24th COBEM - 2017

24th ABCM International Congress of Mechanical Engineering
December 3-8, 2017, Curitiba, PR, Brazil

COBEM-2017-0317
NAVIGATION ROUTINES USING ONLY INTEGER CALCULATIONS FOR MINIMUM CODE FOOTPRINT

Ernani Melo de Sousa Reis
Instituto Tecnológico de Aeronáutica, Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, 12228-900
ernani.reis@gmail.com

Abstract. *This work describe routines developed to aid in the navigation of vehicles, given the present position and destination, the routine computes bearing and range to the destination. The objective was to allow for an implementation in limited processors, requiring little memory and processing power. The program uses no floating point, resulting in small and fast code, with precision adequate to navigation in short steps, around 10km.*

Keywords: *Navigation Algorithm, MAV, UAV, NMEA*

1. INTRODUCTION

The computation of vehicle guidance requires complex unit conversion (degrees of latitude to metres, for example), or large number of significative bits, or large numbers. These conditions can be a problem when the available computational resource is limited, and frequently is. Another trend is the use of FPGA to control vehicles, and the implementation of floating point operations in FPGA can be problematic. The idea was to create routines that compute range and bearing between two points from their geographic coordinates, crucial to navigation, that can be read from a GPS receiver, for example, along with routines that can interpret the NMEA representation of those coordinates. Another application is "return to home" devices. These record a point of departure, read GPS messages, and indicate the way back to the departure point. This kind of application, usually, is implemented in very small machines. This function is common on first person view (FPV) systems, in which the operator flies the vehicle by remote control, and, by a camera installed on the vehicle, video transmitter, receiver and a monitor, has the point of view of a pilot on board the vehicle being remotely piloted.

2. AVAILABLE RESOURCES

Small vehicles autonomous flight is controlled by small computers, that typically will do all the required calculations using floating point numbers. There are many valid approaches, that can be used, largely known. For very small processors, the suitable resources available are very limited, requiring the processing power, not always attainable or convenient. Examples of systems with similar function are many, naming a few: ETH's PIXHawk, mainly for multi rotors ((Meier *et al.*, 2014)), with a 32 bit ARM processor; ENAC's paparazzi ((Remes *et al.*, 2014)), another solid code running on another 32 bit ARM processor; or the famous AuduPilot (Munoz (2014)), running on ATMega with no shortage of memory.

3. APPROACH TAKEN

The first simplification was to consider a spherical geoid. this approximation yields reasonable results, and simplifies the required computations. Another simplification is to treat the surface of the geoid as a plane around the points being processed. The errors caused by these simplifications are reasonable for points within a few tens of kilometres from each other.

In a spherical geoid the latitude is directly related to a distance, as long as the perimeter across the geoid does not deviate significantly from the straight line between the points, in a 6,371 km radius circle, the approach is solid for distances limited to a few tens of kilometres. We will treat all distances in seconds of latitude multiplied by 100, this has a smallest increment of about 30 cm. The latitude (ϕ) will be noted in seconds of arc multiplied by 100 directly, no other processing is required.

The longitude (λ) is related to distances at the equator. For other latitudes, the longitude difference is translated into a distance by $\delta\lambda * \frac{r_l}{r}$, being r_l the local radius around the earth's north-south axis and r the radius of the geoid. The local radius, in the spherical geoid, is $r_l = r * cos(\phi)$. The distance along the longitude is, then, $\delta\lambda * cos(\phi)$ in a representation

similar to the one used for the latitude, seconds of arc multiplied by 100.

All calculations are performed with 32 bit integers, and normalized for best use of the precision available.

The conversions for longitude at latitudes different from the equator require a latitude value. The path include a range of latitudes, that may be different. For those cases, will be used one latitude for the conversion, and will be assumed that the differences will be negligible, certainly the case for few tens of km long paths.

