סדנה: סדנת תנועה רובוטית 2011

פרויקט: סירה רובוטית **(ROBOaTAU**)

צוות הפרויקט:

* Ziv Ramati
* Vova Perelman
* ,Anton Nepomnyashchiy
* Dror Laufer
* Itamar Gilad
* Dan Yaari

מרצה:

* Dan Halperin

**מבוא**

מטרת הפרויקט הייתה לתכנן ולבנות מערכת חומרה ותוכנה של סירה רובוטית.

כבסיס לחומרת הסירה נרכשה סירת MicroMagic של חברת Graupner שהיוותה בסיס עליה הורכבו רכיבי תקשורת, לוגיקה וחיישנים, זאת בהתאם לדרישות התחרות – אליפות העולם בסירות רובוטיות.

תוכנת הסירה שפותחה כוללת אלגוריתמים להשטת הסירה תוך כדי המנעות ממכשולים סטאטים ודינאמים, כמו כן נעשתה התאמה של סימולטור בשם TrackSail לצורך בחינת התוכנה והאלגוריתמם ללא תלות בסירה.

תוך כדי פיתוח הפרויקט הצוות צבר ידע בסיסי בתחום השייט (חלקו ע"י הרצאות שהועברו במהלך הסדנה וחלקו באופן עצמאי ע"י מקורות באינטרנט, התנסות עצמית ע"י השטות של הסירה באגם שבפארק הירקון וכו.).

השאיפה היתה שבסוף הפרויקט הסירה תתמודד בתחרות בינלאומית של השטת סירות רובוטיות שנערכת מדי שנה בחודש אוגוסט (השנה בגרמניה), לכן לפרויקט נוסף גם אלמנט של זמן ותחרותיות, בנוסף לצורך לבצע התאמות לחוקי התחרות ועבודה עם רכיבים חיצוניים (WorldServer).

מטרת מסמך זה היא לתת סקירה כללית של כלל רכיבי הפרויקט.

**חלק א': ארכיטקטורה כללית**



הרעיון הכללי של הארכיטקטורה היה שבשלב ראשון יפותח החלק התוכנתי/אלגוריתמי, שיבדקו על-גבי הסימולטור כסביבת בדיקות. בשלב מאוחר יותרתחובר הספינה לרכיבים שנכתבו עם מינימום שינויים, והסירה המסומלצת והסימולטור יוחלפו בסירה האמיתית והאגם.

בהתאם לכך שניתן, קיימים 2 מצבי עבודה:

1. עבודה מול סירה מסומלצת:



1. עבודה מול סירה אמיתית:



כאשר הרכיבים העיקריים הם:

**ShoreLogic –** אפליקצית השליטה בסירה, מקבלת מידע מסירה אמיתית או מסומלצת ובהתאם לאלגוריתם, מאפייני המירוץ וסוג הסירה מקבלת החלטות תנועה ושולחת אותם בחזרה לסירה. בנוסף, במצב של סירה אמיתית מתבצע עדכון של ה-WorldServer לגבי נתוני הסירה כמתבקש בחוקי התחרות.

רציונאל הפרויקט הוא שזה הרכיב היחיד שמכיר את כל הרכיבים האחרים, ואילו כל רכיב אחר בפרויקט מכיר רק את ה-SL כדי להקטין תלות בין רכיבים ולקבלת מודלריות מירבית המאפשרת גם החלפת כל חלק במערכת מבלי לשכתב את כולה.

**WorldServer –** שרת התחרות, המקבל מכל סירה את נתוני ה-GPS שלה ומאפשר לכל סירה לתשאל נתוני ה-GPS של סירות אחרות ומכשולים דינאמים נוספים.

**Real Boat –** הסירה האמיתית. שולחת נתוני חיישנים (רוח, מצפן, מיקום, מרחק ממכשולים) אותם היא מייצרת ע"י רכיבי החומרה ומקבלת פקודות להזזת ה-Rudder והמפרש הראשני והמשני.

**Race Simulator Server –** סימולטור המדמה שיוט סירה בתנאים פיזיקלים הדומים לעולם האמיתי. משמש כסביבת Debug לפרויקט, בדומה לסירה האמיתית שולח נתוני חיישנים (רוח, מצפן, מיקום, מרחק ממכשולים) אותם הוא מסמלץ וקבל פקודות להזזת ה-Rudder והמפרשים.

**Manual On-Race Instructions –** במהלך התחרות מוצבים ה-Waypoints באופן דינאמי בהתאם לכיוון הרוח המשתנה. בשל כך יש צורך בדרך נוחה ומהירה לתת פרמטרים ללא ביצוע Compilation מחדש של כל הפרויקט, אלא ע"י מנגנון קונפיגורציה חיצוני.

1. **אפליקציית השליטה (ShoreLogic)**

**סביבת פיתוח:** C++

**מבנה כללי:**



נתאר כעת את הרכיבים ביתר פירוט:

**ShoreLogic**

**פונקציות עיקריות:**

Run:

* קריאה ל-Adapter המתאים (סירה/סימולטור) לקריאת נתוני ה-Waypoints במסלול.

הראשון מקבל אותם מקובץ קונפיגורציה והשני מהסימולטור דרך TCP Socket.

* הפעולות הבאות מתבצעות בלולאה עד סיום המירוץ
  + בדיקה האם יש קלט שליטה ידנית (אם כן, ביצוע שלו)
  + קריאת נתוני חיישנים ע"י ה-Adapter המתאים (סירה/סימולטור)
  + קריאת נתוני מכשולים דינאמים/סטטים (מה-Worldserver במקרה של סירה אמיתית, או ע"י פונקציה המייצרת אותם במקרה של סירה מדומה)
  + בדיקה אם סיימנו אם המירוץ (אם לא, ממשיכים)
  + קריאה ל-CalcMovementParams לחישוב נתוני ניווט הסירה.
  + שליחת נתוני ניווט הסירה ע"י ה-Adapter המתאים (סירה/סימולטור).
  + עדכון ה-WorldServer בנתוני הסירה (במקרה של סירה אמיתית).

CalcMovementParams: אחראי על לוגקית ניווט הסירה. מבצע את הפעולות הבאות:

* התאמות למירוץ הספציפי
* הפעלת האלגוריתם על נתוני החיישנים לקבלת כיוון התקדמות רצוי
* בדיקת מצב סוללה (לסירה אמיתית)
* הדפסת מתוני DEBUG ל-LOG
* חישוב תיקון כיוון לפי כיוון/עוצמת רוח ע"י קריאה ל-Adapter המתאים (סירה/סימולטור)
* חישוב כיווני המפרשים לפי כיוון/עוצמת רוח
* חישוב כיוון ה-Rudder לפי סטיית המסלול הנוכחי מהמסלול הרצוי
* כיוונונים ספציפים של ה-Rudder לסימולטור (לא לזוז בזמן תנועה, מעבר בשער וכו.)
* כיוונונים ספציפים לסירה האמיתית (לוגיקה ליום עם רוח חלשה וכו.)

CalcNoWindMovements: לוגיקת תנועה ליום עם רוח חלשה במיוחד (פותח בזמן התחרות)

PostProcessFetchedData: בדיקה לפי נתוני החיישנים החדשים שקיבלנו, האם עברנו את ה-Waypoint הנוכחי.

CalcSail1\_ByWind: חישוב ערך מתאים למפרש הראשי לפי כיוון וחוזק הרוח

CalcSail2\_ByRouteDeviationAndWind: חישוב ערך מתאים למפרש המשני לפי סטיית כיוון הרוח מכיוון ההפלגה הנוכחי.

CalcRudder\_ByError: חישוב ערך מתאים ל-Rudder הסירה, בהתאם לסטיה נוכחית של כיוון הסירה מהכיוון הרצוי.

AdjustWaypointsForRace: לוגיקת טיפול במירוצים ספציפים לתחרות (כגון Station-Keeping)

HandleManualControl: שליטה ידנית באמצעות המקלדת המאפשרת:

* מעבר בין מצב שליטה ידני/אוטומטי בסירה
* שליטה ברמת הפירוט של כתתיבת הודעות ל-LOG (Debug Level 0-3)
* הוספת סמן (Mark) ל-Log המכיל את מיקום הסירה הנוכחית, נתוני החיישנים והערה שניתנה.
* מעבר ל-Waypoint הקודם/הבא.
* סיום האפליקציה.

**BoatAdapter**

**פונקציות עיקריות:**

InitConnection: אתחול Connection ל-Serial COM Port עם פרמטרים מתאימים ועם Flush מיידי.

SendRemoteCommandsData: שליחת פיקודים לסירת הסימולטור דרך Serial COM Port.

