הפרטים שלי כדי להוסיף למסמך הראשי:

בר אנג'ל Bar Angel

ת.ז. 307901587

אימייל: bar.ang16@gmail.com

# תרשים

# התוכנית אשר רצה על מכשיר ה-Raspberry Pi

התוכנית אשר רצה על מכשיר ה Raspberry Pi בנויה בטכנולוגית Windows Universal Platform ונבנתה באמצעות Visual Studio 2017.

התוכנית בנויה במבנה הדומה במידה רבה שרת אינטרנט, כלומר התוכנית מאזינה באופן קבוע לבקשות התחברות מצד הלקוחות. כאשר מתקבלת בקשה, התוכנית מספקת את שירותה, שולחת חיווי ללקוח ואז סוגרת את החיבור.

התוכנית ממשת ארבעה פונקציות עיקריות כמפורט להלן:

## התממשקות לחומרה –

ההתממשקות לחומרה נעשת על ידי מנשק בשם UserHardwareLinker (או בקיצור uhl), זהו למעשה הרחבה של הממשק הבסיסי אשר מתממשק לרכיב hx711 (מידע אודותיו תחת הסעיף "חומרה").

בנוסף לפעולות הבסיסיות של קבלת נתוני השקילה, ה-uhl עוקב אחר המשתמשים ויודע בכל עת האם המשקל נמצא בשימוש, ולהחזיר את פרטי המשתמשים אשר משתמשים במשקל.

כמו כן כתבנו מחלקה נוספת בשם UserHardwareLinker אשר יורשת מ-LinearHX ומוסיפה לה פונקציות נוספות המאפשרות מעקב אחר המשתמשים, ויכולת זיהוי בכל עת האם המשקל נמצא בשימוש, ולהחזיר את פרטי המשתמשים אשר משתמשים במשקל.

## התחברות לענן Microsoft Azure –

ההתחברות לענן נעשית בעזרת IoT Hub. רכיב זה נמצא בענן ומאזין להודעות אשר אשר נשלחות מהתוכנית אל הענן (הסבר מפורט על אופן הטיפול בהודעות בענן נמצא במסמך זה בפרק העוסק בכך).

התוכנית שולחת הודעה לענן בשתי סיטואציות:

* רישום המכשיר לענן, או עדכות כתובתו (הסבר מפורט בפרק העוסק בהתקנת המכשיר)
* שליחת הנתונים שהתקבלו ממד-המשקל יחד עם של המשתמש אשר ביצע את השקילה ותאריך השקילה

## האזנה לבקשות התחברות –

כאשר משתמש רוצה לבצע שקילה עליו להודיע לתוכנית כי ברצונו להישקל. כאשר התוכנית מקבלת את ההודעה, היא יכולה לפנות לממשק ה-uhl (ראה תת סעיף "התממשקות לחומרה") לשם קבלת הנתונים מהסנסור.

שליחת ההודעות נעשית באמצעות פרוטוקול אשר פותח על ידנו בשם Data Request Protocol (או בראשי-תיבות DRP). הסבר מפורט על הפרוטוקול נמצא בחלק הרלוונטי.

DRP הינו פרוטוקול תקשורת אשר יושב מעל פרוטוקול TCP הסטנדרטי. יצירת החיבור בין האפליקציה אצל הלקוח ומכשיר הרספברי פיי ממושת באמצעות Socket. התוכנית מקימה שרת בסיסי אשר מאזין להודעות DRP ומשרת אותן.

## תחזוקת הגדרות בסיסיות, ואספקת ממשק נוח המאפשר גישה להגדרות אלו -

על הלקוח לספק למכשיר מספר הגדרות ראשונית בסיסיות, כלומר הגדרות שיש להגדירן באופן חד פעמי לפני השימוש במכשיר פעם הראשונה. אי-לכך יצרנו ממשק נוח המפאשר למשתמש גישה להגדרות המכשיר ולשינויים.

ההגדרות נשמרות על כרטיס ה-SSD של ה-Raspberry Pi, ולכן אינן משתנות גם לאחר כיבוי והפעלה מחדש של המכשיר.

הנתונים שנשמרים הינם:

* שם מד-המשקל (ניתנת למשתמש האפשרות לתת למכשיר שברשותו שם אשר יוצג למשתמשים הנשקלים. זה יכול להיות נוח עבור משתמשים עסקיים כגון מכוני כושר, קופות חולים וכו')
* שקל,המספר הסידורי של המשקל
* כתובת הIP של המשקל כפי שנצפתה לאחרונה.
* נתוני הכיול של המשקל (הסבר מפורט בסעיף העוסק בחומרה)

הגישה לנתונים אלו נעשית באמצעות ממשק WEB ולכן יכולה להתבצע מכל דפדפן סטנדרטי. המכשיר מתחזק שרת HTTP פשוט אשר מאזין לפורט 9000, ושולח את דף ההגדרות לכל לקוח אשר מנסה לגשת להגדרות המכשיר.

