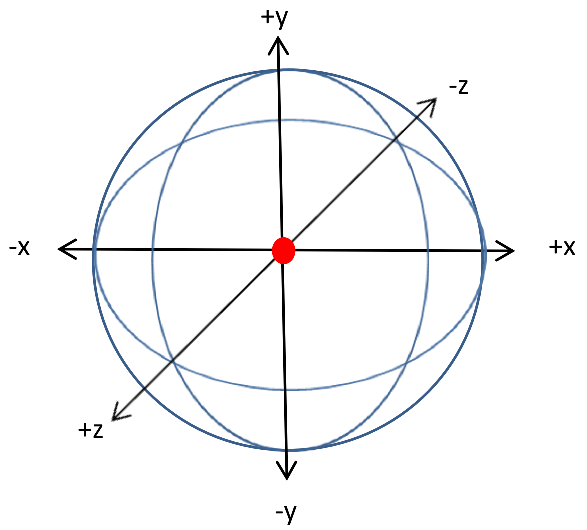
CSE P503 - Project 1: Navigation System.

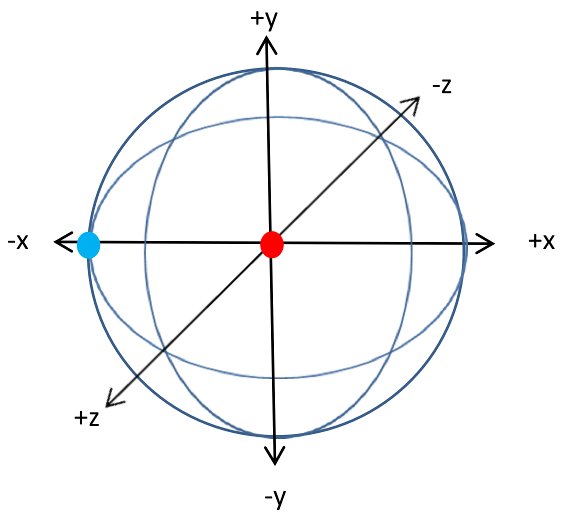
1. Introduction

The goal of this project is to design and verify a probe navigation system. Assume a radio telescope (marked by the red dot) sits at the center of the sphere below.

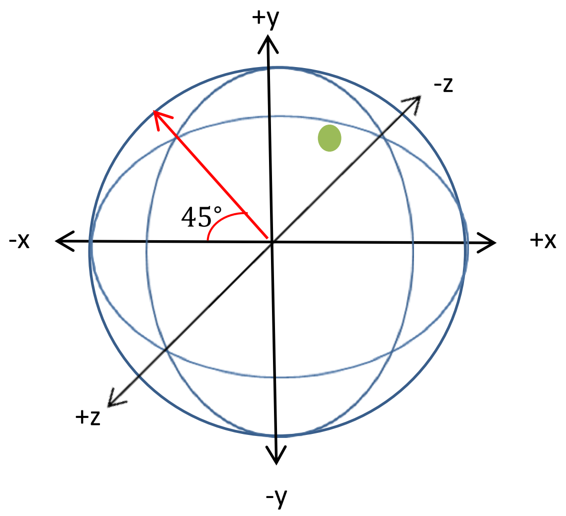


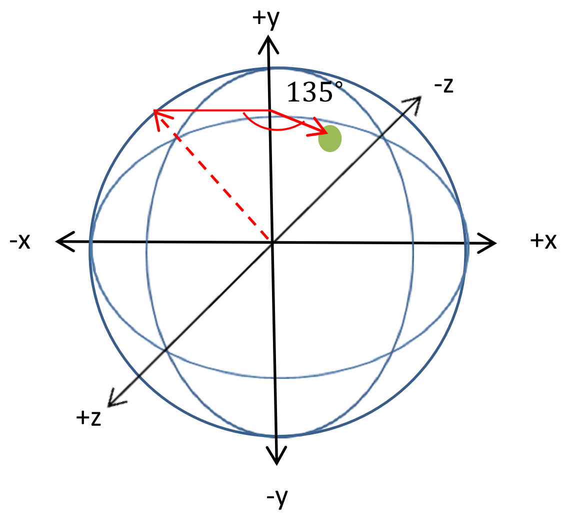
The surface of the sphere represents the universe around the telescope. From the center, the telescope observes distance objects and reports their location using two angles: Declination (δ) and right ascension (α). The telescope does not report the radial distance of objects from the center, because every object can be assumed to be equally far away. Said differently, objects only appear on the shell of the sphere and nowhere else.

