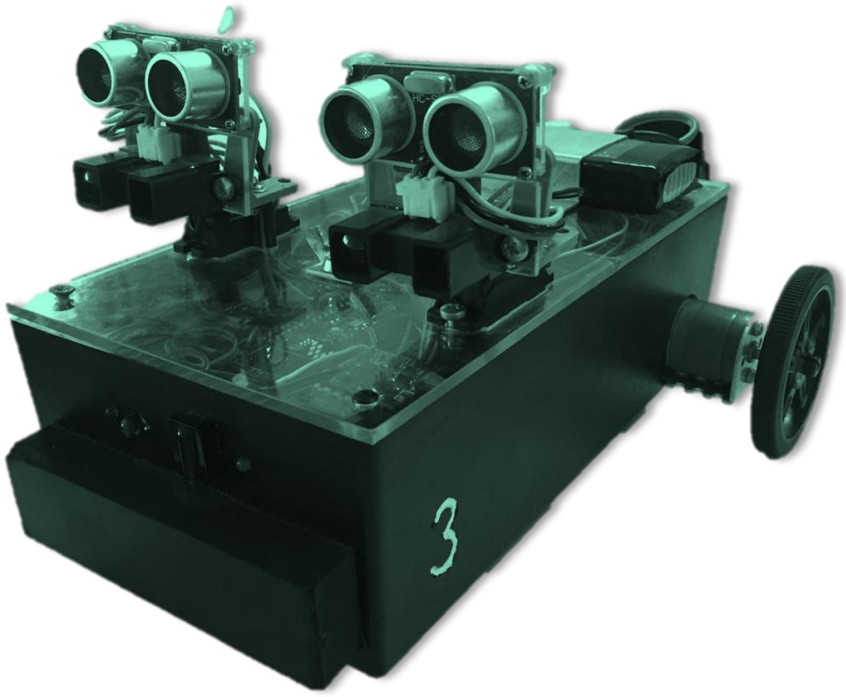
אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

פרויקט גמר

מבנה מחשבים ספרתיים להנדסת מחשבים

רכב אוטונומי



רכב מספר 3
מאור אסייג 318550746
רפאל שטרית 204654891
מנחה : חנן ריבוא

תוכן עניינים

תיאור כללי

תיאור המשימה

תיאור כללי

משימת הגמר הינה תכנון ותכנות של התנהגות אוטונומית עבור רכב המצויד בחומרה המאפשרת זו. חומרה זו מורכבת מחיישני מרחק, זרועות סיבוב, מנועי גלגל מכניים, רכיבי תקשורת אלחוטית ועוד – כפי שיתואר בהמשך.

לכל רכיבי החומרה השונים ניתנו מדריכי קנפוג ועבודה בהתאם ע"י המנחה, חלקים נרחבים מהם אינם מצויינים במסמך זה.

התכנון עבור הרכב הינו להיכנס לזירה בגודל מוגדר מראש, לפגוע במספר מטרות הפזורות במרחב ולצאת מהזירה. למטרה זו ביצענו אינטגרציה בין המודלים השונים תוך כדי תכנון קפדני לדיוק מרבי של אינטגרציה זו.

בשלב הראשון יש לעבור דרך פתח בקיר בכדי להיכנס לזירה – בשלב זה מעורבים חיישני מרחק קול (UltraSonic) למציאת הפתח, חישוב מסלול וכניסה נכונה לזירה.

בשלב השני יש להתקדם בציר המרכזי אווירי, תוך כדי עיבוד אותות מחיישני המרחק שיוצבו בשני צידי הרכב. בחרנו להיעזר בחיישני ה IR, בהם יש דיוק ממוקד למרחק בסדר גודל שהוגדרה הזירה.

במידה וזוהתה מטרה, על הרכב להתכוונן בצורה נכונה לעבר המטרה, ווידוא המרחק ממנה שנית בעזרת חיישני ה IR, פגיעה באובייקט וחזרה לציר המרכזי.

בשלב השלישי, לאחר פגיעה ב 4 מטרות כמתואר בשלב השני, יש לאתר את פתח היציאה מהזירה ולצאת ממנו.

תיאור אבני דרך

- **נסיעה ישרה** - בכדי לסנכרן בין מהירות 2 גלגלי הרכב, ותוך כדי ניצול הרכיבים המבצעים משוב אל הבקר לגבי המהירות המעשית של הגלגלים (בצורה של גל ריבועי בתדירות המשויכת למהירות) – קבוצתנו בחרה לבנות מערכת משוב המעדכנת את מהירות גלגלי הרכב בצורה דינאמית לקבלת תנועה ממוקדת וישרה יותר במהלך נסיעה.

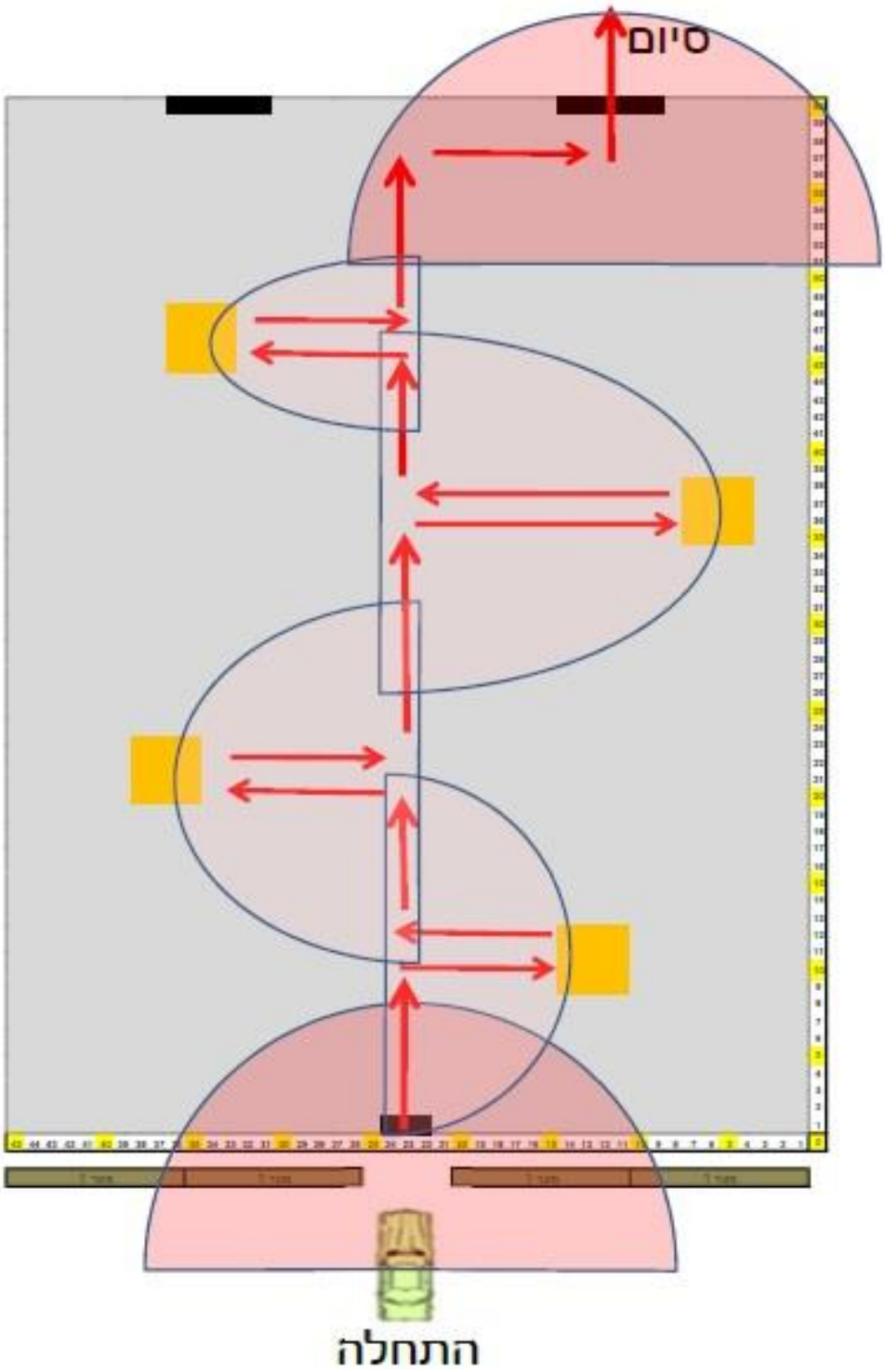
- **עיבוד אותות** – בכדי לאפשר דיוק מירבי למרחקים ארוכים ולזיהוי עצמים היכולים להיות קרובים אחד לשני, עבדנו לסריקת עצמים בעזרת חיישן המרחק IR שאופן פעולתו יפורט בהמשך. לשם פשטות האלגוריתם נמנענו ממערכת משוב משולבת חיישנים, אלא הסתמכנו על בדיקות ווידוא לדיוק נוסף של המרחק מהמטרה.

- **תזוזת זרועות ה Servo** – במהלך המשימה זרועות הServo יהיו או בקדמת הרכב או בצדדיו. בקדמת הרכב הServo ימפו את המרחב ויאפשרו לנו לוודא פגיעה במטרה, ואלו בצידי הרכב ה Servo יאפשרו זווית ל IR למציאת עצמים כפי שהוגדר.

