FFI Norwegian Defence Research Establishment

22/01262

FFI-RAPPORT

FASTmap: A Multi-Layer Sliding Gridmap for Arbitrary Types

Håkon Yssen Rørstad
Thomas Røbekk Krogstad

FASTmap: A Multi-Layer Sliding Gridmap for Arbitrary Types

Håkon Yssen Rørstad
Thomas Røbekk Krogstad

Norwegian Defence Research Establishment (FFI) 20 June 2022

Keywords
Programvare
Digitale kart
Rammeverk
C++ (Progammeringsspråk)
Brukermanualer
Autonomi

FFI report
22/01262

Project number
1604

Electronic ISBN
978-82-464-3415-5

Approvers
Håkon Storli Andersen, *Director of Research*
Martin S. Wiig, *Research Manager*

The document is electronically approved and therefore has no handwritten signature.

Summary

This report presents the Fixed Area Sliding Template map (**FASTmap**), a versatile and efficient C++ mapping framework for storing arbitrary and heterogeneous data types in a sliding grid map. The framework was first developed to map the surroundings of autonomous vehicles as these are often equipped with a diverse suite of sensors gathering heterogeneous measurements from the local environment. Gathering this information in a common data structure can provide an autonomy system with a situational awareness of the local environment. This further enables the autonomy system to adapt its behavior to the terrain or evaluate autonomous task and sensor performance.

After its initial conception, FASTmap has been generalized into a framework supporting various mapping applications, but autonomous vehicles is still considered the main application. A large part of the core functionality is therefore dedicated to moving the map as efficiently as possible without compromising the flexibility to store general types. As such the map is designed to be frequently moved by using internal circular buffers to achieve real time repositioning of data for maps of appropriate sizes and resolutions.

The main focus of the report is to give an introduction to FASTmap suited for new users. To this end the report contains the necessary theory, code snippets and examples to start developing a mapping application. Relevant navigation theory is presented in detail as this subject is tightly coupled with mapping. Especially the interchangeability of grid maps and tangent reference frames is stressed as an important relation. A complete list of the current application programming interface functions, with elaborating comments, is also included as a reference for mapping application developers.

In addition to the user-oriented introduction to the framework, more advanced topics like performance benchmarking and maximum error calculations are discussed in the later chapters. First FASTmap is compared to another similar mapping framework called Gridmap, and it is concluded that FASTmap is more versatile because the choice of data types and memory layout can be specified by the user. Then runtimes for common and frequently used functionality for both maps are compared and examples indicate similar performance for similar map sizes. The results are however not conclusive because of insufficient randomization of the experimental setup, and this is suggested as a topic for a dedicated work. Finally a function for the maximum position error introduced by map movement on the Earth ellipsoid is derived. This error is concluded to be neglectable compared to the error introduced by the grid resolution in most practical cases.

Sammendrag

Denne rapporten presenterer Fixed Area Sliding Template map (**FASTmap**), et fleksibelt og effektivt C++-rammeverk for lagring av arbitrære og heterogene datatyper i et bevegelig rutenett. Rammeverket ble først utviklet for å kartlegge omgivelsene til autonome fartøy, siden disse ofte er utstyrt med et variert utvalg sensorer som innhenter heterogene målinger fra det lokale miljøet. Samling av denne informasjonen i en felles datastruktur kan gi et autonomisystem en forståelse av situasjonen og det lokale miljøet. Autonomisystemet kan videre bruke denne forståelsen til å tilpasse oppførselen sin til terrenget eller evaluere autonom oppgave- og sensorytelse.

Den originale versjonen av FASTmap har senere blitt generalisert til et generelt rammeverk som støtter mange forskjellige kartapplikasjoner, men autonome fartøyer regnes fremdeles som hovedanvendelsen. En stor del av kjernefunksjonaliteten er derfor dedikert til å flyttet kartet så effektivt som mulig uten å miste fleksibiliteten til å lagre generelle typer. Kartet er derfor designet for å flyttes ofte ved å bruke interne ringbuffere til å reposisjonere data i sanntid for kart av realistiske størrelser og oppløsninger.

Hovedfokuset i denne rapporten er å gi en introduksjon til FASTmap tilpasset nye brukere. Rapporten inneholder derfor nødvendig teori, kodeutsnitt og eksempler for å starte utvikling av en kartleggingsapplikasjon. Relevant navigasjonsteori er presentert i detalj siden dette fagfeltet er tett integrert med kartlegging. Det er lagt spesiell vekt på likheten mellom rutenett og tangente referanserammer. En komplett liste over de nåværende programmeringsgrensesnittfunksjonene, med forklaringer, er også inkludert som en referanse for kartapplikasjonsutviklere.

I tillegg til den mer brukerorienterte introduksjonen til rammeverket, blir mer avanserte temaer som ytelsesevaluering og utregning av maksimal feil diskutert i de senere kapitlene. Først blir FASTmap sammenlignet med et annet lignende kartrammeverk kalt Gridmap, og det blir konkludert at FASTmap er mer fleksibelt siden datatyper og struktureringen av minne spesifiseres av brukeren. Deretter blir kjøretidene til felles og hyppig brukt funksjonalitet for begge kart sammenlignet, og kjøretidseksempler indikerer sammenlignbar ytelse for lignende kartstørrelser. Resultatene er derimot ikke nok til å konkludere siden testene ikke ble utført i et tilstrekkelig randomisert miljø, og det anbefales å jobbe videre med dette i et dedikert arbeid. Til slutt blir det utledet en funksjon for den maksimale posisjonsfeilen introdusert ved flytting av kartet på jordellipsoiden. Denne feilen blir konstatert neglisjerbar i forhold til feilen introdusert av rutenettsoppløsningen for de fleste praktiske formål.

Contents

Preface

The authors would like to thank the members of the Underwater Robotics team at The Norwegian Defence Research Establishment (FFI) for their suggestions and comments that has aided the development of the mapping framework. Especially Kenneth Gade for his contributions to the navigation theory presented in this work. Furthermore, the authors are grateful for the discussions regarding computational efficiency and memory layout had with David Kolden, Martin Vonheim Larsen, Ragnar Smestad and Trym Vegard Haavardsholm.

20 June 2022, Håkon Yssen Rørstad and Thomas Røbekk Krogstad

1 Introduction

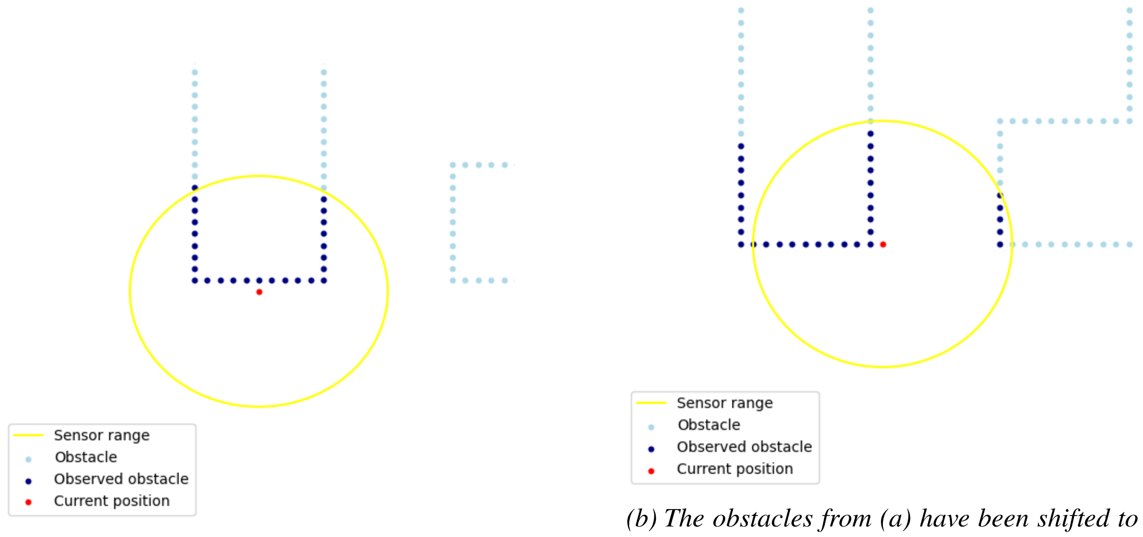
Fixed Area Sliding Template map (**FASTmap**) was first developed to be the backbone of the HUGIN AUVs local world model. This model needed to contain a dense representation of measured and computed values describing the local environment around the AUV for use in autonomous decision making and collision avoidance. As managing dense storage is a computationally complex problem, the need for an efficient and flexible mapping framework quickly became apparent. FASTmap was therefore developed to be easily reconfigurable to adapt to changing requirements while keeping the core functionality efficient to ensure real time operation for maps of realistic sizes.

This report is primarily intended as an introduction and quick reference guide to FASTmap for new users. The report does however also contain descriptions of internal functionality and benchmarking of performance in the later chapters. For application programmers it is recommended to read Chapter 2 and use Chapter 3 as a reference. Chapters 4 and 5 are intended for maintainers and other similarly interested. FASTmap was, as mentioned, originally designed to map the surroundings of moving autonomous vehicles, but the map itself is general enough to be used for most mapping applications. The main features of FASTmap are:

- **Fast** : FASTmap has similar runtime on commonly used functions compared to Gridmap [4].
- **Lightweight** : FASTmap is a header only library making integration into existing projects simple.
- **Versatile** : FASTmap can hold arbitrary C++ types and also heterogeneous types in different layers, becoming an Array of Structures or Structure of Arrays data container at the users discretion.

2 Functional Description

FASTmap is a sliding grid map for holding arbitrary C++ types in its cells. The main use case for the map is ensuring correct positioning of user data while the map moves. In other words, shifting the stored data to be correctly positioned relative to the map origin at all times. This can for example be used for holding sensor data from sensors mounted on a moving platform. Old measurements will then be shifted to reflect the platform motion while the platform moves around gathering new samples. This scenario is illustrated in Figure 2.1.



(a) All observed obstacles are within the sensor range.

(b) The obstacles from (a) have been shifted to reflect the movement of the platform and new obstacles have been added.

Figure 2.1 Plots of a FASTmap mapping obstacles for a moving platform.

2.1 Frames and Maps

In order to use FASTmap correctly it is crucial to have a good understanding of all the reference frames utilized when working with the map. The Earth Centered Earth Fixed (**ECEF**) frame is therefore introduced to establish the global position of the map. ECEF, defined in eg. [3], is a frame with its origin in the center of the earth, its z-axis pointing upwards towards the North pole and its x-axis pointing towards the prime meridian. The y-axis completes the right hand frame and ECEF is illustrated in relation to the Earth in figure 2.2. The ECEF frame is used in this context both as a global reference and to introduce the tangent frames.