הגישה להגדרות מתבצעת ע"י הזנת כתובת ה IP של המכשיר והפורט בשורת הכתובת של הדפדפן.

למשל : <http://10.0.0.2:9000>

# החומרה

--- להוסיף כאן הסברים אודות הרכיבים בהם השתמשנו (מד המשקל, הקומבינטור, המגבר, הרספברי) ---

הרספברי מתחבר ישירות אל המגבר בלבד דרך יציאות GPIO.

הממשק המקשר בין החומרה לתוכנה הינו מחלקה ב-C# אשר מהווה חלק מהתוכנית הראשית שרצה על גבי ה Raspberry Pi. שמו של המנשק הינו LinearHX. זהו למעשה תרגום שביצענו עבור מחלקה שנכתבה עבור ארדואינו בשפת C++: <https://github.com/bogde/HX711>

מעבר לתרגום הקוד הנ"ל כך שיתאים לרספברי פיי, הוספנו בו מספר פונקציות אשר מממשות את תהליך הכיול של המשקל בצורה קלה ונוחה יותר לשימוש עבור הצרכים שלנו.

משך השקילה עורך כ-5 שניות. במהלכה הסנסור מבצע את השקילה מספר רב של פעמים (כאלף פעמים) ומחזיר את ממוצע השקילות שבוצעו. באופן זה אנו ממזערים את אי-דיוקים של הסנסור.

-- להוסיף כאן מידע לגבי התקשורת עם הרכיב hx711. אפשר לקחת את המידע מה-manual של הרכיב באתר של ספארקפאן –

קריאת הנתונים מהסנסור נעשית על ידי מחלקה שכתבתנו בשפת C# ושמה LinearHX. זהו למעשה תרגום שביצענו עבור מחלקה שנכתבה עבור ארדואינו בשפת C++: <https://github.com/bogde/HX711>

# כיול (Calibration)

## הסבר מתמטי

המידע אשר מוחזר מהסנסור הינו מידע חסר יחידות מידה לכן אינו קריא עבור בני אדם. ברצוננו כמובן להציג את המשקל ביחידות מידה של קילוגרמים. לכן, אנו מגדירים העתקה לינארית כאשר הוא הערך הגולמי שהתקבל מסנסור ו- הוא המשקל האמיתי של האובייקט הנשקל (בקילוגרם).

אנחנו אומרים שהמשקל מכויל אם אנחנו יודעים את הפונקציה עבור מד-המשקל שבו אנו משתמשים (שהיא משתנה בין מד-משקל אחד למשנהו, אך לעולם אינה משתנה עבור משקל ספציפי).

משום ש היא פונקציה לינארית, הרי שבהינתן שתי שקילות של שני עצמים שונים שמשקלם האמיתי ידוע מראש ניתן לחשב את הפונקציה. לכן, נגדיר שלושה משתנים:

* - ה"משקל אפס", כלומר הערך שמוחזר על ידי הסנסור כאשר לא מונח עליו דבר.
* - משקל של אובייקט כלשהו שמשקלו ידוע מראש
* - הערך שהוחזר ע"י הסנסור כאשר שקלנו בו את העצם שבחרנו בנקודה הקודמת

עתה מתקיימים הקשרים:

נגדיר:

ולכן ע"ס חישוב אלגברי פשוט נמצא את הפונקציה היחידה שמקיימת את התנאים האלו:

*לכן, אם ידועים לנו הערכים OFFSET וSCALE הייחודיים למד-המשקל המחובר למערכת נוכל לומר שהמשקל מכויל.*

## תהליך הכיול

הכיול מתבצע בעת התקנת המכשיר (כלומר פעם אחת בלבד) ובמהלכו המשתמש אשר מתקין נדרש למדוד ולספק את שלושת הערכים שהוגדרו בתת-סעיף הקודם. את הערכים יספק המתקין ע"י ביצוע השקילות מתאימות במערכת, ואילו את הערך יספק ע"י הזנתו בתיבת-טקסט בדף ההתקנה.