- **טיימרים** – מערכות המשוב מחיישן המרחק UltraSonic ומה DC Motors Encoders מזינים ברגליים ספציפיות אות ריבועי בתדירות מסויימת. על מנת להאזין לתדירות זו ולהיות מסוגלים לנתח אותה, עלינו לקבוע תדירות שעון גבוהה יותר שבעזרתה נוכל לחשב באופן תקין ומדויק את התדירות המופקת מהמשוב, ומשם חישוב בנפרד לכל מודול למטרות השונות.

תיאור מסלול לדוגמה

- סריקות IR הינן קרן לייזר מדויקת, אומנם ישנה אפשרות שימוש בUltraSonic (גל קול) ולכן מצוינת אילוסטרציה של גל מתפשט במרחב.

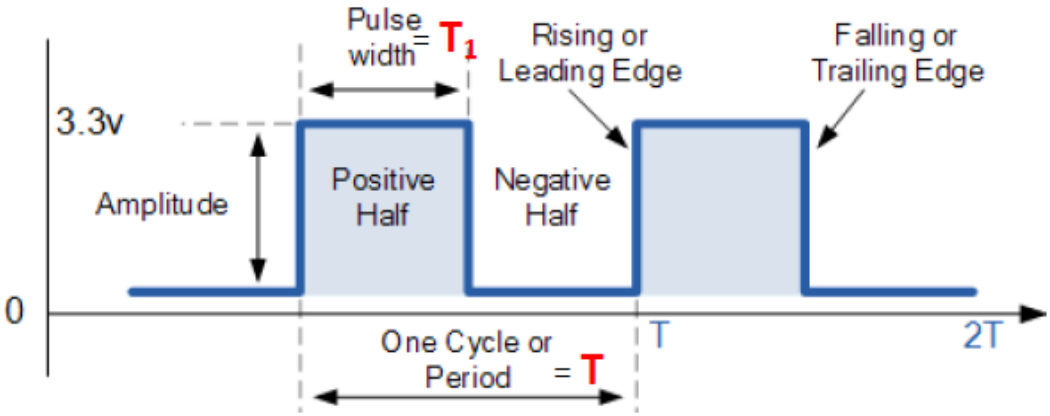


אופני עבודה

רכיבי החומרה ואופני העבודה

UltraSonic	חיישן מרחק
TPM	שעון
PIT	שעון
IR	חיישן מרחק
SERVO	זרוע סיבוב
DC	מנוע

- **DC Motors – 2 מנועי צעד** המחוברים למכניקה של הגלגלים. על ידי הזנת אות ריבועי PWM בתדירות קבועה $40Hz$ בעזרת הטיימרים שבבקר (TPM) נוכל לשלוט על מהירות הגלגלים בעזרת שינוי ה $DutyCycle$ של האות הריבועי המופק. חיווי על המהירות המעשית מופק בעזרת יחידת ה $Encoders$ בצורת גל ריבועי, אליו אנו מאזינים מרגליים ספציפיות בבקר בעזרת $InputCapture$ – עיבוד האות והפקת התדירות שלו לחישובים נוספים בסיוע הספירה של הטיימר הפנימי למשך זמן מחזור אחד. כל **408** עליות שעון של האות שאנו מפיקים להאזנת הגלגלים מתבצע סיבוב גלגל אחד.



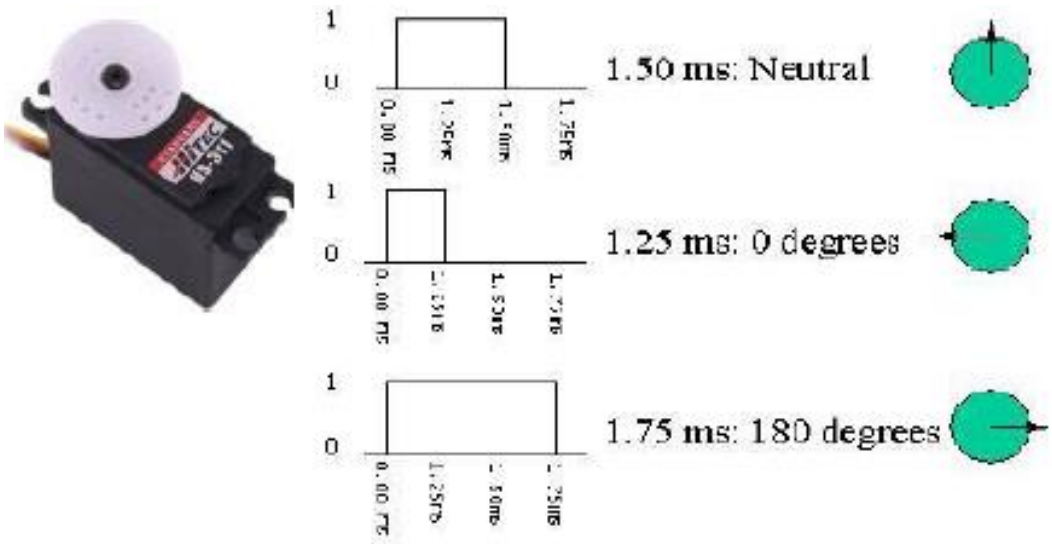
- **Servo Motors – 2 זרועות סיבוב** המותקנות על גב הרכב, ועליהן מחוברים חיישני המרחק השונים ($UltraSonic, IR$). מטרת סיבובן יהיה חשיפת החיישנים למרחב חדש בזירה. ניתן לשלוט בסיבובן על ידי הזנת אות ריבועי PWM בתדירות קבועה $40Hz$, הזווית הרצויה תתקבל על ידי מתן $DutyCycle$ יחסי לזווית הרצויה $0\sim180\ degree \rightarrow DutyCycle\ T_{on}\ 0.568ms \sim 2.2\ ms$.

Servo Motors

$f_{PWM} = 40Hz \rightarrow T_{PWM} = 25msec$

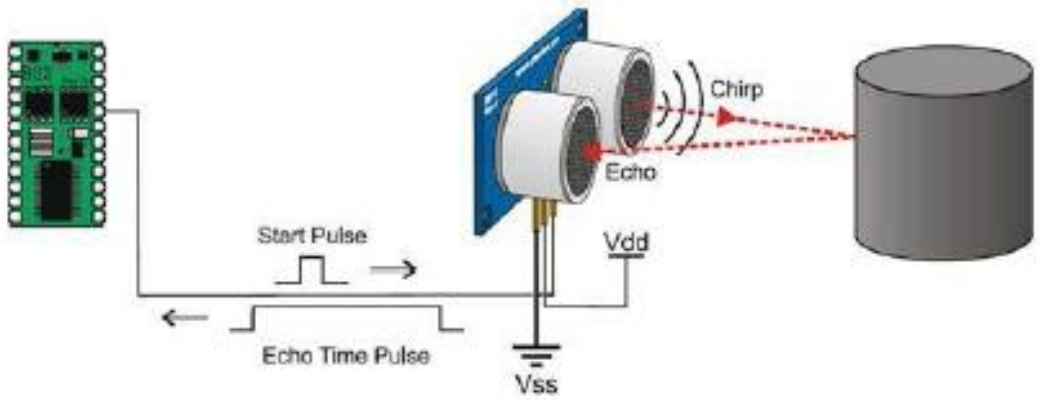
$T_{on} = 0.568msec \rightarrow$ זווית של **0 מעלות**

$T_{on} = 2.2msec \rightarrow$ זווית של **180 מעלות**

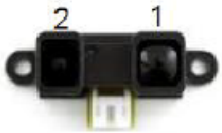


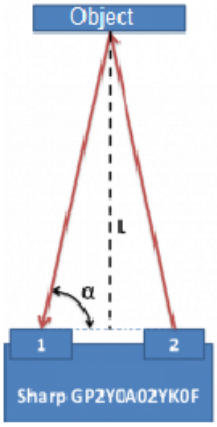
- **UlltraSonic Sensor – 2** חיישני מרחק מבוססים **גלי קול**.
  החיישן יורה גלי קול ומקבל החזרים מן האובייקט, ובהתאם לזמן שעבר מההוצאה לקליטה מופק גל ריבועי בהתאם למרחק. לגל הריבועי ניתן להאזין ברגליים הנכונות להפקת התדירות, ומשם שימוש בנוסחא המסופקת על ידי יצרן הרכיב להפקת המרחק ב cm.
  טווח המדידה האמין של החיישן על פי היצרן הינו עד 4m, אומנם הוא מפיק **גל קול שמתפשט במרחב** ולכן הוא סורק בצורת קונוס כמתואר במסלול לדוגמה בתיאור המשימה לעיל, ולכן בזירה בה המטרות יכולות להיות סמוכות עדיף לעבוד עם חיישן מרחק העובד אחרת. ישנן מספר נוסחאות לחישוב זה, בחרנו בנוסחה הנ"ל :