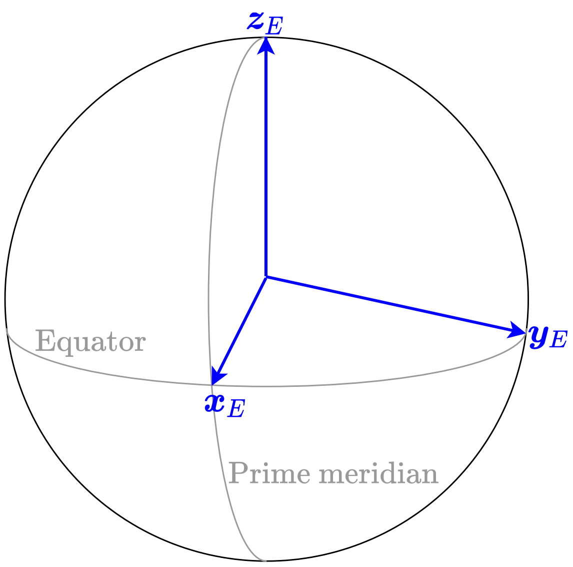
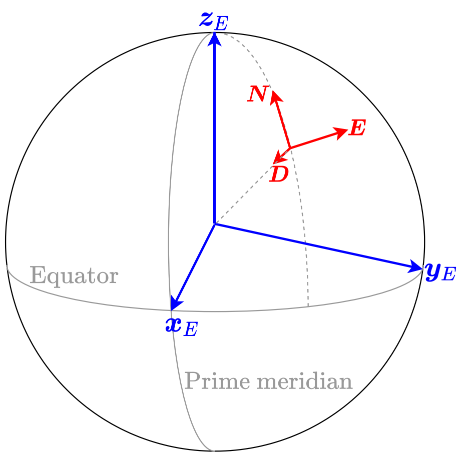
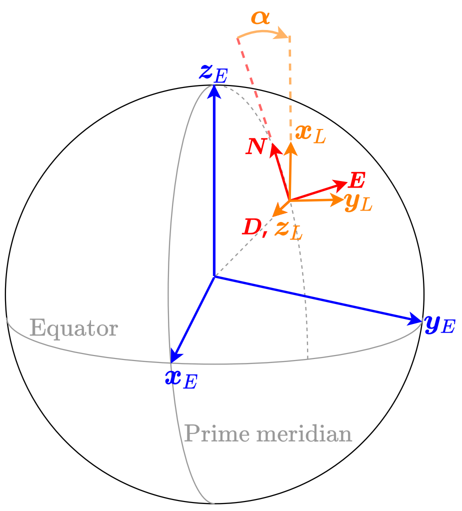


Figure 2.2 The ECEF frame in relation to the earth with the equator and prime meridian.

A tangent frame has its origin in a arbitrary point on the Earths surface and two axis pointing out tangential to the surface at the origin. The last axis completes the right hand frame and therefore points normal to the surface either up or down into the center of the earth. The North East Down(**NED**) frame is a tangent frame with its x-axis always pointing true north, the y-axis points East and the z-axis points down [3]. NED is shown in relation to ECEF and the Earth in Figure 2.3a. If the origin of the NED frame is moved, the axes of the frame are adjusted accordingly such that they always point in the correct directions. This does however result in the NED frame being singular at the poles where the North and East directions are undefined [6].



(a) The NED frame in relation to ECEF frame.



(b) The L frame in relation to the NED frame, away from the initial origin.

Figure 2.3 The tangent frames in relation to the Earth.

To avoid the singularity introduced by the NED frame the Wander azimuth or **L**ocal level frame can be used instead [6]. The L frame is a tangent frame like NED, but has the additional property that its axes do not realign when the origin is moved. In other words, the L frame does not rotate about the z-axis, keeping its original directions for the x and y-axis which are often initialized to point North and East. The z-axis of the L frame points down similarly to the NED frame. The L frame is shown in relation to NED in Figure 2.3b.

Another frame of interest is the body fixed frame, denoted BODY and defined in eg. ch. 2 of [5]. This frame is often defined with its origin in the center of gravity of some rigid body. The axis of the BODY frame are therefore often chosen to point along the rotational axes of the body. This frame is useful for representing quantities or positions closely related to the rigid body in question. For a vehicle this could be the geometry of the craft, forces acting upon it and measurements from onboard sensors. The relation between the BODY frame and the NED frame can be found from the difference in position of the two frames and the euler angles roll(ϕ), pitch(θ) and yaw(ψ) [5]. An example BODY frame in relation to a rigid body is shown in Figure 2.4.

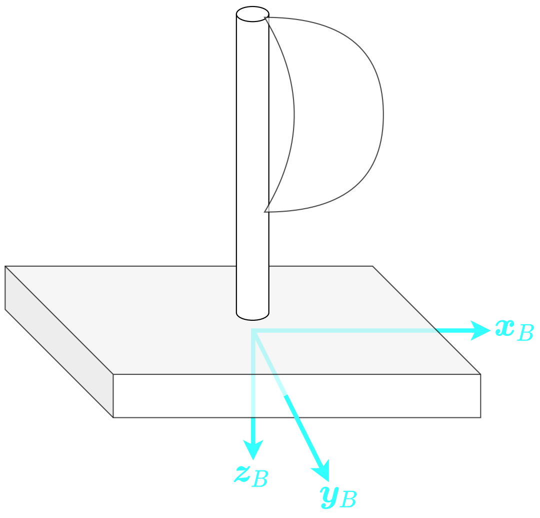
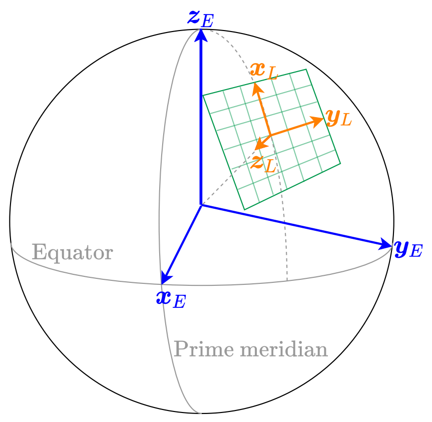
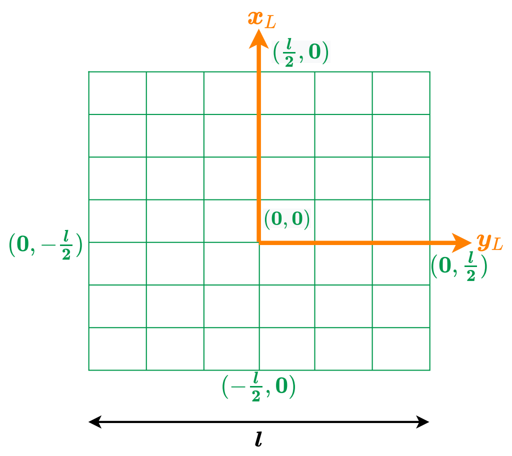


Figure 2.4 The BODY frame in relation to a vehicle-like rigid body.

The reason why the understanding of frames is important for the use of maps is because of the similarities between the two concepts. A flat grid map can be viewed as a tangent reference frame constrained along its axes and with a finite resolution inside the constrained area. In this way FASTmap can be viewed as a discrete and constrained version of the L or NED frame with index (0,0) at the origin. This implies that FASTmap has an x up, y right indexing convention. Figure 2.5a shows an example of a FASTmap in relation to a newly initialized(aligned with North and East) L frame on the spherical Earth and Figure 2.5b shows the map in relation to only the L frame. Notice that negative indices have been included in Figure 2.5b, but the map can also be constrained to the positive quadrant.



(a) *FASTmap(in green) in relation to the L and ECEF frames.*



(b) *FASTmap with indices in relation to the L frame.*

Figure 2.5 FASTmap in relation to the L frame.

The choice of tangent reference frame for FASTmap is left to the user, and will in practice be determined by which frame the user rotates their data positions to before indexing the map. FASTmap does therefore not assume any particular frame in its interface or internals(except for the x up, y right indexing convention), but the L frame is recommended.

2.1.1 Example

Consider a body on the Earths surface with a position defined relative ECEF. Define then a BODY frame with origin in the center of gravity of the body and an L frame with origin coinciding with the BODY frame. These frames are shown in Figure 2.6 with example positions and orientations. In this example FASTmap can be used to store and reposition data measured by sensors on the body while the body moves along the Earth surface. In order for the user to input data from the sensors into the map the position of the data must be rotated from the sensor frame, here assumed to be BODY, to the map tangent frame, assumed to be L. If the data also needs to be interpreted relative to a global reference the rotation from L to ECEF must also be found. For this example the azimuth angle(α) of the L frame is assumed known as this is dependent on the movement of the L frame up to this point in time.

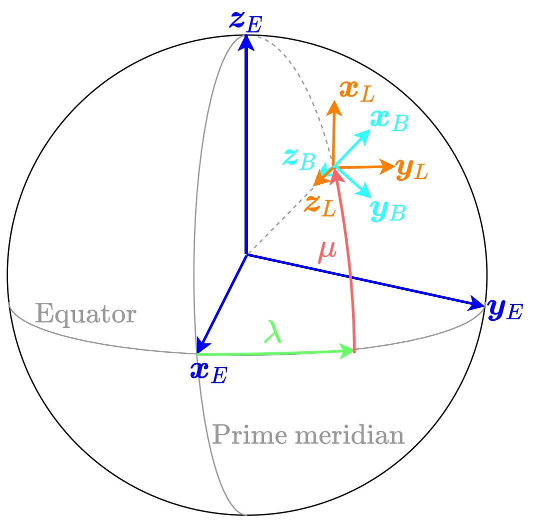


Figure 2.6   Example L and BODY frames relative to ECEF.

Given the orientation of the body represented by the euler angles roll(ϕ), pitch(θ) and yaw(ψ) the rotation from BODY to L can be found from (2.1) [5]. Here $\mathbf{R}_{a,\beta}$ is the rotation matrix representing a rotation of an angle β around an axis a. Notice that the azimuth angle α must be subtracted from the yaw angle as the yaw angle is defined relative North and not the x-axis of the L frame.

$$\mathbf{R}_{LB} = \mathbf{R}_{z,(\psi-\alpha)} \cdot \mathbf{R}_{y,\theta} \cdot \mathbf{R}_{x,\phi}$$

$$\mathbf{R}_{LB} = \begin{bmatrix} cos(\psi-\alpha) & -sin(\psi-\alpha) & 0 \\ sin(\psi-\alpha) & cos(\psi-\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.1}$$

Finding the rotation from L to ECEF requires first the rotation from L to NED and then the position of the NED frame in latitude and longitude. The rotation from L to NED is a simple rotation of α about the z axis of the NED frame as shown in (2.2).

$$\mathbf{R}_{NL} = \mathbf{R}_{z,\alpha} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Given the latitude(λ) and longitude(μ) of the NED frame, NED can be further rotated to the ECEF by (2.3).

$$\mathbf{R}_{EN} = \mathbf{R}_{z,(\lambda)} \cdot \mathbf{R}_{y,(-\mu-\frac{\pi}{2})}$$

$$\mathbf{R}_{EN} = \begin{bmatrix} cos(\lambda) & -sin(\lambda) & 0 \\ sin(\lambda) & cos(\lambda) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos(-\mu-\frac{\pi}{2}) & 0 & sin(-\mu-\frac{\pi}{2}) \\ 0 & 1 & 0 \\ -sin(-\mu-\frac{\pi}{2}) & 0 & cos(-\mu-\frac{\pi}{2}) \end{bmatrix} \tag{2.3}$$

The full rotation from L to ECEF is then given by (2.4), but since the origins of ECEF and L does not align the position of L relative ECEF must be added after rotating if the position is to be interpreted relative to and decomposed in ECEF. This will not be shown here, but the position of L relative ECEF can be calculated from the Latitude, Longitude and a reference ellipsoid [5].

