Structural designs for aerospace integration with python

Introduction;

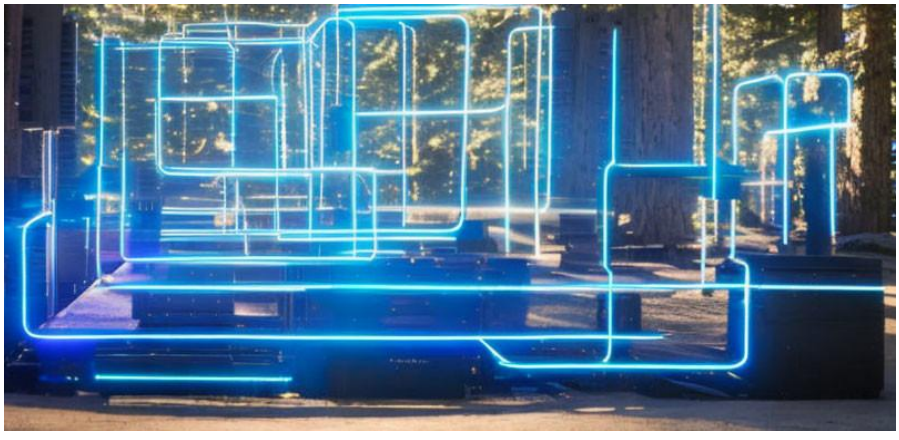
Engineering concepts for aerospace integration with python. In this text I am going to integrate the paradigm of software components with mechanical and analytic systems for the aerospace field. Robotics, the geometry of materials for robotics and aerospace, and Python are compelling fields for research and incorporate novel sensor systems and kinematics into cutting-edge robots with a powerful CPU using RODOS or ROS2 for precise signal navigation with Python.

Furthermore, I would like to undertake some examinations about RODOS, modern robotics, and vxworks with u-boot for the aerospace orbits and terminals for satellites.

By other hand, we have to take into account the field of thermodynamics which is important to notice if we can face problems in the CPU of the satellite for varying levels of pressure.

CHAPTERS;

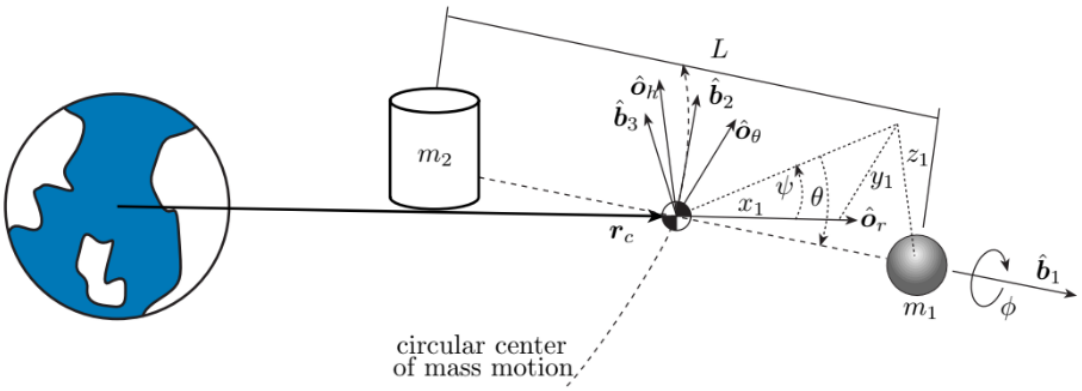
1. Physics (Electrostatic Forces on Three-Dimensional Spacecraft Shapes)
2. Aerospace
3. Orbits (python)
4. Dimensions (Computation)
5. ROS-RODOS
6. Python and u-boot



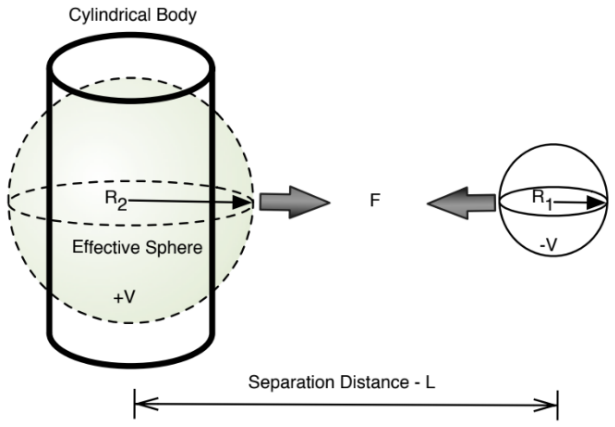
Electrostatic Forces on Three-Dimensional Spacecraft Shapes

The sphere representation allows for very fast evaluations of the electrostatic forces thanks to the analytical closed form solution. The scope of this study only considers line-of-sight electrostatic force solutions between the two the cen-ter of masses. Non-aligned forces and torques are not considered in this work. Further, of interest is over what separation distance range such simplified models yield reasonable electrostatic force approximations. If a spacecraft has long solar panels extended and deployed, then small separation distances can create strong induced charge distributions between the sphere and the nearby solar panel components.