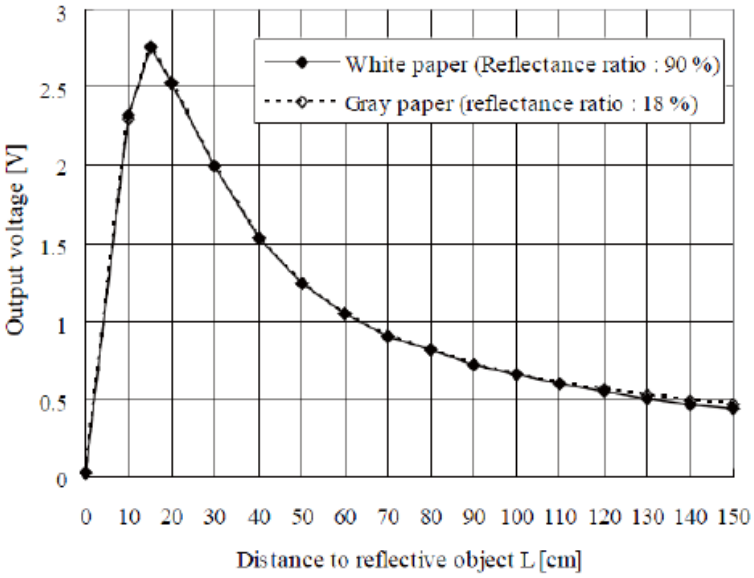
$$\boldsymbol{Range[cm] = Echo\ high\ level\ time \cdot 17000} \quad when\ 17000\left[\frac{cm}{sec}\right] is\ sound\ velocity$$



- **InfraRed Sensor – 2 חיישנים מבוססי לייזר,** החיישן יורה קרן לייזר ומקבל החזר מהאובייקט. מעגל חשמלי ברכיב מתרגם את ההחזר למתח אנלוגי, אותו אנו דוגמים ברגל ספציפית בבקר ומפענחים על פי רשומות היצרן. במשימתנו תפקיד ה IR הינו לזהות מכשולים, פתחי כניסה ויציאה ומיפוי המרחב. בתוכנית שלנו דגימת ערכי ה ADC נמצאת ברקע, ורק במידה וזוהתה דגימה המדמה מכשול במרחק המתאים לגבולות הזירה מכל צד $1.5m$ עולה דגל לטיפול מיוחד בשלבים השונים של המשימה.

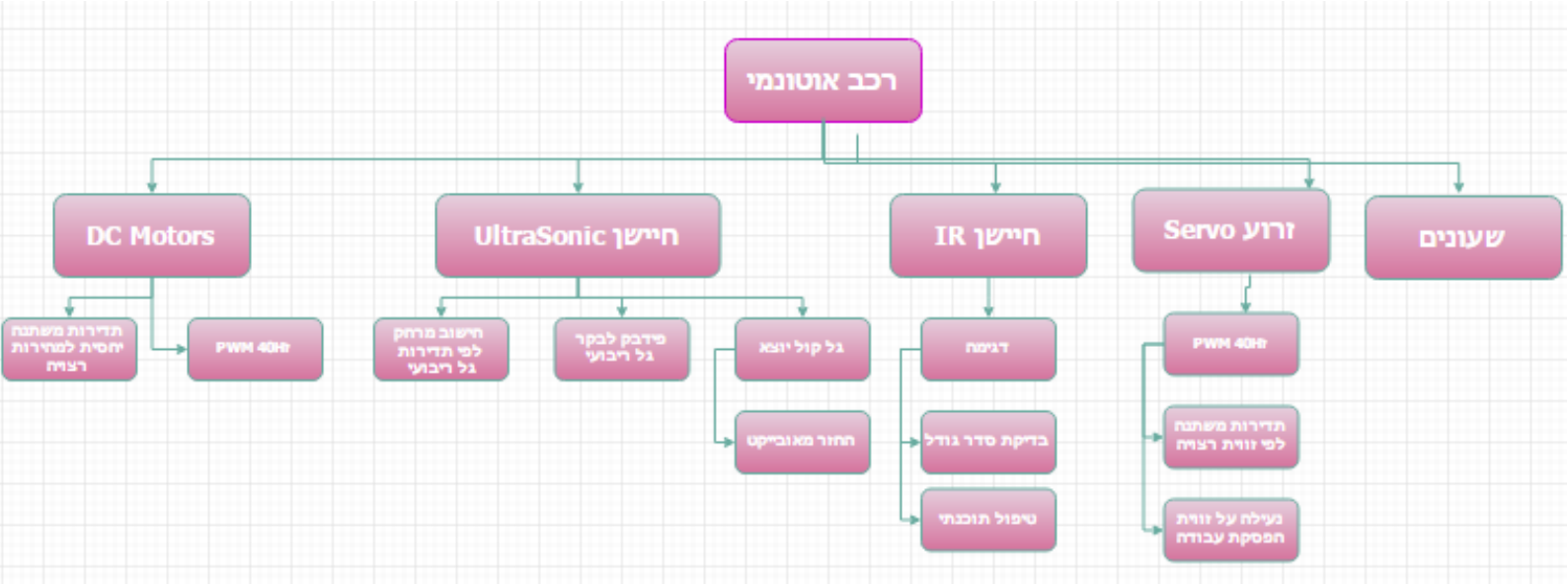






- **Battery -** סוללת הליתיום המחוברת לרכב במצב מלא ערכה הממוצע הוא $8.4\ v$ כאשר מתג הפעלת הרכב במצב ON נכנסת הסוללה לשימוש. הערך המינימאלי המותר במוצא הסוללה הינו $7.4v$. על כרטיס הרכב ישנו מעגל הגנה על הסוללה, כאשר מתח הסוללה יורד לערך לא תקין המנועים מתנתקים, נדלקת נורה ומופעל זמזם.

- **PIT,TPM –** טיימרים פנימיים המאפשר עדכון של דגלים להפעלת פונקציות בזמנים מסויימים בתוכנית לצורך בקרת מהירות, יישור נסיעה, מרחק נותר ממכשול עד לפגיעה ועוד. על ידי קנפוג מתאים ניתן לשלוט בתדירות השעון וכך להיעזר בו בחישובים השונים.
בתכנית שלנו TPM0 עוזר לנו לעקוב אחר חיווי מהירות הגלגלים מה $DC\ Motors\ Encoders$ ע"י חישוב תדירות האות הריבועי המוזן לרגלי הבקר, ובאם נרצה זאת לחישוב המרחק שחיישן ה $UltraSonic$ דגם.



אלגוריתמיקה

תיאור פסאודו קוד איכותי

1. **הכנה לתחילת פעילות** - קנפוג מתאים לפורטים, רכיבי החומרה, השעונים הפנימיים בבקר, אתחול משתנים קבועים.

2. **כניסה לזירה –** התקדמות ערך מוגדר מראש, זיהוי פתח, כניסה לזירה, התקדמות $1.5m$ למתחם המכשולים והכנות לקראת שלב 3 הכוללות – קביעת משתנים מתאימים בשביל מעקב התנועה והמרחק בPIT.

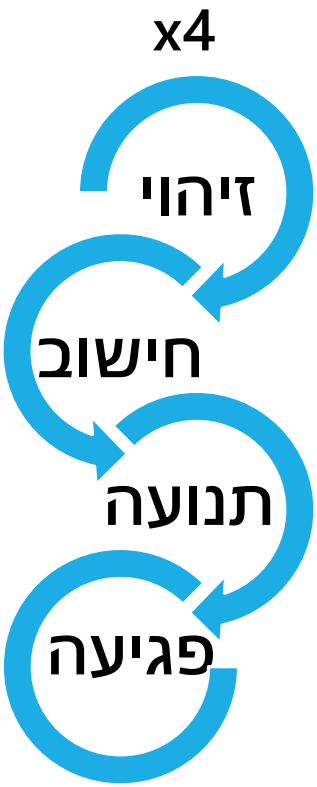
3. **זיהוי מכשול –** תוך כדי תנועה מתבצעת דגימה בתדירות גבוהה של ה IR אם זוהה מכשול במרחק שהרכב צריך להתייחס אליו הרכב יעצור תנועה, יחשב את המסלול אל המכשול תוך כדי סיבוב נכון, יוודא מרחק מהאובייקט מחדש ע"י דגימה נוספת.

4. **תנועה לעבר מכשול –** תוך כדי תנועה מתבצע מעקב אחר המרחק שנותר בעזרת חיווי המהירות של הגלגלים. ווידוא פגיעה תתבצע בעזרת חיישן המרחק. לאחר הפגיעה הרכב יחשב מסלול חזרה לציר המרכזי, יסתובב בהתאם ויינוע עד שיגיע אליו.

5. **המשך תנועה –** חזרה על שלב 2 ואילך, כל עוד לא פגעתי ב 4 מטרות או שהרכב הגיע (בעזרת מיפוי ציר Y) לקצה תחום המטרות, אם כן המשך ל 6.

6. **יציאה מהזירה –** הרכב ייסע עד כדי $0.5m$ מהקצה של הזירה. יסתובב במקביל לקיר המוצא ויחל לחפש בעזרת IR שמאל (דואגים לסובב את ה $Servo$ בהתאם) ערכי דגימה שלא ידמו בטווח $0.5m$.

