import socket  
import sqlite3  
import hashlib  
from rsa import MyRSA # Assuming rsa.py and MyRSA class exist  
from aes import MyAES # Assuming aes.py and MyAES class exist  
import protocol # Assuming protocol.py exists  
import logging  
  
IP\_ADDR = '0.0.0.0'  
PORT = 8443  
QUEUE\_LEN = 1  
# MSG = 'have a nice day' # This global variable seems unused in the main logic  
EXIT\_CMD = 'exit' # This global variable seems unused in the main logic  
EXIT\_RES = 'by by'  
# USER\_INPUT = 'please enter a command: ' # This global variable seems unused  
send\_again = False  
did\_send = False  
last\_massage = '' # Stores the last message sent to a client, used to avoid re-sending identical data or sending "DEFULT"  
msg = '' # Holds the current message to be processed or sent  
IMAGE\_FILENAME = 'my\_pic.jpg'  
IS\_FIRST = True # Flag to indicate if it's the first set of interactions for authentication attempts  
num1 = 0 # Counter for successful authentication attempts to eventually set IS\_FIRST to False  
  
DATABASE\_NAME = "server\_users.db"  
  
# Configure logging at the beginning  
logging.basicConfig(  
 format='%(asctime)s | %(levelname)s | SERVER - %(message)s',  
 datefmt='%m/%d/%Y %I:%M:%S %p',  
 level=logging.INFO,  
 filename='tls\_server.log',  
 filemode='w'  
)  
  
# ------------------- DATABASE FUNCTIONS -------------------  
  
def hash\_password(password: str) -> str:  
 *"""  
 Hash a password using SHA-256.  
  
 Args:  
 password (str): The password to hash.  
  
 Returns:  
 str: The hexadecimal representation of the hashed password.  
 """* return hashlib.sha256(str(password).encode()).hexdigest()  
  
def init\_database() -> None:  
 *"""  
 Initialize the SQLite database and create the 'users' table if it doesn't exist.  
 Logs success or errors during database initialization.  
 """* conn = None  
 try:  
 conn = sqlite3.connect(DATABASE\_NAME)  
 cursor = conn.cursor()  
 cursor.execute('''  
 CREATE TABLE IF NOT EXISTS users (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 password\_hash VARCHAR(64) NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
 )  
 ''')  
 conn.commit()  
 logging.info(f"Database '{DATABASE\_NAME}' initialized successfully.")  
 except sqlite3.Error as e:  
 logging.error(f"Error initializing database '{DATABASE\_NAME}': {e}")  
 finally:  
 if conn:  
 conn.close()  
  
def add\_user(username: str, password: str) -> bool:  
 *"""  
 Add a new user to the database with a hashed password.  
 Uses 'INSERT OR IGNORE' to avoid errors if the username already exists.  
  
 Args:  
 username (str): The username for the new user.  
 password (str): The password for the new user.  
  
 Returns:  
 bool: True if the user was added or already existed (due to IGNORE),  
 False on other SQLite errors (though less likely with IGNORE for primary conflicts).  
 """* conn = None  
 try:  
 conn = sqlite3.connect(DATABASE\_NAME)  
 cursor = conn.cursor()  
 password\_hash = hash\_password(password)  
 cursor.execute(  
 "INSERT OR IGNORE INTO users (username, password\_hash) VALUES (?, ?)",  
 (username, password\_hash)  
 )  
 conn.commit()  
 logging.info(f"User '{username}' processed for addition (added or already existed).")  
 return True  
 except sqlite3.Error as e: # Catches general SQLite errors beyond IntegrityError handled by IGNORE  
 logging.error(f"Error adding user '{username}': {e}")  
 return False  
 finally:  
 if conn:  
 conn.close()  
  
def populate\_initial\_users() -> None:  
 *"""  
 Populate the database with a predefined set of initial users.  
 Calls add\_user for each user in the initial\_users dictionary.  
 """* initial\_users = {"iftach": 1234, "gali": 1212, "yoav": 5647}  
 logging.info("Populating initial users...")  
 for username, password in initial\_users.items():  
 if add\_user(username, str(password)): # Ensure password is treated as string for hashing  
 logging.debug(f"Initial user '{username}' processed.")  
 else:  
 logging.warning(f"Failed to process initial user '{username}'.")  
 logging.info("Finished populating initial users.")  
  
def authenticate\_user(username: str, password: str) -> bool:  
 *"""  
 Authenticate a user against the SQLite database using their username and password.  
  
 Args:  
 username (str): The username to authenticate.  
 password (str): The password to authenticate.  
  
 Returns:  
 bool: True if authentication is successful, False otherwise.  
 """* conn = None  
 try:  
 conn = sqlite3.connect(DATABASE\_NAME)  
 cursor = conn.cursor()  
 password\_hash = hash\_password(password)  
 cursor.execute(  
 "SELECT COUNT(\*) FROM users WHERE username = ? AND password\_hash = ?",  
 (username, password\_hash)  
 )  
 result = cursor.fetchone()  
 if result and result[0] > 0:  
 logging.info(f"User '{username}' authenticated successfully.")  
 return True  
 else:  
 logging.warning(f"Authentication failed for user '{username}'.")  
 return False  
 except sqlite3.Error as e:  
 logging.error(f"Database error during authentication for user '{username}': {e}")  
 return False  
 finally:  
 if conn:  
 conn.close()  
  
# ------------------- SERVER LOGIC -------------------  
  
def send\_image(conn: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Sends an image file (IMAGE\_FILENAME) to the connected client,  
 encrypted with the provided AES protocol. Handles file not found  
 and other exceptions during the process.  
  
 Args:  
 conn (socket.socket): The client's socket connection.  
 aes\_protocol (MyAES): The AES encryption/decryption object for this client.  
 """* try:  
 with open(IMAGE\_FILENAME, 'rb') as f:  
 image\_data = f.read()  
 logging.debug(f"Read image file '{IMAGE\_FILENAME}' for sending.")  
  
 chk\_sum = protocol.calculate\_checksum(image\_data)  
 encrypted\_image\_data = aes\_protocol.encrypt(image\_data)  
 protocol.send\_msg(encrypted\_image\_data, conn)  
 logging.debug(f"Sent encrypted image data to client {conn.getpeername()}.")  
  
 encrypted\_chk\_sum = aes\_protocol.encrypt(chk\_sum.encode())  
 protocol.send\_msg(encrypted\_chk\_sum, conn)  
 logging.debug(f"Sent encrypted checksum to client {conn.getpeername()}.")  
  
 # Wait for client confirmation  
 confirmation\_data = protocol.recv\_encrypted\_msg(conn, aes\_protocol)  
 if not confirmation\_data:  
 logging.warning(f"No confirmation received from client {conn.getpeername()} after sending image.")  
 return # Or handle error appropriately  
  
 confirmation\_msg\_decoded = confirmation\_data.decode()  
 # Assuming protocol for confirmation is "checksum{status}"  
 received\_chk, status\_msg = confirmation\_msg\_decoded.split('{', 1)  
  
 # Loop for resending if confirmation is not "ok"  
 # This loop structure assumes client sends "checksum{again}" or similar  
 # if the image or checksum was not received correctly.  
 retry\_attempts = 0  
 max\_retries = 3 # Prevent infinite loop  
 while not status\_msg == "ok" and retry\_attempts < max\_retries:  
 logging.warning(f"Client {conn.getpeername()} did not confirm image receipt with 'ok' (status: {status\_msg}). Retrying...")  
 protocol.send\_msg(encrypted\_image\_data, conn) # Resend image  
 protocol.send\_msg(encrypted\_chk\_sum, conn) # Resend checksum  
   
 confirmation\_data = protocol.recv\_encrypted\_msg(conn, aes\_protocol)  
 if not confirmation\_data:  
 logging.error(f"No confirmation received on retry {retry\_attempts + 1} from {conn.getpeername()}. Aborting image send.")  
 return  
 confirmation\_msg\_decoded = confirmation\_data.decode()  
 received\_chk, status\_msg = confirmation\_msg\_decoded.split('{', 1)  
 retry\_attempts += 1  
   
 if status\_msg == "ok":  
 logging.info(f"Successfully sent image '{IMAGE\_FILENAME}' to client {conn.getpeername()} and received 'ok'.")  
 else:  
 logging.error(f"Failed to send image '{IMAGE\_FILENAME}' to client {conn.getpeername()} after {max\_retries} retries (final status: {status\_msg}).")  
  
 except FileNotFoundError:  
 logging.error(f"Image file '{IMAGE\_FILENAME}' not found on server.")  
 try:  
 protocol.encrypt\_and\_send\_msg("Image not found".encode(), conn, aes\_protocol)  
 except (socket.error, Exception) as send\_err:  
 logging.error(f"Failed to send 'Image not found' error to client {conn.getpeername() if conn else 'N/A'}: {send\_err}")  
 except (socket.error, AttributeError) as e: # AttributeError for aes\_protocol methods if not MyAES  
 logging.error(f"Socket or AES protocol error sending image to client {conn.getpeername() if conn else 'N/A'}: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending image to client {conn.getpeername() if conn else 'N/A'}: {e}")  
 try:  
 if conn: # Ensure conn is valid before trying to send an error message  
 protocol.encrypt\_and\_send\_msg("Error sending image".encode(), conn, aes\_protocol)  
 except (socket.error, Exception) as send\_err:  
 logging.error(f"Failed to send 'Error sending image' to client {conn.getpeername() if conn else 'N/A'}: {send\_err}")  
  
  
def starting\_sql(rsa\_protocol\_instance: MyRSA, conn: socket.socket) -> MyAES | None:  
 *"""  
 Performs the initial RSA key exchange with the client to establish a shared AES key.  
 Sends the server's public RSA key and receives an AES key encrypted by the client.  
  
 Args:  
 rsa\_protocol\_instance (MyRSA): The server's RSA object (used to get the public key).  
 conn (socket.socket): The client's socket connection.  
  
 Returns:  
 MyAES | None: An initialized MyAES object with the shared key if successful,  
 None otherwise.  
 """* try:  
 public\_key = rsa\_protocol\_instance.export\_public\_key()  
 # temp\_checksum = protocol.calculate\_checksum(public\_key) # Checksum of public key was calculated but not obviously used in original code  
 protocol.send\_msg(public\_key, conn)  
 logging.info(f"Sent public RSA key to client {conn.getpeername()}.")  
  
 # Receive the AES key encrypted by the public key  
 encrypted\_aes\_key = protocol.recv\_encrypted\_msg(conn, rsa\_protocol\_instance) # Decrypt with server's private RSA key  
 if not encrypted\_aes\_key:  
 logging.error(f"Failed to receive encrypted AES key from client {conn.getpeername()}.")  
 return None  
   
 logging.info(f"Received encrypted AES key from client {conn.getpeername()}.")  
 aes\_protocol\_instance = MyAES(encrypted\_aes\_key)  
 logging.info(f"AES protocol established with client {conn.getpeername()}. Starting encrypted communication.")  
 return aes\_protocol\_instance  
 except (socket.error, AttributeError) as e: # AttributeError for rsa/aes\_protocol methods if not MyRSA/MyAES  
 logging.error(f"Socket or crypto protocol error during key exchange with {conn.getpeername()}: {e}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error during key exchange with {conn.getpeername()}: {e}")  
 return None  
  
def receive\_server\_messages(conn: socket.socket, aes\_protocol: MyAES) -> bool:  
 *"""  
 Receives and processes messages from a client.  
 Handles checksum verification, authentication for the first messages,  
 and command processing for subsequent messages.  
  
 Global variables modified: send\_again, did\_send, last\_massage, msg, IS\_FIRST, num1.  
  
 Args:  
 conn (socket.socket): The client's socket connection.  
 aes\_protocol (MyAES): The AES protocol object for this client.  
  