SendRemoteGetControl: שליחה פקודה יחידה ללא תוכן (כגון שינוי מצב ידני/אוטומטי וכו.)

FetchRemoteSensorsData: קריאת נתוני החיישנים שיוצרו בסירה ע"י Serial COM Port.

SetBoatVariablesFromSensors: ביצוע Parsing של המידע המגיע מהסירה לתוך מבנה-נתונים.

FetchWaypointsList: קריאת רשימת ה-Waypoints מרשימה נתונה (שהגיעה בשלב מוקדם יותר מה-WS וה-Configuration) ועיבור פי סוג Waypoint ספציפי. מקבלים את סוגי ה-Waypoints הבאים:

* WAYPOINT – נקודה שיש להגיע אליה ולעבור בתוכה (מייצגת מרכז של שער המורכב מ-2 נק')
* BOUEY – נקודת ציון שיש להקיף מכיוון מוגדר (ימין/שמאל)
* DELAYED – נקודה שכאשר עוברים אותה יש לחכות כמות מוגדרת של זמן לפני שממשיכים לנק' הבאה
* OBSTACLE – מכשול שיש להמנע מלהתקרב אליו

FixRoute\_ByWind: תיקונים לכיוון ההתקדמות של הסירה לפי כיוון ועוצמת הרוח (ספציפי לסירה אמיתית).

**SimulatorAdapter**

**פונקציות עיקריות:**

FetchRemoteSensorsData: קריאת נתוני החיישנים שסומלצו בסימולטור ע"י TCP Socket

SendRemoteCommandsData: שליחת פיקודים לסירת הסימולטור דרך TCP Socket

FetchWaypointsList: קריאת רשימת ה-Waypoints במירוץ מהסימולטור דרך TCP Socket

ועיבוד לפי סוג Waypoint ספציפי. מקבלים את סוגי ה-Waypoints הבאים:

* WAYPOINT – נקודה שיש להגיע אליה ולעבור בתוכה (מייצגת מרכז של שער המורכב מ-2 נק')
* BOUEY – נקודת ציון שיש להקיף מכיוון מוגדר (ימין/שמאל)
* DELAYED – נקודה שכאשר עוברים אותה יש לחכות כמות מוגדרת של זמן לפני שממשיכים לנק' הבאה
* OBSTACLE – מכשול שיש להמנע מלהתקרב אליו

הערה: הפונקציה מבצעת המרות Little/Big Endian מתאימות לפי הצורך שכן JAVA ו-CPP עובדים ב-Endianess שונים.

SimulateDynamicObstacles: פונקציה המייצרת מגוון סוגים של מכשולים דינאמים, מחשבת את התנועה שלהם על המפה ושולחת אותם לסימולטור לתצוגה.

SendSLWaypointsData: שולחת את ה-Waypoints המקוריים ואלו שיוצרו לאחר עיבוד המקוריים (לסימולטור לתצוגה).

FixRoute\_ByWind: תיקונים לכיוון ההתקדמות של הסירה לפי כיוון ועוצמת הרוח (ספציפי לסירה מסומלצת).

**WorldServerAdapter**



מכיוון שה-WorldServer הוא רכיב (JAVA) חיצוני לפרויקט שסופק ע"י צוות התחרות, הוחלט שה-Adapter שעובד מולו ירוץ ב-Thread נפרד, כדי שלא יפריע ללוגיקת ה-SL הסטנדרטית. כתוצאה מכך נוספה לוגיקת סנכרון מול ה-Thread הראשי של ה-SL באמצעות Mutexים רלוונטים.

**פונקציות עיקריות:**

UpdateWorldServer: פונקציה הנקראת מה-Thread של SL ומעדכנת את ה-WS בצורה בטוחה.

SafeGetWaypoints: משמשת לקריאת רישמת ה-Waypoints והמכשולים הסטאטים ממבנה הנתונים הפנימי של ה-WS Adapter ע"י ה-SL.

SafeGetBoats: משמשת לקריאת רישמת הסירות (מכשולים דינאמים) ממבנה הנתונים הפנימי של ה-WS Adapter ע"י ה-SL.

GenerateUniqueID: בניית מזהה חד-ערכי לסירה (יופיע על מפת הסירות התחרות)

GetData: קריאת רשימת אובייקטים מה-WS לתוך וקטור של Map Objects.

קיימים מספר סוגים של אובייקטים מתקבלים:

* Buoy - Waypoint בתחרות
* SailBoat – סירה (מכשול דינאמי)
* CircularObstacle – מכשול סטאטי

SendData: שליחת נתוני הסירה הנוכחת ל-WS

Run: מריצה לולאה אינסופית של שליחת נתונים ל-WS וקריאת מידע ממנו למבני-נתונים בתוך ה-Adapter שהגישה אליהם מתבצעת בצורה Thread-Safe ע"י ה-SL.

**PotentialField/CollisionAlgorithm**

**פונקציות עיקריות:**

getDirectionInRadians: מחזירה כיוון השיוט האופטימלי בהתבסס על אלגוריתם שדה הפוטנציאל (הסבר מפורט על האלגוריתם ינתן בהמשך המסמך).

getDirectionInRadiansCircle: מחזירה כיוון שיוט אופטימלי עפ"י אלגוריתם בחירת הכיוון שמתבסס על הטלת המכשולים על מעגל סביב הסירה.

**קבועים עיקריים:**

* קבוע המשיכה של היעד וקבוע הדחייה של המכשולים. משפיעים על ה- Clearance, המרחק שהסירה תנסה לשמור מהסירות האחרות.

**Misc**

מחלקת שירות כללית המכילה קבועים, הגדרות Macro ופונקציות הנמצאות בשימוש ע"י שאר חלקי הפרויקט.

**סוגי פונקציות עיקריות:**

* Endianess: פונקציות לטיפול בהמרות Little/Big Endian בעבודה מול רשת תקשורת
* Convertion Functions: פונקציות להמרות מטיפוס אחד לשני (String to Double וכו.)
* Log: פונקציות לעבודה מול LOG המערכת
* Bounds/Wrapping: פונקציות לקביעת גבול עליון/תחתון בחישובים
* String Macroes: עבודה בטוחה עם Strings

**טיפוסי נתונים עיקריים:**

* נקודה
* סוגי Waypoint בפרויקט

**קבועים עיקריים:**

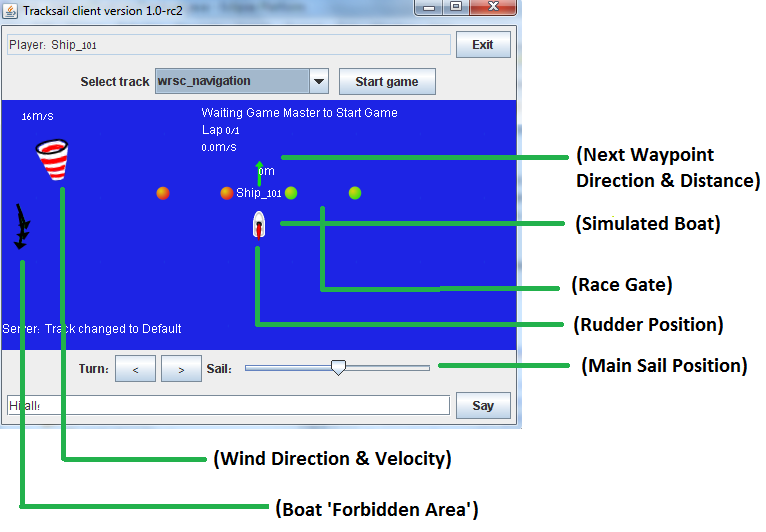
* קבועים מתמטים (פאי, המרות רדיאנים וכו.)
* מיקום קובץ ה-LOG בפרויקט
* הגדרה האם הפרויקט רץ כרגע במצב SIMULATOR או REAL-BOAT.

**2. אפליקציית הסימולטור (TrackSail)**

**סביבת פיתוח:** JAVA

**תיאור כללי:** החלטנו להשתמש בפרויקט בסימולטור Open-Source מהאינטרנט בשם TrackSail, המממש שייט בתנאים פיזיקלים דומים למציאות. לצורך התחרות ופיתוח האלגוריתם נדרשנו להוסיף לסימולטור התאמות ושינויים רבים אותם נתאר כאן.

**מבנה כללי:**



**התאמות שביצענו לסימולטור:**

* Porting ממערכת הפעל LINUX ל-Windows
* סימלוץ של נתוני GPS כיוון ממצפן, וכיוון/עוצמת הרוח ויכולת תשאול שלהם מרחוק,

ע"י ה-SL דרך TCP Socket)

* סימלוץ של Motion Sensors, כאשר מוצבים 4 חיישנים מדומים (ימין-שמאל-קדימה-אחורה).