תרשים זרימה – סדר פעולות הבקר



כניסה לזירה

כניסה לתחום המכשולים

זיהוי מכשול IR

חישוב מסלול והתחלת תנועה

פגיעה במכשול, חישוב מסלול לציר האמצעי והמשך תנועה

יציאה ממתחם המכשולים, יציאה מזירה

תיאור הקוד

קבצי התוכנית

main	התוכנית הראשית, ISR
BoardSupport	קינפוג
UART	פונקציית התקשורת
motors	מנוע גלגלים
servo	מנוע הSERVO
adc	מערכת הדגימה אותות
DistSens	קינפוג הIR
pit	קינפוג שעון הPIT
UltraSonic	קינפוג חיישני ULTRA SONIC

פירוט פונקציות כללי

קובץ main

turnToAngle	מסובב את הרכב בזווית רצויה
ExitArena	פונקצית היציאה מהזירה
EncoderSensing	מחשב את מהירות הרכב על ידי encoder
Motor_Dir_Speed	מגדיר מהירות וכיוון הרכב
FTM0_IRQHandler	פונקצית השירות של TMP
WalkAway	פונקצית הסריקה של הזירה

GetObjectLeft	פונקציה שתפקידה לבצע נגיעה באובייקט מצד שמאל
GetObjectRight	פונקציה שתפקידה לבצע נגיעה באובייקט מצד ימין
EnterArena	כניסה לזירה

קובץ BoardSupport

InitGPIO	קינפוג הLED
InitTPM	קינפוג שעון הTMP
ClockSetup	הגדרת שעון המערכת

קובץ UART

printMenu	מדפיס את תפריט ההתחלה
UART0_IRQHandler	רוטינת השירות של UART
InitUARTs	קיפוג

קובץ motors

MotorConfig	קינפוג המנועים
EncoderConfig	קינפוג הENCODER
EncoderChecker	פונקציה שבודקת את הייציבות בין הגלגלים

קובץ servo

ServoConfig	קינפוג הSERVO
Servo_Movement	אחראי על סיבוב הSERVO
RightExitconfig	קינפוג יציאה שלב א
leftExitconfig	קינפוג יציאה שלב ב
Straightconfig	קינפוג לנסיעה ישרה
CheckWhereTheServo	קינפוג SERVO

קובץ adc

adc_init	קינפוג הADC
ADC0_IRQHandler	רוטינת השירות

קובץ DistSens

DistanceMeasure	פונקצית מדידה
DistSensConfig	קינפוג חישני IR
DistanceMeasuring1	חישוב SENS1
DistanceMeasuring2	חישוב SENS2

קובץ pit

קינפוג הPIT	**pit_init**
רוטינת השירות של PIT	**PIT_IRQHandler**

קובץ UltraSonic

קינפוג הULTRASONIC	**UltraSonicConfig**
הדפסת מרחק	**PrintDistance**

שעונים

מאפשר החלפת אפשור בין מודלי ADC, מיפוי הרכב בציר Y, הרמת דגלים נחוצים לפונקציות בשלבים השונים.	**PIT**
מאפשר חיווי על מהירות הגלגלים באמצעות InputCapture	**TPM0**
מוצא גל ריבועי בתדירות קבועה אך DutyCycle שונה לרכיבי החומר השונים בהתאם לדרישה (סיבוב Servo, מהירות גלגלים)	**TPM1, TPM2, TPM0**

משתנים ודגלים עיקריים

Sens1	דגימת ADC מ IR ימין
Sens2	דגימת ADC מ IR שמאל
Wheel1Speed	מהירות גלגל 1 cm/sec
Wheel2Speed	מהירות גלגל 2 cm/sec
wheel1Duty	$DutyCycle$ שמוזן לגלגל 1
Wheel2Duty	$DutyCycle$שמוזן לגלגל 2
currentY	מיפוי מיקום הרכב בציר Y
currentDistance1	תרגום דגימת IR ימין ל cm
currentDistance2	תרגום דגימת IR שמאל ל cm
Servo1CurrPosition	זווית נוכחית של $Servo$ ימין
Servo2CurrPosition	זווית נוכחית של $Servo$ שמאל
Distance1	מערך דגימות IR להשוואה
Turn1AngleTime	משתנה עזר לסיבוב זוויתי של הרכב עבור מעלות < 90
Turn2AngleTime	משתנה עזר לסיבוב זוויתי של הרכב עבור מעלות < 180

פונקציות עיקריות- פירוט איכותי (ראה קוד בעמודים המפורטים)

קובץ main

- **turnToAngle :**
הפונקציה מקבל זווית ומסובבת את הרכב בהתאם, כאשר ישנה התחשבות בזווית הנוכחית של הרכב. הרכב מתחיל את נסיעתו ב 90 ובהתאם לתוכנית מסתובב כמתבקש.

- **EncoderSensing :**
בעזרת תכונות ה $inputCapture$ האזנו למחזור אחד של גל ריבועי תוך כדי שמירת הספירה של השעון הפנימי. בעזרת מספר זה פונקציה זו מתחשבת בתדירות השעון הפנימי ומניבה את תדירות הגל הריבועי שהאזנו לו, ובהתאם לזאת חישוב מהירות הגלגלים ל cm/sec תוך כדי התחשבות בקוטר הגלגל, 408 עליות שעון לסיבוב שלם.

- **FTM0_IRQHandler :**
כאשר פסיקה זו מאופשרת על ידי טריגר חומרתי (בד"כ עליית שעון של גל ריבועי חיצוני שנרצה לבחון) נשמר מספר עליות השעון של שעון הספירה הפנימי תוך כדי התחשבות ב $TimerOverFlow$.

- **gameOn :**
מאפשר לרכב לבצע את סדרת התיחול המוגדרת והתחלת השלבים השונים במשימה המוגדרת.

- **walkAway :**
לאחר זיהוי מטרה פונקציה זו יודעת להתחשב במיקום הנוכחי של הרכב, מחשבת את המסלול של הרכב בהתאם למיקום האובייקט, נעה לעברו עד פגיעה וחוזרת לקו המרכז האווירי של הזירה.

קובץ Servo

- **Servo_Movment** :
מקבל מספר חיובי המבדיל בין האופציות לפעולה של הפונקציה ($Switch\ Case$), פונקציה זו מזיזה את זרועות ה $Servo$ לזווית רצויה (באמצעות ה Pit שינוי מתון בזווית) בהתאם למשימה, כפי שהוגדר בתיאור החומרה.

קובץ PIT

- **PIT_IRQHandler :**

רוטינת השירות של ה PIT. בתדירות גבוהה ה PIT מאפשר דגימה טורית של ה ADC השונים (IR ימין ו IR שמאל), תוך כדי עדכון מיפוי הרכב בציר Y על ידי המהירות הנוכחית של הגלגלים. דואג בנוסף לחיווי נסיעה ישרה ע"י מערכת משוב הנעזרת בתוצאות ה $Encoders$.

קובץ DistSens

- **DistanceMeasuring1 and DistanceMeasuring2 :**

פונקציה המקבלת את דגימת ה ADC מהIR המתאים לכל צד, המספר שהתקבל מייצג את תרגום המתח הנדגם בהתאם לקנפוג שלנו לערכי $0 - 3.3v \rightarrow 0 - 2^{12}$. הפונקציה נעזרת בפונקציות עזר למציאת הערך הקרוב ביותר במערך הדגימות המובנה מראש ברכב, וכך למצוא באופן יחסית מדויק את המרחק מהאובייקט.

קובץ ADC

- **ADC_IRQHandler :**

עדכון ערכי $sens1$ ו $sens2$ לסירוגין לערכי הדגימות העדכניות של הIR. ישנה אפשרות נוספת לאפשר חיווי לנעילת ה Servo על זווית.

נספח – הקוד כמכלול

ניתן לזהות את שמות הקבצים על פי הכותרות, או לחלופין לפי הסדר הבא :