 Returns:  
 bool: True if a valid non-control message for further processing was received  
 (after initial authentication phase), False otherwise.  
 """* global send\_again, did\_send, last\_massage, msg, IS\_FIRST, num1  
 client\_addr = conn.getpeername() if conn else "Unknown client"  
 try:  
 encrypted\_msg\_data = protocol.recv\_encrypted\_msg(conn, aes\_protocol)  
 if not encrypted\_msg\_data:  
 logging.warning(f"No data received from client {client\_addr}. Connection might be closed.")  
 return False  
   
 full\_msg\_decoded = encrypted\_msg\_data.decode()  
 # Expecting "checksum{message\_content}"  
 try:  
 chk, current\_msg\_content = full\_msg\_decoded.split('{', 1)  
 except ValueError:  
 logging.error(f"Received message from {client\_addr} in unexpected format: {full\_msg\_decoded}")  
 # Optionally send an error back to the client or handle as a protocol violation  
 protocol.encrypt\_and\_send\_msg("error:invalid message format".encode(), conn, aes\_protocol)  
 return False  
  
 chk = chk.lstrip("0") # Remove leading zeros from checksum string  
 logging.debug(f"Received from {client\_addr}: chk='{chk}', content='{current\_msg\_content}'")  
 # Update global msg here, so send\_server\_msg uses the latest  
 msg = current\_msg\_content # This global `msg` will be used by send\_server\_msg if this function returns True  
  
 if not current\_msg\_content: # Empty message content after splitting  
 logging.warning(f"Received empty message content from {client\_addr}.")  
 # Decide how to handle: maybe an error, or just ignore.  
 return False  
  
 if current\_msg\_content == "again":  
 logging.info(f"Client {client\_addr} requested resend ('again').")  
 did\_send = True # Flag to resend last\_massage  
 # The original code sends last\_massage here directly,  
 # but the global `did\_send` should trigger it in `send\_server\_msg`  
 # For consistency, let's stick to the original flow and let send\_server\_msg handle it.  
 # However, the client expects an immediate response.  
 protocol.encrypt\_and\_send\_msg(last\_massage.encode(), conn, aes\_protocol)  
 return False # Not a message for further server logic processing  
  
 elif not protocol.is\_checksum(chk, current\_msg\_content):  
 logging.warning(f"Checksum mismatch for message from {client\_addr}. Requesting resend.")  
 protocol.encrypt\_and\_send\_msg("again".encode(), conn, aes\_protocol)  
 did\_send = True # To prevent send\_server\_msg from sending the faulty `msg`  
 return False # Not a message for further server logic processing  
  
 elif IS\_FIRST: # Authentication phase  
 logging.info(f"Authentication attempt from {client\_addr}: {current\_msg\_content}")  
 try:  
 user\_name, password\_str = current\_msg\_content.split("!", 1)  
 # It's safer to expect password as string from client, then convert if necessary  
 # The original code used int(password), which might fail if password is not purely numeric  
 if authenticate\_user(user\_name, password\_str): # Assuming authenticate\_user can handle string password  
 response\_msg = "you are now connected"  
 last\_massage = response\_msg # Update last\_massage with the response we are sending  
 logging.info(f"User '{user\_name}' from {client\_addr} authenticated. Sending: '{response\_msg}'")  
 protocol.encrypt\_and\_send\_msg(response\_msg.encode(), conn, aes\_protocol)  
 did\_send = True # To indicate a message was just sent, preventing send\_server\_msg from sending again immediately  
 num1 += 1  
 if num1 >= 3: # Assuming 3 successful authentications from \*any\* of the first clients  
 logging.info("Authentication phase completed (IS\_FIRST = False).")  
 IS\_FIRST = False  
 else:  
 response\_msg = "not existing user name or password"  
 last\_massage = response\_msg  
 logging.warning(f"Authentication failed for '{user\_name}' from {client\_addr}. Sending: '{response\_msg}'")  
 protocol.encrypt\_and\_send\_msg(response\_msg.encode(), conn, aes\_protocol)  
 did\_send = True  
 except ValueError: # If msg.split("!") fails  
 response\_msg = "invalid authentication format"  
 last\_massage = response\_msg  
 logging.error(f"Invalid authentication format from {client\_addr}: '{current\_msg\_content}'. Sending: '{response\_msg}'")  
 protocol.encrypt\_and\_send\_msg(response\_msg.encode(), conn, aes\_protocol)  
 did\_send = True  
 except Exception as e: # Catch other potential errors during auth processing  
 response\_msg = f"server error during authentication: {str(e)}"  
 last\_massage = response\_msg # Even server errors update last\_massage for consistency  
 logging.error(f"Server error processing authentication for {client\_addr}: {e}")  
 protocol.encrypt\_and\_send\_msg(response\_msg.encode(), conn, aes\_protocol)  
 did\_send = True  
 return False # Authentication messages are not for further server logic processing in main loop  
  
 else: # Regular message after authentication  
 logging.info(f"Received valid message from {client\_addr} (post-auth): '{current\_msg\_content}'")  
 # The global `msg` is already set to current\_msg\_content  
 return True # Indicates a message for the main server logic  
   
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES protocol error receiving/processing message from {client\_addr}: {e}")  
 return False  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error for message from {client\_addr}: {e}. Message might not be valid UTF-8 after decryption.")  
 # Optionally, send an error message back to the client  
 try:  
 protocol.encrypt\_and\_send\_msg("error:undecodable message".encode(), conn, aes\_protocol)  
 except Exception as send\_err:  
 logging.error(f"Failed to send undecodable message error to {client\_addr}: {send\_err}")  
 return False  
 except Exception as e:  
 logging.error(f"Unexpected error receiving/processing message from {client\_addr}: {e}")  
 return False  
 # Fallback if no other return path was hit (e.g. if msg was empty initially, though guarded)  
 return False  
  
  
def send\_server\_msg(conn: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Sends a message (held in global `msg`) to the client,  
 handling potential resends or default messages.  
  
 Global variables used: msg, last\_massage, did\_send.  
 Global variables modified: did\_send, last\_massage.  
  
 Args:  
 conn (socket.socket): The client's socket connection.  
 aes\_protocol (MyAES): The AES protocol object for this client.  
 """* global msg, last\_massage, did\_send  
 client\_addr = conn.getpeername() if conn else "Unknown client"  
  
 try:  
 if did\_send: # If a message was already sent by receive\_server\_messages (e.g., auth response, "again" response)  
 did\_send = False # Reset flag  
 logging.debug(f"send\_server\_msg: did\_send was true, message already sent by receiver for {client\_addr}. Skipping send.")  
 return  
  
 # Current message to be sent is in global `msg` (set by caller or receive\_server\_messages)  
 message\_to\_send = msg   
  
 if message\_to\_send == last\_massage and message\_to\_send != "DEFULT": # Avoid re-sending "DEFULT" if it was the last message  
 logging.debug(f"Message to send ('{message\_to\_send}') is same as last\_massage for {client\_addr}. Sending 'DEFULT'.")  
 message\_to\_send = "DEFULT"  
   
 # Update last\_massage with what we are about to send  
 # This should happen regardless of whether it's the original msg or "DEFULT"  
 # However, the original code did `last\_massage = msg` which means `msg` itself was updated  
 # Let's keep the global `msg` as what was intended to be sent, and `message\_to\_send` as what is actually sent.  
 # And `last\_massage` should be what was \*actually\* sent.  
   
 logging.info(f"Sending to {client\_addr}: '{message\_to\_send}' (original global msg was: '{msg}')")  
 protocol.encrypt\_and\_send\_msg(message\_to\_send.encode(), conn, aes\_protocol)  
 last\_massage = message\_to\_send # Update last\_massage to what was actually sent.  
  
 if message\_to\_send == "pic": # Or should it be if global msg == "pic"? Original used global msg.  
 # Let's assume if the \*actual sent message\* is "pic"  
 logging.info(f"Message 'pic' sent to {client\_addr}, calling send\_image.")  
 send\_image(conn, aes\_protocol)  
   
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES protocol error sending message to {client\_addr}: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending message to {client\_addr}: {e}")  
  
  
def main() -> None:  
 *"""  
 Main server function.  
 Initializes the database, sets up the server socket, and handles client connections  
 and communication in a sequential manner for three clients.  
 The primary interaction loop involves client1 (conn), client2 (conn1), and client3 (conn2).  
 """* global send\_again, did\_send, last\_massage, msg, IS\_FIRST # num1 is also global but managed within receive\_server\_messages  
  
 logging.info("Server starting...")  
 init\_database()  
 populate\_initial\_users()  
  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 # Allow address reuse  
 server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
   
 # Create RSA instances for each potential client connection early on  
 # This way, if one connection fails, we don't re-generate for others unnecessarily  
 # The original code created them inside the try block.  
 rsa\_main\_client = MyRSA()  
 rsa\_client1\_instance = MyRSA() # For 'conn1'  
 rsa\_client2\_instance = MyRSA() # For 'conn2'  
  
 conn = None  
 conn1 = None  
 conn2 = None  
  
 try:  
 server\_socket.bind((IP\_ADDR, PORT))  
 server\_socket.listen(QUEUE\_LEN)  
 logging.info(f"Server listening on {IP\_ADDR}:{PORT}")  
  
 # Client 1 (conn)  
 logging.info("Waiting for client 1 (conn)...")  
 conn, addr = server\_socket.accept()  
 logging.info(f"Client 1 (conn) connected from {addr}")  
 aes\_protocol\_main = starting\_sql(rsa\_main\_client, conn)  
 if not aes\_protocol\_main:  
 logging.error(f"AES setup failed for client 1 {addr}. Closing connection.")  
 if conn: conn.close()  
 return # Critical failure  
  
 # Initial message exchange with client 1  
 receive\_server\_messages(conn, aes\_protocol\_main) # Handles auth, updates globals  
 # send\_server\_msg will use global `msg` set by receive\_server\_messages or its default logic  
 send\_server\_msg(conn, aes\_protocol\_main)   
  
 # Client 2 (conn1)  
 logging.info("Waiting for client 2 (conn1)...")  
 conn1, addr1 = server\_socket.accept()  
 logging.info(f"Client 2 (conn1) connected from {addr1}")  
 aes\_protocol\_client1 = starting\_sql(rsa\_client1\_instance, conn1)  
 if not aes\_protocol\_client1:  
 logging.error(f"AES setup failed for client 2 {addr1}. Closing connection.")  
 if conn1: conn1.close()  
 if conn: conn.close() # Close previous connections too on critical failure  
 return  
 receive\_server\_messages(conn1, aes\_protocol\_client1) # Auth for client 2  
  
 # Client 3 (conn2)  
 logging.info("Waiting for client 3 (conn2)...")  
 conn2, addr2 = server\_socket.accept()  
 logging.info(f"Client 3 (conn2) connected from {addr2}")  
 aes\_protocol\_client2 = starting\_sql(rsa\_client2\_instance, conn2)  
 if not aes\_protocol\_client2:  
 logging.error(f"AES setup failed for client 3 {addr2}. Closing connection.")  
 if conn2: conn2.close()  
 if conn1: conn1.close()  
 if conn: conn.close()  
 return  
 receive\_server\_messages(conn2, aes\_protocol\_client2) # Auth for client 3  
 # The original code had another receive\_server\_messages for conn2 here.  
 # This implies client3 sends two messages back-to-back for its initial setup.  
 # Or it was an oversight. Assuming client3 sends only one auth message for now.  
 # If two are needed, uncomment:  
 # receive\_server\_messages(conn2, aes\_protocol\_client2)   
  
 # Some initial messages to client 1 again?  
 # The global `msg` might still hold the last response from client3's auth.  
 # This logic is a bit obscure due to globals.  
 send\_server\_msg(conn, aes\_protocol\_main)  
 send\_server\_msg(conn, aes\_protocol\_main)  
  
 # Main communication loop  
 logging.info("All clients connected. Starting main communication loop.")  
 msg = "start" # Initialize global msg for the loop condition  
 while msg != EXIT\_RES: # EXIT\_RES is 'by by'  
 # Receive from client 1 (conn)  
 logging.debug(f"Main loop: Waiting for message from client 1 {addr}")  
 is\_message\_for\_logic = receive\_server\_messages(conn, aes\_protocol\_main)  
 # `msg` global is updated by receive\_server\_messages with client1's message content if valid  
  
 if is\_message\_for\_logic: # A valid data message was received from client 1  
 logging.info(f"Main loop: Processing logic for message from client 1: '{msg}'")  
 original\_client1\_msg = msg # Save it before it's potentially modified  
   
 try:  
 # Assuming client1's message is "value1@value2"  
 part1, part2\_trash = original\_client1\_msg.split('@', 1)  
 except ValueError:  
 logging.error(f"Message from client 1 '{original\_client1\_msg}' not in 'value1@value2' format. Skipping logic.")  
 # Send the original (potentially faulty) message back or an error  
 send\_server\_msg(conn, aes\_protocol\_main) # This will send original\_client1\_msg or DEFULT  
 continue # Go to next iteration of while loop  
  
 temp\_part1\_for\_client2 = part1 # Store part1 for client2 (conn1)  
   
 # Send part1 to client 3 (conn2)  
 msg = part1 # Set global msg to what client3 should receive  
 did\_send = False # Ensure send\_server\_msg actually sends  
 logging.debug(f"Main loop: Sending '{msg}' to client 3 {addr2}")  
 send\_server\_msg(conn2, aes\_protocol\_client2)  
  
 # Receive decision from client 3 (conn2)  
 logging.debug(f"Main loop: Waiting for decision from client 3 {addr2}")  
 receive\_server\_messages(conn2, aes\_protocol\_client2) # Updates global `msg` with client3's decision  
 client3\_decision = msg # Store client3's decision  
  
 if client3\_decision == "kill":  
 logging.info(f"Main loop: Client 3 decided 'kill'. Relaying to client 2 {addr1}.")  
 # Send temp\_part1\_for\_client2 (part1 from client1) to client 2 (conn1)  
 msg = temp\_part1\_for\_client2  
 last\_massage = "hi" # Original code hardcoded this, purpose unclear  
 did\_send = False  
 logging.debug(f"Main loop: Sending '{msg}' (part1) to client 2 {addr1}")  
 send\_server\_msg(conn1, aes\_protocol\_client1)  
  
 # Send part2\_trash (part2 from client1) to client 2 (conn1)  
 msg = part2\_trash  
 did\_send = False  
 logging.debug(f"Main loop: Sending '{msg}' (part2) to client 2 {addr1}")  
 send\_server\_msg(conn1, aes\_protocol\_client1)  
  
 # Receive ack from client 2 (conn1)  
 logging.debug(f"Main loop: Waiting for ack from client 2 {addr1}")  
 receive\_server\_messages(conn1, aes\_protocol\_client1) # Updates global `msg` with client2's ack  
 logging.info(f"Main loop: Received ack '{msg}' from client 2 {addr1} after 'kill' sequence.")  
   