החיישנים בודקים האם יש סירה בזווית מפתח של 45 מעלות, ואם כן מחזירים Min של המרחק לאובייקט הקצר ביותר.

* הוספת תמיכה ב-MultiPlayer (כלומר מירוץ של מספר רב של סירות כאשר כל סירה קשורה דרך TCP PORT נפרד לעותק ייחודי של אפליקצית ShoreLogic המנווטת אותה.
* מספר התאמות לצרכי Debugging ואינטגרציה עם האלגוריתם
  + ציור המכשולים הדינאמים המיוצרים ב-SL ב-Simulator
  + הוספת חיווי על Waypoints שכבר עברנו (שינוי צבע)
  + הוספת חיווי גרפי על המסלול שהאלגוריתם מתכנן שנמשיך אליו בזמן הקרוב
* מגוון התאמות נדרשות עבור התחרות שהופיעו במסמך WRSC, בינהם:
  + הוספת 3 סוגים חדשים של Waypoints מעבר ל-1 (הראשון) שהיה קיים קודם:
    - Gate (שער שמורכב מ-2 מצופים והסירה צריכה לעבור בינהם)
    - Bouey (מצוף יחיד שהסירה צריכה מימין או שמאל)
    - Obstacle (מכשול סטטי שהסירה צריכה לעבור ולא להתקל בו)
    - Delayed Gate (שער שלאחר מעבר בו צריך לחכות באזור מסוים זמן קבוע לפני שממשיכים ל-Waypoint הבא).

לצורך כך שינינו את מבנה קבצי ה-track, ועכשיו הם חייבים להכיל גם שדה TYPE ושדה PARAM שמכיל את הערך הניתן לקנפוג לכל waypoint.

* כדי לדמות בצורה קרובה למציאות את התחרות בנינו3 מסלולים חדשים:
  + wrsc\_navigation - תחרות ניווט מתחילים ב-waypoint שער, מקיפים waypoint

Bouey מימין וחוזרים.

* + wrsc\_station\_keeping - תחרות station keeping, מתחילים ב-waypoint שער,

מחכים X זמן ועוברים דרך waypoint מושהה.

* + wrsc\_collision\_avoidance - המנעות ממכשולים, מתחילים ב-waypoint שער ,עוברים obstacle ומקיפים bouey.
* הוגדר קובץ Configuration המאפשר לקנפג את מידע ה-Param (כגון מאיזה כיוון להקיף Bouey, כמה זמן לחכות ב-Delayed Gate וכו.), מידע המתקבל בזמן תחילת התחרות באופן ידני (ולא מגיע כלל מה-WorldServer).
* שליחה של נתוני המסלול מה-Simulator Server ל-Word Server כדי לדמות מצב אמת כמו בתחרות.

1. **מבנה ה-WorldServer**

**סביבת פיתוח:** JAVA

**תיאור כללי:** את ה-WorldServer קיבלנו מאתר התחרות כרכיב סגור. נדרשנו לעבור על הקוד, ללמוד אותו ולפתח פרוטוקול תקשורת פשוט בינו לבין ה-SL.

ה-WS מקבל מכל סירה את מיקום ה-GPS שלה, ומספק לכל סירה את מיקומי הסירות האחרות (מכשולים דינאמים) ובחלק מהתחרויות גם מכשולים סטאטים (Obstacles).

**חלק ב': תיאור אלגוריתם**

**רקע כללי (מה האלגוריתם אמור לעשות ולמה זה טוב)**

המטרה הכללית של אלגוריתמי שיוט הינה למצוא פתרון לבעיית האופטימיציה של מציאת מסלול בין הסירה ליעד כלשהו כאשר למסלול ישנן מספר הגבלות, ומשקל של כל הגבלה הוא הפרמטרים שלנו. ההגבלות באופן כללי הן: המסלול חייב להיות נקי מהתנגשויות כמו בכל מסלול של תיכנון תנועה. אולם בנוסף לכך נכנסים גם פרמטרים של כמות מינימלית של פניות, מרחק מקסימלי למכשולים כאשר במקרה שלנו המכשולים היו דינאמיים בין היתר.

המטרה הכללית של האלגוריתמים הנ"ל היא לאפשר ללא מגע אדם להגיע ממקור ליעד דרך תווך ימי, כלומר כלי שיט בלתי מיואשים, וזאת במינימיזציה של הסיכונים הנילווים, הזמן והעלות הנדרשים לכך.

**אלגוריתמים נוספים שבדקנו ולא היו מתאימים (Cone, RRT וכו.)**

לפני שבחרנו באלגוריתם מבוסס פוטנציאל לצורך מניעת התנגשויות בדקנו גם אלגוריתמים נוספים. בשניים מהם התמקדנו במיוחד, שכן הם נראו לנו הכי מבטיחים.

הראשון הוא אלגוריתם של Jur van der Berg המאפשר הימנעות ממכשולים דינמיים בדו מימד, שעלולים לנוע לכל כיוון. האלגוריתם מציע לייצג את הזמן כציר אנכי ולמעשה לתכנן תנועה בתלת מימד, כאשר חייבים תמיד לעלות בקצב אחיד. כדי לייצג את המכשולים, סביב אמצע כל מכשול נצייר מעגל שיגדל בקצב ששווה למהירות המרבית של מכשול זה. בתלת מימד, נקבל קונוסים במקום המכשולים. לאלגוריתם יש מספר יתרונות חשובים: הוא גם מחזיר מסלול חלק (כי למעשה המסלול מורכב מישרים העוברים בין הקונסים ומסלולים על שפת הקונסים שניתן לייצג באמצעות ספירלות) ויותר חשוב, אין צורך לנסות ולנבא את המסלולים של הסירות האחרות.

אף על פי כן, גילינו לבסוף שהאלגוריתם פחות מתאים למקרה שלנו. הסיבה הראשונה היא שאחת הדרישות של האלגוריתם היא שמהירות הרובוט (כלומר הסירה שלנו) תהייה תמיד לפחות כמהירות המכשולים (הסירות האחרות). הבעיה היא שבתנאיי רוח משתנים ועקב תופעות כגון גניבת רוח, אין ביכולתנו לספק תנאי בסיסי זה. בעיה נוספת נובעת מכך שהאלגוריתם יכול להחזיר כי בכלל לא קיים מסלול חוקי בין ההתחלה ליעד, ואז יש צורך באלגוריתם קבלת החלטות נוסף. כמו כן, במהלך ריצת האלגוריתם יש צורך במציאת פתרונות נומריים של משוואות, מה שמכביד על זמן הריצה. לבסוף, נתקלנו בבעיה לשלב בין האלגוריתם הזה לתמרונים שעל הסירה לבצע. ייתכן שהמסלול שהאלגוריתם מחזיר עובר בקרבת סירות אחרות, ואם נרצה למשל לבצע תמרונים הכרחיים כדי להמשיך לנוע לכיוון הנכון כגון ה- Tack וה- Jibe תתכן התנגשות (הרי האלגוריתם מניח שאין לרובוט בעיה לנוע על קו ישר לכיוון מסוים).

האלגוריתם השני שבדקנו היה ה- RRT שמאוד דומה לאלגוריתם ה PRM. ההבדל בין שני האלגוריתם הוא שלעומת ה- PRM שבוחר צמתים באקראי ומחבר ביניהם, ה- RRT מתחיל מצומת נתון ומפתח ממנו עץ מסלולים חוקי.

ה- RRT מתחיל מגרף שמכיל צומת בודד – המקום ממנו התחלנו, ומבצע את התהליך הבא עבור מספר גדול של צעדים ומקבל עץ מסלולים: בכל שלב נבחר צומת אקראי מהעץ שכבר ברשותנו. כמו כן, נגריל כיוון תנועה ומרחק תנועה אקראיים וננסה לנוע מהצומת שבחרנו בכיוון שבחרנו. אם התנעה היא חוקית (מצד אחד לא מתנגשת באף מכשול ומצד שני חוקית מבחינת הרוח) נוסיף את נקודת הסיום כצומת לעץ וכן קשת בינו לצומת האקראי שבחרנו.

כעת על העץ שבידינו נוכל להפעיל כל אלגוריתם יעיל למציאת מסלולים קצרים ביותר. ניתן לכל קשת משקל לפי האורך שלה, המהירות המרבית שהסירה תוכל לצבור (לפי הרוח), שינוי הכיוון מהצומת ההתחלתי (נעדיף שהסירה תשנה את כיוונה כמה שפחות, שכן כל שינוי כיוון גורם לאיבוד מהירות) ומרחקה ממכשולים. כמו כן נעדיף מסלולים שמתקרבים ליעד.