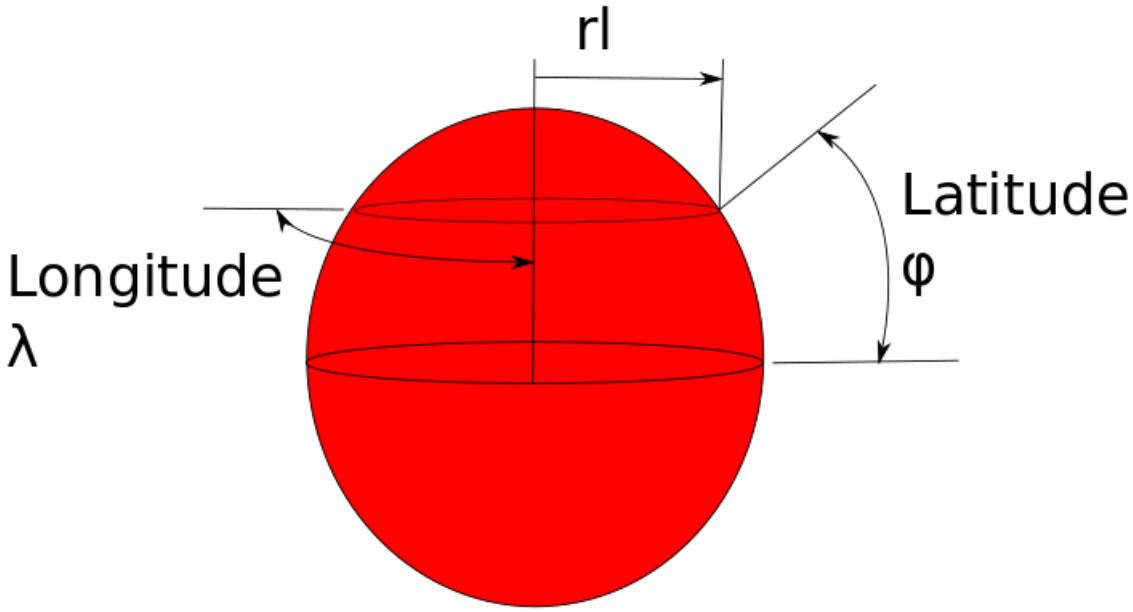

Figure 1. Convertion of Coordinates to Distances

4. IMPLEMENTATION

The routines require computation of sine, cosine, arc sine, arc cosine and square root. The square root was used from Dijkstra (1996). The trigonometric functions were implemented with a lookup table for sine, in full degree steps. This table is used in both directions, to convert angles to sine or cosine, and the other way around. The conversions are made at the optimum sensitivity.

The program flow is:

- For both coordinates of both points, convert the angles from NMEA notation to seconds of arc multiplied by 100
- Calculate the cosine of ϕ_{origin}
- Calculate $\delta\phi$ and $\delta\lambda$ adjusted with the $cos(\phi)$
- Calculate the distance with the adjusted $\delta\phi$ and $\delta\lambda$ - uses a square root
- Calculate the bearing to the second point - uses an arc tan
- Adjust the calculated bearing to the proper quadrant
- Return

5. RESULTS

The size of portions of interest in the code is on table 1, where we can see a code footprint of 3,064 octets. Excluding the NMEA parser (800) and square root (342, if range to waypoint in not required), we have a footprint of 1,922 octets. At the time of writing, a small processor, 8k octets program memory, 8 bit CPU. With 3K octets used in this implementation, leaves about 5K octets for other purposes. A floating point for a processor like this occupies around 3K octets, the navigation routines should double that, at least, leaving less than 2K octets for everything else. The code is open, licensed by GPL, can be found at https://code.google.com/archive/p/cl-osd/.

Table 1. Compiled Code Size - Of Selected Sections

Label	Size
myCos	50
mySin	162
sinData	180
raiz	342
DecG	800
chome	1530

6. TESTS

The tests consisted of arbitrarily choosing seven destination points in all hemispheres, around each of them, chosen thirteen origin points, to cover all quadrants up to 20 km distant (aproximately). The destinations were adjusted so that all origins would be on the same UTM zone. An example of a distribution origins and destination can be seen in figure 2.

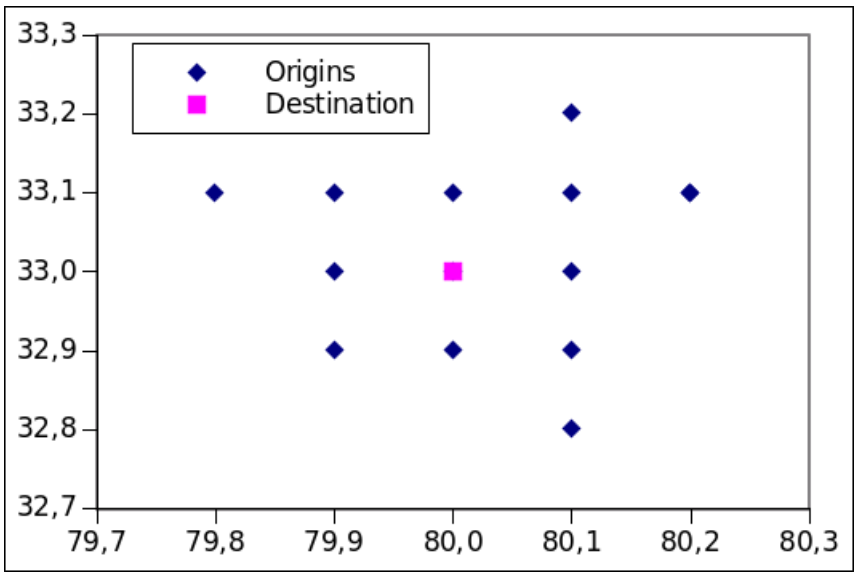

Figure 2. Example of distribution of destination and origins

For all origins, was calculated the range and bearing to the destination, using the routine developed. For each of the destinations, the origins were placed at three different ranges, that depend on the latitude. The range and bearing are illustrated on fig 3.