 while True:  
 pass # This will make the server unresponsive  
 else: # Client 3 did not decide 'kill' (e.g., "no")  
 # Send client3's decision back to client 1 (conn)  
 # `msg` global already holds client3\_decision  
 did\_send = False  
 logging.debug(f"Main loop: Client 3 decision was '{client3\_decision}'. Sending to client 1 {addr}.")  
 send\_server\_msg(conn, aes\_protocol\_main)  
 else: # No new data message from client 1, or it was a control message ("again", auth)  
 # Send a response to client 1 based on current global `msg` or `last\_massage`  
 logging.debug(f"Main loop: No new data message from client 1, or it was control. Sending current/last msg to client 1 {addr}.")  
 # `did\_send` might be true if receive\_server\_messages handled an "again" or auth response.  
 # `send\_server\_msg` will respect `did\_send`.  
 # If `receive\_server\_messages` returned False because connection closed, this send might fail.  
 if conn: # Check if conn is still valid  
 send\_server\_msg(conn, aes\_protocol\_main)  
 else:  
 logging.warning(f"Main loop: Client 1 (conn) seems to be closed. Cannot send message.")  
 break # Exit while loop if primary client is gone  
  
 # Check if global msg became EXIT\_RES due to a client message  
 if msg == EXIT\_RES:  
 logging.info(f"EXIT\_RES ('{EXIT\_RES}') detected. Preparing to exit main loop.")  
 break  
   
 logging.info("Exiting main communication loop.")  
  
 except socket.error as sock\_err:  
 logging.error(f"Socket error in main server operation: {sock\_err}")  
 except KeyboardInterrupt:  
 logging.info("Server shutting down due to KeyboardInterrupt.")  
 except Exception as e:  
 logging.error(f"An unexpected error occurred in main server operation: {e}", exc\_info=True)  
 finally:  
 logging.info("Closing client connections...")  
 if conn:  
 try:  
 conn.close()  
 logging.info(f"Closed connection with client 1 (conn).")  
 except socket.error as e:  
 logging.error(f"Error closing connection conn: {e}")  
 if conn1:  
 try:  
 conn1.close()  
 logging.info(f"Closed connection with client 2 (conn1).")  
 except socket.error as e:  
 logging.error(f"Error closing connection conn1: {e}")  
 if conn2:  
 try:  
 conn2.close()  
 logging.info(f"Closed connection with client 3 (conn2).")  
 except socket.error as e:  
 logging.error(f"Error closing connection conn2: {e}")  
   
 if server\_socket:  
 try:  
 server\_socket.close()  
 logging.info("Server socket closed.")  
 except socket.error as e:  
 logging.error(f"Error closing server socket: {e}")  
 logging.info("Server shutdown complete.")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Logging is configured at the top of the file now  
 main()

## Client 1

*"""  
Author: Iftach kasorla  
Date: 3/6/25  
Description: a simple TLS client (no GUI) for a sensor robot.  
Connects to a server, authenticates, exchanges encrypted messages,  
and interacts with an Arduino via serial communication.  
"""*import logging  
import socket  
import protocol # Assuming protocol.py exists  
from aes import MyAES # Assuming aes.py and MyAES class exist  
from rsa import MyRSA # Assuming rsa.py and MyRSA class exist  
import serial  
import time  
  
# Configure serial port (change 'COM3' to your Arduino's port if different)  
arduino\_port = 'COM3'  
baud\_rate = 9600  
  
HOST\_NAME = '127.0.0.1' # Server's hostname or IP address  
PORT = 8443 # Server's port  
EXIT\_CMD = 'by by' # Command from server that signals client to exit  
# USER\_INPUT = 'please enter a command: ' # Unused global variable  
send\_again = False # Flag to indicate if the last message needs to be resent  
did\_send = False # Flag to indicate if a message was just sent (e.g., by receive\_client\_messages)  
last\_massage = '' # Stores the last message sent by this client  
IMAGE\_FILENAME = "received\_image.jpg" # Filename to save received images  
DEFAULT\_MSG = "DEFAULT" # Default message content, purpose unclear in current logic  
msg = '' # Holds the current message received from server or to be sent (from Arduino)  
  
# Configure logging at the beginning  
logging.basicConfig(  
 format='%(asctime)s | %(levelname)s | CLIENT1 - %(message)s',  
 datefmt='%m/%d/%Y %I:%M:%S %p',  
 level=logging.INFO,  
 filename='tls\_client1.log', # Changed filename to be specific to client1  
 filemode='w'  
)  
  
def connect\_to\_robo\_v3() -> serial.Serial | None:  
 *"""  
 Establishes a serial connection to the Arduino (robot).  
 Sends an initial 'a' byte to the Arduino.  
  
 Returns:  
 serial.Serial | None: The serial connection object if successful, None otherwise.  
 """* try:  
 arduino = serial.Serial(arduino\_port, baud\_rate, timeout=1)  
 logging.info(f"Attempting to connect to Arduino on {arduino\_port} at {baud\_rate} baud.")  
 time.sleep(2) # Wait for the connection to establish  
 if arduino.is\_open:  
 logging.info(f"Successfully connected to Arduino on {arduino\_port}.")  
 print(f"Connected to {arduino\_port}")  
 arduino.write(b'a') # Start robot v3 communication / signal  
 logging.debug("Sent initial 'a' byte to Arduino.")  
 return arduino  
 else:  
 logging.error(f"Failed to open serial port {arduino\_port}.")  
 print(f"Failed to open serial port {arduino\_port}.")  
 return None  
 except serial.SerialException as se:  
 logging.error(f"Serial error connecting to Arduino on {arduino\_port}: {se}")  
 print(f"Serial error on {arduino\_port}: {se}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error connecting to Arduino: {e}")  
 print(f"Unexpected error connecting to Arduino: {e}")  
 return None  
  
def send\_robo\_v3(arduino: serial.Serial) -> None:  
 *"""  
 Sends an 'a' byte to the connected Arduino.  
 Typically used as a signal or keep-alive.  
  
 Args:  
 arduino (serial.Serial): The active serial connection object to the Arduino.  
 """* if arduino and arduino.is\_open:  
 try:  
 arduino.write(b'a')  
 logging.debug("Sent 'a' byte to Arduino (send\_robo\_v3).")  
 except serial.SerialException as se:  
 logging.error(f"Serial error sending 'a' to Arduino: {se}")  
 print(f"Serial error sending 'a': {se}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending 'a' to Arduino: {e}")  
 print(f"Unexpected error sending 'a': {e}")  
 else:  
 logging.warning("Arduino not connected or port not open. Cannot send 'a'.")  
 print("Arduino not connected. Cannot send 'a'.")  
  
  
def receive\_massage\_v3(arduino: serial.Serial) -> str | None:  
 *"""  
 Receives two bytes from the Arduino and formats them as "byte1\_ord@byte2\_ord".  
 This is a blocking call until two bytes are received or timeout occurs on read.  
  
 Args:  
 arduino (serial.Serial): The active serial connection object to the Arduino.  
  
 Returns:  
 str | None: The formatted string from Arduino data if successful, None on error or timeout.  
 """* if not (arduino and arduino.is\_open):  
 logging.warning("Arduino not connected or port not open. Cannot receive message.")  
 print("Arduino not connected. Cannot receive message.")  
 return None  
 try:  
 while True: # Loop to ensure we get the first byte  
 temp1 = arduino.read(1)  
 if temp1:  
 # Now try to get the second byte  
 temp2 = arduino.read(1)  
 if temp2:  
 val1 = ord(temp1)  
 val2 = ord(temp2)  
 formatted\_msg = f"{val1}@{val2}"  
 logging.debug(f"Received from Arduino: byte1={val1}, byte2={val2}. Formatted: '{formatted\_msg}'")  
 return formatted\_msg  
 else:  
 logging.warning("Received first byte from Arduino, but timed out waiting for the second.")  
 # Potentially incomplete message, decide how to handle. Returning None for now.  
 return None   
 # If temp1 is empty, the loop continues due to timeout=1 on Serial.  
 # This could be problematic if Arduino never sends data.  
 # Adding a counter or a more robust timeout mechanism might be needed for production.  
 logging.debug("Waiting for data from Arduino...") # Log if read call returns empty  
 except serial.SerialException as se:  
 logging.error(f"Serial error receiving message from Arduino: {se}")  
 print(f"Serial error receiving: {se}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error receiving message from Arduino: {e}")  
 print(f"Unexpected error receiving: {e}")  
 return None  
  
def receive\_image(my\_socket: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Receives an encrypted image file and its checksum from the server,  
 decrypts it, verifies the checksum, and saves the image.  
  
 Args:  
 my\_socket (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for decryption.  
 """* try:  
 logging.info("Attempting to receive image from server...")  
 encrypted\_data = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_data:  
 logging.error("Failed to receive encrypted image data from server.")  
 return  
  
 encrypted\_chk = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_chk:  
 logging.error("Failed to receive encrypted checksum from server.")  
 return  
   
 chk\_decoded = encrypted\_chk.decode()  
 logging.debug("Received encrypted image data and checksum.")  
  
 # Loop for checksum verification and requesting resend  
 retries = 0  
 max\_retries = 3 # To prevent infinite loops  
 while not protocol.is\_checksum(chk\_decoded, encrypted\_data) and retries < max\_retries:  
 logging.warning(f"Checksum mismatch for received image. Requesting resend (attempt {retries + 1}).")  
 protocol.encrypt\_and\_send\_msg("again".encode(), my\_socket, aes\_protocol)  
   
 encrypted\_data = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_data:  
 logging.error(f"Failed to receive resent encrypted image data (attempt {retries + 1}).")  
 return  
 encrypted\_chk = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_chk:  
 logging.error(f"Failed to receive resent encrypted checksum (attempt {retries + 1}).")  
 return  
 chk\_decoded = encrypted\_chk.decode()  
 retries += 1  
   
 if not protocol.is\_checksum(chk\_decoded, encrypted\_data):  
 logging.error(f"Checksum mismatch after {max\_retries} retries. Aborting image receive.")  
 # Send a final "not ok" or just give up. Original sends "ok" only on success.  
 protocol.encrypt\_and\_send\_msg("error:checksum\_failed".encode(), my\_socket, aes\_protocol) # Example error  
 return  
  
 logging.info("Image checksum verified. Sending 'ok' to server.")  
 protocol.encrypt\_and\_send\_msg("ok".encode(), my\_socket, aes\_protocol)  
  
 try:  
 with open(IMAGE\_FILENAME, 'wb') as f:  
 f.write(encrypted\_data) # Original code writes encrypted\_data, assumes it's decrypted by protocol.recv\_encrypted\_msg  
 # If protocol.recv\_encrypted\_msg returns raw encrypted data, it needs decryption first.  
 # The current protocol.py decrypts it, so this should be decrypted data.  
 logging.info(f"Received and saved image as '{IMAGE\_FILENAME}'")  
 print(f"Received image: {IMAGE\_FILENAME}")  
 except IOError as e:  
 logging.error(f"IOError saving image '{IMAGE\_FILENAME}': {e}")  
 print(f"Error saving image: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error saving image '{IMAGE\_FILENAME}': {e}")  
 print(f"Error saving image: {e}")  
  
 except (socket.error, AttributeError) as e: # AttributeError for aes\_protocol methods  
 logging.error(f"Socket or AES protocol error during image reception: {e}")  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error for checksum during image reception: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error during image reception: {e}")  
  
  
def receive\_client\_messages(conn: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Receives and processes messages from the server.  
 Handles checksums, "again" requests, "pic" commands, and general messages.  
 Updates global variables based on received messages.  
  