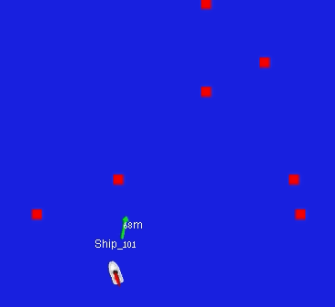
לשיטת ה-RRT הרבה יתרונות. ביניהם זמן ריצה מהיר, אפשרות לפונקצית משקל שכוללת פרמטרים רבים. כמו כן, למשל באמצעות החלקה של המסלול המתקבל יכולנו למנוע מהסירה לעשות תמרונים מיותרים. בנוסף, עקב היכולת לפסול כיווני תנועה בגלל רוח האלגוריתם החזיר מסלולים שכבר כוללים תמרוני Tack.

מצד שני, נתקלנו גם בחסרונות של השיטה. אחד החסרונות הוא העובדה שהאלגוריתם מיעוד להימנעות ממכשולים סטטיים. כדי להתחמק מחסרון זה החלטנו לבנות מסלול עבור המיקום הנוכחי של המכשולים אך לבנות מסלולים חדשים בתדירות גבוהה מספיק כדי שנבחין בתנועה של מכשולים. הבעיה היא שה- RRT כאלגוריתם אקראי נוהג להחזיר מסלולים שונים זה מזה אפילו בהרצה באותם התנאים בדיוק. כדי למנוע תופעה זו, החלטנו לשמר את המסלול ההתחלתי במידה והוא עדיין חוקי מבחינת המכשולים הרוח.

אך הבעיה העיקרית של ה-RRT היא העובדה שהוא מחזיר מסלול. במקרה של סירה ישנן כמה בעיות עם גישה של להיצמד למסלול. הבעיה העיקרית היא שיכולת השליטה בסירה מסוג זה אינה אופטימלית אף עבור שייט מיומן. הסירה אינה יכולה להסתובב במקום (ובמקרים רבים על מנת להסתובב היא צריכה מהירות התחלתית אשר פרופורציונלית ל: מהירות הרוח , עוצמת הרוח וזווית השיוט), אין לה את היכולת לעצור במקום, והיא נסחפת עם הרוח והזרם. עקב כך, ברוב המקרים הסירה פשוט לא יכולה להיצמד למסלול. מכאן שתי בעיות: הראשונה היא שסירה עלולה להיסחף לתוך אחד המכשולים (כי ה-RRT מבטיח רק שהמסלול שהוא מחזיר לא מתנגש במכשולים). הבעיה השנייה היא שאפילו אם נמנענו מהתנגשות אך נסחפנו, כאשר ננסה לחזור למסלול הסירה עלולה להתנגש באחד המכשולים. בעיה נוספת היא ההחלטה האם על האלגוריתם להחליט לחשב מסלול חדש. ייתכן שהסטייה הייתה קטנה וקל לחזור למסלול, אך ייתכן שעקב תנאיי רוח, עדיף לחשב מסלול חדש מאשר לנסות לחזור לקודם וכאן טמונה תורה שלמה בשיוט המשתנה עבור מזג אוויר שונה, תנאי השטח ואפילו מליחות המיים.

ואכן לאחר שחקרנו בספרות המקצועית לגבי שימוש באלגוריתם ה-RRT למציאת מסלול שיוט, גילינו כי נעשה מחקר מקיף ע"י אחד החוקרים המובילים בתחום, ואחת מהמסקנות העקרוניות בו היא שהאלגוריתם אינו מתאים לשימוש בשיוט בגלל אותן הסיבות הנ"ל, למשל בשל כי האלגוריתם אינו מהווה גורם אקראי אמיתי למסלולים שונים כיוון שלקודקודים הראשונים שהוא מגריל יש השפעה חזקה על בחירת שאר הקודקודים, וכן חוזר התלות של הרצות שונות של ה-RRT (שניסינו והצלחנו לפתור).

**הסבר על אלגוריתם הפוטנציאל**



האלגוריתם שאיתו החלטנו להמשיך היה אלגוריתם מבוסס שדות פוטנציאל. האלגוריתם הוא חמדן, כלומר במקום להחזיר מסלול, בהינתן מיקום היעד, מיקומי המכשולים והמיקום הנוכחי של הסירה, הוא יחזיר כיוון תנועה טוב ביותר, שמצד אחד מקדם את הרובוט למטרה ומצד שני דואג שהוא לא יתנגש במכשולים.

שיטת הפעולה של האלגוריתם מבוססת על רעיון פיזיקאלי של כוחות כבידה או כוחות דחייה חשמליים. כל מכשול יפעיל על הרובוט כוח דחייה (פרופורציוני הפוך לריבוע המרחק במימוש הספציפי שלנו) שמיוצג באמצעות וקטור לכיוון הרלוונטי בגודל הרלוונטי, וכן היעד יפעיל על הרובוט כוח משיכה (קבוע במימוש שלנו), שגם כן מיוצג כווקטור מתאים. כדי להחליט על הכיוון האופטימלי, פשוט נסכם את הווקטורים.

לאלגוריתם חמדני כזה יש שתי יתרונות מרכזיים לעומת האלגוריתמים לעיל: בכל נקודה שהסירה יכולה להימצא האלגוריתם ימצא כיוון אופטימלי ואף אם הסירה תיסחף מהמסלול שלה אין זה משנה. יתרון נוסף הוא שילוב מצוין עם אלגוריתם השיוט – היות והאלגוריתם מחזיר כיוון, קל להפעיל על כיוון זה אלגוריתם השיוט. כמו כן לאלגוריתם זמן תגובה טוב, כלומר אם למשל, נצטרך לבצע Tack ונגלה באמצע התמרון שיש סירה שאנחנו עלולים להתנגש בה, האלגוריתם ידע להחזיר כיוון שימנע התנגשות זו. כמו כן, אם סירה תתקרב מאחד הצדדים, האלגוריתם ינסה "לברוח" מהתנגשות עתידית כזאת.

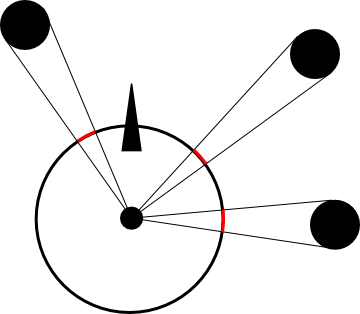
**חסרונות האלגוריתם**

לאלגוריתם הזה יש שני חסרונות עיקריים שמונעים שימוש באלגוריתמים מסוג זה בדרך כלל. מצד אחד, יש מקרה בו האלגוריתם יתעלם ממכשול לגמרי – כאשר היעד, הרובוט והמכשולים נמצאים על אותו ישר כאשר המכשולים נמצא בין הרובוט ליעד. במקרה זה או שהאלגוריתם ינחה את הרובוט לעבור דרך המכשול או הרובוט "ייתקע" על הישר הזה בלי יכולת להגיע ליעד או לרדת מהישר. במקרה שלנו הסירות הן אובייקטים דינאמיים ולכן כיוון שהמכשול ינוע גם כן, הסיכויים שבעיה כזאת תקרה הם קטנים מאוד (אך כחלק מאבטחת האיכות נבנה מנגנון משנה שיימנע התנגשות במקרה זה).

הבעיה השנייה ידועה תחת השם "מינימום מקומי". למעשה אם נחשוב על שדה וקטורי שהמכשולים והיעד יוצרים, היעד הוא נקודת מינימום. אך תתכננה בניות בהן המכשולים והיעד יוצרים בשדה הוקטורי גם נקודות מינימום מקומי, כלומר נקודות בהם הסכום הוקטורי יהיה 0. שוב, במקרה שלנו בגלל שהאובייקטים נמצאים בתנועה וכן יש סחיפה אף אם ניתקל בנקודה כזאת אז או שניסחף ממנה או שהמכשולים יישנו את מיקומם והשדה הוקטורי ישתנה.

**הסבר קצר על האלגוריתם השני שהוכנס**

אלגוריתם נוסף שנבדק אך עקב מחסור בזמן לניסויים בסביבה "רטובה" ולהגעה לתוצאות טובות כמו של האלגוריתם מבוסס פוטנציאל, לא השתמשנו בו.