$$\mathbf{R}_{EL} = \mathbf{R}_{EN} \cdot \mathbf{R}_{NL}$$

$$\mathbf{R}_{EL} = \begin{bmatrix} cos(\lambda) & -sin(\lambda) & 0 \\ sin(\lambda) & cos(\lambda) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos(-\mu-\frac{\pi}{2}) & 0 & sin(-\mu-\frac{\pi}{2}) \\ 0 & 1 & 0 \\ -sin(-\mu-\frac{\pi}{2}) & 0 & cos(-\mu-\frac{\pi}{2}) \end{bmatrix} \cdot \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Using Figure 2.6 the example values $\lambda = \frac{\pi}{4}$, $\mu = \frac{\pi}{4}$, $\begin{bmatrix}\phi & \theta & \psi\end{bmatrix} = \begin{bmatrix}0 & \frac{\pi}{8} & \frac{5\pi}{16}\end{bmatrix}$ and $\alpha = \frac{\pi}{16}$ can be chosen to match roughly with the illustration. Given a point a and its position $\mathbf{p}^B_{Ba} = \begin{bmatrix}1 & 10 & 0\end{bmatrix}^\mathsf{T}$ relative to and decomposed in the origin of the BODY frame, this position can be rotated from BODY to L by (2.5) and from L to ECEF by (2.6). The point a is shown relative to the BODY, L frame and FASTmap in Figure 2.7,

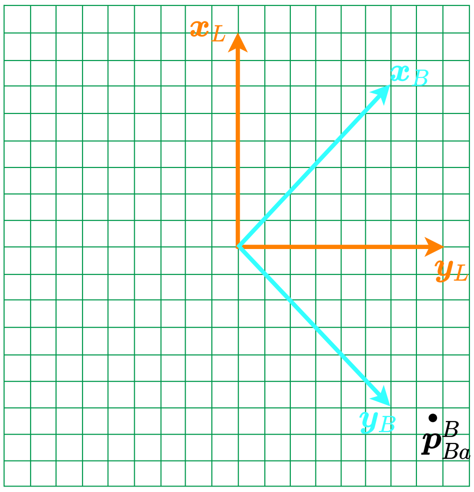


Figure 2.7 Position of example point $\boldsymbol{p}^B_{Ba}$ relative BODY and L.

$$\mathbf{p}_{Ba}^{L} = \mathbf{R}_{LB} \cdot \mathbf{p}_{Ba}^{B} \tag{2.5}$$

$$\mathbf{p}_{Ba}^{L} = \begin{bmatrix} -6.4178 \\ 7.7243 \\ -0.3827 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.9239 & 0 & 0.3827 \\ 0 & 1 & 0 \\ -0.3827 & 0 & 0.9239 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 10 \\ 0 \end{bmatrix}$$

$$\mathbf{p}_{Ba}^{E} = \mathbf{R}_{EL} \cdot \mathbf{p}_{Ba}^{L} \tag{2.6}$$

$$\mathbf{p}_{Ba}^{E} = \begin{bmatrix} -0.3796 \\ 8.5637 \\ -5.2458 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -\frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \\ 0 & 1 & 0 \\ \frac{\sqrt{2}}{2} & 0 & -\frac{\sqrt{2}}{2} \end{bmatrix} \cdot \begin{bmatrix} 0.9808 & -0.1951 & 0 \\ 0.1951 & 0.9808 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -6.4178 \\ 7.7243 \\ -0.3827 \end{bmatrix}$$

2.2 Basic use

Having established the relation between FASTmap and the tangent frames, this section will show the practical connection by introducing the primary Application Programming Interface(**API**) functions of the map. As the L frame is the recommended tangent frame all position or index variables will be denoted as decomposed in the L frame. This is also to clearly differentiate variables representing an index or position from variables that just counts cells.

2.2.1 The at() function

The main use case of FASTmap is to store values in a tangent frame partitioned into cells and automate the reallocation of these values when the origin of the frame changes. A natural point to start is therefore the **at**() function. This function takes an index(x,y), relative to the current origin of the map(tangent frame), as input and returns the address of the value or object at the specified position. This allows for both reading and writing to the specified location at the users discretion. The signatures of the (overloaded) function is shown in Listing 2.1.

*Listing 2.1 The signatures of the **at**() function*

```
auto& at(const Eigen::Vector2i& index_L)

auto& at(int x, int y)
```

2.2.2 The move() function

The origin of the map can at any time be changed by a call to the **move**() function which takes a new map origin, expressed as the number of cells relative to the current, as input. This can also be interpreted as passing the movement of the map as a discretized vector, decomposed in the current frame, to the function. Any values set in the map by previous **at**() calls will after a move be shifted to the correct positions relative to the new origin. This is achieved by shifting all cells in the map a number of times in the up, down, left or right directions until the map origin aligns with the input position. This change of origin is analogous to moving the map to a new tangent frame on the Earths surface. As such all **at**() calls should after the change be interpreted as relative to the new frame. This introduces a position error for data set in the map before the **move**() call, but this error is negligible in most use cases as will be shown in Chapter 4. The signatures of the **move**() function is shown in Listing 2.2.

*Listing 2.2 The signatures of the **move()** function*

```
void move(const Eigen::Vector2i& newOrigin_L)

void move(const int newOriginX_L, const int newOriginY_L)
```

2.2.3 Layers

In order to store not only arbitrary types, but also heterogeneous types in FASTmap the concept of layers must be introduced. A layer is a matrix with homogeneous elements, each corresponding to an index in the map, and heterogeneous storage can be achieved by the use of multiple layers. FASTmap can have an arbitrary number of layers, but the layer types must be specified at compile time by providing the types to the template parameter list of FASTmap as exemplified in Listing 2.3. This functionality is inspired by the layers of Gridmap [4], but is extended to support arbitrary types.

Listing 2.3 Declaration of a FASTmap with template paramters

```
FASTmap<rows, columns, LayerType1, LayerType2> map;
```

The different layers of a FASTmap are accessible by utilizing that the **at**() function is a templated function on layer index. The signatures of the **at**() function is then extended as shown in Listing 2.4, where the layerIdx is the index of its respective type in the class template parameter list, starting at 0. A FASTmap with layers and their respective indices is shown in Figure 2.8.

Listing 2.4 The full signatures of the at function

```
template<int layerIdx = 0>
auto& at(const Eigen::Vector2i& index_L)

template<int layerIdx = 0>
auto& at(int x, int y)
```

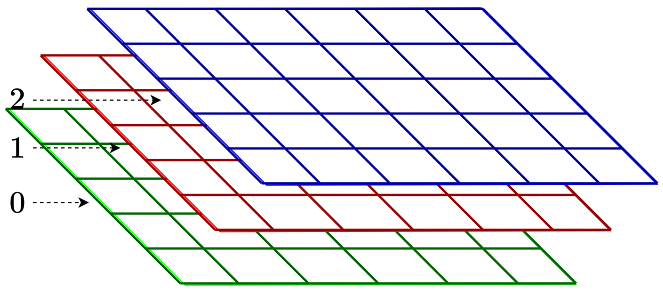


Figure 2.8 A FASTmap with multiple layers and corresponding indices.

The introduction of additional layers naturally impacts the performance and memory use of FASTmap. For a discussion on layers, types, Structures of Arrays and Array of Structures the reader is referred to Chapters 4 and 5.

2.2.4 Example

An example showing the basic functionality of FASTmap is given in Listing 2.5. Here a 5x6 map is created and 3 values are inserted before the map origin is moved to position $(0,3)$. The movement of the map in relation to the L frame and the data points is illustrated in Figure 2.9.

Listing 2.5 Example: Use of basic FASTmap functionality

```
const int rows = 5;
const int columns = 6;
FASTmap<rows, columns, double, double, int> map;

Eigen::Vector2i dataIdx_L{0,5};
map.at<0>(dataIdx_L) = 1.1;
map.at<1>(dataIdx_L) = 1.2;
map.at<2>(dataIdx_L) = 2;

Eigen::Vector2i newOrigin_L{0,3};
map.move(newOrigin_L);

Eigen::Vector2i dataIdxNew_L = dataIdx_L − newOrigin_L;

double shouldBe1point1 = map.at<0>(dataIdxNew_L);
double shouldBe1point2 = map.at<1>(dataIdxNew_L);
int shouldBe2 = map.at<2>(dataIdxNew_L);
```

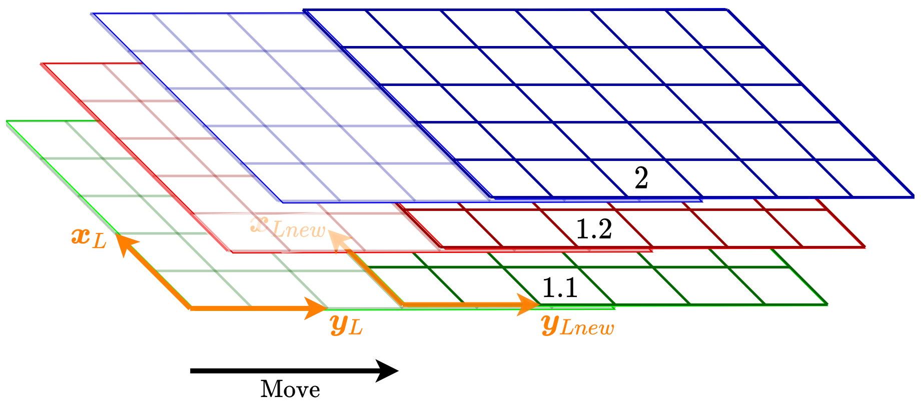
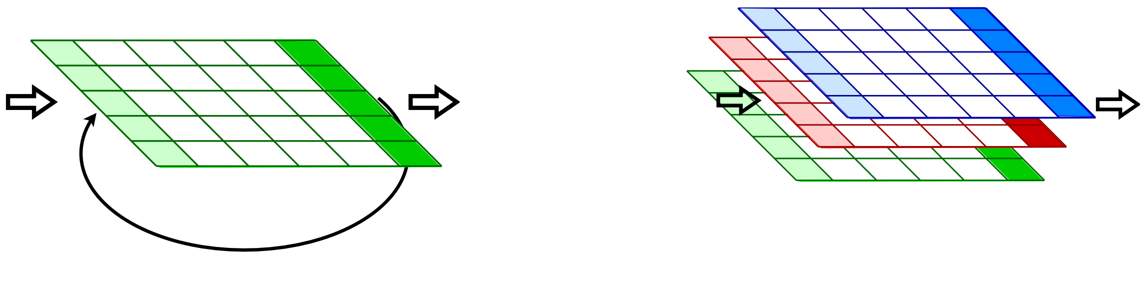


Figure 2.9 Moving a FASTmap with multiple layers and data.

2.3 Advanced use

When cell objects are shifted out of the map they need to be reset in some way in order to be reused as the new objects just shifted in (illustrated in Figure 2.10). If this was not the case the map would grow unbounded with each shift operation and eventually use up all available memory. The reset of objects is performed through the use of the **resetCellObject**() function, which takes the address of a layerType object as input. This function does however pose a design problem. This function needs to modify an object of a user defined type. If the map is to support arbitrary types the type of the specific object can not be assumed known in advance. As such a general, but inefficient reset must be performed. This general reset function is shown in Listing 2.6.



(a) Single layer *(b) Multiple layers*

Figure 2.10 The effect of a shift operation on FASTmap.

Listing 2.6 General reset function

```
template <class layerType>
void resetCellObject(layerType& cellObj)
{
    layerType newObj; // construct
    cellObj = newObj; // copy
} // destruct
```

As seen from Listing 2.6 the general reset function introduces three new assumtions on the layerType. The type must now be fully constructed from its default constructor, have a working destructor and a copy constructor. If the types used satisfies these assumptions and the target application is not notably slowed down by the additional construction and destruction, FASTmap can be used without interacting with the reset function. However, for performance critical applications or for types where the above assumptions are impractical, a second class is provided in addition to FASTmap namely FASTmapBase.