 Args:  
 conn (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for decryption.  
 """* global send\_again, did\_send, last\_massage, msg # msg is updated here  
 try:  
 encrypted\_msg\_data = protocol.recv\_encrypted\_msg(conn, aes\_protocol)  
 if not encrypted\_msg\_data:  
 logging.warning("No data received from server. Connection might be closed.")  
 msg = EXIT\_CMD # Assume server closed, trigger exit  
 return  
   
 full\_msg\_decoded = encrypted\_msg\_data.decode()  
 # Expecting "checksum{message\_content}"  
 try:  
 chk, current\_msg\_content = full\_msg\_decoded.split('{', 1)  
 except ValueError:  
 logging.error(f"Received message from server in unexpected format: {full\_msg\_decoded}")  
 msg = EXIT\_CMD # Treat as critical error, trigger exit  
 return  
  
 chk = chk.lstrip("0") # Remove leading zeros from checksum string  
 logging.debug(f"Received from server: chk='{chk}', content='{current\_msg\_content}'")  
 msg = current\_msg\_content # Update global msg  
  
 if not current\_msg\_content:  
 logging.warning("Received empty message content from server.")  
 # msg is already set to empty, loop might continue depending on EXIT\_CMD check  
 return  
  
 if current\_msg\_content == "again":  
 logging.info("Server requested resend ('again'). Preparing to resend last message.")  
 did\_send = True # Flag to resend last\_massage by send\_client\_message  
 # The original code sends last\_massage here directly.  
 # For consistency with send\_client\_message's logic, we'll let it handle it.  
 # However, the server might expect an immediate response.  
 # Let's send it here as original code implied.  
 if last\_massage: # Ensure there is a last message to send  
 protocol.encrypt\_and\_send\_msg(last\_massage.encode(), conn, aes\_protocol)  
 else:  
 logging.warning("Server requested 'again', but no 'last\_massage' to resend.")  
 # Send a default or error? For now, do nothing more.  
 # `msg` is "again", so the main loop won't process it as a data command.  
  
 elif not protocol.is\_checksum(chk, current\_msg\_content):  
 logging.warning("Checksum mismatch for message from server. Requesting server resend.")  
 protocol.encrypt\_and\_send\_msg("again".encode(), conn, aes\_protocol)  
 did\_send = True # To prevent send\_client\_message from sending the faulty `msg`  
 # `msg` holds the faulty content, but `did\_send` will make `send\_client\_message` skip.  
  
 elif current\_msg\_content == "pic":  
 logging.info("Received 'pic' command from server. Initiating image reception.")  
 receive\_image(conn, aes\_protocol)  
 # `msg` is "pic", main loop might act on this if not EXIT\_CMD.  
  
 else:  
 # This is a general message from the server  
 logging.info(f"Received from server: '{current\_msg\_content}'")  
 print(f"Client received: {current\_msg\_content}")  
 # `msg` is updated with this content for the main loop.  
  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES error receiving message from server: {e}")  
 msg = EXIT\_CMD # Assume critical error, trigger exit  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error for message from server: {e}")  
 msg = EXIT\_CMD  
 except Exception as e:  
 logging.error(f"Unexpected error receiving message from server: {e}")  
 msg = EXIT\_CMD  
  
  
def send\_client\_message(conn: socket.socket, aes\_protocol: MyAES, arduino: serial.Serial | None) -> None:  
 *"""  
 Sends a message to the server. If `did\_send` is True, it means a message  
 was already handled by `receive\_client\_messages` (e.g. "again" response), so it skips.  
 Otherwise, it reads a message from the Arduino and sends it.  
 Updates global variables.  
  
 Args:  
 conn (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for encryption.  
 arduino (serial.Serial | None): The active serial connection to Arduino, or None.  
 """* global msg, last\_massage, did\_send  
   
 if did\_send: # If receive\_client\_messages already sent something (like an "again" confirmation)  
 did\_send = False  
 logging.debug("send\_client\_message: did\_send was true, skipping send from Arduino.")  
 return  
  
 if not arduino:  
 logging.error("Arduino not available, cannot get message to send.")  
 # Optionally, send a placeholder or error message to server  
 # For now, we just don't send anything if Arduino is down.  
 # This might break server's expectation of a message.  
 return  
  
 arduino\_data = receive\_massage\_v3(arduino)  
 if arduino\_data is not None:  
 msg\_to\_send = arduino\_data  
 last\_massage = msg\_to\_send # Update last\_massage with what we are about to send  
 # The global `msg` is also updated, though its immediate use after this send is for server's response.  
 msg = msg\_to\_send   
  
 logging.info(f"Sending to server (from Arduino): '{msg\_to\_send}'")  
 try:  
 protocol.encrypt\_and\_send\_msg(msg\_to\_send.encode(), conn, aes\_protocol)  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES error sending message to server: {e}")  
 # Potentially set global msg to EXIT\_CMD or handle error to break main loop  
 except Exception as e:  
 logging.error(f"Unexpected error sending message to server: {e}")  
 else:  
 logging.warning("Failed to receive message from Arduino. Nothing sent to server.")  
 # Server will eventually timeout or client might send a default heartbeat if implemented.  
  
  
def starting\_sql\_client(my\_socket: socket.socket) -> MyAES | None:  
 *"""  
 Performs the initial handshake with the server:  
 1. Receives the server's public RSA key.  
 2. Generates an AES key, encrypts it with the server's public RSA key, and sends it.  
 3. Sends hardcoded authentication credentials.  
 4. Receives an authentication response from the server.  
  
 Args:  
 my\_socket (socket.socket): The connected socket to the server.  
  
 Returns:  
 MyAES | None: The established AES protocol object if successful, None otherwise.  
 """* try:  
 # Receive the RSA public key  
 public\_key\_data = protocol.recv\_msg(my\_socket)  
 if not public\_key\_data:  
 logging.error("Failed to receive public RSA key from server.")  
 return None  
 logging.info(f"Received public RSA key from server: {public\_key\_data.decode()[:30]}...") # Log snippet  
 print("Received the server's public key.")  
  
 # Create the AES key and send it encrypted with the RSA public key  
 aes\_protocol\_instance = MyAES()  
 encrypted\_aes\_key = MyRSA.encrypt\_with\_key(aes\_protocol\_instance.export\_key(), public\_key\_data)  
   
 logging.info("Sending encrypted AES key to server.")  
 protocol.send\_msg(encrypted\_aes\_key, my\_socket)  
 print("Sent the encrypted AES key.")  
  
 # Authentication  
 user\_name = "iftach" # Hardcoded username  
 password = "1234" # Hardcoded password  
 auth\_msg\_str = f"{user\_name}!{password}"  
 logging.info(f"Sending authentication credentials for user '{user\_name}'.")  
 protocol.encrypt\_and\_send\_msg(auth\_msg\_str.encode(), my\_socket, aes\_protocol\_instance)  
   
 # Receive authentication response  
 # This call updates global `msg`  
 receive\_client\_messages(my\_socket, aes\_protocol\_instance)  
 logging.info(f"Authentication response from server: '{msg}'") # global msg was set by receive\_client\_messages  
  
 if "connected" in msg: # Check if auth was successful based on server's response  
 return aes\_protocol\_instance  
 else:  
 logging.error(f"Authentication failed. Server response: {msg}")  
 return None  
  
 except (socket.error, AttributeError) as e: # AttributeError for MyRSA/MyAES methods  
 logging.error(f"Socket or crypto protocol error during initial handshake: {e}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error during initial handshake: {e}")  
 return None  
  
  
def run\_client() -> None:  
 *"""  
 Main function for the client.  
 Establishes connection with the server, performs handshake and authentication,  
 connects to Arduino, and then enters a loop to send Arduino data to the server  
 and receive responses. Exits if the server sends EXIT\_CMD or on error.  
 """* global msg # Used for EXIT\_CMD check  
 my\_socket = socket.socket()  
 aes\_protocol\_instance = None  
 arduino\_conn = None  
   
 logging.info("Client1 starting...")  
  
 try:  
 logging.info(f"Attempting to connect to server at {HOST\_NAME}:{PORT}...")  
 my\_socket.connect((HOST\_NAME, PORT))  
 print(f"Connected to server: {HOST\_NAME}:{PORT}")  
 logging.info(f"Successfully connected to server: {HOST\_NAME}:{PORT}")  
  
 aes\_protocol\_instance = starting\_sql\_client(my\_socket)  
 if not aes\_protocol\_instance:  
 logging.critical("Failed to establish secure session with server. Exiting.")  
 print("Could not establish secure session. Exiting.")  
 return  
   
 # Initial message after authentication (server might send one)  
 # The original code had a receive\_client\_messages here.  
 # This is already handled at the end of starting\_sql\_client.  
 # If an additional message is expected, uncomment:  
 # receive\_client\_messages(my\_socket, aes\_protocol\_instance)  
 # logging.info(f"Post-authentication message from server: '{msg}'")  
  
  
 arduino\_conn = connect\_to\_robo\_v3()  
 if not arduino\_conn:  
 logging.error("Failed to connect to Arduino. Client will run without Arduino interaction.")  
 # Decide if client should exit or run in a limited mode  
 # For now, it will continue, and send\_client\_message will log errors.  
  
 logging.info("Entering main communication loop...")  
 while True:  
 send\_client\_message(my\_socket, aes\_protocol\_instance, arduino\_conn)  
 receive\_client\_messages(my\_socket, aes\_protocol\_instance) # Updates global `msg`  
   
 if arduino\_conn: # Only send to Arduino if connected  
 send\_robo\_v3(arduino\_conn) # Send keep-alive or trigger  
   
 if msg == EXIT\_CMD:  
 logging.info(f"Received EXIT\_CMD ('{EXIT\_CMD}') from server. Exiting client.")  
 print("Server requested exit. Closing client.")  
 break  
 # Add a small delay to prevent busy-looping if desired, e.g., time.sleep(0.1)  
  
 except socket.error as sock\_err:  
 logging.error(f"Socket error in client: {sock\_err}")  
 print(f"Socket error: {sock\_err}")  
 except serial.SerialException as ser\_err:  
 logging.error(f"Serial communication error in client: {ser\_err}")  
 print(f"Serial error: {ser\_err}")  
 except KeyboardInterrupt:  
 logging.info("Client shutdown initiated by user (KeyboardInterrupt).")  
 print("\nClient exiting...")  
 except Exception as e:  
 logging.error(f"An unexpected error occurred in client: {e}", exc\_info=True)  
 print(f"An error occurred: {e}")  
 finally:  
 if arduino\_conn and arduino\_conn.is\_open:  
 try:  
 arduino\_conn.close()  
 logging.info("Arduino serial connection closed.")  
 print("Arduino connection closed.")  
 except Exception as e:  
 logging.error(f"Error closing Arduino connection: {e}")  
 if my\_socket:  
 try:  
 my\_socket.close()  
 logging.info("Socket connection to server closed.")  
 print("Socket closed.")  
 except Exception as e:  
 logging.error(f"Error closing socket: {e}")  
 logging.info("Client1 finished.")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Logging is configured at the top of the file  
 run\_client()

## Client2

*"""  
Author: Iftach Kasorla  
Date: 3/6/25  
Description: A simple TLS client (no GUI) for an actuator robot (e.g., a "killer robot").  
Connects to a server, authenticates, exchanges encrypted messages,  
and interacts with an Arduino to perform actions based on server commands.  
"""*import logging  
import socket  
import protocol # Assuming protocol.py exists  
from aes import MyAES # Assuming aes.py and MyAES class exist  
from rsa import MyRSA # Assuming rsa.py and MyRSA class exist  
import serial  
import time  
  
# Configure serial port (change 'COM4' to your Arduino's port if different)  
arduino\_port = 'COM4'  
baud\_rate = 9600  
  
HOST\_NAME = '10.100.102.21' # Server's hostname or IP address (Note: Different from client1)  
PORT = 8443 # Server's port  
EXIT\_CMD = 'by by' # Command from server that signals client to exit  
# USER\_INPUT = 'please enter a command: ' # Unused global variable  
send\_again = False # Flag to indicate if the last message needs to be resent  
did\_send = False # Flag to indicate if a message was just sent (e.g., by receive\_client\_messages)  
last\_massage = '' # Stores the last message sent by this client  
IMAGE\_FILENAME = "received\_image.jpg" # Filename to save received images  
DEFAULT\_MSG = "DEFAULT" # Unused global variable  
msg = '' # Holds the current message received from server or to be sent  
  
# Configure logging at the beginning  
logging.basicConfig(  
 format='%(asctime)s | %(levelname)s | CLIENT2 - %(message)s', # Log identifier  
 datefmt='%m/%d/%Y %I:%M:%S %p',  
 level=logging.INFO,  
 filename='tls\_client2.log', # Specific log file for client2  
 filemode='w'  
)  
  
def connect\_to\_robo\_v3() -> serial.Serial | None:  
 *"""  
 Establishes a serial connection to the Arduino (robot).  
  
 Returns:  
 serial.Serial | None: The serial connection object if successful, None otherwise.  
 """* try:  
 arduino = serial.Serial(arduino\_port, baud\_rate, timeout=1)  
 logging.info(f"Attempting to connect to Arduino on {arduino\_port} at {baud\_rate} baud.")  
 time.sleep(2) # Wait for the connection to establish  
 if arduino.is\_open:  
 logging.info(f"Successfully connected to Arduino on {arduino\_port}.")  
 print(f"Connected to {arduino\_port}")  
 return arduino  
 else:  
 logging.error(f"Failed to open serial port {arduino\_port}.")  
 print(f"Failed to open serial port {arduino\_port}.")  
 return None  
 except serial.SerialException as se:  
 logging.error(f"Serial error connecting to Arduino on {arduino\_port}: {se}")  
 print(f"Serial error on {arduino\_port}: {se}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error connecting to Arduino: {e}")  
 print(f"Unexpected error connecting to Arduino: {e}")  
 return None  
  
def send\_robo\_v3(arduino: serial.Serial) -> None:  
 *"""  
 Sends an 'a' byte to the connected Arduino.  
 Its specific purpose in this client's main loop is not immediately clear  
 from the original `run\_client` logic if `send\_client\_message` always sends 'h'.  
  
 Args:  
 arduino (serial.Serial): The active serial connection object to the Arduino.  
 """* if arduino and arduino.is\_open:  
 try:  
 arduino.write(b'a')  
 logging.debug("Sent 'a' byte to Arduino (send\_robo\_v3).")  
 except serial.SerialException as se:  
 logging.error(f"Serial error sending 'a' to Arduino: {se}")  
 print(f"Serial error sending 'a': {se}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending 'a' to Arduino: {e}")  
 print(f"Unexpected error sending 'a': {e}")  
 else:  
 logging.warning("Arduino not connected or port not open. Cannot send 'a'.")  
 print("Arduino not connected. Cannot send 'a'.")  
  
def receive\_massage\_v3(arduino: serial.Serial) -> None:  
 *"""  
 Waits to receive a single byte from the Arduino.  
 This function blocks until a byte is received or timeout occurs.  
 The received byte is not returned or used by this function directly.  
 It seems to function as a blocking acknowledgment or signal wait.  
  