הרעיון הכללי של האלגוריתם הוא הבא:

נצייר מעגל סביב הנקודה בה נמצאת הסירה ונטיל את המכשולים (שמיוצגים כמעגלים) על המעגל. ניתן לנקודה על המעגל משקל לפי הקרבה לכיוון הרצוי ולפי קרבת המכשולים. נבחר בנקודה עם המשקל הגבוהה ביותר.

**חלק ג': ארכיטקטורת החומרה**

**רקע כללי :**

הסירה כולה נבנתה על בסיס דגם הסירה micro magic RTR זה דגם של סירת מפרש על שלט רחוק (מפרט טכניניתן למצוא באתר: <http://www.graupner.de/en/products/81053d1e-0ce0-48d3-94b4-9fd7240b5cfc/2014.200/product.aspx>).

על הסירה היה צורך להרכיב את המרכיבים הבאים:

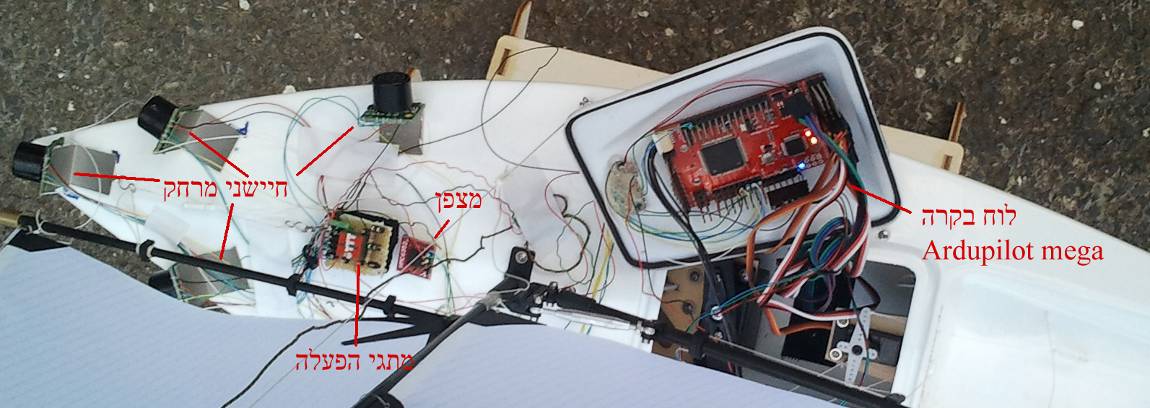
1. מיקרו מעבד
   1. אנחנו בחרנו להשתמש ב: Ardupilot mega שהוא לוח בקרה הכולל לא רק מיקרו מעבד וזכרון אלא גם כניסות I/O אנלוגיות, דיגיטליות וPWM ופורטים לממשקי I2C וserial interface.
2. חיישני קירבה
   1. החיישנים שבחרנו עבור עבודה בסביבה חיצונית ועל סף מיים, תוך כדי התחשבות בצריכת הספק ומחיר היו חיישני אולטרא סאונד מסוג Maxsonr EZ0 המספקים חישה לטווח של 6 מטר. דף נתונים: <http://www.maxbotix.com/documents/MB1000_Datasheet.pdf>
3. מצפן
   1. המצפן משמש רק כדי לדעת את כיוון החרטום של הסירה והדגם שנבחר הוא: HMC6352 – זהו מצפן פשוט במיוחד שמתקשר בפרוטוקול I2C, והעמידות שלו לרעש היא ברמת 2 מעלות כיוון עבור כל מעלה הטיה בציר המשלים לכיווניות (קרי אם הוא ממשיך לפנות לאותו הכיוון אך חץ התפון משמש כציר הטיה), עקב כך החיבור בוצע בסיבוב של 90 מעלות כדי לקבל עמדיות להטיית הספינה. דף נתונים: <http://www.alldatasheet.com/datasheet-pdf/pdf/205923/HONEYWELL/HMC6352.html>
4. GPS
   1. המודול שנבחר הוא הEM406 זהו מודל קטן בעל רגישות גבוהה ודיוק טוב עבור המחיר, דפי נתונים: <http://www.sparkfun.com/datasheets/GPS/EM-406A_User_Manual.PDF>
5. חיישן כיוון רוח.
   1. על מנת לחשב את כיוון הרוח השתמשנו במקודד מגנטי סיבובי הנותן רזולוציה של 12 ביט עבור 360 מעלות ועליו הרכבנו דגל/שבשבת על מנת ללכוד את הרוח, הדגם הנבחר היה: EMS22P דפי נתונים: <http://www.bourns.com/data/global/pdfs/EMS22P.pdf>.
6. חיישן עוצמת רוח.
   1. של מנת לחשב את עוצמת הרוח השתמשנו בחיישן המבוסס נגדים תלויי טמפרטורה אשר מחוממים על ידי זרם קבוע לטמפ' קבועה ואז משב הרוח מצנן אותם לטמפ' הקובעת את ההתנגדות מחדש וזו מחושבת לנתינת ערך מתח נמדד. לא קיימים דפי נתונים למוצר זה היות וזה מוצר עצמאי אך ניתן לראות תרשים חשמלי בכתובת: <http://cdn.shopify.com/s/files/1/0038/9582/files/WindSensorSchematicRevC1.png?100218>
7. מערכת תקשורת עם החוף.
   1. על מנת לתקשר עם החוף נבחרה מערכת דו צדדית המורכבת משני רכיבי XBEE explorer, כאשר צד החוף מתחבר למחשב בעזרת מחבר יעודי לכרטיס עם מוצא USB וצד הסירה מתחבר ללוח הבקרה. דפי נתונים: <http://ftp1.digi.com/support/documentation/90000976_G.pdf> .
8. מייצב מתח – ספק המתח של הסירה הייתה בטריית 2 תאים שסיפקה מתח לסרווים והוציאה 7.4 וולט, על מנת לעבוד עם אלקטרוניקה הדורשת 5v היה צורך במייצב (לא ליניארי היות וליניארי התחמם יתר על המידה) המייצב הנבחר היה UBEC -5A דפי התונים נמצאים: <http://www.hobbywing.com/upload/manual/5A-HV%20ubec_e.pdf>.

**ארכיטקטורה כללית**

מבנה הסירה הבסיסי נשמר זהה. כשהאלקטרוניקה הנוספת הייתה:

1. לוח הבקרה הותקן מתחת למכסה של הסירה, זה איפשר לחווט בנוחות את כל הרכיבים על ידי כלים (שהוכנו ידנית) ולנתק אותו במקרה הצורך.
2. חיישני המרחק הותקנו מסביב לסיפון בזוויות: 0, 35± ו- 80±, הזוויות נועדו לתת כיסוי טוב של הסביבה מסביב לסירה כדי לגלות מכשולים במיקומים שאליהם הסירה עלולה להגיע בתמרונים נפוצים.
3. המצפן הותקן על הסיפון, כדי למנוע הפרעות מגנטיות מספקי הDC בתוך גוף הסירה.
4. כרטיס התקשורת הורכב בתוך גוף הסירה והתחבר ללוח הבקרה, הסיבה למיקום הפנימי הייתה הרגישות הגבוהה למיים בגוף מכוסה המתכת וצורה שהקשתה מאד על הגנה בחוץ, כמו כן היה שימוש במיים בתור משטח אדמה כללי עבור האנטנה, כאשר התקנה בגובה עלולה ליצור החזרים עקב הפרשים במשטחי האדמה.
   1. הפסדי הטווח (והתווך) שנמדדו עקב מיקום פנימי לא היוו בעיה ביחס לצרכי התחרות – הכרטיסים משדרים בעוצמה של 60mW או 13dBm וסף הרגישות הוא -96dBm.
5. כרטיס הGPS הותקן גם כן מתחת לסיפון, אך מוצמד לסיפון בצידו התחתון, המיקום היה קרוב ככל האפשר למרכז הסירה ונתן חיבור נוח ללוח הבקרה.
6. חיישן עוצמת הרוח הותקן בראש התורן תחת ההנחה שעוצמה הרוח בראש התורן מעל המפרש תיתן את עוצמת הרוח האמיתית ללא שינויים עקב מיקום המפרש.
7. חיישן כיוון הרוח הותקן בראש התורן על מנת לאפשר זרימת אוויר מלאה ולמנוע התנגשות.
8. פלטת מתגים – כדי למנוע בעיות חיווט מרובות ועל מנת לתת מוחות גבוהה יותר על גבי הסיפון הותקנה פלטת מתגים, המתגים איפשרו לכבות ולהדליק את המערכת ולבצע restart כך שלא יהיה צורך לפתוח את הסירה כל פעם כדי לתכנת מחדש את הarduino או כדי לכבות ולהדליק אותה. כמו כן המתק שחובר ישירות למייצב המתח מנע זליגת זרם לשאר חלקי הסירה וכך לשמר הספק. כמו כן כל חיווטי המתח של הרכיבים נותבו לפלטה ומשם הועברו לתוך הסירה.
9. מייצב המתח הותקן התוך הסירה כשהמוצא של הולך לפלטת המתגים.

