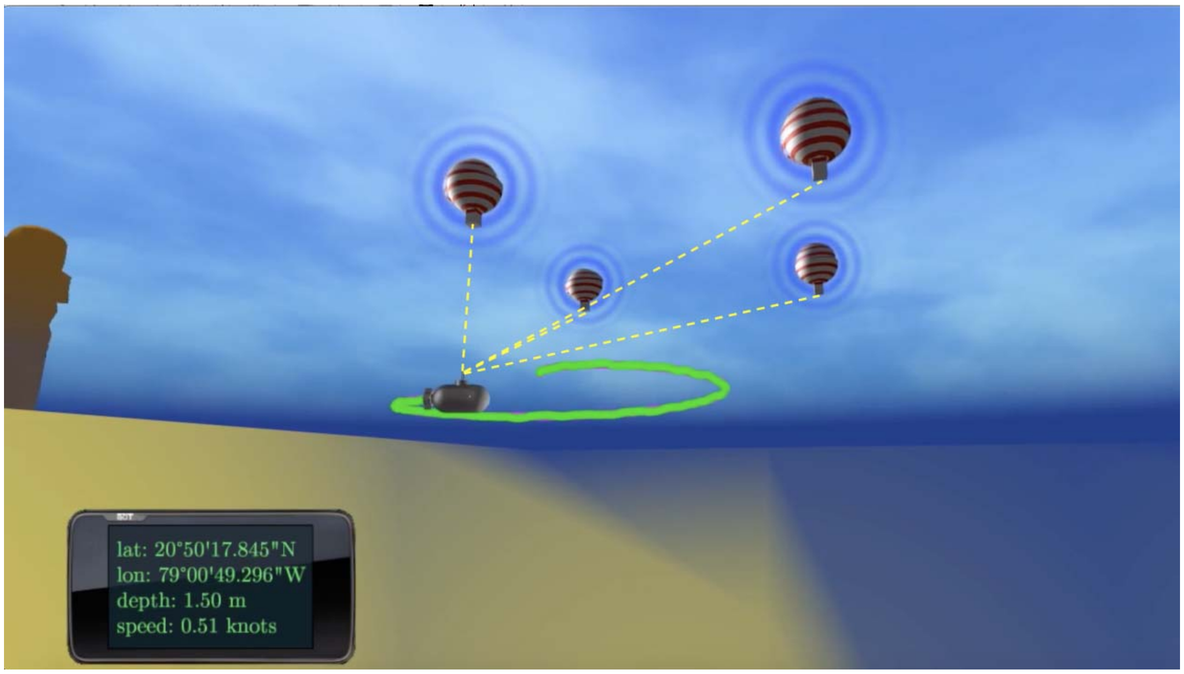
Target Localisation

Overview



Knowing where a target or platform is located in uncontrolled environments is one of the challenging problems in robotics. The position of a robot arm is relatively straightforward through the use of encoders on arm joints whereas a robot in a city, the air or underwater has greater challenges. GNSS like GPS, GLONASS, Galileo and BeiDou can provide some degree of accuracy but tend to fail in 'urban canyons' or provide too slow or imprecise updates for UAVs operating in tight environments and cannot work when underwater at all.

You are to implement a tracking system for targets operating near a set of tracking buoys. These buoys can get range observations to the target and their own position is well known.

You will have to decode the messages from the buoys, identify which of these messages relate to the targets of interest and solve the set of linear equations that give you the target location. Not all buoys will manage an observation every time. Buoys may also drift over time and will provide updated position information.

Buoy Messages

Buoys will transmit messages to you through `stdin`. There are two types of messages that can be transmitted:

- Buoy Positions: `$BP,<time>,<buoy ID>,<x>,<y>,<z>*<checksum>\n`

- Target Ranges: `$TR,<time>,<buoy ID>,<target ID>,<range>*<checksum>\n`

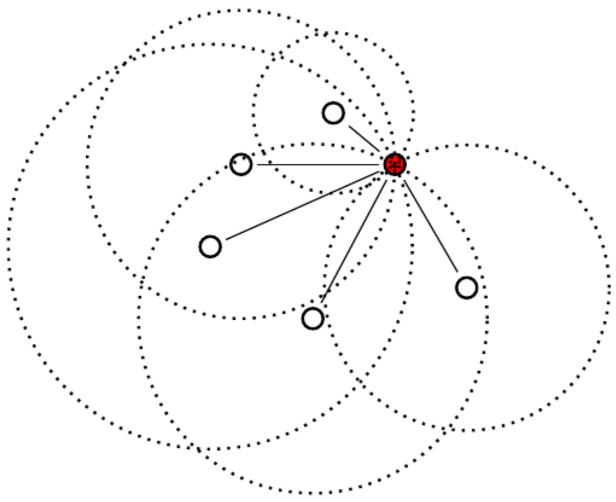
Times are all floating point values in seconds, IDs are integers and will not overlap (no buoy can have the same ID as a target). Coordinates and ranges are all given in metres in a local coordinate system and with x being north, y being east and z down.

Checksum is the decimal representation of an 8 bit unsigned value made of the last 8 bits (modulo 256) of the sum of all characters between (but not including) '$' and '*'. If the checksum is invalid the message is to be ignored.

Target Position

A target or set of targets will be localised for each invocation of the program at one or multiple times. The location to be solved for will be in three dimensions and will be achieved through the use of multiple range measurements from buoys or sensors at known locations. Given range measurements a number of methods can be employed to solve for the positions. The method you are using here is based on solving sets of linear equations.

A depiction of the concept (reduced to two dimensions) is shown below. Each of the (white) buoys measures range to the red target. The dotted circles represent the locations that are the same range from the respective buoys. The location where these circles intersect is the estimated position. Real world observations like the range estimate always have a measure of uncertainty or noise included and will not be exact.



Posing the Problem

Range measurements from the `i` th of `n` buoy positions `(x_i, y_i, z_i)` to the unknown target position `(x_t, y_t, z_t)` can be expressed as:

$$r_i^2 = (x_i - x_t)^2 + (y_i - y_t)^2 + (z_i - z_t)^2$$

These equations are nonlinear in unknowns `x_t, y_t, z_t` however we can create a set of `n-1` equations linear in the target coordinates by subtracting the range equation from a selected buoy - choosing buoy `i=1` here for example as our reference buoy:

$$r_i^2 - r_1^2 = (x_i - x_t)^2 + (y_i - y_t)^2 + (z_i - z_t)^2 - (x_1 - x_t)^2 - (y_1 - y_t)^2 - (z_1 - z_t)^2$$

Expanding the right hand side we find the equation linear in our target coordinates and further rearrangement moves the known terms to the left hand side. This gives our final equations, valid for all buoys after the first as:

$$2(x_i - x_1)x_t + 2(y_i - y_1)y_t + 2(z_i - z_1)z_t = r_1^2 - r_i^2 + (x_i^2 + y_i^2 + z_i^2) - (x_1^2 + y_1^2 + z_1^2)$$

Which can be treated as a 3 column matrix **A** made of the coefficients of the target coordinates , **x** a column vector of our three unknowns
and **b** of the accumulated right hand sides of the equations. Each linear equation adds a row to **A** and **b**.

$$\mathbf{Ax} = \mathbf{b}$$

Expanding these matrices for a few rows gives the forms:

$$\begin{bmatrix} 2\left(x_2 - x_1\right) & 2\left(y_2 - y_1\right) & 2\left(z_2 - z_1\right) \\ 2\left(x_3 - x_1\right) & 2\left(y_3 - y_1\right) & 2\left(z_3 - z_1\right) \\ \vdots & \vdots & \vdots \\ 2\left(x_j - x_1\right) & 2\left(y_j - y_1\right) & 2\left(z_j - z_1\right) \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} r_1^2 - r_2^2 + \left(x_2^2 + y_2^2 + z_2^2\right) - \left(x_1^2 + y_1^2 \right. \\ r_1^2 - r_3^2 + \left(x_3^2 + y_3^2 + z_3^2\right) - \left(x_1^2 + y_1^2 \right. \\ \vdots \\ r_1^2 - r_j^2 + \left(x_j^2 + y_j^2 + z_j^2\right) - \left(x_1^2 + y_1^2 \right. \end{bmatrix}$$

Solving Linear Equations

Solving the linear equations can be achieved as long as there are least as many equations as unknown variables. We have three unknown variables (coordinates of the target) and each buoy after the first generates a new (ideally independent) equation, so a minimum of four buoys is required to give three equations. `n` buoys gives `m = n - 1` equations.