 Args:  
 arduino (serial.Serial): The active serial connection object to the Arduino.  
 """* if not (arduino and arduino.is\_open):  
 logging.warning("Arduino not connected or port not open. Cannot receive message.")  
 print("Arduino not connected. Cannot receive message.")  
 return  
 try:  
 logging.debug("Waiting for a byte from Arduino (receive\_massage\_v3)...")  
 while True:  
 temp = arduino.read(1)  
 if temp:  
 logging.debug(f"Received a byte from Arduino: {temp}")  
 break # Exit once a byte is received  
 # If temp is empty, loop continues due to serial timeout=1  
 logging.debug("Still waiting for byte from Arduino...")  
 except serial.SerialException as se:  
 logging.error(f"Serial error waiting for byte from Arduino: {se}")  
 print(f"Serial error receiving: {se}")  
 except Exception as e:  
 logging.error(f"Unexpected error waiting for byte from Arduino: {e}")  
 print(f"Unexpected error receiving: {e}")  
  
  
def receive\_image(my\_socket: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Receives an encrypted image file and its checksum from the server,  
 decrypts it, verifies the checksum, and saves the image.  
 (Identical in structure to client1's receive\_image)  
  
 Args:  
 my\_socket (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for decryption.  
 """* try:  
 logging.info("Attempting to receive image from server...")  
 encrypted\_data = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_data:  
 logging.error("Failed to receive encrypted image data from server.")  
 return  
  
 encrypted\_chk = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_chk:  
 logging.error("Failed to receive encrypted checksum from server.")  
 return  
   
 chk\_decoded = encrypted\_chk.decode()  
 logging.debug("Received encrypted image data and checksum.")  
  
 retries = 0  
 max\_retries = 3  
 while not protocol.is\_checksum(chk\_decoded, encrypted\_data) and retries < max\_retries:  
 logging.warning(f"Checksum mismatch for received image. Requesting resend (attempt {retries + 1}).")  
 protocol.encrypt\_and\_send\_msg("again".encode(), my\_socket, aes\_protocol)  
   
 encrypted\_data = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_data: logging.error(f"Failed to receive resent image (attempt {retries + 1})."); return  
 encrypted\_chk = protocol.recv\_encrypted\_msg(my\_socket, aes\_protocol)  
 if not encrypted\_chk: logging.error(f"Failed to receive resent checksum (attempt {retries + 1})."); return  
 chk\_decoded = encrypted\_chk.decode()  
 retries += 1  
   
 if not protocol.is\_checksum(chk\_decoded, encrypted\_data):  
 logging.error(f"Checksum mismatch after {max\_retries} retries. Aborting image receive.")  
 protocol.encrypt\_and\_send\_msg("error:checksum\_failed".encode(), my\_socket, aes\_protocol)  
 return  
  
 logging.info("Image checksum verified. Sending 'ok' to server.")  
 protocol.encrypt\_and\_send\_msg("ok".encode(), my\_socket, aes\_protocol)  
  
 try:  
 with open(IMAGE\_FILENAME, 'wb') as f:  
 f.write(encrypted\_data)  
 logging.info(f"Received and saved image as '{IMAGE\_FILENAME}'")  
 print(f"Received image: {IMAGE\_FILENAME}")  
 except IOError as e:  
 logging.error(f"IOError saving image '{IMAGE\_FILENAME}': {e}")  
 print(f"Error saving image: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error saving image '{IMAGE\_FILENAME}': {e}")  
 print(f"Error saving image: {e}")  
  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES protocol error during image reception: {e}")  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error for checksum during image reception: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error during image reception: {e}")  
  
  
def receive\_client\_messages(conn: socket.socket, aes\_protocol: MyAES) -> None:  
 *"""  
 Receives and processes messages from the server.  
 Handles checksums, "again" requests, "pic" commands, and general messages.  
 Updates global variables based on received messages.  
 (Identical in structure to client1's receive\_client\_messages)  
  
 Args:  
 conn (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for decryption.  
 """* global send\_again, did\_send, last\_massage, msg # msg is updated here  
 try:  
 encrypted\_msg\_data = protocol.recv\_encrypted\_msg(conn, aes\_protocol)  
 if not encrypted\_msg\_data:  
 logging.warning("No data received from server. Connection might be closed.")  
 msg = EXIT\_CMD # Assume server closed, trigger exit  
 return  
   
 full\_msg\_decoded = encrypted\_msg\_data.decode()  
 try:  
 chk, current\_msg\_content = full\_msg\_decoded.split('{', 1)  
 except ValueError:  
 logging.error(f"Received message from server in unexpected format: {full\_msg\_decoded}")  
 msg = EXIT\_CMD  
 return  
  
 chk = chk.lstrip("0")  
 logging.debug(f"Received from server: chk='{chk}', content='{current\_msg\_content}'")  
 msg = current\_msg\_content  
  
 if not current\_msg\_content:  
 logging.warning("Received empty message content from server.")  
 return  
  
 if current\_msg\_content == "again":  
 logging.info("Server requested resend ('again'). Preparing to resend last message.")  
 did\_send = True  
 if last\_massage:  
 protocol.encrypt\_and\_send\_msg(last\_massage.encode(), conn, aes\_protocol)  
 else:  
 logging.warning("Server requested 'again', but no 'last\_massage' to resend.")  
  
 elif not protocol.is\_checksum(chk, current\_msg\_content):  
 logging.warning("Checksum mismatch for message from server. Requesting server resend.")  
 protocol.encrypt\_and\_send\_msg("again".encode(), conn, aes\_protocol)  
 did\_send = True  
  
 elif current\_msg\_content == "pic":  
 logging.info("Received 'pic' command from server. Initiating image reception.")  
 receive\_image(conn, aes\_protocol)  
 else:  
 logging.info(f"Received from server: '{current\_msg\_content}'")  
 print(f"Client received: {current\_msg\_content}")  
  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES error receiving message from server: {e}")  
 msg = EXIT\_CMD  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error for message from server: {e}")  
 msg = EXIT\_CMD  
 except Exception as e:  
 logging.error(f"Unexpected error receiving message from server: {e}")  
 msg = EXIT\_CMD  
  
  
def send\_client\_message(conn: socket.socket, aes\_protocol: MyAES, arduino: serial.Serial | None) -> None:  
 *"""  
 Sends a hardcoded message 'h' to the server after waiting for an Arduino acknowledgment.  
 This typically serves as an acknowledgment from this client (robot).  
 Updates global variables.  
  
 Args:  
 conn (socket.socket): The connected socket to the server.  
 aes\_protocol (MyAES): The AES protocol object for encryption.  
 arduino (serial.Serial | None): The active serial connection to Arduino, or None.  
 """* global msg, last\_massage, did\_send  
   
 if did\_send: # If receive\_client\_messages already sent something  
 did\_send = False  
 logging.debug("send\_client\_message: did\_send was true, skipping send of 'h'.")  
 return  
  
 if not arduino:  
 logging.error("Arduino not available, cannot get acknowledgment before sending 'h'.")  
 # Proceeding to send 'h' anyway as per original logic, but this might be unintended.  
 # Consider if 'h' should only be sent if Arduino ack is received.  
 else:  
 receive\_massage\_v3(arduino) # Wait for Arduino ack (byte received and discarded)  
 logging.debug("Received acknowledgment from Arduino (or timeout).")  
  
 msg\_to\_send = 'h' # This client always sends 'h' as its data message  
 last\_massage = msg\_to\_send  
 msg = msg\_to\_send # Update global msg  
  
 logging.info(f"Sending acknowledgement '{msg\_to\_send}' to server.")  
 try:  
 protocol.encrypt\_and\_send\_msg(msg\_to\_send.encode(), conn, aes\_protocol)  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES error sending '{msg\_to\_send}' to server: {e}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending '{msg\_to\_send}' to server: {e}")  
  
def starting\_sql\_client(my\_socket: socket.socket) -> MyAES | None:  
 *"""  
 Performs the initial handshake with the server.  
 (Identical in structure to client1's starting\_sql\_client)  
  
 Args:  
 my\_socket (socket.socket): The connected socket to the server.  
  
 Returns:  
 MyAES | None: The established AES protocol object if successful, None otherwise.  
 """* try:  
 public\_key\_data = protocol.recv\_msg(my\_socket)  
 if not public\_key\_data:  
 logging.error("Failed to receive public RSA key from server.")  
 return None  
 logging.info(f"Received public RSA key from server: {public\_key\_data.decode()[:30]}...")  
 print("Received the server's public key.")  
  
 aes\_protocol\_instance = MyAES()  
 encrypted\_aes\_key = MyRSA.encrypt\_with\_key(aes\_protocol\_instance.export\_key(), public\_key\_data)  
   
 logging.info("Sending encrypted AES key to server.")  
 protocol.send\_msg(encrypted\_aes\_key, my\_socket)  
 print("Sent the encrypted AES key.")  
  
 user\_name = "iftach" # Hardcoded  
 password = "1234" # Hardcoded  
 auth\_msg\_str = f"{user\_name}!{password}"  
 logging.info(f"Sending authentication credentials for user '{user\_name}'.")  
 protocol.encrypt\_and\_send\_msg(auth\_msg\_str.encode(), my\_socket, aes\_protocol\_instance)  
   
 receive\_client\_messages(my\_socket, aes\_protocol\_instance) # Updates global `msg`  
 logging.info(f"Authentication response from server: '{msg}'")  
  
 if "connected" in msg:  
 return aes\_protocol\_instance  
 else:  
 logging.error(f"Authentication failed. Server response: {msg}")  
 return None  
  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or crypto protocol error during initial handshake: {e}")  
 return None  
 except Exception as e:  
 logging.error(f"Unexpected error during initial handshake: {e}")  
 return None  
  
  
def send\_stop\_num\_to\_arduino(num\_str: str, arduino: serial.Serial | None) -> None:  
 *"""  
 Converts a number string to an integer and sends it as a single byte to the Arduino.  
 This is likely a command or data point for the Arduino.  
  
 Args:  
 num\_str (str): The string representation of the number to send.  
 arduino (serial.Serial | None): The active serial connection to Arduino, or None.  
 """* if not (arduino and arduino.is\_open):  
 logging.warning("Arduino not connected or port not open. Cannot send number.")  
 print("Arduino not connected. Cannot send number.")  
 return  
 try:  
 # The original code printed global `msg` here, which might be confusing  
 # if `num\_str` is different. Let's log `num\_str`.  
 logging.debug(f"Attempting to send number string '{num\_str}' to Arduino.")  
 print(f"Sending to Arduino: {num\_str}") # Original code printed global `msg`  
   
 stop\_number = int(num\_str)  
 if 0 <= stop\_number <= 255: # Ensure it fits in one byte  
 arduino.write(bytes([stop\_number]))  
 logging.info(f"Sent number {stop\_number} (as byte) to Arduino.")  
 else:  
 logging.error(f"Number {stop\_number} is out of byte range (0-255). Cannot send to Arduino.")  
 print(f"Number {stop\_number} is out of byte range.")  
 except ValueError:  
 logging.error(f"Cannot convert '{num\_str}' to an integer to send to Arduino.")  
 print(f"Error: '{num\_str}' is not a valid number.")  
 except serial.SerialException as se:  
 logging.error(f"Serial error sending number to Arduino: {se}")  
 print(f"Serial error sending number: {se}")  
 except Exception as e:  
 logging.error(f"Unexpected error sending number to Arduino: {e}")  
 print(f"Unexpected error sending number: {e}")  
  
  
def run\_client() -> None:  
 *"""  
 Main function for client2.  
 Connects to server, authenticates, connects to Arduino.  
 Enters a loop to receive commands (numbers) from the server, send them to Arduino,  
 and send an acknowledgment ('h') back to the server.  
 """* global msg # Used for EXIT\_CMD check and by send\_stop\_num\_to\_arduino via print  
 my\_socket = socket.socket()  
 aes\_protocol\_instance = None  
 arduino\_conn = None  
  
 logging.info("Client2 starting...")  
  
 try:  
 logging.info(f"Attempting to connect to server at {HOST\_NAME}:{PORT}...")  
 my\_socket.connect((HOST\_NAME, PORT))  
 print(f"Connected to server: {HOST\_NAME}:{PORT}")  
 logging.info(f"Successfully connected to server: {HOST\_NAME}:{PORT}")  
  
 aes\_protocol\_instance = starting\_sql\_client(my\_socket)  
 if not aes\_protocol\_instance:  
 logging.critical("Failed to establish secure session with server. Exiting.")  
 print("Could not establish secure session. Exiting.")  
 return  
   
 # Server might send an initial message after auth, handled by starting\_sql\_client's call  
 # to receive\_client\_messages. Global `msg` has that response.  
  
 arduino\_conn = connect\_to\_robo\_v3()  
 if not arduino\_conn:  
 logging.error("Failed to connect to Arduino. Client will attempt to run but Arduino interaction will fail.")  
 # Client will proceed, but send\_stop\_num\_to\_arduino and receive\_massage\_v3 will log errors.  
  