SPHERE-SPHERE ELECTROSTATIC FORCE MODEL



While the sphere-sphere electrostatic interaction is useful for demonstration purposes, it does not closely relate to the geometry of actually deployed spacecraft. Therefore, it is of interest to study generic 3D shapes and their electrostatic interaction with a spherical vehicle. It is beyond the scope of this text to consider interactions between multiple 3D shapes.



expressions for q1 and q2, Eq. (6) is separated into two equations.

V1 = kc ✓ q1 R1+ q2 L

V2 = kc ✓q1 L + q2 R2

Programming the electrostatic interaction

The intensity of the force between two nodes is determined by the distance between them (D), the charge of the opposite node (C), the strength of the link (L) and the simulation alpha (A), using the following formula: ACL/D^2.

In the case of a full-mesh group of nodes, the strength of the links is equal for all node pairs, rendering L a system constant. When modeling gravity this would be your gravitational constant G (and C each node's mass), while in an electrostatic system it would represent the Coulomb's law constant k (and C each node's electrical charge).

Node charges (C) can be positive or negative. Positive means that this node will attract other nodes with the specified intensity, while a negative charge represents a repelling force towards other nodes. Keep in mind that, unlike electrical charge, two positive charges do not repel each other, and two opposite charges do not mutually attract each other.

```
import d3ForceMagnetic from 'd3-force-magnetic';
d3.forceSimulation()
    .nodes(<myNodes>)
    .force('magnetic', d3.forceMagnetic()
        .strength(0.8)
    );

export default function(x) {
  return function() {
    return x;
  };
}
```

Magnetic forces for structural components example;

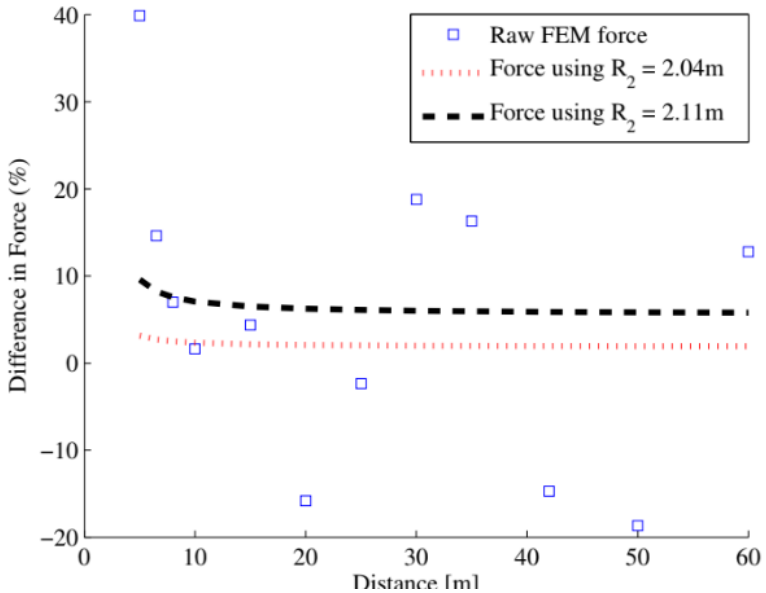
src/magnetic.js

```
import constant from './constant';
import {binarytree} from 'd3-binarytree';
import {quadtree} from 'd3-quadtree';
import {octree} from 'd3-octree';

export default function() {
  let nDim,
```

```
    nodes = [],
    links = [],
    id = (node => node.index),        // accessor: node unique id
    charge = (node => 100),         // accessor: number (equivalent to node
mass)
    strength = (link => 1),         // accessor: number (equivalent to G
constant)
    polarity = ((q1, q2) => null),      // boolean or null (asymmetrical)
    distanceWeight = (d => 1/(d*d)),    // Intensity falls with the square of
the distance (inverse-square law)
    theta = 0.9;

  function force(alpha) {
    if (links.length) { // Pre-set node pairs
      for (let i = 0; i < links.length; i++) {
        const link = links[i],
          dx = link.target.x - link.source.x,
          dy = (link.target.y - link.source.y) || 0,
          dz = (link.target.z - link.target.z) || 0,
          d = distance(dx, dy, dz);

        if (d === 0) continue;

        const relStrength = alpha * strength(link) * distanceWeight(d);

        const qSrc = charge(link.source),
          qTgt = charge(link.target);

        // Set attract/repel polarity
        const linkPolarity = polarity(qSrc, qTgt);

        const sourceAcceleration = signedCharge(qTgt, linkPolarity) *
relStrength;
        const targetAcceleration = signedCharge(qSrc, linkPolarity) *
relStrength;

        link.source.vx += dx / d * sourceAcceleration;
        link.target.vx -= dx / d * targetAcceleration;
        if (nDim > 1) {
          link.source.vy += dy / d * sourceAcceleration;
          link.target.vy -= dy / d * targetAcceleration;
        }
        if (nDim > 2) {
          link.source.vz += dz / d * sourceAcceleration;
          link.target.vz -= dz / d * targetAcceleration;
        }
      }
    } else { // Assume full node mesh if no links specified
   const tree =
          (nDim === 1 ? binarytree(nodes, d => d.x)
          :(nDim === 2 ? quadtree(nodes, d => d.x, d => d.y)
          :(nDim === 3 ? octree(nodes, d => d.x, d => d.y, d => d.z)
          :null
        ))).visitAfter(accumulate);
```

```
    const etherStrength = alpha * strength();

    for (let i = 0; i < nodes.length; i++) {
      const node = nodes[i],
  nodeQ = charge(node);
tree.visit((treeNode, x1, arg1, arg2, arg3) => {
```

