SubSkipper Documentation

Filip Socko

August 14, 2015

1 Introduction

The GitHub SubSkipper repository contains the core logic of the app, such that it can be verified or used for other projects. The documentation contains the principles and equations on which the logic is based, and some method documentation.

The main purpose of the repository and documentation is to record techniques and methods of early submarine attack techniques in a way which are simple to employ in computer programs (i.e. showing mathematical equations where possible), as well as acting as a reference for Submarine Simulators.

The Android App will be developed from this repository as a separate, polished product.

2 Requirements

The Requirements for SubSkipper are the following:

- Calculate Constant Bearing Solutions using the following methods:
  - Dick O'Kane method
- Show a representation of the *Is/Was* or *AngriffScheibe* calculator for easy input, along with its data
- Calculate target location with a course and speed
- Calculate AOB from aspect ratio
- Calculate Speed via *Fixed Wire* method
- Feature a Modular timer/stopwatch

3 Unit Conversions

3.1 Speed

1.0 knots	1.0 NM per hour
1.0 m/s	1.94384449 knots
1.0 m/s	3.6 km/h

3.2 Length

1.0 NM	1852.0 m
1.0 km	0.539956803 NM
1.0 NM	2025.37 yards
1.0 NM	6076.12 feet
1.0 m	3.2808399 feet
1.0 m	1.0936133 yards

4 O'Kane Torpedo Solution

The Dick O'Kane method was devised by members of the Subsim.com forums. It is a constant bearing method which relies on calculating a lead angle – an angle on which torpedoes, if launched will intercept the course of the target– to which the periscope is pointed. As parts of the target ship cross the bearing, torpedoes are fired along it. The O'Kane method relies on being ahead of the target, and the final AOB – at which the torpedo strikes the target– to be 90°.

Calculates *lead angle* based on target and torpedo speed.
The solution requires submarine to be ahead of target.
Captain inserts target speed into TDC, puts the scope on the lead bearing, fires as the target crosses the bearing.

The Equation for lead angle is as follows:

$$LeadAngle = 90 - \arctan\left(\frac{TorpedoSpeed}{TargetSpeed}\right)$$

4.1 Computational Solution

The method *oKSolution()* in the class *OKane.class* in the package *coreLogic* is used to calculate the O'Kane Lead Bearing when using the O'Kane method. The periscope is pointed to the lead bearing calculated by *oKane()*.

The method takes the following arguments:

- int AOB - For determining whether AOB is Port or Starboard
- double targS - Target Speed
- Torpedo fireS - Torpedo Speed

4.2 Errors

Attempting to compute O'Kane Lead angle given the following situations will return the flag -1.

4.2.1 Submarine is not ahead of target:

The following code checks if AOB is ahead of the target, either port or starboard. Furthermore, if the submarine is at an AOB of 0, or 180, the O'Kane lead angle cannot be calculated.

```
if(AOB <= 90 && AOB<360){
            stbd = true;
        }
        else if(AOB>=270 && AOB<360 ){
            stbd = false;
        }
        else if(AOB == 0){
            invalidSol = true;
```

4.2.2 Torpedo Speed or Target Speed are less than one:

torpS and targS are verified to be ¿ 1 in the following code:

```
 if(torpS < 1 || targS <1){
           return -1;}
```

4.2.3 Lead is greater than 90

If lead is more than 90°, it means we would be aiming the torpedo backwards. This means the solution is invalid, as either the target is too fast, or the torpedo too slow. This error is handled in the following code:

```
if(lead > 90){
lead = -1;} //Okane relies on being ahead of the target,
                //we would be aiming backwards.
```

4.3 OKSolution Code

```
 public double OKSolution(int AOB, double targS, double torpFireS)
    {
        double solBearing = -1;

        boolean stbd = true;
```

```
    //check if the position is correct. Sub needs to be in front of
        target, on either
    //Stbd or port side. this means AOB is either 0-90 for stbd or
        270-360 for port
    //if anything else happens, solution is invalid, and we return a
        flag.
    boolean invalidSol = false;
    if(AOB <= 90 && AOB<360){
        stbd = true;
    }
    else if(AOB>=270 && AOB<360 ){
        stbd = false;
    }
    else if(AOB == 0){
        invalidSol = true; //for now let's assume the user is an
            idiot if AOB = 0
    }
    else{invalidSol = true;}

    if(invalidSol){
        solBearing = -1;
    }
    //if stbd, subtract from 360.
    else if(stbd){
        solBearing = 360-okaneLead(torpFireS, targS);
    }
    //If port, add to 0 for lead bearing.
    else{
        solBearing = 0 + okaneLead(torpFireS, targS);
    }
    return solBearing; //tSpeed into TDC, set your scope to this
        bearing, fire
}

//Input: torpedo speed (kn), target speed (kn)
public double okaneLead(double torpS, double targS){
    if(torpS < 1 || targS <1){
        return -1;}
    double lead = 0;
    //90 - inverseTan(torpS/targS)
    double rad = (torpS/targS);
    lead = 90-Math.toDegrees(Math.atan(torpS/targS));
    if(lead > 90){
        lead = -1;} //Okane relies on being ahead of the target,
    //we would be aiming backwards.

    return lead;

}
```

5 Calculating Distance To Target

No methods for calculating Distance To Target will be provided as solutions such as a periscope stadimeter and *sonar* are readily available.

6 Calculating AOB Based on Aspect Ratio

AOB can be determined given the following data:

- Range To Target
- Observed Mast Height
- Observed Ship Length
- Reference Aspect Ratio (i.e. $\frac{ReferenceLength}{ReferenceMastHeight}$)

1. Determine an observed and reference *aspect ratio*.

$$AR_{observed} = \frac{ObservedLength}{ObservedMastHeight}$$

Do the same with data from the recognition manual for the reference Aspect Ratio (*ARreference*).

2.

$$AOB = arcSin\frac{AR_{observed}}{AR_{reference}}$$

N.B: This method is less accurate as AOB approaches 0.