 logging.info("Entering main communication loop...")  
 while True:  
 # Receive first command/number from server  
 receive\_client\_messages(my\_socket, aes\_protocol\_instance) # global `msg` now holds server data  
 if msg == EXIT\_CMD: break  
 send\_stop\_num\_to\_arduino(msg, arduino\_conn) # Send it to Arduino  
  
 # Receive second command/number from server  
 receive\_client\_messages(my\_socket, aes\_protocol\_instance) # global `msg` updated  
 if msg == EXIT\_CMD: break  
 send\_stop\_num\_to\_arduino(msg, arduino\_conn) # Send it to Arduino  
  
 # Send acknowledgment 'h' to server  
 send\_client\_message(my\_socket, aes\_protocol\_instance, arduino\_conn) # This will set global `msg` to 'h'  
  
 if msg == EXIT\_CMD: # Check if the 'h' somehow became EXIT\_CMD (unlikely here)  
 logging.info(f"Received/Sent EXIT\_CMD ('{EXIT\_CMD}'). Exiting client.")  
 print("Server/Client requested exit. Closing client.")  
 break  
 # Add a small delay if necessary: time.sleep(0.1)  
  
 except socket.error as sock\_err:  
 logging.error(f"Socket error in client: {sock\_err}")  
 print(f"Socket error: {sock\_err}")  
 except serial.SerialException as ser\_err:  
 logging.error(f"Serial communication error in client: {ser\_err}")  
 print(f"Serial error: {ser\_err}")  
 except KeyboardInterrupt:  
 logging.info("Client shutdown initiated by user (KeyboardInterrupt).")  
 print("\nClient exiting...")  
 except Exception as e:  
 logging.error(f"An unexpected error occurred in client: {e}", exc\_info=True)  
 print(f"An error occurred: {e}")  
 finally:  
 if arduino\_conn and arduino\_conn.is\_open:  
 try:  
 arduino\_conn.close()  
 logging.info("Arduino serial connection closed.")  
 print("Arduino connection closed.")  
 except Exception as e:  
 logging.error(f"Error closing Arduino connection: {e}")  
 if my\_socket:  
 try:  
 my\_socket.close()  
 logging.info("Socket connection to server closed.")  
 print("Socket closed.")  
 except Exception as e:  
 logging.error(f"Error closing socket: {e}")  
 logging.info("Client2 finished.")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Logging is configured at the top of the file  
 run\_client()

## client3

*"""  
Author: Iftach Kasorla  
Date: 3/6/25  
Description: GUI TLS Client with Login and Interactive Interface.  
Allows a user to connect to the server, authenticate, and make decisions  
based on messages received from the server.  
"""*import tkinter as tk  
from tkinter import messagebox, ttk  
import logging  
import socket  
import protocol # Assuming protocol.py exists  
from aes import MyAES # Assuming aes.py and MyAES class exist  
from rsa import MyRSA # Assuming rsa.py and MyRSA class exist  
import threading  
import queue  
  
HOST\_NAME = '127.0.0.1' # Server's hostname or IP address  
PORT = 8443 # Server's port  
EXIT\_CMD = 'by by' # Command from server that might signal client to exit (not explicitly handled here)  
IMAGE\_FILENAME = "received\_image.jpg" # Not used in this client's current logic  
MSG\_DISPLAY\_TIME = 5000 # 5 seconds in milliseconds for how long a message might be highlighted or screen stays  
  
class ClientGUI:  
 *"""  
 Manages the GUI application for the secure client, including login,  
 displaying server messages, and sending user decisions.  
 """* def \_\_init\_\_(self):  
 *"""  
 Initializes the main Tkinter window, network components, GUI frames,  
 and starts the message queue processor.  
 """* self.root = tk.Tk()  
 self.root.title("Secure Client GUI") # More descriptive title  
 self.root.geometry("450x350") # Slightly larger for better layout  
  
 logging.info("Initializing ClientGUI application.")  
  
 # Network components  
 self.socket: socket.socket | None = None  
 self.aes\_protocol: MyAES | None = None  
 self.message\_queue = queue.Queue() # Thread-safe queue for communication between network thread and GUI thread  
  
 # GUI frames  
 self.frames = {}  
 self.\_create\_frames()  
 self.show\_frame("LoginScreen")  
  
 # Handle window close event  
 self.root.protocol("WM\_DELETE\_WINDOW", self.\_on\_closing)  
  
 # Start queue handler to process messages from network threads in the main GUI thread  
 self.root.after(100, self.process\_queue)  
 logging.info("ClientGUI initialized and queue processor started.")  
  
 def \_create\_frames(self) -> None:  
 *"""  
 Creates all the different frames (screens) used in the application  
 and stores them in the self.frames dictionary.  
 """* logging.debug("Creating GUI frames.")  
 # Login Screen  
 login\_frame = ttk.Frame(self.root, padding="10")  
 self.\_setup\_login\_screen(login\_frame)  
 self.frames["LoginScreen"] = login\_frame  
  
 # Main Screen  
 main\_frame = ttk.Frame(self.root, padding="10")  
 self.\_setup\_main\_screen(main\_frame)  
 self.frames["MainScreen"] = main\_frame  
  
 # Wait Screen  
 wait\_frame = ttk.Frame(self.root, padding="10")  
 self.\_setup\_wait\_screen(wait\_frame)  
 self.frames["WaitScreen"] = wait\_frame  
  
 # Decision Screen  
 decision\_frame = ttk.Frame(self.root, padding="10")  
 self.\_setup\_decision\_screen(decision\_frame)  
 self.frames["DecisionScreen"] = decision\_frame  
  
 for frame\_name, frame\_obj in self.frames.items():  
 frame\_obj.grid(row=0, column=0, sticky="nsew")  
 logging.debug(f"Frame '{frame\_name}' created and gridded.")  
 self.root.grid\_rowconfigure(0, weight=1)  
 self.root.grid\_columnconfigure(0, weight=1)  
  
  
 def \_setup\_login\_screen(self, frame: ttk.Frame) -> None:  
 *"""Sets up the widgets for the login screen."""* ttk.Label(frame, text="Username:").pack(pady=5)  
 self.username\_entry = ttk.Entry(frame, width=30)  
 self.username\_entry.pack(pady=5)  
 self.username\_entry.focus\_set() # Set focus to username entry  
  
 ttk.Label(frame, text="Password:").pack(pady=5)  
 self.password\_entry = ttk.Entry(frame, show="\*", width=30)  
 self.password\_entry.pack(pady=5)  
  
 ttk.Button(frame, text="Connect", command=self.\_attempt\_login, style="Accent.TButton").pack(pady=20)  
 # Allow Enter key to trigger login from password field  
 self.password\_entry.bind("<Return>", lambda event: self.\_attempt\_login())  
  
  
 def \_setup\_main\_screen(self, frame: ttk.Frame) -> None:  
 *"""Sets up the widgets for the main control screen."""* ttk.Label(frame, text="Main Control Panel", font=("Arial", 16)).pack(pady=20)  
 ttk.Button(frame, text="START System Process", command=self.\_send\_start, style="Accent.TButton").pack(pady=20, ipadx=10, ipady=5)  
 ttk.Button(frame, text="Logout / Disconnect", command=self.\_logout).pack(pady=10)  
  
  
 def \_setup\_wait\_screen(self, frame: ttk.Frame) -> None:  
 *"""Sets up the widgets for the waiting screen."""* self.status\_label = ttk.Label(frame, text="Waiting for server message...", font=("Arial", 12), anchor="center")  
 self.status\_label.pack(pady=50, expand=True)  
  
  
 def \_setup\_decision\_screen(self, frame: ttk.Frame) -> None:  
 *"""Sets up the widgets for the decision-making screen."""* ttk.Label(frame, text="Server Request:", font=("Arial", 14)).pack(pady=10)  
 self.request\_label = ttk.Label(frame, text="", font=("Arial", 12, "bold"), foreground="blue", wraplength=380)  
 self.request\_label.pack(pady=10)  
  
 button\_frame = ttk.Frame(frame)  
 ttk.Button(button\_frame, text="Send Killer Robot",  
 command=lambda: self.\_send\_decision("kill"), style="Danger.TButton").pack(side=tk.LEFT, padx=20, ipadx=5, ipady=5)  
 ttk.Button(button\_frame, text="Don't Send",  
 command=lambda: self.\_send\_decision("no"), style="Success.TButton").pack(side=tk.RIGHT, padx=20, ipadx=5, ipady=5)  
 button\_frame.pack(pady=20)  
  
 def show\_frame(self, name: str) -> None:  
 *"""  
 Raises the specified frame (screen) to be visible.  
  
 Args:  
 name (str): The key name of the frame to show.  
 """* try:  
 if name in self.frames:  
 frame = self.frames[name]  
 frame.tkraise()  
 logging.info(f"Showing frame: {name}")  
 else:  
 logging.warning(f"Attempted to show non-existent frame: {name}")  
 except Exception as e:  
 logging.error(f"Error showing frame '{name}': {e}")  
  
  
 def \_attempt\_login(self) -> None:  
 *"""  
 Retrieves username and password from entry fields and starts  
 the server connection process in a new thread.  
 """* username = self.username\_entry.get()  
 password = self.password\_entry.get()  
 logging.info(f"Login attempt for username: {username}")  
  
 if not username or not password:  
 messagebox.showwarning("Login Error", "Username and password cannot be empty.")  
 logging.warning("Login attempt with empty username or password.")  
 return  
  
 # Disable button to prevent multiple clicks while connecting  
 # This would require access to the button widget, e.g., self.login\_button  
 # For simplicity, this is omitted here but is good practice.  
  
 threading.Thread(target=self.\_connect\_to\_server, args=(username, password), daemon=True).start()  
  
  
 def \_connect\_to\_server(self, username: str, password: str) -> None:  
 *"""  
 Connects to the server, performs RSA/AES key exchange, and authenticates.  
 This method is designed to run in a separate thread.  
 Puts results or errors into the message\_queue.  
  
 Args:  
 username (str): The username for authentication.  
 password (str): The password for authentication.  
 """* logging.info(f"Thread '{threading.current\_thread().name}': Attempting to connect to server {HOST\_NAME}:{PORT}.")  
 try:  
 self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.socket.settimeout(10) # 10-second timeout for connection  
 self.socket.connect((HOST\_NAME, PORT))  
 logging.info("Socket connected to server.")  
  
 # RSA Key Exchange  
 logging.debug("Receiving public RSA key from server...")  
 public\_key = protocol.recv\_msg(self.socket)  
 if not public\_key:  
 raise ConnectionError("Failed to receive public RSA key from server.")  
 logging.info("Public RSA key received.")  
  
 self.aes\_protocol = MyAES() # Client generates its AES key  
 encrypted\_aes\_key = MyRSA.encrypt\_with\_key(self.aes\_protocol.export\_key(), public\_key)  
 logging.debug("Sending encrypted AES key to server...")  
 protocol.send\_msg(encrypted\_aes\_key, self.socket)  
 logging.info("Encrypted AES key sent.")  
  
 # Send credentials  
 auth\_msg\_str = f"{username}!{password}"  
 logging.debug(f"Sending encrypted credentials for user '{username}'...")  
 protocol.encrypt\_and\_send\_msg(auth\_msg\_str.encode(), self.socket, self.aes\_protocol)  
 logging.info("Credentials sent.")  
  