קובץ pit	טיימר
קובץ Servo	זרוע סיבוב
קובץ UltraSonic	חיישן מרחק
קובץ adc	חומרת דגימה
קובץ battery	בטריה
קובץ BoardSupport	קנפוג חומרה בבקר
קובץ DistSens	חיישן IR
קובץ main	תוכנית ראשית
קובץ motors	גלגלי רכב

```c
#include "pit.h"
#include "BoardSupport.h"
#include "TFC.h"
/* Initializes the PIT module to produce an interrupt every second
void pit_init(void)
{
    // Enable PIT clock
    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;
    // Turn on PIT
    PIT_MCR = 0;
    // Configure PIT0 to produce an interrupt every 4ms
    PIT_LDVAL0 = 0x0000FFFF;//Trigger to adc
    PIT_LDVAL1 = 0x000DFD40;//check sensors
    PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK+ PIT_TCTRL_TEN_MASK;//Enable interrupt
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // Enable timer
    currsens=6;
    MultiDrop=2;
    counter=0;//temp
    enable_irq(INT_PIT-16); // disable PIT IRQ on the NVIC
    set_irq_priority(INT_PIT-16,1);  // Interrupt priority = 0 = max
  // ADC0_SC1B = ADC_SC1_ADCH(currsens) + ADC_SC1_AIEN_MASK;//Connecting to the current
sensor1
}
void PIT_IRQHandler(void)
{
    //         IR sensor      ///////

    if(MoveServo==0){
        if(counter==0){
            ADC0_SC1B = ADC_SC1_ADCH(currsens) + ADC_SC1_AIEN_MASK;//Connecting to the
current sensor1
            //
            //temp[i]=sens1;
            //i++;

        }else if (counter==1){
            ADC0_SC1B = ADC_SC1_ADCH(currsens+1) + ADC_SC1_AIEN_MASK;//Connecting to
the current sensor2
            //
            //temp[j]=sens2;
            //j++;
        }
    }
    counter=(counter+1)%2;//chose the IR
    RoundOfPit=(RoundOfPit+1)%4;//move the servo
    RoundOfPit2=(RoundOfPit2+1)%2;// EncoderChecker or init_TPM0

    //             move servo                 //
    if(MoveServo==1){
    if(scanum>0 && RoundOfPit==0){
        Servo_Movement(2);
        scanum--;//number of the servo left to go to the side
    }
    if(scanum==0){
        ScanPhase=0;
        MoveServo=0;//if the servo is on moving don't get IR measurements
        Exit=0;//chose if left or right servo
        }
    }

    //               Encoder               //
    if(currentSpeed>0){
        if(RoundOfPit2==0)
            EncoderChecker();
```

```
        else
            init_TPM0(1);
    }

    //          distance over          //
    if ( DistLeft > 0){
        DistLeft = DistLeft - (double)fmax(wheel1Speed,wheel2Speed)*0.04584; // current
speed * pit timer per entry
    }

    if (Yflag==1){
        currentY = currentY + (double)fmax(wheel1Speed,wheel2Speed)*0.04584;
    }

    //                      delay                   //
    if(delay>0)
        delay--;
    // Clear interrupt
    PIT_TFLG1 = PIT_TFLG_TIF_MASK;
}
```

```c
#include "TFC.h"
#include "mcg.h"
#define MaxDuty 0x660//1632
#define MinDuty 0x1AF//431
#define DeltaDuty 0x4F1//1201

void ServoConfig(){
    //Servo1 Configuration
    //Servo1_EN
    PORTE_PCR21 = PORT_PCR_MUX(3); // PTE21 pin TPM1_CH1- ALT3
    // Servo1_samp
    PORTE_PCR29 = PORT_PCR_MUX(0); // PTE29 pin ADC0_SE4b- ALT1
    //Servo2 Configuration
    //Servo2_EN
    PORTE_PCR31 = PORT_PCR_MUX(3); // PTE31 pin TPM0_CH4- ALT3
    // Servo2_samp
    PORTE_PCR30 = PORT_PCR_MUX(0); // PTE30 pin ADC0_SE23- ALT1
    backServo=-1;
    TPM1_C1V = 0x1AF;// 0 duty cycle
    TPM0_C4V =0x660;// 0 duty cycle
    TPM0_C4SC |= TPM_CnSC_MSB_MASK + TPM_CnSC_ELSB_MASK;//En_Servo2
    TPM1_C1SC |= TPM_CnSC_MSB_MASK + TPM_CnSC_ELSB_MASK;//Enable_Servo1
}

void Servo1SetPos (int Servo1Position){
    if(backServo==1){
        Servo1Position=Servo1Position-5;
    }else if(backServo==0){
        Servo1Position=Servo1Position+5;
    }
    if(Servo1Position>90){
        Servo1Position=90;
    }
    if(Servo1Position<0)
    {
        Servo1Position=0;
    }
    int Duty = (int)((Servo1Position*DeltaDuty/180) + MinDuty);
    TPM1_C1V=Duty;
    Servo1CurrPosition=Servo1Position;
}

void Servo2SetPos (int Servo2Position){
    if(backServo==1){
        Servo2Position=Servo2Position+5;
        }else if(backServo==0){
        Servo2Position=Servo2Position-5;
        }
    if(Servo2Position>180){
        Servo2Position=180;
    }
    if(Servo2Position<70)
    {
        Servo2Position=70;
    }
    int Duty = (int)((Servo2Position*DeltaDuty/180) + MinDuty);
    TPM0_C4V=Duty;
    Servo2CurrPosition=Servo2Position;
}

//-----------------------------------------------------------------
// PRESS 1 Function CASES
//-----------------------------------------------------------------
void Servo_Movement(int mode){
```

```c
    PORTE_PCR31 = PORT_PCR_MUX(3); // PTE31 pin TPM0_CH4- ALT3
    switch(mode){
        //get the servo to the sides of the car
        case 1:
            backServo=0;
            scanum=35;
            MultiDrop=4;
            MoveServo=1;
            option = 0;
            while(scanum>0);
            break;
        //move the servo in dir*multiDrop degrees
        case 2:
            //TPM0_SC |= TPM_SC_CMOD(1); //start the TPM0 PWM counter
            if(Exit==1||Exit==2){//Exit logic
                if(Exit==1){
                    if(Servo1CurrPosition<90)
                        Servo1SetPos(Servo1CurrPosition);}
                else{
                    if(Servo2CurrPosition>90)
                        Servo2SetPos(Servo2CurrPosition);
                    if(Servo1CurrPosition>0)
                        Servo1SetPos(Servo1CurrPosition-6);
                }

            }else{
                Servo1SetPos(Servo1CurrPosition);
                Servo2SetPos(Servo2CurrPosition);
            }
            option = 0;
            break;
        // move to the front of the car
        case 3:
            backServo=1;
            scanum=35;
            MultiDrop=4;
            MoveServo=1;
            while(scanum>0);

    }
    option = 0;
}

void RightExitconfig(){
    TurnOnADC();
    Exit=1;
    turnToAngle(180);
    Servo_Movement(1);

}
void leftExitconfig(){
    TurnOnADC();
    Exit=2;
    turnToAngle(0);
    Servo_Movement(1);
    //ExitADCright();
}
void Straightconfig(){
    CheckWhereTheServo(0);//check  where the servo and change if needed
    TurnOnADC();
    delay=3;
    while(delay>0);
}
void CheckWhereTheServo(int position){
```

```
    if(Servo1CurrPosition==90&&position==0){
        Servo_Movement(3);
    }else if(Servo1CurrPosition==0&&position==90)
        Servo_Movement(1);
}
```

```c
 * UltraSonic.c

#include "DistSens.h"
#include "TFC.h"

void UltraSonicConfig(){
    PORTC_PCR3 = 0; // assign PTC3 as TPM0_CH2
    PORTC_PCR4 = 0; // assign PTC4 as TPM0_CH3
    PORTD_PCR1 = PORT_PCR_MUX(4); // PTD1(TPM0_CH1) - output
    PORTD_PCR2 = PORT_PCR_MUX(4); // PTD2(TPM0_CH2) - inputCapture
    PORTD_PCR3 = PORT_PCR_MUX(4); // PTD3(TPM0_CH3) - inputCapture
}

//-----------------------------------------------------------------
//  Function to Print the speed
//-----------------------------------------------------------------
void PrintDistance(){
    double DistanceToPrint1;
    double DistanceToPrint2;
    //calculate the distance
    if (state == 3 || state == 4){ //IR
        currentDistance1 = DistanceMeasuring1(sens1);
        currentDistance2 = DistanceMeasuring2(sens2);
        DistanceToPrint1 = currentDistance1;
        DistanceToPrint2 = currentDistance2;
    }
    if (state ==5){
        UltraSonicSensing();
        DistanceToPrint1 = currentDistanceUltra1;
        DistanceToPrint2 = currentDistanceUltra2;
    }
    // then print distance 1
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    UARTprintf(UART0_BASE_PTR,"\r\n Distance 1 from the object is : \r\n");//logic to
print the current motor speed
    int k=4;
     while(k>=0){
        if(UART0_S1 & UART_S1_TDRE_MASK){ // TX buffer is empty and ready for sending
          if(k==4){ UART0_D=((int)(DistanceToPrint1/100)%10)+48;} //X00.0 DIGIT
          if(k==3){ UART0_D=(int)(DistanceToPrint1/10)%10+48;}//0X0.0 DIGIT
          if(k==2){ UART0_D=(int)(DistanceToPrint1)%10 + 48;} //00X.0 DIGITONE DIGIT
          if(k==1){ UART0_D=46;}