FASTmapBase is a base class with a **resetCellObject**() function that needs to be defined by the user. The default FASTmap class is also derived from this base class and defines the function in Listing 2.6 as its only reset function. For a custom derived class the user needs to override the reset function once for each different layer type utilized. This allows specifying how an object should be reset when it goes out of map range and can often be achieved without the need for construction of a new object or destruction of the old.

A final note about the inheritance of FASTmapBase is that this is done through the use of the Curiously Recurring Template Pattern(**CRTP**) [10]. This can be seen from the presence of the derived class in the template parameter list of FASTmapBase shown in Listing 2.7. The use of CRTP in this context is to avoid the performance hit that virtual functions can introduce in regular C++ inheritance [11] and because runtime polymorphism is not required in this case.

Listing 2.7 The template parameter lists of FASTmapBase

```
template<class DerivedMap, int rows, int columns, class... layerTypes>
class FASTmapBase
```

2.3.1 Example

A typical derived class from FASTmapBase is exemplified in Listing 2.8.

Listing 2.8 CRTP inheritance of FASTmapBase

```
const int rows = 10;
const int columns = 10;

class DerivedMap : public FASTmapBase<DerivedMap, rows, columns, layerType1, layerType2>
{
public:
    // The reset functions needs to be public such that FASTmapBase
    // can call them using CRTP.

    void resetCellObject(layerType1& cellObj)
    {
        // layerType1 specific reset
    }

    void resetCellObject(layerType2& cellObj)
    {
        // layerType2 specific reset
    }
};
```

2.4 Known Limitations

2.4.1 The at() function, dependent names and inheriting FASTmapBase

When inheriting FASTmapBase the reader might encounter strange compilation errors when trying to use the **at**<layerIdx>() function in the derived class internals. This problem arises when the derived class is also a class template and some of its template parameters are also used in the template parameter list of FASTmapBase. This makes FASTmapBase a dependent name which is not looked up before the derived class is instantiated [7]. As such during declaration of the derived class the compiler cannot known if a call to **at**<layerIdx>() is a call to a templated function or a comparison of an object named at and the layerIdx, it assumes the latter. The way around this is the use of the template keyword to tell the compiler that the angle brackets is passing a template parameter. A toy example showing the problem and the solution is shown in Listing 2.9. Here CRTP has been omitted for simplicity and a dummy class FASTmap is used instead of FASTmapBase.

*Listing 2.9 Problem with dependent names and the **at**<layerIdx>() function*

```
template<int size, class layerType>
class FASTmap
{
public:
    template<int layerIdx>
    void at() { }
};

template<int size>
class doubleMap : public FASTmap<size, double> // FASTmap depends on size
{
public:

    void useAt()
    {
        // This results in a compilation error as < is interpreted as the less than operator.
        // From g++-7: "error: invalid operands of types '<unresolved overloaded function type>'
        // and 'int' to binary 'operator<'"
        this->at<0>();

        // This compiles. The template syntax tells the compiler that at is a function that takes a
        // template parameter.
        this->template at<0>();
    }
};

int main()
{
    doubleMap<10> dMap;
    dMap.useAt();

    FASTmap<10, int> iMap;

    // Fine because this instance of FASTmap does not depend on any non-specified template
```

```
        // parameters.
        iMap.at<0>();

        iMap.template at<0>(); // This always works, but is redundant here.

    }
```

2.4.2 Bool as a layer type

Since each layer in FASTmap i stored as a std::vector using bool as a layer type imposes the same problems as creating an vector of bools. std::vector<bool> does not give a vector of bools, which is represented using a byte, but a vector of bits where the bit value represents wether the bool is true or false. This does indeed save memory, but this implicit conversion has the unfortunate side effect of breaking the assumption that the layerTypes passed to FASTmap is what is actually stored in the layers. This results in multiple errors of the type "error: cannot bind non-const lvalue reference of type 'std::_Bit_reference&' to an rvalue of type 'std::vector<bool, std::allocator<bool> >::reference {aka std::_Bit_reference}'" from g++-7 when trying to compile FASTmap with such a layer. It is therefore recommended to use an unsigned char as a layer type instead and treat a value of 0 as false and all other values as true.

3 Interface

This chapter will present all the API functions of FASTmap with some comments about usecases, parameters and return types. The chapter is provided as a reference for application programers. The **move**() and **at**() functions are thoroughly described in Chapter 2 and will therefore not be restated here, but it will be mentioned that FASTmap also has overloads of the **at**() functions returning const references.

3.1 Constructors

Both the constructor of FASTmap and FASTmapBase takes an offset as their only non-template parameter. This offset specifies the number of cells, in the x and y direction, between the cell in the lower left corner of the map and the cell aligned with the tangent frame origin. In other words this offset specifies the cell with index (0,0). An example is the map in figure 2.5b where the offset is set to (3,3). The offset defaults to the the cell in the lower left corner if no value is passed to the constructors. The signatures of the constructors are given in 3.1.

Listing 3.1 The signatures of the FASTmap and FASTmapBase construtors

```
FASTmap(const Eigen::Vector2i& offset = Eigen::Vector2i(0, 0))

FASTmapBase(const Eigen::Vector2i& offset = Eigen::Vector2i(0, 0))
```

3.2 shift()

The **shift**() function is the function called by **move**() internally to do the actual shifting of the map. **shift**() takes a direction, defined by the enum **shiftDir**, and shifts the map in the that direction. This results in all cells being moved in the the specified direction and the column or row that goes out of map length is reset and wraps around to the other side. Note here that moving the map origin a single cell in one direction corresponds to shifting the map in the opposite direction. The reader is advised to consider this before using **move**() and **shift**() interchangeably. The **shiftDir** enum and the signature of the **shift**() function is given in 3.2.

*Listing 3.2 The signatures of the **shift()** function and the **shiftDir** enum*

```
enum shiftDir
{
    up,
    down,
    left,
    right,
    none
};

void shift(shiftDir direction)
```

3.3 reset()

The **reset**() function simply resets the map by resetting all cells in all layers. This is in practice done by calling the **resetCellObject**() function on each cell.

*Listing 3.3 The signature of the **reset()** function*

```
void reset()
```

3.4 resetBlock()

The **resetBlock**() function resets a block of the map defined as the square with the index **fromIdx_L** as the upper left corner and the index **toIdx_L** as the lower right corner. The cells in the square is reset for all layers.

*Listing 3.4 The signatures of the **resetBlock()** function*

```
void resetBlock(const Eigen::Vector2i& fromIdx_L, const Eigen::Vector2i& toIdx_L)

void resetBlock(int xFrom, int xTo, int yFrom, int yTo)
```

3.5 resize()

The **resize**() function changes the size of the map to the specified number of rows and columns. The **reset**() function is also called in the internals of **resize**() meaning all values in the map is reset when the map changes size. The reason for this and the fact that no conservative resize exists is because FASTmapBase does not assume that the layerTypes have valid copy constructors. Without this assumption it is difficult to create a general and computational efficient resize that retains map data.

*Listing 3.5 The signature of the **resize()** function*

```
void resize(int nRows, int nColumns)
```

3.6 Bresenham()

The **Bresenham**() function is an implementation of Bresenhams line algorithm[1] returning indices forming a line on the grid from **fromIdx_L** to **toIdx_L**. If **fromIdx_L** or **toIdx_L** is outside the map the valid part of the line is returned. This function is useful for a number of applications and some examples are: iterating over a line, searching along a line, edge detection and projecting positions outside the map to the closest index in the map.

*Listing 3.6 The signatures of the **Bresenham()** function*

```
const std::vector<Eigen::Vector2i> Bresenham(const Eigen::Vector2i& fromIdx_L,
                                             const Eigen::Vector2i& toIdx_L) const

const std::vector<Eigen::Vector2i> Bresenham(int xFrom, int xTo, int yFrom, int yTo) const
```

3.7 getRows()

Returns the current number of map rows.

*Listing 3.7 The signature of the **getRows()** function*

```
int getRows() const
```

3.8 getColumns()

Returns the current number of map columns.

*Listing 3.8 The signature of the **getColumns()** function*

```
int getColumns() const
```

3.9 getNlayers()

Returns the number of layers as a constant expression.

*Listing 3.9 The signature of the **getNlayers()** function*

```
constexpr int getNlayers() const
```

3.10 setRows()

Calls resize with the number of rows passed to the function and the current number of columns.

*Listing 3.10 The signature of the **setRows()** function*

```
void setRows(int nRows)
```

3.11 setColumns()

Calls resize with the number of columns passed to the function and the current number of rows.

*Listing 3.11 The signature of the **setColumns()** function*

```
void setColumns(int mColumns)
```

3.12 getOffset()

Returns the current offset of the map. The map offset is defined in Section 3.1.

*Listing 3.12 The signature of the **getOffset()** function*

```
Eigen::Vector2i getOffset() const
```

3.13 setOffset()

Set the map offset. The map offset is defined in Section 3.1.

Listing 3.13 The signature of the ***setOffset()*** *function*

```
void setOffset(const Eigen::Vector2i& newOffset)
```

3.14 cellIdx2idx()

Converts the number of map cells counted from the lower left corner, in the x and y direction, to a map index relative to the origin. This is equivalent to subtracting the map offset from the number of cells(**cellIdx**). The map offset is defined in Section 3.1.

Listing 3.14 The signature of the ***cellIdx2idx()*** *function*

```
Eigen::Vector2i cellIdx2idx(const Eigen::Vector2i& cellIdx) const
```

3.15 idx2cellIdx()

Converts a map index to the number of map cells counted from the lower left corner in the x and y direction. This is equivalent to adding the map offset to the index(**index_L**). The map offset is defined in Section 3.1.

Listing 3.15 The signature of the ***idx2cellIdx()*** *function*

```
Eigen::Vector2i idx2cellIdx(const Eigen::Vector2i& index_L) const
```

3.16 isIdxInMap()

Returns true if the index passed to the function is within the map, false otherwise.

Listing 3.16 The signatures of the ***isIdxInMap()*** *function*

```
bool isIdxInMap(const Eigen::Vector2i& index_L) const

bool isIdxInMap(int x, int y) const
```

3.17 getLayer()

The **getLayer**() function returns a reference to the underlying std vector holding the objects in the layer with index **layerIdx**. This function should be used with **CAUTION** as access to modify the vector is given. If the vector is modified the internal functionality of FASTmap might break. This function exists for users with the need to access the underlying vector or raw data pointer in order to employ operations on the data which is not supported by the FASTmap API. The return type of this function will be a row times columns long vector of the specific layer type.

*Listing 3.17   The signature of the **getLayer()** function*

```
template<int layerIdx = 0>
auto& getLayer()
```

4 Efficiency and Accuracy

4.1 Map shifting and time complexity

For any real-time mapping application the computational efficiency of the map functions is paramount. First of all the map must be able to store away and shift data at a rate faster than new data is received to avoid build up of a data queue. Secondly as the speed of shift operations must in some way be dependent on the number of cells a faster map can be larger or have higher resolution during execution. This directly increases the effectiveness and usefulness of the map for the target application. The function internals of FASTmap is therefore optimized for speed wherever possible without compromising the necessary versatility of a general map.