Declination and right ascension explain how far up and to the right one should turn the telescope, assuming the telescope is currently focused towards the –x axis. For example, the blue point on the sphere is at $\delta = 0°,\ \alpha = 0°$.



To find a point using these two angles, first turn the telescope towards the +y axis if $\delta > 0°$ or towards the –y axis if $\delta < 0°$. It is always the case that $-90° \leq \delta \leq 90°$. Next, rotate the telescope counter clock-wise about the y axis for α degrees. The right ascension can take the full range from values from $0 \leq \alpha \leq 360$. For example, to find the green point at $\delta = 45°$, $\alpha = 135°$ the telescope performs the following maneuvers:





Given the orientation of the axes (which conveniently align with XNA framework that we shall use for simulation), the conversion from angles to points on a units sphere is given by:

$$x = -\cos(\alpha)\cos(\delta)\,, \qquad z = \sin(\alpha)\cos(\delta)\,, \qquad y = \sin(\delta)$$

2. Reference Frame

The International Celestial Reference Frame (Version 2) is a set of over 3,400 points on the sphere that have been observed by radio telescopes over the last 30 years. The ICRF2 database provides landmarks for describing the exact positions of objects in this spherical coordinate system. We shall use these landmarks to navigate.

In practice, astronomers do not use degrees to describe declination and right ascension. Declination is described by three values: Whole degrees (d), minutes of an arc (moa), and seconds of an arc (soa). The first two values are integers while the third may be fractional. The conversion to degrees is given by:

$$ToDegrees(d, moa, soa) = \begin{cases} \left(d + \dfrac{moa}{60} + \dfrac{soa}{3600}\right) & if\ d \geq 0 \\ -\left(d + \dfrac{moa}{60} + \dfrac{soa}{3600}\right) & if\ d < 0 \end{cases}$$

Right ascension is described by three values: Hours (h), minutes (m), and second (s). Again, the first two values are integers. Note that **minutes** and **minutes of an arc** are two different units. (Also, seconds and seconds of an arc are different.) The conversion to degrees is:

$$ToDegrees(h, m, s) = 15h + \frac{m}{4} + \frac{s}{240}$$

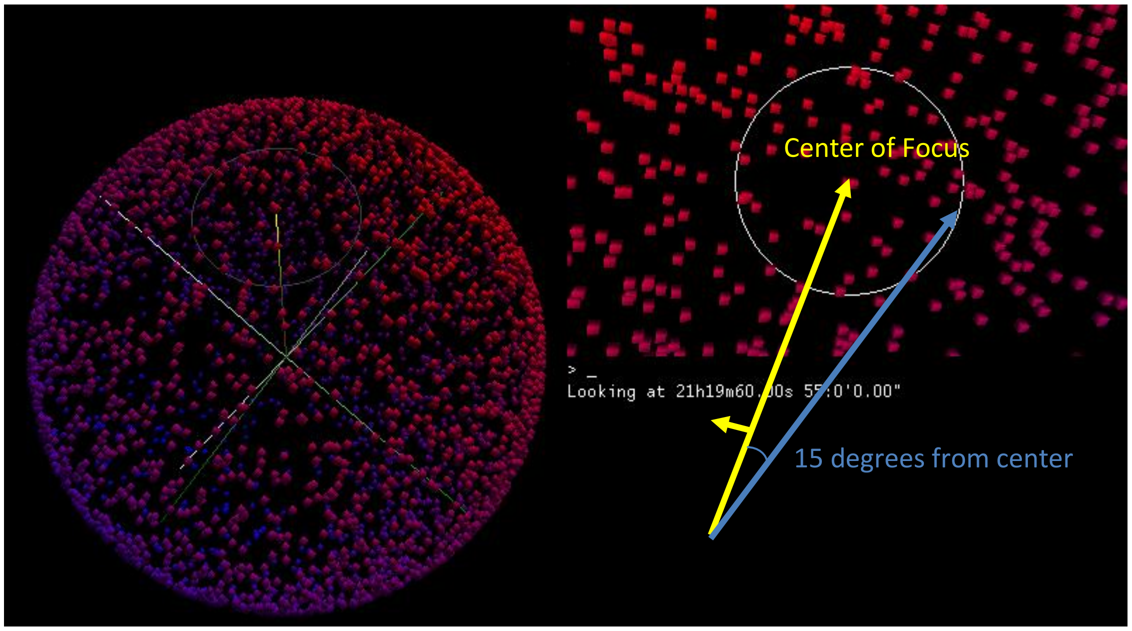
Hours are in the interval [0, 23]. Minutes/MOA are in the interval [0, 59]. Seconds/SOA are in the half-open interval [0,60). Whole degrees are in the interval [-90, 90]. If the whole degrees of declination are exactly -90 or 90, then moa and soa must be zero.