 # Verify authentication  
 logging.debug("Waiting for authentication response from server...")  
 response\_data = protocol.recv\_encrypted\_msg(self.socket, self.aes\_protocol)  
 if not response\_data:  
 raise ConnectionError("No authentication response from server.")  
   
 response\_str = response\_data.decode()  
 # Assuming server sends "checksum{message}"  
 \_, auth\_response\_content = response\_str.split('{', 1)  
 logging.info(f"Authentication response received: '{auth\_response\_content}'")  
  
 if "connected" in auth\_response\_content:  
 logging.info(f"User '{username}' authenticated successfully.")  
 self.message\_queue.put(("show\_main", None))  
 else:  
 logging.warning(f"Authentication failed for user '{username}': {auth\_response\_content}")  
 self.message\_queue.put(("error", f"Invalid credentials or server error: {auth\_response\_content}"))  
 if self.socket: self.socket.close(); self.socket = None # Close on auth failure  
 self.aes\_protocol = None  
  
 except socket.timeout:  
 logging.error("Connection to server timed out.")  
 self.message\_queue.put(("error", "Connection failed: Timed out"))  
 except socket.error as se:  
 logging.error(f"Socket error during connection/handshake: {se}")  
 self.message\_queue.put(("error", f"Connection failed: {se}"))  
 except ConnectionError as ce: # Custom for logical connection steps  
 logging.error(f"Connection process error: {ce}")  
 self.message\_queue.put(("error", f"Connection process error: {ce}"))  
 except Exception as e:  
 logging.error(f"Unexpected error in \_connect\_to\_server: {e}", exc\_info=True)  
 self.message\_queue.put(("error", f"Connection failed: An unexpected error occurred ({type(e).\_\_name\_\_})"))  
 finally:  
 # If socket exists but AES protocol was not set (meaning full success wasn't reached)  
 if self.socket and not self.aes\_protocol:  
 try:  
 self.socket.close()  
 self.socket = None  
 except Exception as e\_close:  
 logging.error(f"Error closing socket in \_connect\_to\_server finally block: {e\_close}")  
  
  
 def \_send\_start(self) -> None:  
 *"""  
 Sends a "start" command to the server and transitions to the WaitScreen.  
 Initiates message receiving in a new thread.  
 """* if self.socket and self.aes\_protocol:  
 try:  
 logging.info("Sending 'start' command to server.")  
 protocol.encrypt\_and\_send\_msg("start".encode(), self.socket, self.aes\_protocol)  
 self.show\_frame("WaitScreen")  
 threading.Thread(target=self.\_receive\_messages, daemon=True).start()  
 except (socket.error, AttributeError) as e: # AttributeError if socket/aes\_protocol is None  
 logging.error(f"Error sending 'start' command: {e}")  
 self.message\_queue.put(("error", f"Failed to send 'start': {e}. Please reconnect."))  
 except Exception as e:  
 logging.error(f"Unexpected error in \_send\_start: {e}", exc\_info=True)  
 self.message\_queue.put(("error", f"Unexpected error: {e}"))  
 else:  
 logging.warning("Attempted to send 'start' but not connected.")  
 self.message\_queue.put(("error", "Not connected to server. Please login again."))  
  
  
 def \_receive\_messages(self) -> None:  
 *"""  
 Continuously listens for messages from the server in a loop (original broke after one).  
 Puts received messages or errors into the message\_queue.  
 This method is designed to run in a separate thread.  
 """* logging.info(f"Thread '{threading.current\_thread().name}': Starting to listen for server messages.")  
 # The original code had `while True` but then `break`.  
 # If it's meant to receive only one message then transition, the `break` is fine.  
 # If it's meant to be a continuous listener until an error or explicit stop, the `break` should be conditional.  
 # For this server logic, it seems one message comes, then user decision, then another wait.  
 # So, receiving one message then stopping the thread is appropriate.  
 if self.socket and self.aes\_protocol:  
 try:  
 logging.debug("Waiting to receive encrypted message from server...")  
 encrypted\_msg\_data = protocol.recv\_encrypted\_msg(self.socket, self.aes\_protocol)  
 if not encrypted\_msg\_data:  
 # This indicates server closed connection or an issue with recv\_encrypted\_msg  
 logging.warning("No data received from server in \_receive\_messages, server might have closed connection.")  
 self.message\_queue.put(("error", "Connection lost with server."))  
 return # Exit thread  
  
 msg\_decoded = encrypted\_msg\_data.decode()  
 # Assuming server sends "checksum{message}"  
 \_, msg\_content = msg\_decoded.split('{', 1)  
 logging.info(f"Message received from server: '{msg\_content}'")  
 self.message\_queue.put(("show\_message", msg\_content))  
 # Original had break here, which is fine if only one message is expected before next user action.  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Socket or AES error in \_receive\_messages: {e}")  
 self.message\_queue.put(("error", f"Network error: {e}. Connection may be lost."))  
 except UnicodeDecodeError as e:  
 logging.error(f"Unicode decode error in \_receive\_messages: {e}")  
 self.message\_queue.put(("error", "Received undecodable message from server."))  
 except Exception as e:  
 logging.error(f"Unexpected error in \_receive\_messages: {e}", exc\_info=True)  
 self.message\_queue.put(("error", f"Error receiving data: {e}"))  
 else:  
 logging.warning("\_receive\_messages called but not connected.")  
 self.message\_queue.put(("error", "Not connected. Cannot receive messages."))  
  
  
 def \_send\_decision(self, decision: str) -> None:  
 *"""  
 Sends the user's decision ("kill" or "no") to the server.  
 Transitions to WaitScreen and starts listening for more messages.  
  
 Args:  
 decision (str): The decision made by the user.  
 """* if self.socket and self.aes\_protocol:  
 try:  
 logging.info(f"Sending decision '{decision}' to server.")  
 protocol.encrypt\_and\_send\_msg(decision.encode(), self.socket, self.aes\_protocol)  
 self.show\_frame("WaitScreen")  
 # Start a new thread to listen for the next server message  
 threading.Thread(target=self.\_receive\_messages, daemon=True).start()  
 except (socket.error, AttributeError) as e:  
 logging.error(f"Error sending decision '{decision}': {e}")  
 self.message\_queue.put(("error", f"Failed to send decision: {e}. Please reconnect."))  
 except Exception as e:  
 logging.error(f"Unexpected error in \_send\_decision: {e}", exc\_info=True)  
 self.message\_queue.put(("error", f"Unexpected error: {e}"))  
 else:  
 logging.warning(f"Attempted to send decision '{decision}' but not connected.")  
 self.message\_queue.put(("error", "Not connected to server. Please login again."))  
  
  
 def process\_queue(self) -> None:  
 *"""  
 Periodically checks the message\_queue for messages from network threads  
 and updates the GUI accordingly. This runs in the main Tkinter thread.  
 """* try:  
 # Process all available messages in the queue without blocking  
 while True:  
 msg\_type, content = self.message\_queue.get\_nowait()  
 logging.debug(f"Processing queue message: type='{msg\_type}', content='{content}'")  
 if msg\_type == "show\_main":  
 self.show\_frame("MainScreen")  
 elif msg\_type == "show\_message":  
 self.\_display\_message(content) # This will show DecisionScreen  
 elif msg\_type == "error":  
 logging.error(f"Displaying error from queue: {content}")  
 messagebox.showerror("Error", content)  
 # Attempt to clean up and reset  
 if self.socket:  
 try:  
 self.socket.close()  
 except Exception as e\_close:  
 logging.error(f"Error closing socket in process\_queue after error: {e\_close}")  
 self.socket = None  
 self.aes\_protocol = None  
 self.show\_frame("LoginScreen")  
 self.message\_queue.task\_done() # Signal that a message was processed  
 except queue.Empty:  
 pass # No messages in the queue, which is normal  
 except Exception as e:  
 logging.error(f"Unexpected error in process\_queue: {e}", exc\_info=True)  
 finally:  
 # Reschedule this method to run again after 100ms  
 self.root.after(100, self.process\_queue)  
  
  
 def \_display\_message(self, message: str) -> None:  
 *"""  
 Displays the received message on the DecisionScreen.  
 The original code had a timer to re-show the DecisionScreen, which is  
 redundant if it's already shown.  
  
 Args:  
 message (str): The message from the server to display.  
 """* try:  
 self.show\_frame("DecisionScreen")  
 self.request\_label.config(text=message)  
 logging.info(f"Displayed message on DecisionScreen: '{message}'")  
 # The original self.root.after(MSG\_DISPLAY\_TIME, self.show\_frame, "DecisionScreen")  
 # would just re-raise the same screen. If the intent was to timeout the decision,  
 # it should transition to a different screen or take a default action.  
 # For now, removing the confusing `after` call here. If a timeout is needed:  
 # self.root.after(MSG\_DISPLAY\_TIME, self.\_handle\_decision\_timeout)  
 except Exception as e:  
 logging.error(f"Error displaying message '{message}': {e}")  
  
 def \_logout(self) -> None:  
 *"""Handles user logout: closes socket and returns to login screen."""* logging.info("Logout requested by user.")  
 if self.socket:  
 try:  
 # Optionally send a "disconnect" message to the server  
 # protocol.encrypt\_and\_send\_msg("disconnecting".encode(), self.socket, self.aes\_protocol)  
 self.socket.close()  
 logging.info("Socket closed during logout.")  
 except Exception as e:  
 logging.error(f"Error closing socket during logout: {e}")  
 self.socket = None  
 self.aes\_protocol = None  
 self.username\_entry.delete(0, tk.END) # Clear username  
 self.password\_entry.delete(0, tk.END) # Clear password  
 self.show\_frame("LoginScreen")  
 logging.info("User logged out, returned to LoginScreen.")  
  
  
 def \_on\_closing(self) -> None:  
 *"""Handles the event when the main window is closed."""* logging.info("Main window closing event triggered.")  
 if messagebox.askokcancel("Quit", "Do you want to quit the Secure Client?"):  
 logging.info("User confirmed quit. Shutting down.")  
 if self.socket:  
 try:  
 # Consider sending a graceful disconnect message to the server here if protocol supports it  
 # protocol.encrypt\_and\_send\_msg(EXIT\_CMD.encode(), self.socket, self.aes\_protocol)  
 self.socket.close()  
 logging.info("Socket closed on application exit.")  
 except Exception as e:  
 logging.error(f"Error closing socket on application exit: {e}")  
 self.root.destroy()  
 logging.info("Application window destroyed. Exiting.")  
 else:  
 logging.info("User cancelled quit.")  
  
  
 def run(self) -> None:  
 *"""Starts the Tkinter main event loop."""* logging.info("Starting ClientGUI main loop.")  
 # Apply some ttk styles for a slightly more modern look  
 style = ttk.Style()  
 style.configure("Accent.TButton", foreground="white", background="#0078D7") # Example accent button  
 style.configure("Danger.TButton", foreground="white", background="#DC3545") # Example danger button  
 style.configure("Success.TButton", foreground="white", background="#28A745") # Example success button  
  
 self.root.mainloop()  
 logging.info("ClientGUI main loop exited.")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 logging.basicConfig(  
 format='%(asctime)s | %(levelname)s | GUI\_CLIENT - %(message)s', # Log identifier  
 datefmt='%m/%d/%Y %I:%M:%S %p',  
 level=logging.INFO, # Use logging.DEBUG for more verbose output during development  
 filename='gui\_client.log',  
 filemode='w'  
 )  
 try:  
 client\_app = ClientGUI()  
 client\_app.run()  
 except Exception as e:  
 logging.critical(f"Critical error starting GUI client: {e}", exc\_info=True)  
 # Fallback if GUI itself fails to initialize, print to console  
 print(f"A critical error occurred: {e}")

## Protocol

*"""  
Author: Iftach Kasorla  
Date: 3/6/25  
Description: communication protocol. the protocol contains a 3 digit length (send as a string) followed by the message  
"""*import socket  
from crypto\_base import CryptoBase  
  
LEN\_LEN = 1000  
  
  
def calculate\_checksum(data: bytes) -> str:  
 *"""  
 Calculates a simple checksum for a given byte string.  
  
 Args:  
 data: The byte string to calculate the checksum for.  
  
 Returns:  
 The calculated checksum as an integer.  
 """* checksum = 0  
 for byte in data:  
 checksum = (checksum + byte) & 0xFF # Keep within 8 bits  
  
 return str(checksum)  
  
  
def encrypt\_and\_send\_msg(msg: bytes, comm: socket.socket, enc\_dec: CryptoBase) -> None:  
 *"""  
 send the passed message over the socket within the described protocol* ***:param*** *msg: the message to send* ***:param*** *comm: the socket to send the message over* ***:param*** *enc\_dec: the rsa protocol instance* ***:return****: None  
 """* checksum = calculate\_checksum(msg)  
 msg = msg.decode()  
 msg = checksum + '{' + msg  
 cipher\_msg = enc\_dec.encrypt(msg.encode())  
  
 return send\_msg(cipher\_msg, comm)  
  
  
def send\_msg(msg: bytes, comm: socket.socket) -> None:  
 *"""  
 send the passed message over the socket within the described protocol* ***:param*** *msg: the message to send* ***:param*** *comm: the socket to send the message over* ***:return****: None  
 """* msg\_len = len(msg)  
  
 actual\_lmsg = str(msg\_len).zfill(LEN\_LEN).encode() + msg  
 comm.sendall(actual\_lmsg)  
  
  
def recv\_encrypted\_msg(comm: socket.socket, dec: CryptoBase) -> bytes:  
 *"""  
 receive a message from the socket within the described protocol and decrypt it with the passed rsa\_protocol* ***:param*** *comm: the socket to receive the message* ***:param*** *dec: the decrypt object to decrypt with* ***:return****: the received message as bytes  
 """* msg = recv\_msg(comm)  
 if msg != b'':  
 msg = dec.decrypt(msg)  
 return msg  
  
  
def recv\_msg(comm: socket.socket) -> bytes:  
 *"""  
 receive a message from the socket within the described protocol* ***:param*** *comm: the socket to receive the message* ***:return****: the received message as bytes  
 """* msg\_len = ''  
 while len(msg\_len) < LEN\_LEN:  
 tmp = comm.recv(LEN\_LEN - len(msg\_len))  
 if not tmp:  
 msg\_len = 0  
 break  
 msg\_len += tmp.decode()  
 msg = b''  
 while len(msg) < int(msg\_len):  
 tmp = comm.recv(int(msg\_len) - len(msg))  
 if not tmp:  
 msg = b''  
 break  
 msg += tmp  
 return msg  
  
  
def is\_checksum(chk: str, data) -> bool:  
 if isinstance(data, str):  
 data = data.encode()  
 if chk == calculate\_checksum(data):  
 return True  
 else:  
 return False