This quadratic equation is solved for q1. Because q1 is now known, q2 is solved by using Eq. (7). All parameters except for the effective radius, R2. R2 can be easily solved to obtain the second body's effective spherical radius. Computing the effective radius of a known sphere has helped illustrate numerical accuracy.



Code-references
https://github.com/vasturiano/d3-force-magnetic.git

The electrostatic charges and forces change between the two objects, at the time the space object is in the atmosphere, the electrostatic analysis and thermodynamic analysis in programming languages such as python and the magnitudes are important to determine some important aspects of commutability problems between magnitudes.

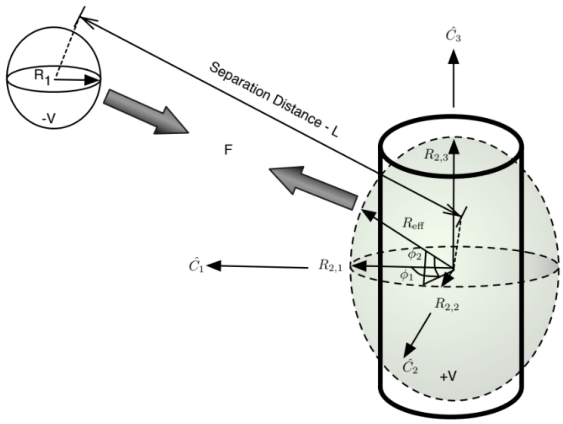
```
const relStrength = alpha * strength(link) * distanceWeight(d);

      const qSrc = charge(link.source),
        qTgt = charge(link.target);

      // Set attract/repel polarity
      const linkPolarity = polarity(qSrc, qTgt);
```

```
        const sourceAcceleration = signedCharge(qTgt, linkPolarity) *
relStrength;
        const targetAcceleration = signedCharge(qSrc, linkPolarity) *
relStrength;

        link.source.vx += dx / d * sourceAcceleration;
        link.target.vx -= dx / d * targetAcceleration;
        if (nDim > 1) {
          link.source.vy += dy / d * sourceAcceleration;
          link.target.vy -= dy / d * targetAcceleration;
        }
```

Cylinder electrostatic forces in the geometric vectors;

```
x = R2,1 cos2 cos
1 (15a)

y = R2,2 cos2 sin
1 (15b)

z = R2,3 sin

2 (15c)

Result:
R2 = Reff = px2 + y2 + z2
```

Cylinder example (electrostatic forces for satellites development and programming control-structures)