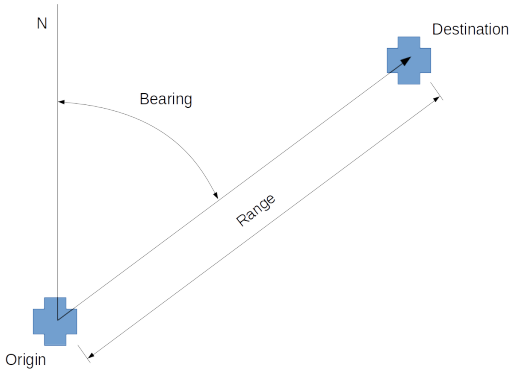

Figure 3. Range and Bearing definition

For all points, their coordinates were converted to UTM assuming WGS84, using (Karney and Deakin, 2010), on a

gnumeric spreadsheet. From the calculated UTM positions, were calculated the same bearing and range for each case. The two results for each case were compared. For each o the chosen destinations, the modulus of the differences between the data computed by the proposed program, and by the UTM positions. The modulus of the differences were averaged for each of the three approximated ranges. The results are presented in table 2.

Table 2. Error Observed for Various Situations

Destination			Error	
Lat (Deg)	Lon (Deg)	Range (m)	Range	Bearing (Deg)
2	2	11124	0,3%	0,52
2	2	15731	0,3%	0,17
2	2	20250	18,3%	7,40
-20	40	10790	0,3%	0,67
-20	40	15304	0,5%	0,61
-20	40	20250	16,2%	6,21
40	-125	9844	0,2%	1,29
40	-125	14073	0,5%	1,29
40	-125	20250	9,1%	3,25
-60	125	8343	0,2%	1,73
-60	125	12486	0,4%	1,73
-60	125	18570	7,8%	1,90
70	125	7453	0,6%	1,88
70	125	11786	0,3%	1,88
70	125	17531	6,5%	2,12
80	33	6508	1,4%	0,52
80	33	11302	0,2%	0,65
80	33	16850	6,0%	0,85
88	-33	5728	7,5%	0,52
88	-33	11132	0,3%	0,50
88	-33	16607	5,8%	0,30

7. CONCLUSIONS

This paper describes the development of a routine to determine the bearing and range to the next waypoint in a navigation . The developed rotine was intended to be used in minimal machines, with little memory or computing power. The function developed presented reasonable precision for distances under around 10km, a reasonable compromise for small vehicles. Longer distances showed poor results, a reasonable workaround would be adding intermediary waypoints. The assumptions made proved reasonable, yielding the desired performance for the intended conditions, requiring very limited resources, allowing for an implementation in small microprocessors or FPGA.
This code has been used on the CL-OSD, a system to indicate the point of takeoff of a "FPV" (Fist Person View), camera and rf transmitter installed in a airmodel, allowing the pilot to fly as being on board. The OSD (On Screen Display) superimposes flight data to the image transmitted, including a pointer and the range to the point of takeoff. The system was implemented on a ATMEGA8, with 8K octets of program storage space, insufficient to the floating point library and necessary code. The system was extensively discussed in RCGroups (www.rcgroups.com/forums/showthread.php?1490446-CL-OSD-a-open-source-osd-software-for-E-OSD-and-G-OSD).

8. FUTURE WORK

Next step in this line of work is an implementation on a FPGA, with an interleaved inertial sensor acquisition, for very high speed flight controller.

9. REFERENCES

Dijkstra, W., 1996. “Arm code square root routines”. *news:comp.sys.arm*, Vol. 1, No. 1, p. 1.

Karney, C.F.F. and Deakin, R.E., 2010. “F.w. bessel (1825): The calculation of longitude and latitude from geodesic measurements”. *Astronomische Nachrichten*, Vol. 331, No. 8, pp. 852–861. ISSN 1521-3994. doi:

10.1002/asna.201011352. URL http://dx.doi.org/10.1002/asna.201011352.

Meier, L., Tanskanen, P., Fraundorfer, F. and Pollefeys, M., 2014. "Pixhawk: A system for autonomous flight using onboard computer vision".

Munoz, J., 2014. *ArduPilot 2.x manual*.

Remes, B., Esden-Tempski, P., Van Tienen, F., Smeur, E., De Wagter, C. and De Croon, G., 2014. "Lisa-s 2.8g autopilot for gps-based flight of mavs". *IMAV 2014: Proceedings of the International Micro Air Vehicle Conference and Competition 2014*.

10. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.