Given the matrix of coefficients of the unknowns which is an `mx3` matrix **A** and the `mx1` vector **b** associated with it we want to solve for the unknown `3x1` vector **x**:

$$\mathbf{Ax} = \mathbf{b}$$

A is not necessarily square so we cannot simply invert **A** and multiply on the left. An alternate approach is multiple on the left by the transpose of **A**.

$$\mathbf{A^T A x} = \mathbf{A^T b}$$

`A^T A` is a `3x3` matrix, `A^T b` is a `3x1` column vector. We still need to solve for **x**. This can be done via `3x3` matrix inversion - but this isn't as simple as the `2x2` case and doesn't extend well if we have a larger matrix to invert.

A common set of approaches are based on decomposing the matrix `A^T A` into two or three matrices that are easier to invert or solve. Variations of this generally involve diagonal, upper and lower triangular matrices. One method that words well in this situation is using a Cholesky Decomposition which decomposes `A^T A` into two square triangular matrices which are the transpose of each other. The two common ways of expressing these are **L** left triangular and **R** right triangular matrices such that `L L^T = R^T R = A^T A`.

This enables the expression for **x** to be reposed using triangular matrices - for which we have a solve function in the last quiz.

$$\mathbf{A^T A x} = \mathbf{A^T b}$$

$$\mathbf{L L^T x} = \mathbf{A^T b}$$

$$\mathbf{L y} = \mathbf{A^T b},\ \text{solve for y}$$

$$\mathbf{L^T x} = \mathbf{y},\ \text{solve for x}$$

A number of algorithms are possible for the Cholesky decomposition. One is possible that uses the symmetric matrix rank update operation that we have previously developed. A code like version of this algorithm is shown below that performs the decomposition in place (in the same memory location) as the input symmetric matrix to give the lower triangular matrix **L**:

Text

```
for j=1:n
    A(j, j) = sqrt(A(j, j))

    A((j+1):n, j) /= A(j, j)

    // symmetric matrix rank update, k=1, alpha = -1.0
    A((j+1):n, (j+1):n) += - A((j+1):n, j) * A^T((j+1):n, j)

endfor
```

The Program

Your program `localise` will take input from `stdin` that contains the range and position information from the buoys. Command line arguments will provide the target id(s) of interest - multiple targets are an extension and not required for most of the marks. Information relating to other targets will be ignored. Your program will output time and coordinates for the target for points in time at which the position can be determined by the above algorithms.

Specification

1. The program shall be called `localise`.

2. The program shall be built by a `Makefile`
3. The `Makefile` shall have a clean option that deletes all compiled and linked modules (excluding supplied `lar.o` if using).
4. The program shall accept the argument `-t` followed by the integer ID for a target of interest. This can be repeated multiple times. If not present no target localisation output is required.
5. The program shall accept the argument `-b` followed by the highest buoy ID in use. The program shall assume a value of 4 if not specified.
6. If `-b` is repeated an error message shall be printed to `stderr` of "`Error: -b was repeated.`" and the program shall return a value of 1.
7. If `-b` is supplied with an argument lower than 3 an error message of "`Error: less than 4 buoys is insufficient to localise a target.`" and the program shall exit with a return value of 3.
8. If `-t` is supplied with a value that is equal to or below the maximum buoy ID supplied with `-b` an error message of "`Error: Target ID within buoy ID range.`" shall be printed to `stderr` and the program shall exit with a return value of 4.
9. The program shall accept the argument `-c` that indicates checksum errors should be printed to `stderr`. See later condition for required response.
10. Any other argument shall be rejected and to `stderr` shall be printed error message "`Error: Unknown argument '-%c'.`" where `%c` indicates the unknown character provided and the program shall return the value 2.
11. Program arguments shall be responded to left to right and errors handled as they occur.
12. The program shall receive lines from `stdin` that contain messages from the buoys.
13. All buoy messages shall start with '$ and end with '*' followed by a decimal number in the range 0 to 255 and a newline character.
14. The final decimal number after '*' is the checksum that shall be calculated as the sum of all (unsigned) characters between, but not including, the '$' and '*' characters modulo 256 (remainder after dividing by 256).
15. If a checksum doesn't match the message contents an error message shall be printed to `stderr`. The error message shall be "`Error: checksum for message '%s' failed.`" where the message without the newline character should be included at `%s`. Program execution should continue and the failed message shall be ignored for all other purposes.
16. Components of buoy messages shall be separated by a comma character ',' with no spaces.