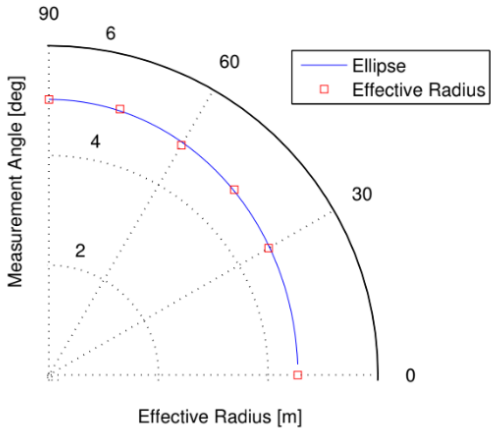


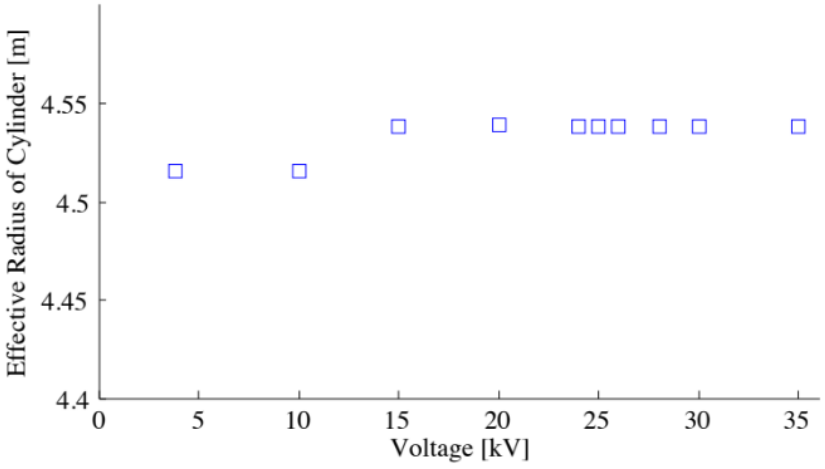
R2 = R2,1 = Reff if linearized about

1 =

2 = 0.

This fact makes formation dynamics simpler if constant attitudes are held between the cylinder and sphere. Elliptical distributions of the effective radius may not occur for every 3D body geometry.the voltage between the two craft but, it is uncertain whether changes in voltage will cause the effective radius to change. Thus, a sweep across voltages is performed holding the cylinder and sphere in the orientation seen is obtained by changing the voltage on both bodies (equal and opposite) and then calculating the effective radius.





Aerospace(Gravity gradient stabilization)

SPIN STABILIZATION and DUAL-SPIN STABILIZATION focused on torque-free motion of a spinning spacecraft, and that of a platform with a spinning wheel, respectively. In contrast, this lesson takes gravity gradient torques into account, and aims to perform a stability analysis similar to those previously seen. Via judicious design, naturally-occurring force fields can be exploited for the purpose of passive attitude stabilization that requires no power, control laws and sensing.

• FI : inertial frame fixed to (but not rotating with) Earth
• FO: orbiting frame, with origin fixed to spacecraft, 3-axis towards Earth's center, 2-axis anti-parallel to orbital angular momentum,~h
• FB: body-fixed frame, with origin at spacecraft center of mass

The spacecraft's position and angular velocity (with respect to FI) can be resolved in FB. Assume the selected body-fixed frame is a principal axes frame; that is, FB ≡ FP . Using the diagonal moment of the inertia matrix, I, associated with such a frame, the gravity gradient disturbance torques can be determined as derived in disturbance variables.

Forces and gravity:

```
  force.initialize = function(initNodes, nDims) {
    const numDimensions = [1,2,3].includes(nDims) ? nDims : 2;

  force.initialize = function(initNodes, ...args) {
    nodes = initNodes;
    nDim = numDimensions;
    nDim = args.find(arg => [1, 2, 3].includes(arg)) || 2;
    initialize();
  };
```

The polarity of the earth in equations and programming electrostatic forces across the gravity gradient;

```
function force(alpha) {
    if (links.length) { // Pre-set node pairs
      for (let i = 0; i < links.length; i++) {
        const link = links[i],
          dx = link.target.x - link.source.x,
          dy = (link.target.y - link.source.y) || 0,
          dz = (link.target.z - link.target.z) || 0,
          d = distance(dx, dy, dz);
```

Gravity-gradient stabilization;

Combining these results with the I1 > I3 condition for pitch stability, we observe that three-axis attitude stability is guaranteed for I2 > I1 > I3. In other words, a spacecraft with its major axis along the orbital plane's normal and its minor axis in the nadir direction is gravity gradient-stabilized.