3. Navigation

The space probe carries a radio telescope and travels in the direction the telescope is looking. Therefore, it must determine where the telescope is looking with respect to the sphere to ensure it is traveling in the right direction. Fortunately, the objects observed by the telescope are so far away that the motion of the probe does not affect the angular distances between objects. In other words, we can assume the probe is always at the center of the sphere.

To determine where the telescope is looking, it must try to match the pattern of points it sees with those in the ICRF2 database. Once the coordinates of two points are identified, then the direction of travel can be computed.

The navigation system receives an array of points from a radio telescope with 30 degree field-of-view (FOV). In other words, the telescope observes all points that are up to (and including) 15 degrees from the center of its focus, as shown below.



```
> _
Looking at 21h19m60.00s 55:0'0.00"
```

Because the telescope does not know where it is pointed, it reports its observations assuming it is pointed as $\delta = 0°$, $\alpha = 0°$. Thus, the coordinates reported by the telescope cannot be directly used to compute direction. They are rotated by an unknown amount. You must compute the center of focus using these points and the ICRF2 database.

4. Design Navigation Algorithm

Design the navigation algorithm by filling in the contents of the following method:

```
public CelestialPoint[] Locate(CelestialPoint[] observations)
{

}
```

The input to the method is a set of points observed by the telescope and reported from its local frame of reference (assuming the center of focus $\delta = 0°$, $\alpha = 0°$). The output of the method is an array containing the possible positions the telescope could be pointed. These should be expressed in the true coordinate system. Typically, the observations should provide enough information to identify a unique point.

As you design your algorithm write the requirements and guarantees on methods using Code Contracts. Apply static analysis to detect potential bugs in the code and correct these when possible.

5. Getting Started

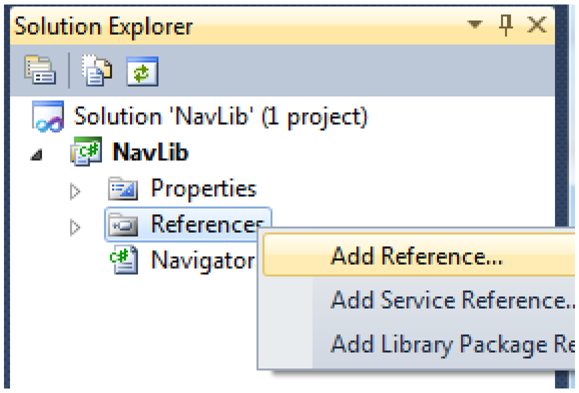
1. Install Code Contracts: http://msdn.microsoft.com/en-us/devlabs/dd491992