          if(k==0){ UART0_D=(int)(DistanceToPrint1*10)%10 + 48;} //00.X DIGITONE DIGIT
AFTER DOT
          k--;
         }
     }
    UARTprintf(UART0_BASE_PTR,"  [cm per second]  \r\n");
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    //-----------------------------------//
    // then print distance 2
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    UARTprintf(UART0_BASE_PTR,"\r\n Distance 2 from the object is : \r\n");//logic to
print the current motor speed
    k=4;
     while(k>=0){
        if(UART0_S1 & UART_S1_TDRE_MASK){ // TX buffer is empty and ready for sending
          if(k==4){ UART0_D=((int)(DistanceToPrint2/100)%10)+48;} //X00.0 DIGIT
          if(k==3){ UART0_D=(int)(DistanceToPrint2/10)%10+48;}//0X0.0 DIGIT
          if(k==2){ UART0_D=(((int)(DistanceToPrint2)%10))+48;} //00X.0 DIGITONE DIGIT
          if(k==1){ UART0_D=46;}
          if(k==0){ UART0_D=(int)(DistanceToPrint2*10)%10+48;} //00.X DIGITONE DIGIT
AFTER DOT
          k--;
```

```
        }
     }
    UARTprintf(UART0_BASE_PTR,"  [cm per second]  \r\n");
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    option = 0;
}
//-----------------------------------------------------------------
// Print UltraSonicDistance
//-----------------------------------------------------------------
void UltraSonicSensing(){
    currentDistanceUltra1 = wheel1Delta * 0.0226666666666667; // [time echo is
high]*17000 -> (wheel1Delta / 750000) * 17000;
    currentDistanceUltra2 = wheel2Delta * 0.0226666666666667; // [time echo is
high]*17000 -> (wheel1Delta / 750000) * 17000;
}
//-----------------------------------------------------------------
// PRESS 5 Function - UltraSonic
//-----------------------------------------------------------------
void UltraSonic_Sensor(){
    option=0;
    init_TPM0(3);
}
void UltraDistance(){
    UltraSonicSensing();
}
```

```c
#include "adc.h"
/*  adc_init()
void adc_init(void)
{
    // Enable clocks
    SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK  ; // ADC0  clock
    // Configure ADC
    ADC0_CFG1 = 0; // Reset register
    ADC0_CFG1 |= (ADC_CFG1_MODE(1)  |   // 12 bits mode
                  ADC_CFG1_ADICLK(0)|   // Input Bus Clock (20-25 MHz out of reset (FEI
mode))
                  ADC_CFG1_ADIV(1)) ;   // Clock divide by 2 (10-12.5 MHz)

    ADC0_SC2 |=  ADC_SC2_ADTRG_MASK;// Hardware trigger

    ADC0_SC3 |= ADC_SC3_AVGE_MASK |     // Enable HW average
                ADC_SC3_AVGS(2)   |     // Set HW average of 32 samples
                ADC_SC3_CAL_MASK;       // Start calibration process
    ADC0_CFG2 |= ADC_CFG2_MUXSEL_MASK;//The msb
    ADC0_SC1B |= ADC_SC1_ADCH(31) + ADC_SC1_AIEN_MASK; // Disable module
    printDistisPressed=0;
    enable_irq(INT_ADC0-16); // enable PIT IRQ on the NVIC
    set_irq_priority(INT_ADC0-16,0);  // Interrupt priority = 0 = max
}
void ADC0_IRQHandler(void){
    if((counter==1||Exit==2)&&Exit!=1){
        sens1=ADC0_RB;
        //ADC0_CV1=sens1-50;
        //ADC0_CV2=sens1+50;
        //RightScan= DistanceMeasuring1(sens1);
        //UpgradeRight();//to do
    }else if ((counter==0||Exit==1)&&Exit!=2){
        sens2=ADC0_RB;
        /*if(Exit==1){
            TurnOffADC();
            DistLeft=-1;
            ExitRight=1;
        }*/
        //ADC0_CV1=sens2-50;
        //ADC0_CV2=sens2+50;
        //LeftScan= DistanceMeasuring1(sens2);
        //UpgradeLeft();//to do
    }
    /*else{
        batteryVOLT=ADC0_RB;
        ADC0_CV1=batteryVOLT-10;
        ADC0_CV2=batteryVOLT+10;
    }*/
}

// function to update the map due to right scan
// nextIndexForTargetRight = 0, tempTargetRight = 0,
void UpgradeRight(){
    int x;//index
    int y;//index
    if (RightScan < 150){
        x = currentX + (int) (RightScan * sin(Servo1CurrPosition) * 0.1); // x index of
the target
        y = currentY + (int) (RightScan * cos(Servo1CurrPosition) * 0.1);  // y index
of the target
        if (x <= 44 && y <= 44 && map[x + y * 44] == 0) { // the target is legit
            for (i = currentX; i < x; i++)       if (map[i + currentY * 44]==0) {map[i
+ currentY * 44] = 1;}
```

```
            if (Servo1CurrPosition < 90){
                for (i = currentY ; i < y; i++)  if (map[currentX + i * 44]==0)
{map[currentX + i * 44] = 1;}}
            if (Servo1CurrPosition >= 90){
                for (i = currentY ; i < y; i++)  if (map[currentX + i * 44]==0)
{map[currentX + i * 44] = 1;}}

            if (tempTargetRight[0] == 0){
                tempTargetRight[0] = x;
                tempTargetRight[1] = y;
            }
            map[x + y * 44] = 2; // its a target
        }
    }
}

// function to update the map due to left scan
// nextIndexForTargetLeft = 0, tempTargetLeft = 0,
void UpgradeLeft(){
    int x;//index
    int y;//index
    if (LeftScan < 150){
        x = currentX - (int) (RightScan * sin(180 - Servo1CurrPosition) * 0.1); // x
index of the target
        y = currentY + (int) (RightScan * cos(180 - Servo1CurrPosition) * 0.1);  // y
index of the target
        if (x <= 44 && y <= 44 && map[x + y * 44] == 0) { // the target is legit
            for (i = x + 1; i < currentX; i++)       if (map[i + currentY * 44]==0)
{map[i + currentY * 44] = 1;}

            if (Servo1CurrPosition > 90){
                for (i = currentY ; i < y; i++)  if (map[currentX + i * 44]==0)
{map[currentX + i * 44] = 1;}}
            if (Servo1CurrPosition <= 90){
                for (i = currentY ; i < y; i++)  if (map[currentX + i * 44]==0)
{map[currentX + i * 44] = 1;}}

            if (tempTargetLeft[0] == 0){
                tempTargetLeft[0] = x;
                tempTargetLeft[1] = y;
            }
            map[x + y * 44] = 2; // its a target
        }
    }
}

void ExitADCright(){                                    //temp do delet
    ADC0_SC2 |= ADC_SC2_ACFE_MASK;//compare mode
    ADC0_CV1=600;
}
void TurnOffADC(){
    PIT_TCTRL0 =0; //Disable timer
}
void TurnOnADC(){
    PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // Enable timer
}
```

```c
 * battery.c
#include "battery.h"

void BattLedConfig (){
    PORTE_PCR22 = PORT_PCR_MUX(1); // PTE22 pin ADC0_SE3- ALT1
    //
    PORTB_PCR8 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK; // PTB8 pin as GPIO
    PORTB_PCR9 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK; // PTB9 pin as GPIO
    PORTB_PCR10 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;// PTB10 pin as GPIO
    GPIOB_PDDR |= PORT_LOC(8) + PORT_LOC(9) + PORT_LOC(10); //Setup as output pin
}


// turn 3 LEDS *_*_*_* for battery level 1 to 8
void BattLedState (double BattSamp){// BattSamp = from 6.05 to 7.5 v
    float currentBatt = 1.23 - BattSamp;
    int levelBatt =8 - (currentBatt/0.21)*8; // 7.5-6.05=1.45v
    if (levelBatt & 0x001){
        TFC_BAT_LED0_ON;
    }else{
        TFC_BAT_LED0_OFF;
    }
    if(levelBatt & 0x010){
        TFC_BAT_LED1_ON;
    }else{
        TFC_BAT_LED1_OFF;
    }
    if(levelBatt & 0x100){
            TFC_BAT_LED2_ON;
    }else{
        TFC_BAT_LED2_OFF;
    }
    batteryLevel = levelBatt; // for global variable
}

void PrintBattery(){
    double temp = (batteryVOLT/1217.0);// 1217 is 1V
    BattLedState(temp);
    // then print battery level
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    UARTprintf(UART0_BASE_PTR,"\r\n battery level is : ");//logic to print the current
motor speed
    int k = 0;
    while (k==0){
        if (UART0_S1 & UART_S1_TDRE_MASK){
            UART0_D=(int)(batteryLevel+48); //X.0 DIGIT
            k++;
        }
    }
    UARTprintf(UART0_BASE_PTR,"\r\n \r\n");
    state = 0;
}
```

```c
#include "TFC.h"
#include "mcg.h"

#define MUDULO_REGISTER  0x493E //18,750
// set I/O for switches and LEDs
void InitGPIO()
{
    //SIM_SCGC6 |=  SIM_SCGC6_DAC0_MASK;
    //enable Clocks to all ports - page 206, enable clock to Ports
    SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTC_MASK |
SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;
    //DAC0_C0 |= DAC_C0_DACEN_MASK + DAC_C0_DACRFS_MASK+DAC_C0_LPEN_MASK; //enable

    //GPIO Configuration - LEDs - Output
        PORTD_PCR1 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK;  //Blue