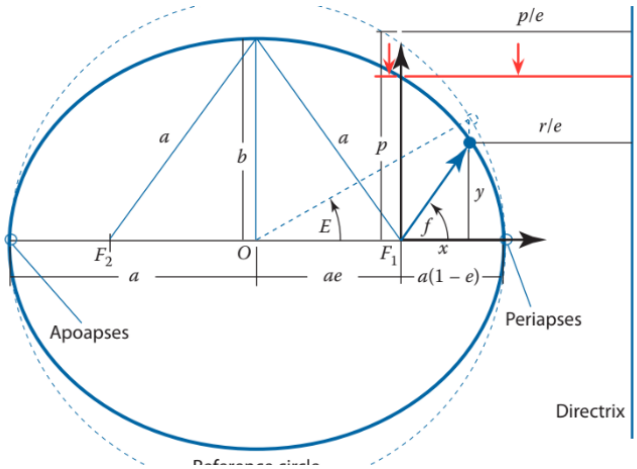
```
 const linkPolarity = polarity(qSrc, qTgt);

        const sourceAcceleration = signedCharge(qTgt, linkPolarity) *
relStrength;
        const targetAcceleration = signedCharge(qSrc, linkPolarity) *
relStrength;

        link.source.vx += dx / d * sourceAcceleration;
        link.target.vx -= dx / d * targetAcceleration;
        if (nDim > 1) {
          link.source.vy += dy / d * sourceAcceleration;
          link.target.vy -= dy / d * targetAcceleration;
        }
```

We can calculate the acceleration and the polarity across the gravity gradient equations in programming.

Analytical mechanics of the space vectors,

Orbits (python)

Change spherical to planar to make it say consider the unforced planar pendulum equations and change spherical to planar to make it say that our **problems to** consider the unforced planar pendulum equations and change spherical to planar to consider the inverted planar pendulum problem...The p and p/e distances are incorrectly indicated. The guidelines should be adjusted as shown in red below.



Vectors

The coordinate frame

```
B : {O,^b1,^b2,^b3}
```

Illustrated in Figure 1 is defined through its origin O and the three mutually orthogonal unit direction vectors:

```
{
^b1,
^b2,
^b3}
```

A vector with a ˆ symbol denotes a unit length di-rection vector. A right-handed coordinate frame sat-isfies ˆb1 × b2 = ˆb3. If this coordinate frame B is attached to a rigid body, then describing the orien-tation of the rigid body is equivalent to studying the orientation of B.

Variables to model gravity-gradient equations:

a < 0 for any positive energy in our diagram

- Apoapsis
- Periapsis
- Directrix

```
import sys
sys.path.append('/home/liamk/mirror/superdarn/davitpy')
import models.igrf as igrf #use the davitpy wrapper on the IGRF
import numpy as np
from numpy import cos,sin

import apex_converter
import datetime
import pdb

def drawApexContours(year,dayofyear,secofday,hemi='N'):
      """
      Display apex latitude and longitude lines on an orthographic
      projection of the globe.

      Parameters
```

```
      ----------
            year : int
            dayofyear : int
            secofday : int
                  Date and time for apex transform.
            satloc : [lat,lon] - int
                  Geographic Latitude and Longitude for POV of projection
                  (perspective of looking down from a satellite at satloc)

      Returns
      -------
            map - Basemap instance for the orthographic projection
      """
```

Vector variables representation:

A vector ω represents a series of components as a 3×1 matrix, we must specify with respect to which frame the vector components have been taken. If some possible B frame components are used as in Eq. (3), then the left superscript nota-tion is used again.

```
Bω =B
ω1
ω2
ω3
```

Orthographic programming projection across gravity gradients and electrostatic forces in the atmosphere:

Here we have some geometry coordinates and lambda functions to represent orthographic projection of the map and support our gravity gradient torques:

```
#Compute the datetime for this year, dayofyear, secofday
      #Used by nightshade
      # set up orthographic map projection with
      # perspective of a satellite looking down at 50N, 100W.
      # use low resolution coastlines.
      doy2datetime = lambda doy,year,secofday:
datetime.datetime(year,1,1,0,0,0)+datetime.timedelta(days=doy)+datetime.timede
lta(seconds=secofday)
      time = doy2datetime(year,dayofyear,secofday)


      #Set up the map
      #--------------
```

Altitudes and gravity gradient are important to extend the atmospheric knowledge as well as dimensional 3D background frames with initial inertial systems (I)

```
# make a regular lat/lon grid.
      nlats = 73; nlons = 145; delta = 2.*np.pi/(nlons-1)
      lats = (0.5*np.pi-delta*np.indices((nlats,nlons))[0,:,:])*180./np.pi
      lons = (delta*np.indices((nlats,nlons))[1,:,:])*180./np.pi
      alts = np.ones_like(lats)*800. #altitudes of 800 km

      #Get the apex latitude and longitude for all grid points at an altitude
of 800 km for a reference height of 110 km
      alat,alon,qdlat =
converter.geo2apex(lats.flatten(),lons.flatten(),alts.flatten(),hr=110.)

# compute native map projection coordinates of lat/lon grid.
      x, y = map(lons, lats)
```

In the process of how to differentiate a vector r expressed in B and the vector components. To discuss the time evolution of r, an observer frame must be specified.