17. The first component of a buoy message shall be two alphanumeric characters that indicates the type of the message.

18. The second component of a buoy message shall be a transmission time given in seconds accurate to two decimal places.

19. The third component shall be an integer indicating the ID of the buoy supplying the message.

20. If the message type is "TR" or target range the fourth and fifth components shall be the integer IDs of the target and the floating point range in metres to that target from the buoy.

21. If the message type is "BP" or buoy position the fourth, fifth and sixth components shall be the new coordinates of the buoy in metres given in the order north, east and down from the origin location.

22. Unknown message types shall be silently ignored.

23. The minimum value for Buoy IDs shall be 0.

24. The minimum value for Target IDs shall be one greater than the highest buoy ID in use.

25. The maximum value for Target IDs shall be 32767.

26. Messages that arrive within the same second are to be treated as simultaneous. As an example messages between 4.00 and 4.99 inclusive are to be treated as simultaneous - with buoy position updates being applied before ranges are used to localise.

27. All valid range observations of the targets shall be used.

28. Estimation of the position of the target shall be performed using the algorithms described above.

29. Output for all targets that can be validly observed in each time period shall be to `stdout` and take the form "`$TP,<time of last used message>,<target id>,<x>,<y>,<z>*<checksum>\n`"

30. The time of last used measurement refers to the last measurement in that time step that is valid and refers to a buoy or tracked target.

31. Output coordinates shall have at least 3 values after the decimal point.

32. Output times shall have at least 2 values after the decimal point.

33. Execution shall cease when all data has been read in and the EOF flag is set. The program shall return 0.

Libraries

You may use any functions in the C11 Standard Library and getopt - part of the GNU C Library (no extra linker flags are required). You may additionally reuse your code from Quiz 3's Linear Algebra Routines or a lar.h/lar.o module that will be supplied. You may not use any external sources of code, regardless of licensing. The following libraries are likely to be most useful:

- Header `<stdio.h>` has input/output functions: `fgets()`, `fprintf()`
- Header `<stdlib.h>` has text to numeric functions `strtod()` and more.
- Header `<string.h>` has character finding and string splitting functions `strchr()`, `strtok()` and more.
- Header `<assert.h>` has the bug-catching macro `assert()`
- Header `<getopt.h>` has the command line parsing helper `getopt()`.

Academic Honesty

This assignment is to represent individual work. You are not to collaborate with your peers, this is an individual assignment. You may discuss your approach to the problem with other students, but you may not share code, look at another student's code, or allow another student to look at your code. Similar code will be treated with suspicion. All submissions will be checked for similarity.

For those who haven't completed quiz 3 a compiled module will be supplied that implements those functions using the lar.h header. You shall not use another students code for the quiz functions. You may use code you previously submitted for that quiz.

Marking

Compliance with the specification will tested by Ed's automated testing process. Compliance with the specification is worth 75% of this assignment.The remaining 25% of the assignment is awarded for quality of your code. Things that will be assessed include, but are not limited to:

- Appropriate and consistent formatting and style
- Project layout
- Good coding practices (e.g. no global variables or "magic numbers")
- Self-documenting code, including appropriate commenting
- Appropriate use of functions

Late Submission

Late submissions will be penalised at a rate of 5% of the value of the assignment per day or part thereof. Submissions that are more than 10 days late will not be accepted. The last submission is the one that will be assessed, regardless of whether an earlier one has a greater value in test cases solved.