The function internals of FASTmap is inspired by the similar map Gridmap[4], but with some key differences. Both maps utilize circular buffers to implement shift functionality, but Gridmap utilizes an Eigen matrix as its internal datastructure while FASTmap uses a std::tuple of std::vectors. This is where the important distinction between the maps originate, Gridmap is locked to storing only floats while, as previously stated, FASTmap can stor arbitrary heterogeneous types.

Returning to the circular buffers and its implementation in FASTmap. The buffer consists of an indexing and resetting scheme hidden from the user. An additional origin, named internal origin, is defined and used to access objects in the layers. This internal origin offsets the indexing from the **at**() function to ensure that the correct object, relative to the map movement is returned to the user. To shift FASTmap in a direction the internal origin needs only to be incremented or decremented in the appropriate direction and the row or column that is "shifted out" needs to be reset. This is a highly computationally efficient shift operation compared to shifting the position of each element for each layer. The operation is $O(N \cdot L)$ where N is the number of rows or columns, depending on the direction of the shift, and L is the number of layers.

The shift example from Section 2.2.4 is shown together with the corresponding operations on the internal data structure in Figure 4.1 and 4.2. Here 3 shift operations are executed sequentially incrementing the internal origin and resetting the column that will be used as the column just shifted in to the map. As seen from the figure the index needs to wrap around the edges of the structure in order to access the reset columns, but this can be achieved by a simple check on the index and subtraction by the number of rows or columns. The same operation is applied to the internal origin if it is shifted out of the structure.

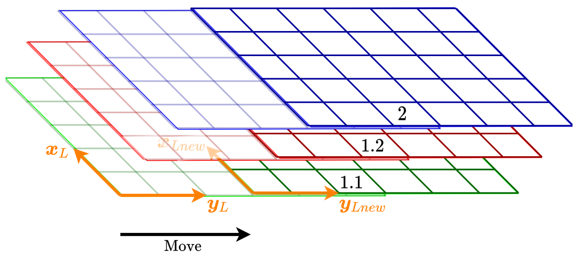


Figure 4.1 Example from 2.2.4 added for convenience.

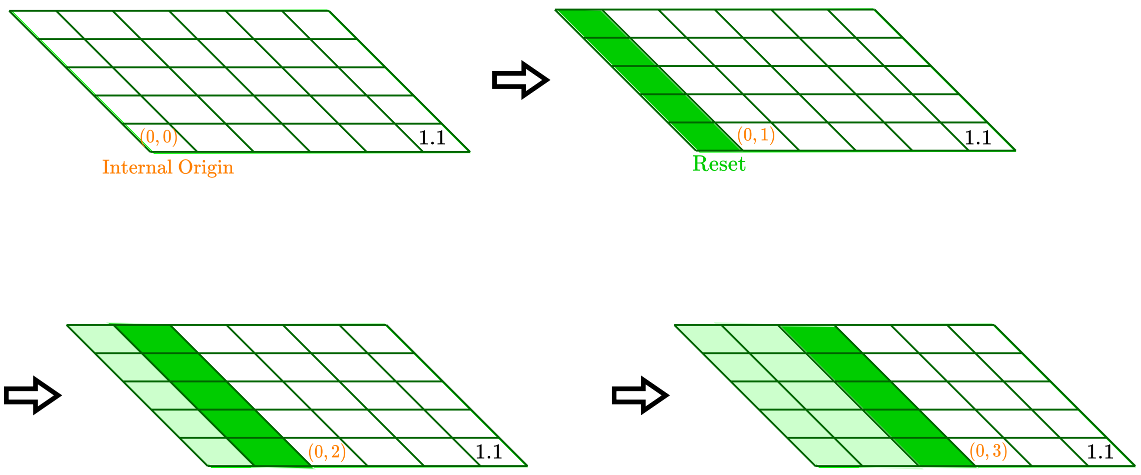


Figure 4.2 Operations on one layer of the internal data structure during execution of the example from 2.2.4.

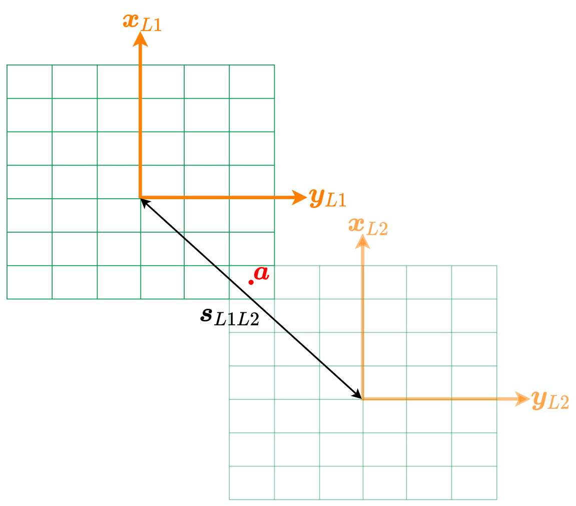
4.2 Position error of map data

Any tangental mapping of values on the Earths surface will introduce a position error because of the curvature of the ellipsoid. FASTmap is no exception and the user is expected to account for this error in the target application. However, since FASTmap is also moving, an additional error is introduced which does not effect stationary tangent frames. This is the error from interpreting data as relative to the current origin while they where inserted into the map relative to a previous origin. This will be the subject for the remainder of this chapter, but first the greatest contributor to position errors in FASTmap must be established.

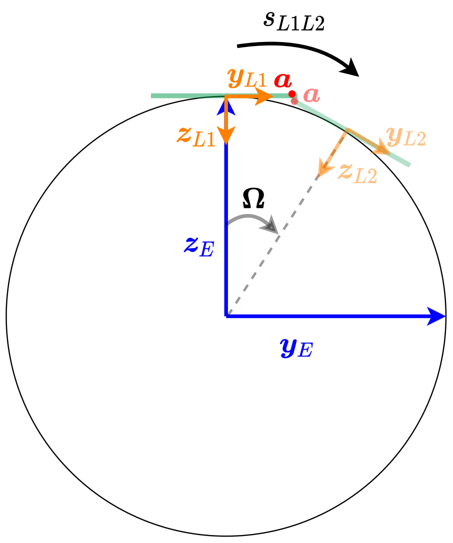
For most practical use cases the grid resolution will be the leading cause of position errors for data in the map. This is because the worst case error introduced for the position of a data point is $\frac{1}{2} \cdot l_c$ where l_c is the map resolution or length of one side of a square map cell. The resolution can be decreased to achieve a lower error, but the map will quickly run in to memory or runtime constraints for maps of realistic sizes. Following the convention from [6] any additional errors at less than 10% of the resolution error will be considered negligible.

As stated in Chapter 2 the move and shift operations are comparable to moving the origin of the tangent frame along the surface of the ellipsoid. However, the shifting of the actual cells internally in FASTmap assume a flat Earth horizontal move for computational efficiency. As a result any old values interpreted relative to the new origin will have a position error introduced by the movement of the frame along the curved surface of the Earth. This position error is dependent on the curvature itself, but also on the distance between the two origins. The values in the map are however deleted when they are shifted out the map range. This ensures that the error introduced by moving is bounded by the map length, which for most practical applications are bounded by computational or memory constraints before the position error becomes significant compared to the error introduced by the grid resolution. This will be shown in the following section.

Consider a FASTmap on the surface of the earth. The surface is assumed to be an ellipsoid and the map origin is assumed to align with an L frame L_1 on the ellipsoid. A sample a is inserted into the map at one of the map corners, in other words the position furthest away from the origin. The map is then moved, assuming flat Earth, the longest distance possible while keeping the inserted sample inside the map range as shown in Figure 4.3a. The new position of the map is a new L frame denoted L_2 and in this frame the map will have the sample positioned in the opposite corner. On the ellipsoid this move is projected onto the surface, illustrated in Figure 4.3b using a sphere, resulting in the sample having the maximum possible error introduced by the movement of the map.



(a) Flat move, neglecting the error.



(b) Spherical move, showing the error.

Figure 4.3 The worst case FASTmap move on the flat and spherical Earth.

Since the Earth is elliptic the error introduced by the movement is dependent on the global position of the map while executing the move operation. To avoid this dependency the maximum curvature of the Earth is assumed. This corresponds to using a spherical Earth model, as in Figure 4.3b, with the maximum curvature radius r_{em} given by (4.1). Here a_e is the semi-major axis of the earth ellipsoid and b_e is the semi-minor axis. A proof for the relation in (4.1) is given in Appendix 6. Using parameters from WGS84 [9] for the semi-major and minor axis gives a maximum curvature radius of $r_{em} \approx 6335439.3\ m$.

$$r_{em} = \frac{b_e^2}{a_e} \tag{4.1}$$

4.2.1 Move on spherical Earth

To find the error introduced by assuming flat earth movement while the actual move is on the sphere the position of the sample relative to L_2, denoted $\boldsymbol{p}_{L_2a}^{L_2}$, will be calculated using first spherical and then flat Earth. For the spherical case the calculations requires a homogeneous transform of the sample position relative L_1, denoted $\boldsymbol{p}_{L_1a}^{L_1}$, to the position relative L_2 as given by (4.2). Here

$\boldsymbol{R}_{L_2L_1}$ is the rotation matrix from L_1 to L_2 and $\boldsymbol{p}^{L_1}_{L_2L_1}$ is the vector from L_2 to L_1 decomposed in L_1. Since L_1 to L_2 is the position of the map at two different points in time the vector $\boldsymbol{p}^{L_1}_{L_2L_1}$ is assumed to be observed from the E frame ensuring that it is independent of the Earths rotation and, as a consequence, time.

$$\boldsymbol{p}^{L_2}_{L_2a} = \boldsymbol{R}_{L_2L_1}(\boldsymbol{p}^{L_1}_{L_1a} + \boldsymbol{p}^{L_1}_{L_2L_1}) \tag{4.2}$$

Focusing first on $\boldsymbol{p}^{L_1}_{L_2L_1}$, this vector can be found by differencing the positions of L_1 and L_2. As these positions are on the surface of a sphere they can be represented by two unit length normal vectors to the sphere, $\boldsymbol{n}^E_{EL_1}$ and $\boldsymbol{n}^E_{EL_2}$, called n-vectors [6]. The angle Ω between the n-vectors can then be found from (4.3), where s_{L1L2} is the length of the move along the sphere shown in Figure 4.3b

$$\Omega = \frac{s_{L1L2}}{r_{em}} \tag{4.3}$$

Assuming that the map movement is only in the y and z directions of the E frame and that L_1 is located at the intersection between the sphere and the z-axis of the E frame, the two general n-vectors can be simplified to (4.4) and (4.5). These simplified versions still represent a move between any two points as the Earth is assumed spherical and the E frame can be re-orientated in order for the n-vectors to have any position on the sphere.

$$\boldsymbol{n}^E_{EL_1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.4}$$

$$\boldsymbol{n}^E_{EL_2} = \begin{bmatrix} 0 \\ sin(\Omega) \\ cos(\Omega) \end{bmatrix} \tag{4.5}$$

From the n-vectors the vector between the frames L_2 and L_1 decomposed in E can be found from (4.6).

$$\boldsymbol{p}^E_{L_2L_1} = (\boldsymbol{n}^E_{EL_1} - \boldsymbol{n}^E_{EL_2}) \cdot r_{em} \tag{4.6}$$

$$\boldsymbol{p}^E_{L_2L_1} = \begin{bmatrix} 0 \\ -r_{em} \cdot sin(\Omega) \\ r_{em} \cdot (1 - cos(\Omega)) \end{bmatrix}$$