Dimensions (Computation)

Enumerative combinatorics applied to higher dimensions for computational algorithms research;

The main goal of enumerative combinatorics is to count the elements of a finite set. Most frequently, we encounter a family of sets T0, T1, T2, T3, . . . and we need to find the number tn = |Tn| for n = 1, 2, ...

For example, the tilings of Figure 1.1 correspond, respectively, to 1 + 1 + 1 + 1, 1 + 1 + 2, 1 + 2 + 1, 2 + 1 + 1, 2 + 2. These sums are easy to count. If there are k summands equal to 2 there must be n−2k summands:

an = bX n/2c , k=0 (n − k) k = n0 + (n − 1) ,1 + (n − 2) 2
+ · · · .

We can apply this formula to computational variables:

f(x + h, y + k) variables

Decomposition

```
variable subzero = 0.0012
variable h = 2.12
variable y = 1.25
variable k = 2.00
```

f(x,y) + [hf'x(x,y) + KF'y(x,y) = 0

```
import Tfidf Vectorizer, CountVectorizer
From sklearn.decomposition import NMK, lateratDirichletAllocation
```

And then model the algorithm to research higher dimensions in a imaginary computation in high scales Rather to think on small computing scales. Recurrence. Let n ≥ 2. In a domino tiling, the leftmost column of a 2 × n can be covered by a vertical domino or by two horizontal dominoes. If the leftmost domino is vertical, the rest of the dominoes tile a 2 × (n − 1) rectangle, so there are an−1 such tilings. On the other hand, if the two leftmost dominoes are horizontal, the rest of the dominoes tile a 2 × (n − 2) rectangle, so there are an−2 such tilings. We obtain the recurrence relations:

a0 = 1, a1 = 1, an = an−1 + an−2 for n ≥ 2.

Explicit formula 2. There is a well established method that turns linear recurrence relations with constant coefficients, such as (2), into explicit formulas. We will review it in Theorem 2.4.1. In this case, the method gives

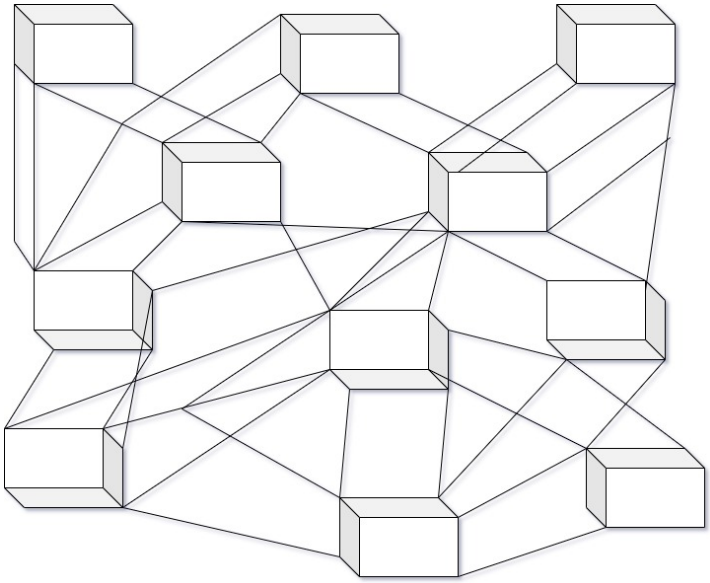
an = 1 √5 1 + √5 square 2 (!n+1) − 1 −√5 square2 !n+1

$$a_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right).$$

Generating function. The last kind of answer we discuss is the generating function. This is perhaps the strangest kind of answer, but it is often the most powerful one.

Consider the infinite power series A(x) = a0 + a1x + a2x 2 + · · · . We call this the generating function of the sequence a0, a1, a2, . .