**תמונות:**

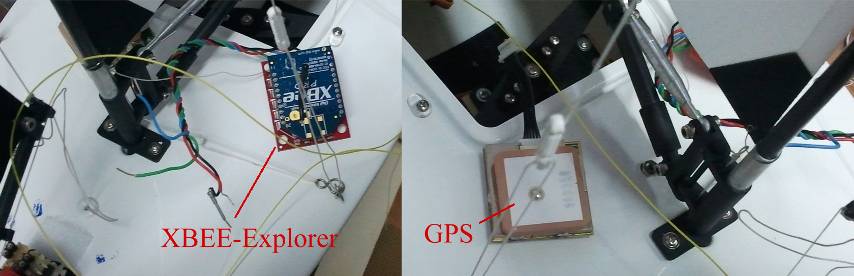


עד סיום התחרות חיישני המרחק כבר לא תפקדו עקב חדירת מיים ונזק פיזי מהמסע. הם כוסו בניילון למניעת קצרים ונותקו מהלוח.

החיבורים שלהם לספינה הם זוויתני מגנזיום שעוצבו בתעשייה אווירית וכוסו בנייר דבק דו צדדי פשוט.

מסקנות חומרה:

1. אחרי בחינת הספינות הגרמניות מומלץ לחבר את הסרווים מחדש לא לפי הוראות ההרכבה אלא בצורה חיצונית כך שישלטו ישירות בזויות המפרש.
2. יש להחליף את חיישני המרחק בחיישנים מוגנים ככל הנראה לייזר.
3. סביבת הwireless היא סביבה רועשת מבחינה ספקטרלית, כדאי למצו מיכשור העובד בתדרים אחרים, או להגיע עם ספקטרום ולמצוא איזור שקט.



**עבודה עם חיישני הארדוינו**

**רקע כללי (מה נותן לנו הארדוינו, איזה חיישנים היו שם ואיזה פקודות הם יכולים לקבל)**

לוח הארדואינו הינו לוח microcontroller open source. הלוח מאפשר הרצה של תוכנית, הנכתבה בשפה ייעודית (הנקראת ארדואינו), אשר דומה בסינטקט לג'אווה ומבוססת על C.  
התוכנית אשר מורצת על הלוח מוגבלת מאוד עקב מגבלות הלוח, ובעצם מורכבת משלושה חלקים:  
- החלק הראשון הוא ספריות חיצוניות אשר מאפשרות לה להריץ פקודות מסויימות, הספריות יכולות הן להיות בשפת ארדואינו והן ב-C (עם מספר מצומצם של מגבלות על גישה לזכרון וכו').  
- החלק השני נקרא ה-setup. זוהי מעין פרוצדורה שתרוץ פעם אחת בתחילת כל ריצה של הלוח, בפרוצדורה זו בדרך כלל מבצעים את כל פעולות האתחול עבור הרכיבים על הלוח.  
- החלק השלישי הינו ה-main. זוהי לולאת while true שתרוץ תמיד לאחר ההרצה הראשונית של ה-setup.  
בלולאה ממומשת הפעולה העיקרית שאותה רוצים שהלוח יבצע.

ללוחות ארדואינו ישנם רכיבים רבים שחברות שונות מייצרות, חלקם פשוטים ומאפשרים קריאה פשוטה של מתחים וחלקם מסובכים יותר. אנחנו השתמשנו במספר רכיבים:  
- חיישני ultrasonic לקריאת מרחק: אלו חיישנים עם אלומה די רחבה אשר מאפשרים קריאת מחרק מדויקת מאוד עד למרחק של שלושה מטרים. את קריאת החיישנים ניתן לקבל על ידי קריאה לפונקציה analogRead שממומשת כפונקציית ליבה של ארדואינו. הפונקציה תחזיר את המתח הנקרא על ידי החיישן (ערך בין 0 ל-1023) אותו ניתן להמיר למרחק. עקב בעיות טכניות לא השתמשנו לבסוף בחיישנים.  
- חיישני עוצמת רוח: זהו חיישן אשר מודד לפי שינוי טמפרטורה את עוצמת הרוח הנושבת עליו ומראה זאת על ידי שינוי במתח שהוא שולח. חיישן זה, כפי חיישן ה-ultrasonic נקרא על ידי הפונקציה analogRead.  
- חיישן כיוון רוח: זהו חיישן שיודע לשלוח את זווית השינוי שבה הוא נמצא יחסית למצב ההתחלתי, כאשר הוא יכול להסתובב רק על ציר אחד. חיישן זה מחזיר מידע דיגיטלי אשר המרנו למידע אנלוגי וקראנו על הלוח שוב באמצעות הפונקציה analogRead.  
- מצפן: זהו רכיב שיודע להחזיר את הזווית בה הוא נמצא ביחס לצפון שאליו הוא מאותחל. קריאות המצפן רגישות ממש להטייתו בציר מסויים, אך פחות בהרבה בציר האנכי לו. על כן, המצפן הותקן מאונך לתנועת הסירה על מנת למנוע את ההטיה (אשר הייתה מתרחשת כאשר הסירה הייתה נוטה לצדדים).  
המצפן מחובר דרך חיבורים ייעודים לו ומשתמש בספריית Wire שהינה גם כן ספריית ליבה של ארדואינו ומאפשרת לקרוא את המידע ממנו באמצעות הפקודות Wire.requestFrom ו- Wire.receive (כאשר יש לבצע כמה הטיות על הקריאה בשביל להמיר אותה לקריאה בזוויות).  
- GPS: זהו רכיב המאפשר לקרוא נתוני GPS בכמה פורמטים, כאשר ברירת המחדל הוא פרוטוקול ה-NMEA.  
חיישן זה הינו המסובך ביותר, היות וקריאת המידע ופרסורו דורשת עבודה רבה. על כן, נעשה שימוש בספריות עזר חיצוניות אשר מאפשרות פשוטה יחסית של הנתונים.  
החיישן יכול להחזיר מגוון די רחב של נתונים, ממיקומו, מהירותו, הכיוון בו הוא נע ועד כדי הזמן ופרטים נוספים אשר לא השתמשנו בהם העוברים דרך הודעות NMEA.  
- Servo: זהו רכיב אותו ניתן לסובב בהתאם למתח הנשלח אליו. אנחנו השתמשנו בשלושה עבור לסובב את שני המפרשים ואת ה-rudded. רכיבים אלה גם כן דורשים ספריות חיצוניות על מנת להתמודד עם תזמון שליחת האותות אליהם. על מנת להזיז סרבו צריך להגדירו ואז לשלוח לו את הפקודה write עם הזווית אותה מעוניינים לקבוע (ערך בין 0 ל-180 כאשר ראינו כי בטוח רק לשלוח ערכים בין 10 ל-170, ובמפרש הראשי רק בין 100 ל-170).  
בנוסף, צריך לשים לב לעבודה עתידית שלא ניתן להגדיר יותר מרכיב servo יחיד ועל מנת לעקוף את מכשול זה, צריך להגדיר את הרכיב כמערך של servo.  
- Xbee: זהו הרכיב המאפשר תקשורת wireless בין שני רכיבים מסוג זה, כאשר אחד ממוקם על הלוח ואחד מחובר באמצעות דונגל מתאם למחשב. מעבר להעברת מידע בקצבים שונים (כאשר ניתן לקבוע את הקצב של העברת המידע ובכך את כמות המידע המקסימלית שניתן להעביר בשנייה), מאפשר גם להעלות תוכניות חדשות ללוח עצמו דרך ה-IDE של ארדואינו (הסבר מפורט בהמשך).

**פורמט הפיקוד ושליחת התוצרים (הפורמט הטקסטואלי בין ה-ShoreLogic לרכיב הקוד של הארדאוינו)**

קבענו פורמט בו אנו שולחים אל הלוח מידע על מנת לשנות את מצב ה-servos וכן מקבלים מידע על החיישנים השונים.  
על מנת לשנות את מצב ה-servos, על רכיב ה-shorelogic לשלוח באמצעות כתיבה לפורט הסיריאלי (אשר אותו מסמלץ רכיב ה-Xbee) את הפקודות מהפורמט <Char><Degree>. כשאר האות הינה A, B או C עבור כל סרבו בנפרד והזווית הינה ערך בין 15 ל-165 ותמיד מכיל 3 ספרות, כלומר 10 יתורגם ל-010 (כמו כן, B שהינו הסרבו למפרש הראשי אמור לקבל ערכים בין 100 ל-165 וערכים נמוכים מ-100 יתורגמו ל-100).