And $\boldsymbol{p}^E_{L_2L_1}$ can be rotated to L_1 by the rotation matrix in (4.7) giving (4.8). The matrix $\boldsymbol{R}_{L_1E}$ is found by inspection of Figure 4.3b.

$$\boldsymbol{R}_{L_1E} = \begin{bmatrix} cos(-\pi) & 0 & sin(-\pi) \\ 0 & 1 & 0 \\ -sin(-\pi) & 0 & cos(-\pi) \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.7}$$

$$\boldsymbol{p}^{L_1}_{L_2L_1} = \begin{bmatrix} 0 \\ -r_{em} \cdot sin(\Omega) \\ r_{em} \cdot (cos(\Omega) - 1) \end{bmatrix} \tag{4.8}$$

In order to complete the homogeneous transform in (4.2) the rotation matrix between the L_1 and L_2 must be found. This is achieved by observing that both frames lay on the sphere with x-axes pointing in the same direction. This ensures that the rotation between the frames is a simple rotation about the x-axis of the L_1 frame by an angle $-\Omega$. The resulting rotation matrix is given in (4.9).

$$\boldsymbol{R}_{L_2L_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\Omega) & sin(\Omega) \\ 0 & -sin(\Omega) & cos(\Omega) \end{bmatrix} \tag{4.9}$$

Having all the pieces of the homogeneous transformation the value of $\boldsymbol{p}^{L_1}_{L_1a}$ must be determined to calculate $\boldsymbol{p}^{L_2}_{L_2a}$. Assuming, without loss of generality, that the sample a is inserted into the lower left corner of the map, as shown in Figure 4.4, $\boldsymbol{p}^{L_1}_{L_1a}$ takes the value given by (4.10). Here the grid is zero indexed, r_n is the number of map rows, c_n the number of map columns and l_c is the map resolution or length of one side of a square map cell.

$$\boldsymbol{p}^{L_1}_{L_1a} = \begin{bmatrix} -l_c \cdot \frac{r_n-1}{2} \\ l_c \cdot \frac{c_n-1}{2} \\ 0 \end{bmatrix} \tag{4.10}$$

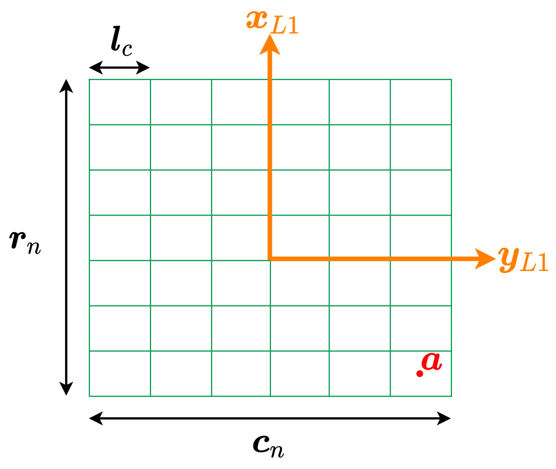


Figure 4.4 Position of sample a in FASTmap

In order to simplify some calculations the L_1 and L_2 frames are rotated $\frac{pi}{8}$ about their z axis ensuring that the movement of the map is only along the y and z axis of the frames. This results in (4.10) becoming (4.11).

$$\boldsymbol{p}_{L_1 a}^{L_1} = \begin{bmatrix} 0 \\ l_c \cdot \sqrt{\frac{r_n - 1}{2}^2 + \frac{c_n - 1}{2}^2} \\ 0 \end{bmatrix} \tag{4.11}$$

Now the value of $\boldsymbol{p}_{L_1 a}^{L_1}$ can be inserted into (4.2) giving (4.12) after some simplifications and canceling of terms.

$$\boldsymbol{p}_{L_2 a}^{L_2} = \begin{bmatrix} 0 \\ l_c \cdot \sqrt{\frac{r_n - 1}{2}^2 + \frac{c_n - 1}{2}^2} \cdot cos(\Omega) - r_{em} \cdot sin(\Omega) \\ -l_c \cdot \sqrt{\frac{r_n - 1}{2}^2 + \frac{c_n - 1}{2}^2} \cdot sin(\Omega) + r_{em} \cdot (1 - cos(\Omega)) \end{bmatrix} \tag{4.12}$$

4.2.2 Move on flat Earth

For the flat case the calculations are trivial and the position can be seen directly from figure 4.3a and is given in (4.13). Here the^denotes an approximated value found by assuming flat Earth.

$$\hat{\boldsymbol{p}}_{L_2 a}^{L_2} = \begin{bmatrix} l_c \cdot \frac{r_n}{2} \\ -l_c \cdot \frac{c_n}{2} \\ 0 \end{bmatrix} \tag{4.13}$$

Again, rotating the L_1 and L_2 frames $\frac{pi}{8}$ about their z axis simplifies (4.13) to (4.14).

$$\hat{\boldsymbol{p}}_{L_2 a}^{L_2} = \begin{bmatrix} 0 \\ -l_c \cdot \sqrt{\frac{r_n}{2}^2 + \frac{c_n}{2}^2} \\ 0 \end{bmatrix} \tag{4.14}$$

4.2.3 Maximum position error from move

To obtain the maximum position error from the movement of the tangent frame (4.12) can be subtracted from (4.14) giving $\boldsymbol{e} = \hat{\boldsymbol{p}}_{L_2 a}^{L_2} - \boldsymbol{p}_{L_2 a}^{L_2}$. Taking the norm of the error gives as scalar value (4.15) suited for comparing with the resolution error.

$$e = \left\| \begin{bmatrix} 0 \\ -l_c \cdot (\sqrt{\frac{r_n}{2}^2 + \frac{c_n}{2}^2} + \sqrt{\frac{r_n - 1}{2}^2 + \frac{c_n - 1}{2}^2} \cdot cos(\Omega)) + r_{em} \cdot sin(\Omega) \\ l_c \cdot \sqrt{\frac{r_n - 1}{2}^2 + \frac{c_n - 1}{2}^2} \cdot sin(\Omega) + r_{em} \cdot (1 - cos(\Omega)) \end{bmatrix} \right\| \tag{4.15}$$

As (4.15) depends on the number of map columns, rows and the resolution, a surface plot of the error is given in Figure 4.5. The number of rows have been set equal to the number of columns and is denoted Map length in order to have only two variables. A plane showing 10% of the resolution error for each map size have also been added. In the figure all parts of the surface beneath the red plane indicates a map size where the error from assuming a flat earth while moving is negligible compared to the resolution error. As seen this area contains most maps of practical sizes including a map with 10000 rows/columns and a resolution of 3 m giving a map length of 30 km.

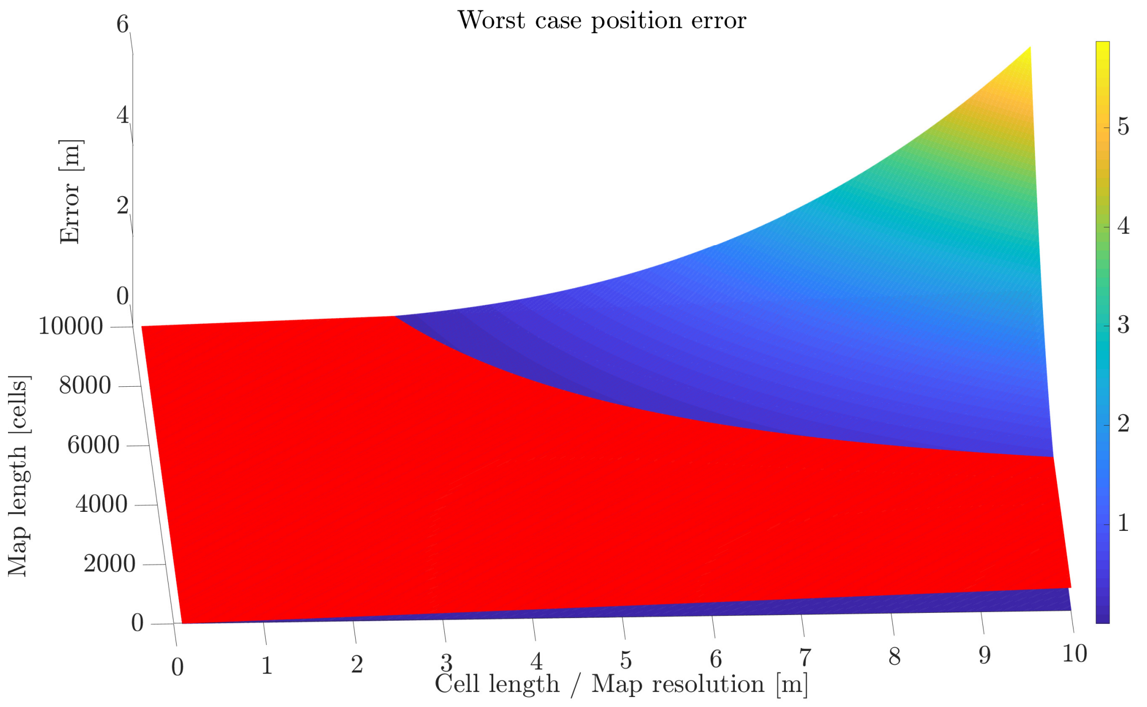


Figure 4.5 *The maximum position error introduced by moving the map as a function of map size and resolution. The red plane indicates 10% of the resolution error for the corresponding map.*

5 Comparison with Gridmap

Comparing FASTmap and Gridmap is made possible by the fact that both maps are designed to solve the same problems, they have similar APIs and they are both written in C++. However comparing these two maps quantitatively is difficult because of the effect of measurement biases in computer systems. This effect is thoroughly explained in [8] and implies that, at least, comparing runtime performance is a complicated task deserving of a dedicated work. Nevertheless some examples of runtime-benchmarking will be given in this chapter to show plausibility of comparable runtimes for FASTmap and Gridmap. The chapter will however start with a qualitative discussion regarding Array of Structures (**AoS**) and Structures of Arrays (**SoA**).

5.1 Arrays of Structures or Structures of Arrays

As stated in Chapter 2 FASTmap can have multiple layers of arbitrary types. FASTmap can as such be turned into an AoS map by having complete structures in one layer. Similarly FASTmap can become a SoA map by splitting the members of a structure and storing each type in a separate layer. Additionally a hybrid map can be constructed by adding the AoS layer to the SoA map. This flexibility is added to ensure that the optimal memory layout can always be chosen as this depends on the target application [12]. Gridmap only supports SoA with floats, meaning the memory layout of Gridmap is efficient for only a subset of applications, but ensuring that the user can not choose a layout that is suboptimal for a situation where SoA would be adequate. In this way the versatility of FASTmap is a design choice based on the C++ mindset: The programmer knows best.

5.2 Runtime Examples

The basis for comparing the time complexity of the two maps will be a set of frequently used operations that both maps provide. The runtimes of these operations will be sampled 30 times and presented as box plots to show some of the statistical properties of the series. As Gridmap and FASTmap are configurable to have different sizes, resolutions and types the map configurations must be chosen such that the resulting data structure is equal in both maps. This is achieved by a FASTmap with one 1000x1000 layer of floats and a Gridmap with a 1000x1000 layer of resolution 1.0. This should result in a more or less equal underlying structure consisting of an array of floats with length 1000000. These configurations will be used to sample the operation runtimes of the two maps in the following sections. However, as stated at the start of this chapter, these samples are not gathered in a sufficiently randomized environment to be considered conclusive evidence as to which map has lowest runtime.