A graphical example of higher computing dimensions;



Spatial relations between particles and byte states:

Functions and explanations of bits and the relations with higher order dimensions states of particles. The functions of n bits modulus p. DH function security mod p. suppose p is n bits long

$$e^{\blacktriangle}\left(\sqrt[3]{n}\right)$$

$$e^{\sqrt[3]{1024\ bits}} = e^{10\ bits}$$

The idea is to take as reference a non-determined sequence of n bits or a possible quantum space & the replication of u elements. We have a particle such 2 [Y] , [N] as dimensions y and n for several elements or n bits. Also, the following example is represented in a quantum space scenario. The element can replicate itself in another way or another dimension of space-time but is not a dual relationship between particles or n bits set. A single wall with particle interchange between n, p

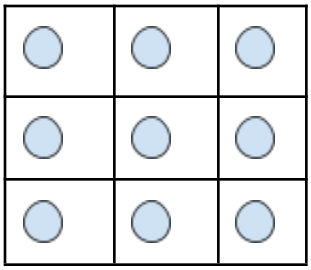
Different n particle replication or not (two different states)

The main idea is to identify that photons can travel in different states in a wall or given space. The exponential replication is a non-determined sequence of n bits or particles such that it occurs in quantum replication of elements in a set u. We can compare states between particles and a set of n bits because permutations and transmission of different states and replications take place across different high order dimensions.

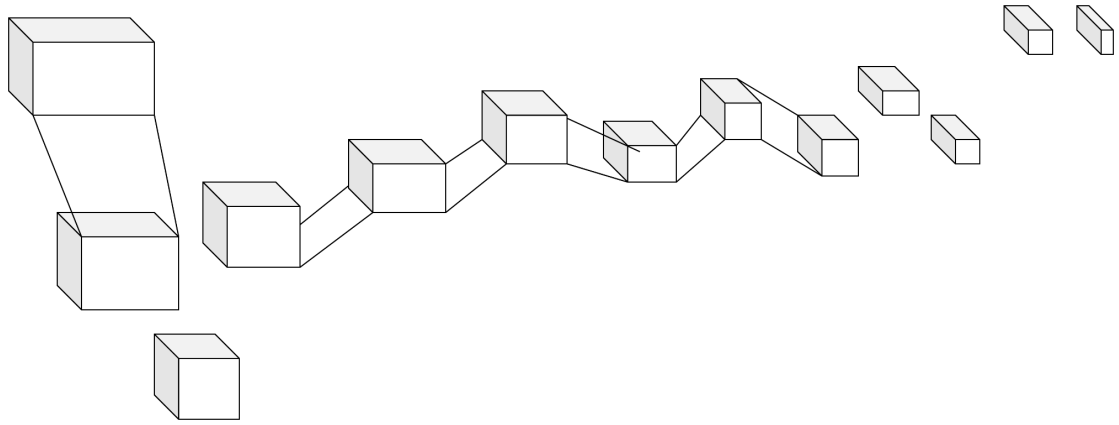
What can I see in a new mathematical cubic format?

I want to introduce you to the cubic dimension to understand relations between a set of elements and bits.

$e^{\blacktriangle}\left(\sqrt[3]{n}\right)$ => $\left(\sqrt[3]{n}\right)$ => cubic space of $\left(\sqrt{n}\right)$ bits



The probability and the proportions of n bits-particles can be chaotic if the proportions are high and the probable collision of elements is also high. In the example above we have a cubic relationship between bits or possible particles. **Graphical example of dimensions replication:**
Replication Process-partitions-5D



The modern robotic satellites CPU (RODOS-ROS)

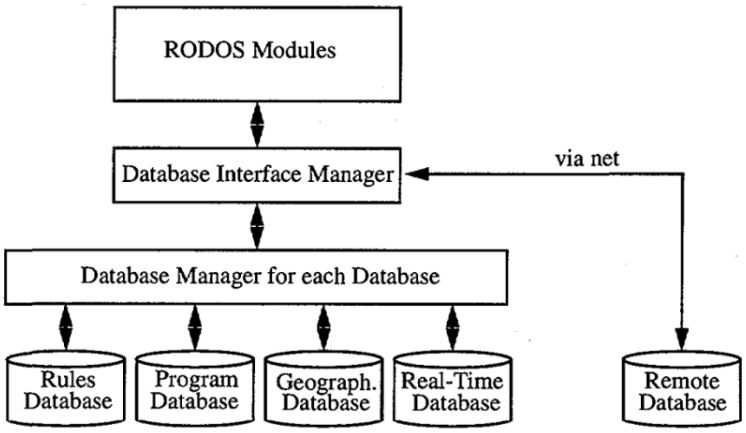
The RODOS system is based on the Client-Server principle. It is built of modules, which are connected via a Communication Interface. Each of these modules can either be a

• Server, which provides special services to other modules, or a
• Client, which requests services from other modules, or both. Well defined data structures allow the exchange of data between the client and the server.