## Aes

*"""  
Author: Iftach Kasorla  
Date: 3/6/25  
Description: a simple AES encryption/decryption class  
"""*from Crypto.Cipher import AES  
from Crypto.Random import get\_random\_bytes  
import base64  
from crypto\_base import CryptoBase  
  
  
class MyAES(CryptoBase):  
 KEY\_SIZE = 32  
  
 def \_\_init\_\_(self, key: bytes = None):  
 self.key = key  
 if key is None:  
 self.key = get\_random\_bytes(self.KEY\_SIZE)  
 else:  
 if len(key) not in [16, 24, 32]:  
 raise ValueError("AES key must be 16, 24, or 32 bytes long")  
  
 def export\_key(self):  
 *"""  
 export the encryption key as a byte string* ***:return****: the key as a byte string  
 """* return self.key  
  
 def encrypt(self, text: bytes) -> bytes:  
 *"""  
 encrypt the passed message* ***:param*** *text: the message to be encrypted* ***:return****: the encrypted message bytes  
 """* cipher = AES.new(self.key, AES.MODE\_EAX)  
 nonce = cipher.nonce  
 ciphertext, tag = cipher.encrypt\_and\_digest(text)  
 return nonce + ciphertext  
  
 def decrypt(self, cipher\_text: bytes,) -> bytes:  
 *"""  
 decrypt the passed ciphertext* ***:param*** *cipher\_text: the cipher to decrypt* ***:return****: the decrypted message bytes  
 """* nonce = cipher\_text[:16]  
 cipher\_text = cipher\_text[16:]  
 cipher = AES.new(self.key, AES.MODE\_EAX, nonce=nonce)  
 return cipher.decrypt(cipher\_text)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 aes = MyAES()  
 msg = input("please input a message to be encrypted: ")  
 encrypted = aes.encrypt(msg.encode())  
 print("base64 encrypt: ", base64.b64encode(encrypted).decode())  
 print("the decrypted message is: ", aes.decrypt(encrypted).decode())  
 print('------------------------------------')  
 print('now with an external key')  
 aes = MyAES(get\_random\_bytes(16))  
 msg = input("please input a message to be encrypted: ")  
 encrypted = aes.encrypt(msg.encode())  
 print("base64 encrypt: ", base64.b64encode(encrypted).decode())  
 print("the decrypted message is: ", aes.decrypt(encrypted).decode())

## RSA

*"""  
Author: Iftach Kasorla  
Date: 3/6/25  
Description: a simple RSA encryption/decryption class  
"""*from Crypto.PublicKey import RSA  
from Crypto.Cipher import PKCS1\_OAEP  
import base64  
from crypto\_base import CryptoBase  
  
  
class MyRSA(CryptoBase):  
 KEY\_LEN = 2048  
  
 def \_\_init\_\_(self):  
 *"""  
 initialize the RSA key  
 """* self.key = RSA.generate(self.KEY\_LEN)  
  
 @staticmethod  
 def encrypt\_with\_key(message: bytes, p\_key: bytes) -> bytes:  
 *"""  
 encrypt the passed message with the passed key* ***:param*** *message: the message to be encrypted* ***:param*** *p\_key: the key used to encrypt the message* ***:return****: the encrypted message bytes  
 """* key = RSA.import\_key(p\_key)  
 cipher = PKCS1\_OAEP.new(key)  
 return cipher.encrypt(message)  
  
 def encrypt(self, message: bytes) -> bytes:  
 *"""  
 encrypt the passed message* ***:param*** *message: the message to be encrypted* ***:return****: the encrypted message bytes  
 """* cipher = PKCS1\_OAEP.new(self.key.publickey())  
 return cipher.encrypt(message)  
  
 def decrypt(self, cipher\_text: bytes) -> bytes:  
 *"""  
 decrypt the passed ciphertext* ***:param*** *cipher\_text: the cipher to decrypt* ***:return****: the decrypted message bytes  
 """* cipher = PKCS1\_OAEP.new(self.key)  
 return cipher.decrypt(cipher\_text)  
  
 def export\_public\_key(self) -> bytes:  
 *"""  
 export the public key as bytes* ***:return****:  
 """* return self.key.publickey().exportKey()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 rsa = MyRSA()  
 msg = input("please input a message to be encrypted: ")  
 encrypted = rsa.encrypt(msg.encode())  
 print("base64 encrypt: ", base64.b64encode(encrypted))  
 print("the decrypted message is: ", rsa.decrypt(encrypted))

## CryptoBase

*"""  
Author: Nir Dweck  
Date: 24/10/24  
Description: an interface for encryption and decryption  
"""*import abc  
  
  
class CryptoBase:  
 *"""  
 a base class for the cryptography module  
 """* @abc.abstractmethod  
 def encrypt(self, message: bytes) -> bytes:  
 pass  
  
 @abc.abstractmethod  
 def decrypt(self, cipher\_text: bytes) -> bytes:  
 pass

## Arduino1

//www.elegoo.com

#include <Servo.h>

Servo myservo;

//Line Tracking IO define

#define LT\_R !digitalRead(10)

#define LT\_M !digitalRead(4)

#define LT\_L !digitalRead(2)

#define ENA 5

#define ENB 6

#define IN1 7

#define IN2 8

#define IN3 9

#define IN4 11

int rightDistance = 0, leftDistance = 0, middleDistance = 0;

#define distance\_cm 20

int Echo = A4;

int Trig = A5;

char stop\_num = 0;

char nothing = 255;

bool is\_white = false;

bool next\_stop = false;

bool stop\_at\_two = false;

#define carSpeed 120

void forward(){

analogWrite(ENA, carSpeed);

analogWrite(ENB, carSpeed);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

}

void back(){

analogWrite(ENA, carSpeed);

analogWrite(ENB, carSpeed);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

}

void left(){

analogWrite(ENA, carSpeed);

analogWrite(ENB, carSpeed);

digitalWrite(IN1, LOW);

digitalWrite(IN2, HIGH);

digitalWrite(IN3, LOW);

digitalWrite(IN4, HIGH);

}

void right(){

analogWrite(ENA, carSpeed);

analogWrite(ENB, carSpeed);

digitalWrite(IN1, HIGH);

digitalWrite(IN2, LOW);

digitalWrite(IN3, HIGH);

digitalWrite(IN4, LOW);

}

void stop(){

digitalWrite(ENA, LOW);

digitalWrite(ENB, LOW);

}

int Distance\_test() {

digitalWrite(Trig, LOW);

delayMicroseconds(2);

digitalWrite(Trig, HIGH);

delayMicroseconds(20);

digitalWrite(Trig, LOW);

float Fdistance = pulseIn(Echo, HIGH);

Fdistance= Fdistance / 58;

return (int)Fdistance;

}

char Full\_distance() {

int n = 90;

myservo.write(n);

int min\_dis = 100;

char deg = 0;

int temp = 0;

for (int i =0; i<22; i++) {

temp = Distance\_test();

if (temp <= distance\_cm) {

if (temp< min\_dis) {

min\_dis = temp;

deg = n;

}

}

n += 5;

myservo.write(n);

delay(100);

}

if (min\_dis == 100) {

deg = 0;

}

myservo.write(90);

delay(100);

return deg;

}

void setup(){

myservo.attach(3,700,2400); // attach servo on pin 3 to servo object

Serial.begin(9600);

pinMode(Echo, INPUT);

pinMode(Trig, OUTPUT);

pinMode(10,INPUT);

pinMode(4,INPUT);

pinMode(2,INPUT);

myservo.write(90);

while (!Serial.available()) {

}

char t = Serial.read();

}

void loop() {

if(LT\_M){

is\_white = false;

forward();

}

else if(LT\_R) {

is\_white = false;

right();

}

else if(LT\_L) {

is\_white = false;

left();

}

else {

if (!is\_white) {

stop();

is\_white = true;

stop\_num = stop\_num + 1;

if (next\_stop) {

next\_stop = false;

while(!Serial.available()) {

}

}

}

if (stop\_num != 5) {

middleDistance = Full\_distance();

if(middleDistance != 0) {

Serial.write(stop\_num);

delay(500);

Serial.write(middleDistance);

next\_stop = true;

}

}

if (stop\_num == 5) {

stop\_num = 0;

delay(1000);

}

}

}

}

## Arduino2

// Motor driver pins (TB6612)

#define PIN\_Motor\_PWMA 5

#define PIN\_Motor\_PWMB 6

#define PIN\_Motor\_AIN\_1 7

#define PIN\_Motor\_BIN\_1 8

#define PIN\_Motor\_STBY 3

// Ultrasonic sensor pins

#define TRIG\_PIN 13

#define ECHO\_PIN 12

// Distance threshold (cm)

#define OBSTACLE\_DISTANCE 20

// Line tracking sensor pins

#define SENSOR\_LEFT A0

#define SENSOR\_MIDDLE A1

#define SENSOR\_RIGHT A2

#define SPEED 70 // Base speed (0-255)

// Helper macros (assuming LOW means line detected)

#define LT\_L (digitalRead(SENSOR\_LEFT) == LOW)

#define LT\_M (digitalRead(SENSOR\_MIDDLE) == LOW)

#define LT\_R (digitalRead(SENSOR\_RIGHT) == LOW)

bool did\_stop = false;

int stop\_num = 4;

int num = 1;

// Ultrasonic distance function

int Distance\_test() {

digitalWrite(TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(10); // 10us pulse

digitalWrite(TRIG\_PIN, LOW);

unsigned long duration = pulseIn(ECHO\_PIN, HIGH);

int distance\_cm = duration / 58;

return (distance\_cm > 200) ? 200 : distance\_cm;

}

void setVoltageForDuration(float targetVoltage) {

// Validate input voltage range (0-5V)

if (targetVoltage < 0.0) {

targetVoltage = 0.0;

} else if (targetVoltage > 5.0) {

targetVoltage = 5.0;

}

// Convert voltage to PWM value (0-255)

int pwmValue = (int)((targetVoltage / 5.0) \* 255 + 0.5); // +0.5 for rounding

// Set pin 11 as output

pinMode(11, OUTPUT);

int i = 0;

while(i!=600) {

digitalWrite(11, HIGH);

// Maintain voltage for 5 seconds

delay(3);

// Optional: Turn off the output after duration

digitalWrite(11, LOW);

delay(2);

i++;

}

// Apply the PWM signal

}

void driveMotors(bool dirA, int speedA, bool dirB, int speedB) {

digitalWrite(PIN\_Motor\_AIN\_1, dirA ? HIGH : LOW);

digitalWrite(PIN\_Motor\_BIN\_1, dirB ? HIGH : LOW);

analogWrite(PIN\_Motor\_PWMA, speedA);

analogWrite(PIN\_Motor\_PWMB, speedB);

}

void stopMotors() {

analogWrite(PIN\_Motor\_PWMA, 0);

analogWrite(PIN\_Motor\_PWMB, 0);

}

void forward() {

driveMotors(true, SPEED, true, SPEED);

Serial.println("Forward");

}

void left() {

// Pivot left: left wheel backward, right wheel forward

driveMotors(false, SPEED, true, SPEED);

Serial.println("Pivot Left");

}

void right() {

// Pivot right: left wheel forward, right wheel backward

driveMotors(true, SPEED, false, SPEED);

Serial.println("Pivot Right");

}

void stop\_and\_kill() {

unsigned long rotate\_start, rotate\_duration;

int distance;

// 1. Rotate left until obstacle is detected

rotate\_start = millis();

while (true) {

right();

distance = Distance\_test();

Serial.print("Distance: ");

Serial.println(distance);

if (distance <= OBSTACLE\_DISTANCE) {

stopMotors();

break;

}

delay(50); // Small delay to avoid spamming sensor

}

rotate\_duration = millis() - rotate\_start;

delay(300);

// 2. Stop for 5 seconds

Serial.println("Obstacle detected. Stopping for 5 seconds...");

stopMotors();

setVoltageForDuration(3);

// 3. Rotate right for the same time to return to original heading

Serial.println("Returning to starting position...");

rotate\_start = millis();

while (millis() - rotate\_start < rotate\_duration) {

left();

delay(10);

}

stopMotors();

Serial.println("Returned to starting position.");

delay(300);

// End program (do nothing)

while (true) {

stopMotors();

delay(1000);

}

}

void setup() {

// Motor pins

pinMode(PIN\_Motor\_STBY, OUTPUT);

pinMode(PIN\_Motor\_PWMA, OUTPUT);

pinMode(PIN\_Motor\_PWMB, OUTPUT);

pinMode(PIN\_Motor\_AIN\_1, OUTPUT);

pinMode(PIN\_Motor\_BIN\_1, OUTPUT);

digitalWrite(PIN\_Motor\_STBY, HIGH); // Enable motors

// Sensor pins

pinMode(SENSOR\_LEFT, INPUT);

pinMode(SENSOR\_MIDDLE, INPUT);

pinMode(SENSOR\_RIGHT, INPUT);

// Ultrasonic pins

pinMode(TRIG\_PIN, OUTPUT);

pinMode(ECHO\_PIN, INPUT);

Serial.begin(9600);

while (!Serial.available()) {

}

char t = Serial.read();

stop\_num = t;

}

void loop() {

// Read sensors

bool leftSensor = LT\_L;

bool middleSensor = LT\_M;

bool rightSensor = LT\_R;

// Line tracking logic

if (!middleSensor) {

did\_stop = false;

forward(); // On track

} else if (!rightSensor) {

did\_stop = false;

right(); // Too far left, pivot right

} else if (!leftSensor) {

did\_stop = false;

left(); // Too far right, pivot left

} else {

if (!did\_stop) {

did\_stop = true;

stopMotors(); // Lost line, stop

delay(1000);

Serial.println("Stop/Lost");

if(stop\_num == num) {

stop\_and\_kill();

}

num ++;

}

}

}