5.2.1 Access map

The time spent to iterate across the entire map setting each cell to 1.0 is shown for the two maps in Figure 5.1 and 5.2. A sample is the runtime recorded for completion of the specified operation, here accessing the whole map. Iterating in the x or y directions of the maps in the outer most loop is shown as two separate cases as, depending on the indexing of the internal array, cache lines might be loaded with the next elements or the ones in the next row or column.

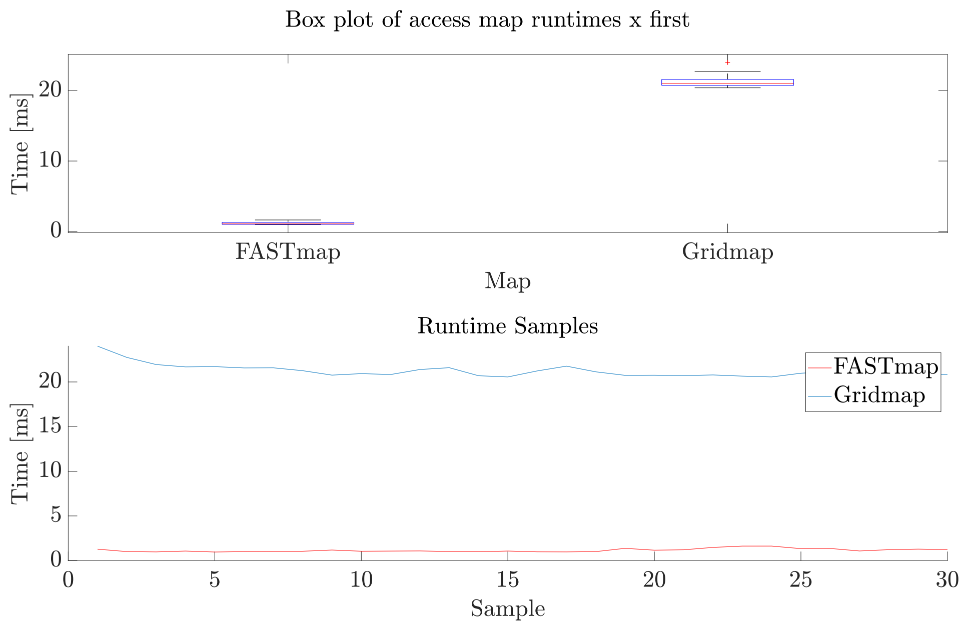


Figure 5.1 Box and sample plots of access whole map runtimes with x in the outer most loop.

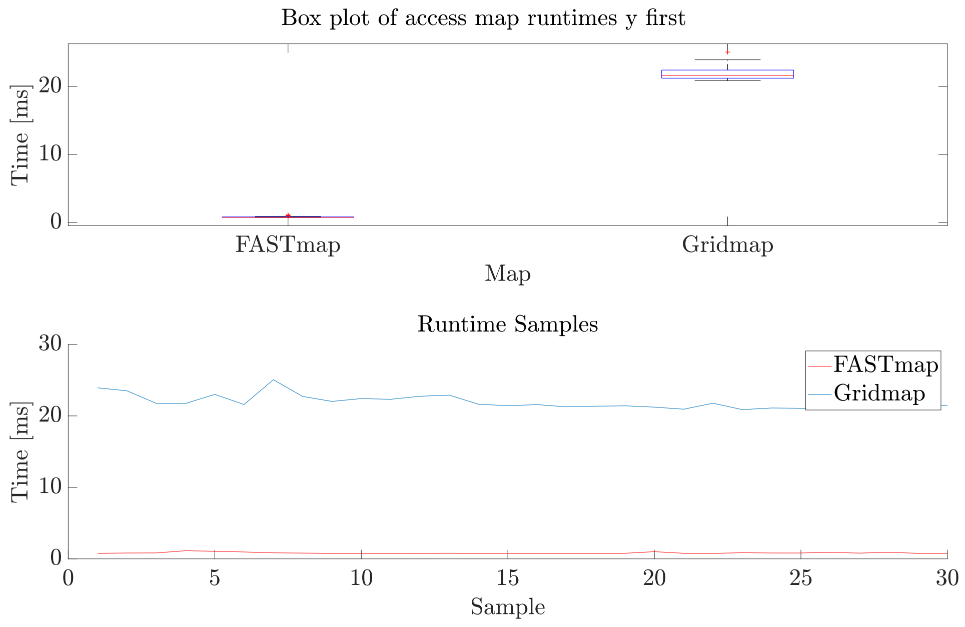


Figure 5.2 Box and sample plots of access whole map runtimes with y in the outer most loop.

5.2.2 Shift map

The time spent shifting the map in the right and up directions for the two maps are shown in Figure 5.3 and 5.4. Again up and right are shown as two separate cases because of potential differences in cache performance. Figure 5.5 and 5.6, which exclude the first sample, are also provided as both maps have outlier values for the first shift operation in the sample series. This is probably because the cache at this point does not contain any values from the maps resulting in extra cache misses.

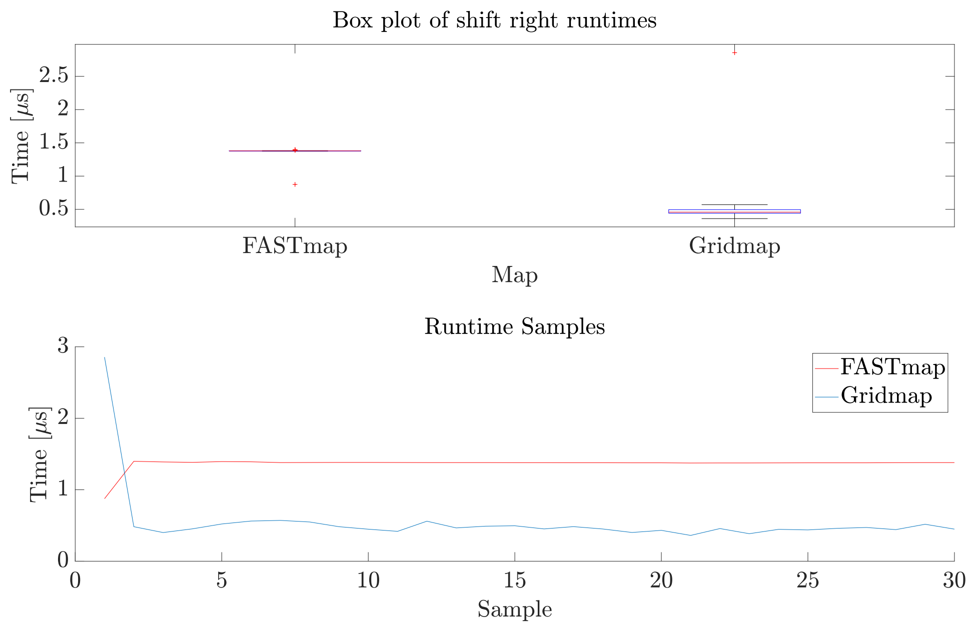


Figure 5.3 Box and sample plots of shift right runtimes.

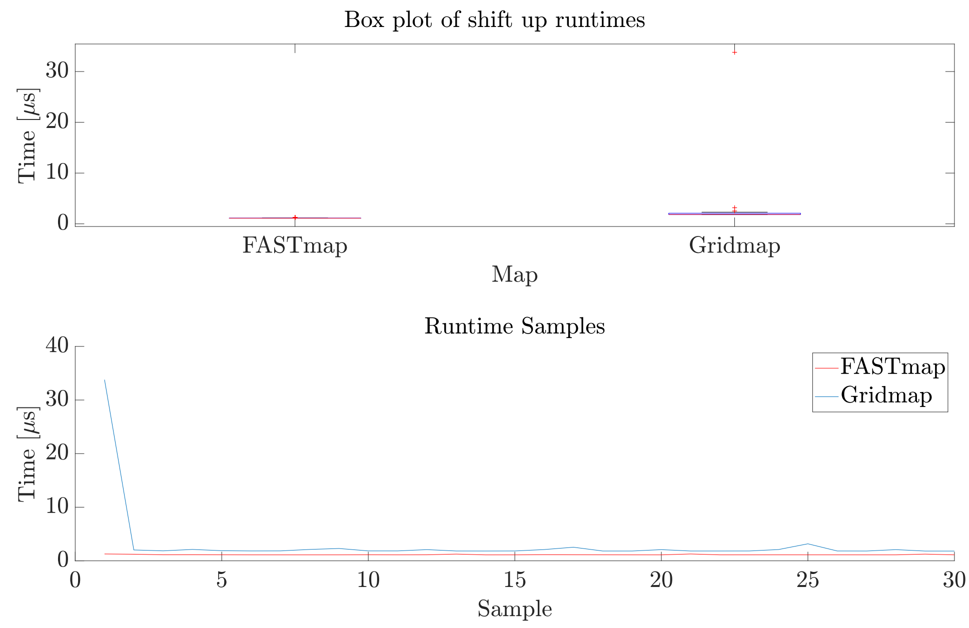


Figure 5.4 Box and sample plots of shift up runtimes.

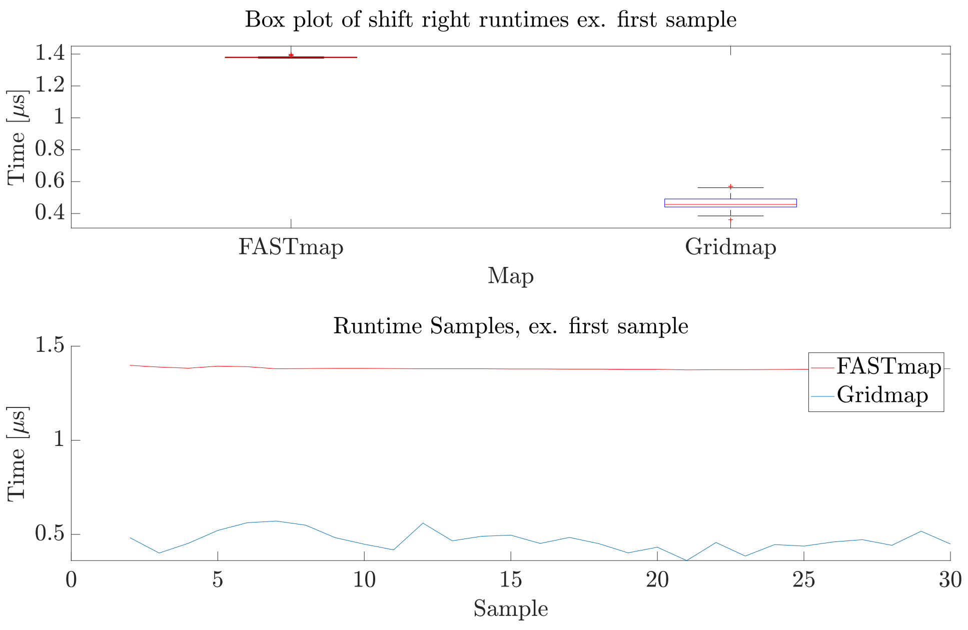


Figure 5.5 Box and sample plots of shift right runtimes excluding the first sample.