        GPIOD_PDDR |= BLUE_LED_LOC; //Setup as output pin
        PORTB_PCR18 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK; //Red
        PORTB_PCR19 = PORT_PCR_MUX(1) | PORT_PCR_DSE_MASK; //Green
        GPIOB_PDDR |= RED_LED_LOC + GREEN_LED_LOC; //Setup as output pins
        RGB_LED_OFF;

}
//-----------------------------------------------------------------
// TPMx - Initialization
//-----------------------------------------------------------------
void InitTPM(char x){  // x={0,1,2}
    switch(x){
    case 0:
        TPM0_SC = 0; // to ensure that the counter is not running
        TPM0_SC |= TPM_SC_PS(5) + TPM_SC_TOIE_MASK; //Prescaler =32, up-mode,
counter-disable
        TPM0_MOD = MUDULO_REGISTER; // PWM frequency of 40Hz = 24MHz/(32x18,750)
        TPM0_C2SC |= TPM_CnSC_ELSA_MASK ; // PTC3
        TPM0_C3SC |= TPM_CnSC_ELSA_MASK ; // PTC4
        TPM0_CONF = TPM_CONF_DBGMODE_MASK;

        TPM0_SC |= TPM_SC_CMOD(1); //start the TPM0 PWM counter
        Servo2CurrPosition=180;
        enable_irq(INT_TPM0-16);
        set_irq_priority(INT_TPM0-16,2);
        break;

    case 1://Motor2 TPM1_CH0 PTE20 pin ALT3
        TPM1_SC = 0; // to ensure that the counter is not running
        TPM1_SC |= TPM_SC_PS(5); //Prescaler =32, up-mode, counter-disable
        TPM1_MOD = MUDULO_REGISTER; // PWM frequency of 40Hz = 24MHz/(32x18,750)
        TPM1_C0SC |= TPM_CnSC_MSB_MASK + TPM_CnSC_ELSB_MASK;
        TPM1_C0V = 0;// 0 speed
        TPM1_CONF = TPM_CONF_DBGMODE_MASK;

        Servo1CurrPosition=0;
        TPM1_SC |= TPM_SC_CMOD(1); //start the TPM1 PWM counter
        break;

    case 2: //Motor1 TPM2_CH1 PTE23 pin ALT3
        TPM2_SC = 0; // to ensure that the counter is not running
        TPM2_SC |= TPM_SC_PS(5); //Prescaler =32, up-mode, counter-disable
        TPM2_MOD = MUDULO_REGISTER; // PWM frequency of 40Hz = 24MHz/(32x18,750)
        TPM2_C1SC |= TPM_CnSC_MSB_MASK + TPM_CnSC_ELSB_MASK;
        TPM2_C1V = 0;//speed
        TPM2_CONF = TPM_CONF_DBGMODE_MASK;
        TPM2_SC |= TPM_SC_CMOD(1); //start the TPM2 PWM counter
        break;
    }
```

```
}
//-----------------------------------------------------------------
// TPMx - Clock Setup
//-----------------------------------------------------------------
void ClockSetup(){
        pll_init(8000000, LOW_POWER, CRYSTAL,4,24,MCGOUT); //Core Clock is now at 48MHz
using the 8MHZ Crystal
        //Clock Setup for the TPM requires a couple steps.
        //1st,  set the clock mux
        //See Page 124 of f the KL25 Sub-Family Reference Manual
        SIM_SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;// We Want MCGPLLCLK/2=24MHz (See Page
196 of the KL25 Sub-Family Reference Manual
        SIM_SOPT2 &= ~(SIM_SOPT2_TPMSRC_MASK);
        SIM_SOPT2 |= SIM_SOPT2_TPMSRC(1); //We want the MCGPLLCLK/2 (See Page 196 of
the KL25 Sub-Family Reference Manual
        //Enable the Clock to the TPM0,TPM1,TPM2 and PIT Modules
        //See Page 207 of f the KL25 Sub-Family Reference Manual
        SIM_SCGC6 |= SIM_SCGC6_TPM0_MASK +SIM_SCGC6_TPM1_MASK + SIM_SCGC6_TPM2_MASK;
        // TPM_clock = 24MHz , PIT_clock = 48MHz
}
```

```c
 * DistSens.c
#include "DistSens.h"
#include "TFC.h"
#include "mcg.h"

float
Distance1[130]={        3050.0  ,2980.0,2914.0, 2848.0, 2782.0, 2716.0,2657.0,2601.0,25
45.0,2489.0,2433.0,2376.0,2321.0,2266.0,2211.0,2156.0,  2103.0,2054.0,2005.0,1956.0,190
7.0,1865.0,1827.0,1789.0,1751.0,1713.0,1678.0,  1644.0,1610.0,1576.0,1542.0,1517.0,1494
.0,1471.0,1448.0,1425.0,1397.0,1371.0,  1345.0,1319.0,1293.0,1271.0,1253.0,1235.0,1217.
0,1199.0,1179.0,1160.0,1141.0   ,1122.0,1103.0,1091.0,1081.0,1071.0,1061.0,1051.0,1036.
0,1023.0,1010.0,997.0   ,984.0,972.0,962.0,952.0,942.0,932.0,919.0,908.0,897.0,886.0,87
5.0,869.0,863.0 ,857.0,851.0,845.0,840.0,836.0,832.0,828.0,824.0,819.0,817.0,815.0,813.
0,811.0 ,807.0,805.0,803.0,801.0,799.0,784.0,772.0,765.0,752.0,743.0,735.0,730.0,725.0
720.0,710.0,700.0,691.0,684.0,677.0,670.0,667.0,666.0,665.0,664.0,663.0 }; // IR 1
float
Distance2[130]={        3050.0  ,2980.0,2914.0, 2848.0, 2782.0, 2716.0,2657.0,2601.0,25
45.0,2489.0,2433.0,2376.0,2321.0,2266.0,2211.0,2156.0,  2103.0,2054.0,2005.0,1956.0,190
7.0,1865.0,1827.0,1789.0,1751.0,1713.0,1678.0,  1644.0,1610.0,1576.0,1542.0,1517.0,1494
.0,1471.0,1448.0,1425.0,1397.0,1371.0,  1345.0,1319.0,1293.0,1271.0,1253.0,1235.0,1217.
0,1199.0,1179.0,1160.0,1141.0   ,1122.0,1103.0,1091.0,1081.0,1071.0,1061.0,1051.0,1036.
0,1023.0,1010.0,997.0   ,984.0,972.0,962.0,952.0,942.0,932.0,919.0,908.0,897.0,886.0,87
5.0,869.0,863.0 ,857.0,851.0,845.0,840.0,836.0,832.0,828.0,824.0,819.0,817.0,815.0,813.
0,811.0 ,807.0,805.0,803.0,801.0,799.0,784.0,772.0,765.0,752.0,743.0,735.0,730.0,725.0
720.0,710.0,700.0,691.0,684.0,677.0,670.0,667.0,666.0,665.0,664.0,663.0 }; //IR 2
int startMeasure = 0;
int index = 0;

void DistSensConfig(){
    SIM_SOPT7 |= SIM_SOPT7_ADC0ALTTRGEN_MASK | SIM_SOPT7_ADC0PRETRGSEL_MASK |
SIM_SOPT7_ADC0TRGSEL(4);//set up the adc-b and connect to pit0
}

void DistSensStop(){
    //PIT_TCTRL0 &= ~PIT_TCTRL_TEN_MASK; // Disable timer
}

double DistanceMeasuring1 (int DisSamp1){
    int index1 = indexDistance1(DisSamp1);
    index1=index1+20;
    return index1;
}

double DistanceMeasuring2 (int DisSamp2){
    int index2 = indexDistance2(DisSamp2);
    index2=index2+20;
    return index2;
}

int indexDistance1 (int DisSamp1){
    int index1 = 0;
    while (index1<13&&Distance1[index1*10]>DisSamp1){
        index1++;
    }
    index1=index1*10;
    if(index1==130){index1--;}
    while(index1>0&&Distance1[index1]<DisSamp1){
        index1--;
    }
    return index1;
}
int indexDistance2 (int DisSamp2){
    int index2 = 0;
    while (index2<13&&Distance2[index2*10]>DisSamp2){
```

```
        index2++;
    }
    index2=index2*10;
    if(index2==130){index2--;}
    while(index2>0&&Distance2[index2]<DisSamp2){
        index2--;
    }
    return index2;
}

//-----------------------------------------------------------------
//  PORTD - ISR = Interrupt Service Routine
//-----------------------------------------------------------------
void PORTD_IRQHandler(void){
    volatile unsigned int i;
    while(!(GPIOD_PDIR & (SW_POS)));// wait of release the button
    // check that the interrupt was for switch
    if (PORTD_ISFR & SW_POS){
      if(index!=13){
          DistanceMeasure(index);
          index +=1;
      }
      else{
          startMeasure=0;
          index = 0;
      }
    }
    for(i=100000 ; i>0 ; i--); //delay, button debounce
    PORTD_ISFR |= 0x0000004;  // clear interrupt flag bit of PTD7
}

void DistanceForArray (){
    if(index!=14){
            DistanceMeasure(index);
            DistanceMeasureIR2(index);
         index +=1;
    }
    else {
        startMeasure=0;
        DistSensStop();
        index = 0;
    }
}

void DistanceMeasure (int i){
    int k=0;
    i = i*10;
    int temp = sens1;
    if(i==0){
        Distance1[i]=temp;
    }else{
      Distance1[i]=temp;
      temp = temp - Distance1[i-10];
      while(k<9){
          k++;
         Distance1[i-k] = Distance1[i+1-k]-(temp/10);
      }
    }
}

void DistanceMeasureIR2 (int i){
    int k=0;
    i = i*10;
    int temp = sens2;
```

```
    if(i==0){
        Distance2[i]=temp;
    }else{
      Distance2[i]=temp;
      temp = temp - Distance2[i-10];
      while(k<9){
          k++;
         Distance2[i-k] = Distance2[i+1-k]-(temp/10);
      }
    }
}

//-----------------------------------------------------------------
// PRESS 3,4 Function CASES
//-----------------------------------------------------------------
void Distance_Sensor(int mode){
    switch(mode){
        // start printing the distance
        case 1:
            PrintDistance();
            //Servo_Movement(1); // return to the front
            break;
    }
}
```

```c
# include "TFC.h"
# define  Turn1AngleTime 3.5 //need to check
# define  Turn2AngleTime 3.75 //need to check

int main(void){
    ClockSetup();
    InitGPIO();
    InitTPM(0);
    InitTPM(1);
    InitTPM(2);
    InitGPIO();