על מנת לקבל מידע מהלוח, על רכיב ה-shorelogic לקרוא מידע מהפורט הסיריאלי כאשר הפורמט הינו ערכי החיישנים כאשר הם מופרדים על ידי ; (עם תו | שמסמן את תחילת וסוף ההודעה). בנוסף מועברת אות אשר מסמנת האם מצב הלוח הוא כעת בשליטה ידנית או אוטומאטית (הסבר בהמשך).  
את פרסור המידע מצבעים ב-shorelogic עצמו וכמו כן חשוב לשים לב כי ישנה מגבלה לקצב שליחת המידע וייתכן כי חלק מההודעת לא יגיעו כראוי (עם כי מרביתן מצליחות להיקרא).

**קוד הארדואינו**

הקוד עובד באופן הבא:  
כפי המתואר לעיל, הוא מתחיל מהרצת פרוצדרות אתחול אשר מאתחלת את כל החיישנים ומאפסת אותם כראוי, וכמו כן שמה את כל הסרבויים למצב הביניים.  
לאחר מכן מתחילה הריצה של הלולאה עצמה, הלולאה תהיה באחד משני מצבים, אוטומאטי או ידני.  
כאשר הלולאה במצב אוטומאטי, השליטה על הלוח מתבצעת על ידי רכיב ה-shorelogic אשר יקרא נתונים מהחיישנים וישלח פקודות לסרבויים בהתאם לצורך.  
כאשר הלולאה במצב ידני, השליטה על הסירה היא דרך השלט.  
ישנם מספר אפשרויות למעבר דרך המצבים, ניתן לשלוח פקודה לרכיב עצמו אשר מאפשרת מעבר בין המצבים. כמו כן, אם הרכיב נמצא במצב אוטומאטי ולא מקבל פקודה במשך 30 שניות, הוא יעבור בעצמו למצב ידני מחשש לבעיה בשליחת הנתונים מהמחשב (בדרך כלל הודעות נשלחות באופן רציף ללוח).

בכל אחד משני המצבים, עדיין מתבצעת קריאה מכל החיישנים והם עדיין נשלחים (בנוסף לאות המסמנת באיזה מצב אנו נמצאים). במצב אוטומאטי הלוח מנסה לקרוא מידע הנשלח לה מה-shorelogic בפורמט אשר מתואר לעיל, ולפיו לקבוע את זוויות הסרבויים. במצב ידני הלוח קורא את הנתונים הנשלחים על ידי השלט וקובע את הסרבויים לפיו.

**חוסרים**

ישנם מספר חלקים שניתן לשפר בעבודה על החומרה. הבעיה העיקרית היא המגבלה שלוח האדואינו מציב.  
כפי שנאמר, הלוח יכול להריץ רק לולאה אחת כתוכנית העיקרית שלו. מעבר לבעיות שמגבלה זו יוצרת, אשר את חלקן ניתן לעקוף בדרכים שונות, עולה כי החולשה העיקרית של הלוח היא שהוא יכול להריץ פעולה אחת בכל זמן נתון.  
מגבלה זו באה במיוחד לידי ביטוי בקריאה מחיישנים. למרות שהמתחים על כל החיישנים נתונים בכל עת, ניתן לקרוא מידע מחיישן אחד בכל זמן. בעיה זו מתעצמת כאשר הקריאה מחיישן מסויים עלולה לקחת זמן רב (לדוגמא קריאה מחיישן ה-ultrasonic לוקחת כ-50 מילי שניות, וכאשר מנסים לקרוא חמישה כפי שניסינו, מתבזבזת רבע שנייה בה הלוח אינו מסוגל לבצע פעולה נוספת).

הבעיה הזו היא מובנית בלוח, ולא נראה כי יש דרך להתמודד איתה ישירות (היות ולא ניתן להריץ מספר threads במקביל).  
בעיה נוספת מתרחשת כאשר ישנן בעיות של מתח אספקה ללוח, במצב זה הקוד עלול עדיין לרוץ אך שולח מידע לא נכון או לא רלוונטי.

**חלק ד': סביבת בדיקות ו-Debug**

**בדיקות "יבשות" במהלך הפיתוח**

במהלך הפיתוח, ובעיקר בשלב בחירת אלגוריתם מניעת התנגשויות, רצינו כמובן לוודא שהאלגוריתמים יעבדו. כמובן שלאורך הפיתוח לא תמיד יכולנו לבדוק את האלגוריתמים ישר בבדיקות "רטובות", כלומר באגם. אחת הסיבות לכך היא שהיות ואנחנו לא מכירים את נושא השיוט לעומק, לא תמיד היינו יכולים להבחין האם החלטות שגויות נובעות מאלגוריתם מניעת התנגשויות, אלגוריתם השיוט או "באגים" בתוכנה. כמו כן, הסביבה של שייט היא מאוד דינמית והיה קשה לשחזר את התנאים שבהם האלגוריתם קיבל החלטות שגויות.

עקב כך, הבדיקות הראשוניות נעשו באופן "יבש" לגמרי. הכוונה היא שבנינו סביבת בדיקה (נקודת התחלה, נקודת סיום, מכשולים ורוח) והרצנו את האלגוריתמים תחילה בסביבה שבנינו מראש, כאשר באמצעות קבצי לוג וסקריפטים בפייטון (כסביבת עבודה עם אפשרות המחשה של גרפיקה דו ממדית נוחה ומפותחת) יכולנו להציג על המסך את המסלולים שהאלגוריתם בנה ולכייל אותו שייבחר במסלול הטוב ביותר מביניהם.

בשלב הבא בנינו מפות (קבצי tack) הדומים למסלולים שהוגדרו לנו להתמודד בתחרות ובחנו את התנהגות הסירה בהם, ובשלב הסופי הוספנו תמיכה בריבוי סירות והרצת מירות בינהם במקביל, כאשר כל סירה מחוברת לעותק נפרד של אפליקצית SL.

בנוסף, כדי להקל על בדיקות האלגוריתמים וכדי לשלול אפשרות שבאגים בסימולטור מפריעים לבדיקה תקינה שלהם, נכתבו מספר Scripts בשפת Python, המדמים את המסלול שבוחר האלגוריתם למעבר בין מכשולים ללא הפרעות.

**בדיקות "רטובות" בבריכה ובאגם** כאשר הסירה האמיתית היתה מוכנה לאינטגרציה מול רכיבי התוכנה, עברנו לביצוע בדיקות בבריכת גומי שנרכשה לצורך העניין והוצבה מחוץ לבניין שרייבר. הבדיקות כללו בעיקר כיול של החיישנים ובדיקה של יכולות הסיבוב של ה-Rudder וכיוונון המפרשים.

בשלב הבא עברנו לבדיקה אמיתית באגם שבפארק הירקון (תל-אביב), בו ביצענו בעיקר השטות ידניות של הסירה בעזרת השלט כדי להתנסות בטקטיקות השטת סירות.

בשבועיים האחרונים לפרויקט עברנו לאגם גדול יותר, בפארק הירקון (רמת-גן), בו ביצענו השטות הסירה בעזרת האלגוריתם. בשלב זה בוצעו מספר רב מאוד של התאמות וכיוונונים לתוכנה ולאלגוריתם.

**חלק ה': הפניות למקורות רלוונטים**

אתר התחרות: http://acg.cs.tau.ac.il/courses/workshop/spring-2011/projects/roboat/world-robotic-sailing-competition-2011/training-and-competition

אתר הפרויקט (ומסמכים נלווים): http%3A//acg.cs.tau.ac.il/courses/workshop/spring-2011/projects/roboat/boat-project-general-design

קוד הפרויקט: https://www.assembla.com/spaces/cs-workshop

**נספח 1- בדיקות במהלך התחרות בגרמניה ותובנות לקבוצות עתידיות**

במהלך התחרות התנסנו במספר רב של תחרויות, התרשמנו ולמדנו מטכניקות של קבוצות אחרות. היה שיפור מדי יום ושכתבנו חלק מהלוגיקה לאור עצות ותובנות שקיבלנו במהלך התחרות.

מספר דברים עיקריים שלמדנו –

* בתנאים דינאמים כמו תחרות באגם חשוב מאוד לשמור LOGים מפורטים, שכן קשה מאוד להבין בזמן אמת למה הסירה ביצעה החלטה שגויה כזאת או אחרת, ועד שהסירה חוזרת לחוף עובר לפעמים זמן רב.