כמו כן, אם *הערכים OFFSET וSCALE ידועים למתקין מראש הוא יוכל לספקם ישירות ללא צורך בביצוע השקילות.*

# Data Request Protocol (DRP)

--- להוסיף הסבר על מטרתו ואופן השימוש בו –

הפרוטוקול מבוסס על מבנה JSON ולו הם השדות שלו:

* **Protocol** – שם הפרוטוקול. תמיד יכיל את אותה מחרוזת "DRP$" (כדי להבדיל אותו מהודעות שאינן רלוונטיות שעלולות "להתעופף" בענן)
* **Dev Type** – אותו דבר כמו בהודעות MEP. יכיל את אחת המחרוזות APP או RBPI.
* **Source ID -** המספר הסידורי של שולח ההודעה.
* **Dest ID -** המספר הסידורי של שולח ההודעה.
* **Username** – שם המשתמש שרוצה להישקל
* **Data** – מכיל את תוצאת השקילה.
* **Status** - שדה זה מכיל את סוג ההודעה. המכשיר המקבל את ההודעה צריך לדעת כיצד להתייחס לשדות של הפרוטוקול בהתאם לסוג ההודעה. סוגי ההודעה האפשריים הם:
  + **scanned** – מיד לאחר סריקת הקוד האפליקציה לרספברי הודעה עם הסטטוס הזה ("הרגע סרקתי אותך"). ההודעה תכיל את שם המשתמש המחובר לאפליקציה.
  + **data** – מעביר מידע לאפליקציה (בשדה data) בלי לסגור את החיבור.
  + **ack** – בכל פעם שהרספברי/אפליקציה מקבל הודעה (עם סטטוס שונה מ-ACK) הוא יענה בהודעת ACK. בהודעת ACK חשוב שימצאו המספרים הסידוריים (שאר השדות יכולים להכיל זבל).
  + **InUse** – לא ניתן להישקל היות והמשקל נמצא בשימוש ע"י משתמש אחר
  + **HardwareError** – לא ניתן להישקל בעקבות בעיות בחומרה
  + **Illegal** – הודעה לא חוקית **או** הודעה שאינה ממקור אמין (ע"ס הToken)
* **Date** – תאריך ושעת שליחת ההודעה (עדיין לא לגמרי סגור על הפורמט)

מכיוון שרכיבי החומרה הפרוייקט שלנו מחוברים זה לזה בכמעין "שרשרת" היינו צריכים לבדוק כל אחד מהם בנפרד לפני שיכולנו לחבר את כולם יחד.

לשם כך השתמש ברכיב Load cell אשר מסוגל לשקול עד 1 ק"ג בלבד. היות ומדובר בסנסור בודד (במקום באיחוד של ארבעה סנוסורים) יכולנו לחבר אותו לרכיב ה-Amplifier ללא צורך בקומבינטור. לכן, בעזרתו יכולנו לכתוב אתה הקוד המשתממשק לחומרה ולדבג אותו. ורק לאחר שווידאנו שהוא עובד על מד-המשקל הקטן חיברנו את ה- Amplifier למד המשקל שבו אנו משתמשים בפרוייקט.

חיבור רכיב ה-Combinator היוו אף הוא אתגר לא קטן. בהתחלה ניסינו לבנות את הרכיב בעצמנו (משום שלא היה ניתן להשיג אותו אצל הספק שעובד עם אונ' תל אביב). נעזרנו במדריך הזה לשם כך. אולם לא הצלחנו לגרום לסנסור לעבוד ולבסוף העדפנו לקנות את הסנסור בעצמנו מספק אחר במקום לבנות אותו לבד.

נעזרנו במנחה שלנו, סיוון טולדו, כדי לבצע את פעולות הלחמה של הרכיבים.

היינו אופטימים במחשבה שנוכל להוריד מ-github קוד אשר קורא נתונים מהרכיב hx711, "לשתול" אותו בתוכנית שלנו ולקוות שהוא יעבוד. כמובן שלא כך היה המצב, מהר מאוד הבנו שללא קריאה מעמיקה של הדוקומנטציה של הרכיב, לא נצליח לכתוב את המחלקה כך שהיא תעבוד כמו שצריך. הקוד אשר אחראי על קריאת הנתונים עבר מספר רב של גירסאות עד שהגענו לגירסה הסופית שלו.

חששנו מהשימוש ב socket programming לשם יצירת התקשורת בין האפליקציה לRPI. חשבנו בהתחלה ליצור את התקשורת דרך הענן, אולם לבסוף התברר לנו שמובר בעבודה קשה שמהווה מתכון לתקלות בזמן ששימוש בsockets הוא פשוט יותר ואמין יותר. החלטנו להשתמש בענן למטרות של אחסון מידע בלבד, ולא למטרת העברת הודעות בין מכשירים.