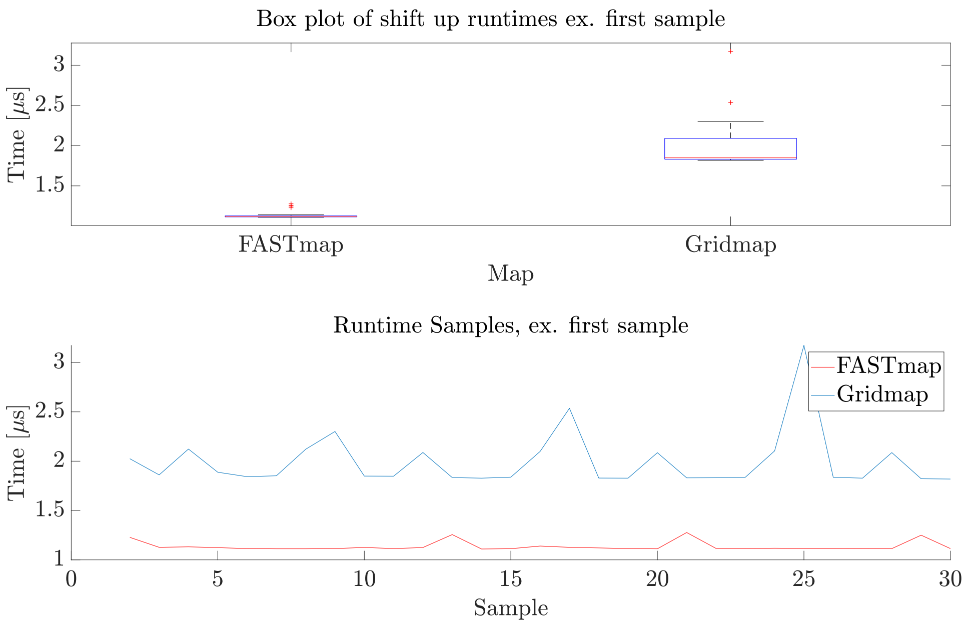


Figure 5.6 Box and sample plots of shift up runtimes excluding the first sample.

5.2.3 Discussion

As seen from the figures 5.1 to 5.6 FASTmap has shorter runtime than Gridmap in all cases except the shift right operation. However as mention earlier these examples are not sufficient to conclude that any map is faster than the other, but it exemplifies that FASTmap can have shorter runtimes than Gridmap. Proving which map is most computationally efficient is set aside for a future work where randomized experimental setups are used to statistically determine if any of the two maps are significantly more efficient than the other.

6 Conclusion

Fixed Area Sliding Template map (FASTmap) has been shown to be a versatile and efficient mapping framework with competitive execution times compared to the similar Gridmap. Even though FASTmap was not statistically proven faster than Gridmap in this report, FASTmap had lower execution time in three out of four examples. The worst case position errors of the map have also been discussed and exemplified for maps of realistic sizes. It was shown that the error introduced by the grid resolution will be the dominant error for most practical use cases, but a function for calculating the worst case error introduced by movement on the Earth ellipsoid was also provided. This to aid users that need to utilize the full range of available map sizes.

As the main focus of this report has been documenting the functionality of the map, a thorough presentation of the application programming interface has been provided. Necessary navigation theory, function signatures and code examples has been included to give the reader sufficient information to create their own mapping application. FASTmap is at the time of writing only utilized by a limited number of mapping systems. If the map becomes more widely used a need for more functionality or standardization of common code might arise. Therefore, it is suggested that further work on the mapping framework includes extensions to the API in order to suit the needs of new target applications. Furthermore a separate work dedicated to benchmarking the performance of the map, in a sufficiently randomized environment, should be conducted. As mentioned, this work should be able to conclude if the map is statistically faster than other similar maps, and could be used to evaluate further optimizations of the framework.

Appendix

A Circle Corresponding to Maximum Ellipse Curvature

Inspired by [2] the radius of a circle approximating the maximum curvature of an ellipse can be found by combining the equations for a circle (6.1) and an ellipse (6.2). It is assumed that the center of the ellipse is in the origin and that the circle has its center in $(a - r, 0)$ as shown in Figure 6.1. Here a is the semi-major axis, b the semi-minor axis and r the radius of the circle.

$$(x - (a - r))^2 + y^2 = r^2 \tag{6.1}$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{6.2}$$

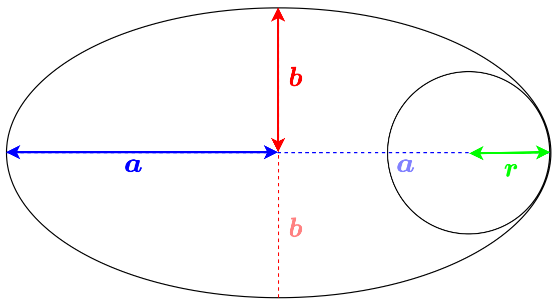


Figure 6.1 The circle with radius r corresponding to the greatest curvature of an ellipse.

Rewriting (6.1) to (6.3) and inserting into (6.2) gives (6.4) which is recognized as an quadratic equation in x.

$$y^2 = r^2 - (x - (a - r))^2 \tag{6.3}$$

$$0 = (\frac{1}{a^2} - \frac{1}{b^2}) \cdot x^2 + \frac{2(a - r)}{b^2} \cdot x + \frac{r^2 - (a - r)^2 - b^2}{b^2} \tag{6.4}$$

Solving the quadratic equation in (6.4) for x gives (6.5)

$$x = \frac{-\frac{2(a-r)}{b^2} \pm \sqrt{\frac{4(a-r)^2}{b^4} - 4(\frac{1}{a^2} - \frac{1}{b^2}) \cdot \frac{r^2-(a-r)^2-b^2}{b^2}}}{2(\frac{1}{a^2} - \frac{1}{b^2})} \tag{6.5}$$

Inserting (6.3) into (6.2) has implicitly assumed that x and y are equal for the ellipse and the circle. This is only the case in the intersection point of the two shapes. As there is only on intersection point, as seen from Figure 6.1, the discriminant of (6.5) must be zero. Setting the discriminant equal to zero, as in (6.6), and simplifying the expression yields (6.7) which is a new quadratic equation in r.

$$0 = \frac{4(a-r)^2}{b^4} - 4(\frac{1}{a^2} - \frac{1}{b^2}) \cdot \frac{r^2 - (a-r)^2 - b^2}{b^2} \tag{6.6}$$

$$0 = \frac{r^2}{b^2} - \frac{2r}{a} + \frac{a^2}{b^2} \tag{6.7}$$

Solving (6.7) for r gives (6.8), which is an equation for r using only the known quantities a and b.

$$r = \frac{b^2}{a} \tag{6.8}$$

References

[1] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4 (1):25–30, 1965. doi: 10.1147/sj.41.0025.

[2] dfnu (https://math.stackexchange.com/users/480425/dfnu). A circle inside an ellipse. Mathematics Stack Exchange, 2019. URL `https://math.stackexchange.com/q/3085954`. URL:https://math.stackexchange.com/q/3085954 (version: 2019-01-24).

[3] Olav Egeland and Jan Tommy Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics, 01 2002.

[4] Péter Fankhauser and Marco Hutter. A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation. In Anis Koubaa, editor, *Robot Operating System (ROS) – The Complete Reference (Volume 1)*, chapter 5. Springer, 2016. ISBN 978-3-319-26052-5. doi: 10.1007/978-3-319-26054-9{_}5. URL `http://www.springer.com/de/book/9783319260525`.

[5] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley and Sons, Ltd, 05 2011. ISBN 978-1119991496. doi: 10.1002/9781119994138.

[6] Kenneth Gade. A non-singular horizontal position representation. *Journal of Navigation*, 63 (3):395–417, 2010. doi: 10.1017/S0373463309990415.

[7] Rainer Grimm. Dependent names. Modernes C++, 2021. URL `https://www.modernescpp.com/index.php/dependent-types`. Accessed: 2022-01-19.

[8] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter Sweeney. Producing wrong data without doing anything obviously wrong! In *Sigplan Notices - SIGPLAN*, volume 44, pages 265–276, 03 2009. doi: 10.1145/1508284.1508275.

[9] NIMA. Department of defence world geodetic system 1984: Its definition and relationships with local geodetic systems. Technical report, National Imagery and Mapping Agency, 2000. NIMA Technical Report TR8350.2, 3rd edn.

[10] David Vandevoorde, Nicolai.M Josuttis, and Douglas Gregor. *C++ Templates, The Complete Guide*. Addison-Wesley, 2.nd edition, 2018. ISBN 978-0-321-71412-1.

[11] Todd Veldhuizen. Techniques for scientific C++. Technical report, Indiana University, Department of Computer Science, 09 2000. Technical Report # 542, Version 0.4.

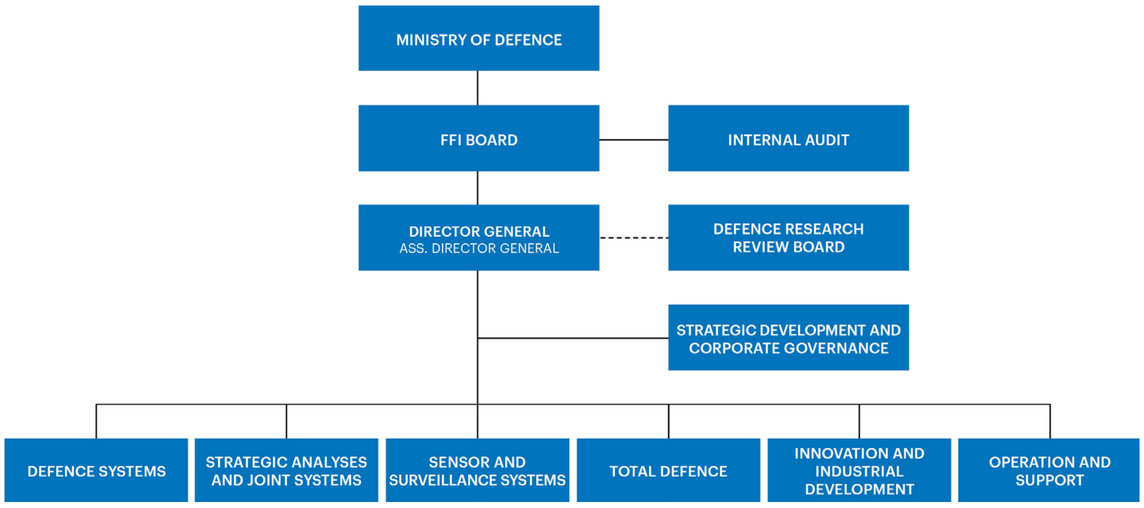
[12] Nicolas Weber and Michael Goesele. Auto-tuning complex array layouts for GPUs. In *EGPGV@ EuroVis*, pages 57–64, 2014.

About FFI
The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's mission
FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's vision
FFI turns knowledge and ideas into an efficient defence.

FFI's characteristics
Creative, daring, broad-minded and responsible.


MINISTRY OF DEFENCE
FFI BOARD
INTERNAL AUDIT
DIRECTOR GENERAL
ASS. DIRECTOR GENERAL
DEFENCE RESEARCH REVIEW BOARD
STRATEGIC DEVELOPMENT AND CORPORATE GOVERNANCE
DEFENCE SYSTEMS
STRATEGIC ANALYSES AND JOINT SYSTEMS
SENSOR AND SURVEILLANCE SYSTEMS
TOTAL DEFENCE
INNOVATION AND INDUSTRIAL DEVELOPMENT
OPERATION AND SUPPORT

Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: post@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: post@ffi.no