2. Install the XNA redistributable framework: http://www.microsoft.com/download/en/details.aspx?id=20914

3. Download the file CSEP503_Project1.zip from the class website and unzip it to location **$Project**.

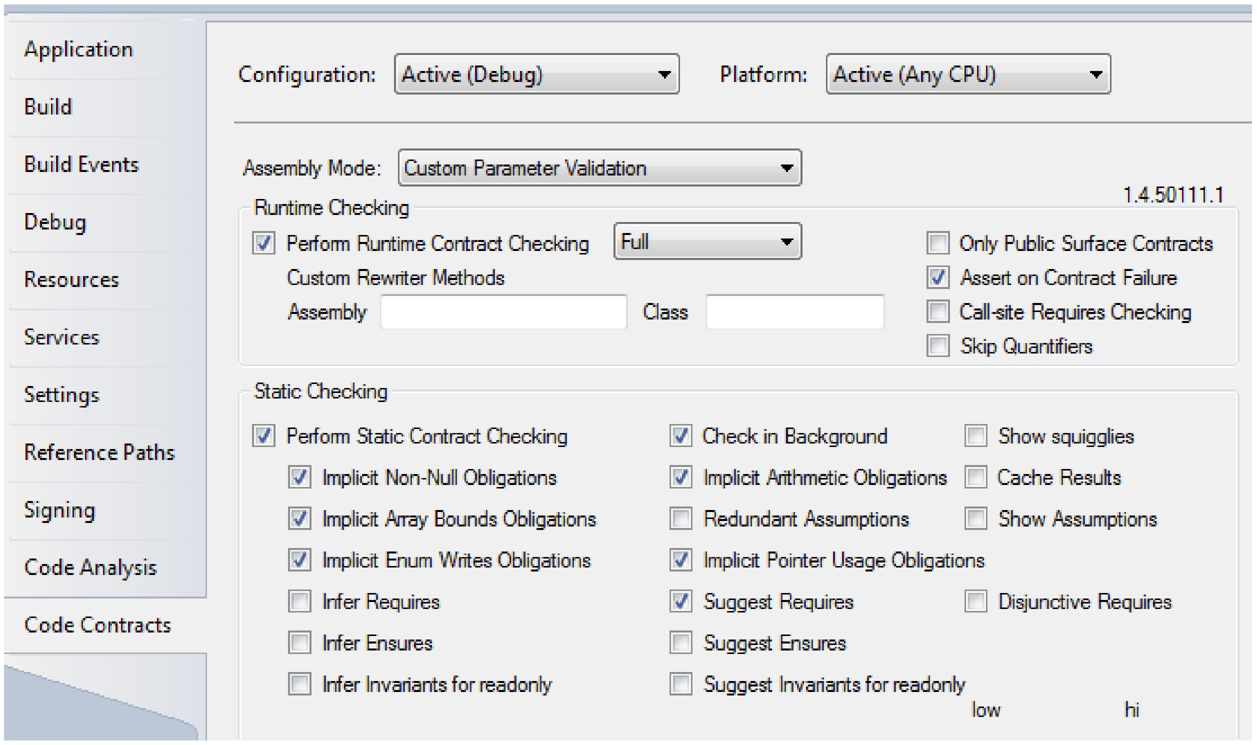
4. Open the solution **$Project\NavLib\NavLib.sln** in Visual Studio.

5. Add a reference to **CelestialUnits.dll**. From the **Solution Explorer**, right-click on **References**, and click **Add Reference**. (See figure below)



Click Browse and browse **$Project\NavSim.** Double-click on **CelestialUnits.dll**, then click **Add** and then **Close** the dialog box.

6. Enable code contracts by selecting **Project -> NavLib Properties**... You should select the following options:



Configuration: Active (Debug) Platform: Active (Any CPU)