חלק מהצוותים מימשו גם GUI יפה ואינדיקטיבי שעובד מעל ה-LOG.

* דרך מוצלחת להבנת תופעות בסירה היא לאפשר שמירת MARK ב-LOG, השומר לפי דרישה את מיקום הסירה ברגע נתון, כלל נתוני החיישנים שלה והערת מפעיל שהוכנה ברגע השמירה.

לאחר התחרות אפשר למצוא את ה-MARK הצמוד להערה ולנתונים ולנתח את הבעיה בצורה טובה.

* בתנאי אמת יש לפעמים סטיות חדות ורגעיות במדדים אותם מחזירים החיישנים, סטיות כאלה יכולות לפגוע מאוד בהתנהגות של הסירה ולגרום לה להחלטות שגויות. דרך מוצלחת להתמודד עם זה הוא מימוש Averaging חכם לקריאות החיישנים, כך שסטיה רגעית לא תשפיע בצורה מהותית על ההתנהגות.
* ניווט ביום שבו כמעט אין רוח שונה מאוד מניווט ביום עם רוח, במהלך התחרות יצא לנו לכתוב לוגיקה מיוחדת לניווט בתנאים כאלה, מומלץ להיפגש עם צוות שייטים מקצועיים או שייטי סירה על שלט רחוק ולהגיע איתם להשטה של הסירה – חשוב שלא לקבל הרצאה כללית בע"פ, אלא לקבל מהם מידע כשהם מתנסים עם הסירה עצמה.
* אנטנות גדולות המשדרות אות חזק יכולות לשבש תקשורת נתונים ולגרום לאיבוד קשר עם הסירה מעבר למרחק מסוים, תופעה שלא נתקלנו בה בארץ למרות שבדקנו טווחים גדולים הרבה יותר. לכן חשוב שתהיה לוגיקת חירום למקרה כזה.
* חשוב לממש מד סוללה שיתריע במקרה שמתקרבים לקצה גבול היכולת של הסוללה (עשינו זאת במהלך התחרות).

**נספח 2 – תפעול הארדוינו (תפעול ותיעוד כללי)**

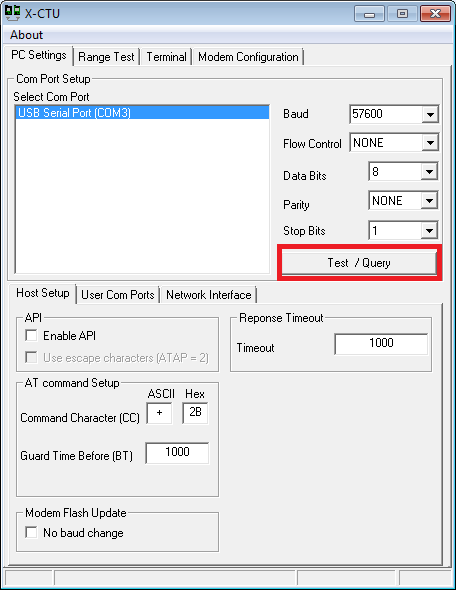
בפרוייקט בוצעה טעינה והפעלה של המערכת על ידי שימוש ברכיבי ה-XBEE ולא דרך כבל USB, החלק הבא מסביר כיצד לבצע את הפעולות הללו:

**העלאת קוד לארדואינו דרך xbee**

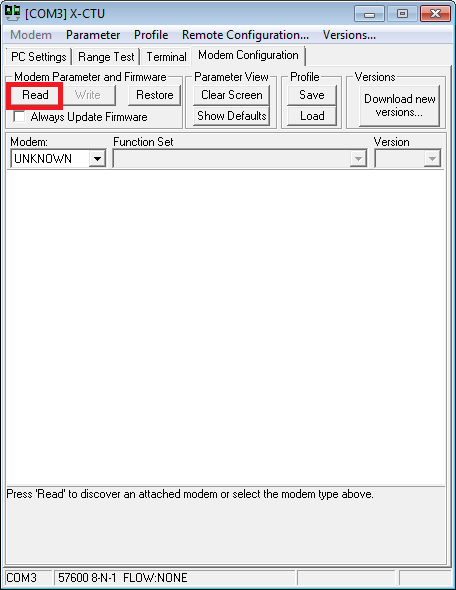
החומרה הנדרשת: לוח ארדואינו, שני מודולי xbee, xbee explorer usb dongle ו-xbee explorer regulated.

התוכנות הנדרשות: arduino ide (<http://arduino.cc/en/Main/Software>) ו-X-CTU (לחפש גרסא עדכנית, זו המסופקת באתר הינה מ-2008 ויוצרת בעיות תאימות עם הרכיבים החדשים יותר - גרסא אחרונה לזמן זה - מאי 2010). ftdi drivers (<http://www.ftdichip.com/Drivers/VCP.htm>)

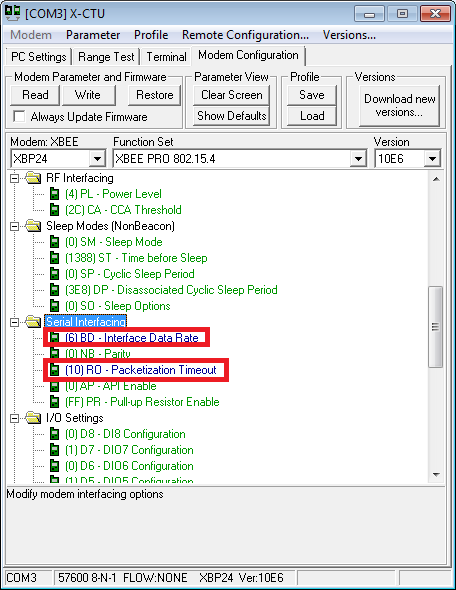
שלב 1: לחבר את רכיבי ה-xbee דרך ה-xbee explorer usb dongle למחשב.  
שלב 2: להריץ את x-ctu, לראות תחת PC Settings שהפורט COM שדרכו מחובר הרכיב מצויין ונבחר ב-Select Com Port ולהריץ Test / Query.



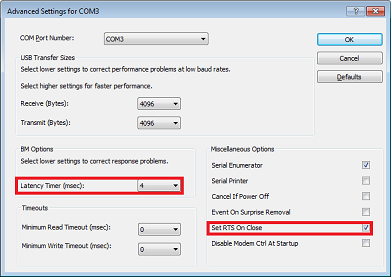
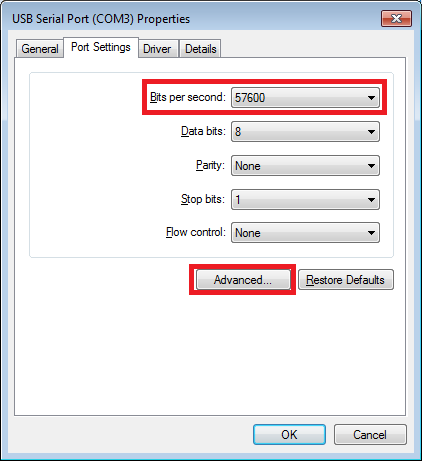
שלב 3: תחת Modem Configuration ללחוץ על Read.



שלב 4: לשנות תחת את Serial Interfacing את Interface Data Rate לקצב המתאים ללוח ארדואינו (ניתן לראות את הקצבים עבור לוחות בקובץ boards.txt בתיקיית ה-Arduino IDE - עבור ArduPilot Mega הקצב הוא 57600). וכן, לשנות את Packetization Timeout ל-10. לעשות זאת עבור שני ה-Xbee.



שלב 5: לשנות את הפורט המתאים תחת Device Manager לקצב שקבעתם בשלב 4, וכן, תחת Advanced, לסמן את Set RTS on close ואת ה-latency לשנות ל-4.



שלב 6: ב-Arduino IDE לשנות את Board ו-Serial Port שיתאימו לרכיבים.

שלב 7: לאחר כתיבת הקוד ובדיקה שהוא מתקמפל ב-Arduino IDE, לחבר את כל הרכיבים, להפעיל את הלוח וללחוץ ולהחזיק את כפתור ה-reset שנמצא על לוח הארדואינו. כעת ללחוץ על על Upload, לחכות עד שמופיעה השורה "Binary sketch size: ..." ורק בזמן הזה לעזוב את כפתור ה-reset.



שלב 8: הקוד אמור כעת לעלות ולרוץ על הרכיב. כעת אפשר לראות את הקלט שחוזר תחת serial monitor ב-Arduino IDE.