    InitUARTs();
    MotorConfig();
    adc_init();
    EncoderConfig();
    BattLedConfig ();
    pit_init();
    DistSensConfig();
    ServoConfig();
    printMenu(); // print the menu

    target=4 ; //to begin
    currentAngle = 90;
    distFromStartingPoint = 60; // change by hanan
    delay=180;
    while(delay>0);
    gameOn();
    while(1){
        wait();
    }
    return 0;
}
// TPM0 Function
void init_TPM0(int mode){
// Motor Movement Function
void Motor_Dir_Speed (int Direction, int Speed){
    currentSpeed=Speed;
    GPIOC_PDOR = 0;// stop motors
    if (Direction!=0){// Direction = 1 is forward , Direction = 2 is backward,
Direction =0 is stop
        wheel2Duty =  5526; //speed for motor 2
        TPM1_C0V = wheel2Duty;
        wheel1Duty =  5200; //speed for motor 1
        TPM2_C1V = wheel1Duty;

        if (Direction == 1){
            GPIOC_PDOR |= PORT_LOC(6);
            int i;
            for (i=8000; i>0; i--);
            GPIOC_PDOR |=PORT_LOC(10);//PTC10 direction forward for motor 2
        }
        else if (Direction == 2){
            GPIOC_PDOR |= PORT_LOC(5);
            GPIOC_PDOR |= PORT_LOC(7);//PTC7 direction backward for motor 2
        }
    }
    if(Speed==0){
        wheel1Speed=0;
        wheel2Speed=0;
        TPM1_C0V=0;
        delay=50;
        while(delay>0);
        TPM2_C1V=0;
    }
```

```
    else init_TPM0(1);
}
//the car at start in angle 90'
void turnToAngle(int angle){
//  Function to Print the speed
void PrintSpeed(){
// PRESS 1 Function CASES
void Motor_Movement(int mode){
// Encoder Sensing to calculate the speed of the motors
void EncoderSensing(){
// TPM0- ISR = Interrupt Service Routine
void FTM0_IRQHandler(){

void gameOn(){
    EnterArena();
    while(target>0){
        delay=30;
        while(delay>0);
        while(target>0){
        WalkAway();
        }
    }
    DistLeft=100;
    Motor_Dir_Speed(1,8);
    while(DistLeft>0);
    Motor_Dir_Speed(0,0);
    delay=30;
    while(delay>0);
    ExitArena();
    stopAll();
}
void EnterArena(){//NEED TO DO SAVE THR CAR BEFOR INTARE
    // servo to 90 degree
    DistLeft = distFromStartingPoint + 150; // 1.5m from the starting point
    Motor_Dir_Speed(1,8);//start moving forward
    while(DistLeft > 0);
    Motor_Dir_Speed(0,0);//stop
    delay=10;
    while(delay>0);
    currentY = 1.5;
}
void ExitArena(){//                                          looking for the Exit
    RightExitconfig();//get the servo 1 to 90
    Motor_Dir_Speed(1,5);
    DistLeft=220;//
    while(DistLeft>0){
        if(sens2<800){//
            DistLeft=0;
            TurnOffADC();
            ExitRight=1;
        }
    }
    Motor_Dir_Speed(0,0);
    delay=3;
    while(delay>0);
    if(ExitRight==1){
        Motor_Dir_Speed(1,1);
        DistLeft=40;
        while(DistLeft>0);
        Motor_Dir_Speed(0,0);
        delay=5;
        while(delay>0);
        turnToAngle(90);
        delay=5;
```

```c
        while(delay>0);
        Motor_Dir_Speed(1,5);
        DistLeft=200;//
        while(DistLeft>0);
        Motor_Dir_Speed(0,0);
    }else{

        leftExitconfig();
        Motor_Dir_Speed(1,5);
        DistLeft=1000;//
        while(DistLeft>0){
            if(sens1<600){//
                DistLeft=0;
                TurnOffADC();
                ExitLeft=1;
            }
        }
        Motor_Dir_Speed(0,0);
        delay=5;
        while(delay>0);
        if(ExitLeft==1){
            Motor_Dir_Speed(1,1);
            DistLeft=40;
            while(DistLeft>0);
            Motor_Dir_Speed(0,0);
            delay=5;
            while(delay>0);
            turnToAngle(90);
            delay=5;
            while(delay>0);
            Motor_Dir_Speed(1,5);
            DistLeft=200;
            while(DistLeft>0);
            Motor_Dir_Speed(0,0);
        }
    }
}
void WalkAway(){//                                    Search targets function
    BLUE_LED_ON;
    int OldTarget=target;
    CheckWhereTheServo(90);//the servoes is equal
    Motor_Dir_Speed(1,10);//
    delay=10;
    while(delay>0);
    Yflag=1;
    while(!(sens1>2000||sens2>2000));//
    delay=20;
    while(delay>0);
    BLUE_LED_OFF;
    RED_LED_ON;
    Yflag=0;
    Motor_Dir_Speed(0,0);
    delay=20;
    while(delay>0);
    int Objectright=DistanceMeasuring1(sens1);
    int Objectleft =DistanceMeasuring2(sens2);
    if(Objectleft<120){//
        RGB_LED_OFF;
        GREEN_LED_ON;
        GetObjectLeft();
    }
    if(Objectright<120){
        RGB_LED_OFF;
        GREEN_LED_ON;
```

```
        RED_LED_ON;
        GetObjectRight();
    }
    if(OldTarget!=target){
        DistLeft=40;//                                  the length of box//
        while(DistLeft>0);
    }
    RGB_LED_OFF;
}
void GetObjectLeft(){//                                   go to the Object in the left
side
    turnToAngle(0);
    Straightconfig();//get the servo to 0 and 180
    delay=20;
    while(delay>0);
    if(sens1>700 || sens2>700){
        DistLeft=fmax(DistanceMeasuring1(sens1),DistanceMeasuring2(sens2));
        int temp=DistLeft;
        Motor_Dir_Speed(1,8);
        while(DistLeft>0);
        Motor_Dir_Speed(0,0);
        delay=20;
        while(delay>0);
        DistLeft=temp;
        Motor_Dir_Speed(2,8);
        while(DistLeft>0);
        target--;
    }
    Motor_Dir_Speed(0,0);
    delay=20;
    while(delay>0);
    turnToAngle(90);
    CheckWhereTheServo(90);//get the servo to 90
    RGB_LED_OFF;
    delay=20;
    while(delay>0);
}
void GetObjectRight(){//                                  go to the Object in the right
side
    turnToAngle(180);
    Straightconfig();//get the servo to 0 and 180
    delay=20;
    while(delay>0);
    if(sens1>700||sens2>700){
        DistLeft=fmax(DistanceMeasuring1(sens1),DistanceMeasuring2(sens2));
        int temp=DistLeft;
        Motor_Dir_Speed(1,8);
        while(DistLeft>0);
        Motor_Dir_Speed(0,0);
        delay=20;
        while(delay>0);
        DistLeft=temp;
        Motor_Dir_Speed(2,8);
        while(DistLeft>0);
        target--;
    }
    Motor_Dir_Speed(0,0);
    delay=20;
    while(delay>0);
    turnToAngle(90);
    CheckWhereTheServo(90);//get the servo to 90
    RGB_LED_OFF;
    delay=20;
    while(delay>0);
```

```
}
void stopAll(){
    stop();
}
```

```c
#include "TFC.h"
#include "mcg.h"
#define MUDULO_REGISTER  0x2EE0

// motor - MotorConfig
void MotorConfig(){
        //GPIO Configuration  - Output
        // Motor 1 direction
        PORTC_PCR5 = PORT_PCR_MUX(1); // assign PTC5
        GPIOC_PDDR |= PORT_LOC(5);  // PTC5 is Output
        PORTC_PCR6 = PORT_PCR_MUX(1); // assign PTC6
        GPIOC_PDDR |= PORT_LOC(6);  // PTC6 is Output
        // Motor 2 direction
        PORTC_PCR7 = PORT_PCR_MUX(1); // assign PTC5
        GPIOC_PDDR |= PORT_LOC(7);  // PTC7 is Output
        PORTC_PCR10 = PORT_PCR_MUX(1); // assign PTC6
        GPIOC_PDDR |= PORT_LOC(10);  // PTC10 is Output
        // Motor 1 enable
        PORTE_PCR23 = PORT_PCR_MUX(3); // PTE23 pin TPM2_CH1- ALT3
        // Motor 2 enable
        PORTE_PCR20 = PORT_PCR_MUX(3); // PTE20 pin TPM1_CH0- ALT3
}

void EncoderConfig(){
    //PORTD_PCR1 = 0; // PTD1(TPM0_CH1) - output
    //PORTD_PCR2 = 0; // PTD2(TPM0_CH2) - inputCapture
    //PORTD_PCR3 = 0; // PTD3(TPM0_CH3) - inputCapture
    PORTC_PCR3 = PORT_PCR_MUX(4); // assign PTC3 as TPM0_CH2
    PORTC_PCR4 = PORT_PCR_MUX(4); // assign PTC4 as TPM0_CH3
}

void EncoderChecker(){//check the encoders for high accuracy only straight
    if(wheel1Speed > 35){
           wheel1Duty = wheel1Duty - 45;
           //wheel2Duty = wheel2Duty + (wheel2Duty/wheel2Speed)*0.3;
          // TPM1_C0V = wheel2Duty;
           TPM2_C1V = wheel1Duty;
    }else{
           wheel1Duty = wheel1Duty + 45;
           TPM2_C1V = wheel1Duty;
    }
    if(wheel2Speed > 35){
           wheel2Duty = wheel2Duty - 45;
          // wheel1Duty = wheel1Duty + (wheel1Duty/wheel1Speed)*0.3;
           TPM1_C0V = wheel2Duty;
          // TPM2_C1V = wheel1Duty;
    }else{
           wheel2Duty = wheel2Duty + 45;
           TPM1_C0V = wheel2Duty;
    }
}
```