Assembly Mode: Custom Parameter Validation

1.4.50111.1

Runtime Checking

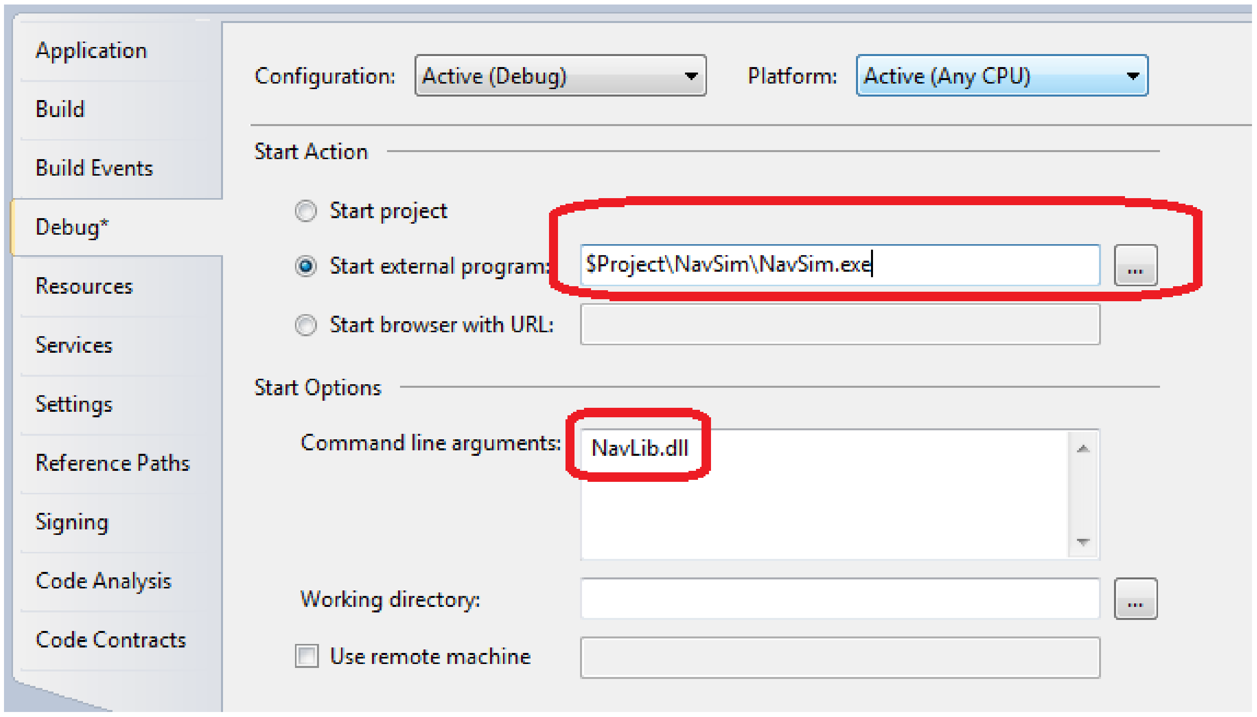
- [✓] Perform Runtime Contract Checking Full
- Custom Rewriter Methods
- Assembly Class
- [ ] Only Public Surface Contracts
- [✓] Assert on Contract Failure
- [ ] Call-site Requires Checking
- [ ] Skip Quantifiers

Static Checking

- [✓] Perform Static Contract Checking
- [✓] Implicit Non-Null Obligations
- [✓] Implicit Array Bounds Obligations
- [✓] Implicit Enum Writes Obligations
- [ ] Infer Requires
- [ ] Infer Ensures
- [ ] Infer Invariants for readonly
- [✓] Check in Background
- [✓] Implicit Arithmetic Obligations
- [ ] Redundant Assumptions
- [✓] Implicit Pointer Usage Obligations
- [✓] Suggest Requires
- [ ] Suggest Ensures
- [ ] Suggest Invariants for readonly
- [ ] Show squigglies
- [ ] Cache Results
- [ ] Show Assumptions
- [ ] Disjunctive Requires

low hi

7. Set the debug executable. Click on the Debug tab in the above window, and provide the following settings. (See figure on next page.)



Configuration: Active (Debug)
Platform: Active (Any CPU)

- ○ Start project
- ● Start external program: $Project\NavSim\NavSim.exe
- ○ Start browser with URL:

Command line arguments: NavLib.dll

Working directory:

☐ Use remote machine

8. Pressing F5 should start the debugger using the NavSim.exe simulator. Type **locate** in the simulator to call your Locate method. You may also set a break point on your Locate method to catch this event. Type **help** in the simulator to see a list of commands.