The dialogue between RODOS and a user can be organized in two different modes. In the so-called "automatic mode" the system automatically presents all information which is relevant to decision making and quantifiable in accordance with the current state of knowledge in the real cycle time.

The exchange of messages between the modules of the RODOS system is controlled by the Communication Interface. Each module can send messages to and receive messages from other modules. The messages contain fields which define the type, sender and recipient of the message. Three types of messages are considered by the Communication Interface:

• Requests are sent to other modules to ask for special services.
• Notification is sent back by the recipient if the request was successfully completed.
• Failures are sent back if some error occurred during the service.



The program database contains parameters and results of the application software implemented in RODOS. The real-time database will comprise all kinds of environmental monitoring data and measurements. The information in the rules database consists of expert judgments, facts, rules and preferences required for both evaluating alternative countermeasure combinations and controlling the user interaction and program flow in RODOS.

Code-examples for satellites

Definition_1: Attribute characterizes a piece of information, which can be expressed by a sin-gle value.

Definition_2: Value is the magnitude of an attribute. It can be a number, a description or a key which is an abbreviation of a complex description or a link to another entry in the database.

```
#include "rodos.h"
namespace RODOS {
bool        getIsHostBigEndian()          { return isHostBigEndian; }
int32_t        getSpeedKiloLoopsPerSecond() { return 313330; }    // see
rodos-tutorials/development-tests/cpuspeedtest.cpp
int32_t        getMaxIdleCntPerSecond()     { return 4992266; }   // see
rodos-tutorials/development-tests/cpuspeedtest.cpp
int32_t        getYieldTimeOverhead()       { return 990; }       // see
tutorials/core/yieldtime.cpp
const char* getHostCpuArch()               { return "x86"; }
const char* getHostBasisOS()              { return "baremetal"; }
} // namespace RODOS
```

```
#pragma once
13
14
// #include "hw_datatypes.h"
15
16
// _____________________  only defines, not in namespace
17
18
19
#define OSVERSION "Linux-x86"
20
21
/*************** System Configuration *********/
22
23
//#define UART_GATEWAY                //< activates the Interrupt fo incomming
networkmessages for the UART Gateway
24
//#define ENABLE_LINUX_CAN_INTERRUPT  //< Uncomment to enable Linux CAN
Interrupt (this may hang when using UDP!!)
```

RODOS-SERVERS

Timers for satellites

```
void Timer::start() {
120
    itimerval params;
121
    int       retval;
122
    params.it_interval.tv_sec  = static_cast<time_t>(microsecondsInterval /
1000000);
123
    params.it_interval.tv_usec =
static_cast<suseconds_t>(microsecondsInterval % 1000000);
124
    params.it_value.tv_sec     = params.it_interval.tv_sec;
125
    params.it_value.tv_usec    = params.it_interval.tv_usec;
126
    retval                     = setitimer(ITIMER_REAL, &params, 0);
127
    RODOS_ASSERT(retval == 0); // error during call to setitimer
128
}
```

ROS-SERVERS

We can build robotic server-client systems inside the spatial object to reinforce the communications between two parties and satellite signals, but we need to keep in mind the thermodynamics in outer space and the conditions such as asteroids and dark matter. Here I have some ROS server files with the headers. The idea is to communicate satellites with strong signals and space research with RODOS, linux arm x86 and ROS for the modern robots that can be placed inside the satellites. We can also add RPA functionality to the robots brain.

```
#include "ros/ros.h" // ROS Default Header File
// SrvTutorial Service File Header (Automatically created after build)
#include "ros_tutorials_service/SrvTutorial.h"
#include <cstdlib> // Library for using the "atoll" function
int main(int argc, char **argv) // Node Main Function
{
ros::init(argc, argv, "service_client"); // Initializes Node Name
if (argc != 3) // input value error handling
{
ROS_INFO("cmd : rosrun ros_tutorials_service service_client arg0 arg1");
ROS_INFO("arg0: double number, arg1: double number");
return 1;
}
```

```
int64 a
int64 b
---
```

int64 result
Nucleus CPU

I would like to create server-nodes and robotic-programs testing for aerospace research in order to find new forms of life and improve the materials of the earth and the communications.

Here is another server in ROS

```
ros::ServiceClient ros_tutorials_service_client =
nh.serviceClient<ros_tutorials_service::SrvTutorial>("ros_tutorial_srv");
// Declares the 'srv' service that uses the 'SrvTutorial' service file
ros_tutorials_service::SrvTutorial srv;
// Parameters entered when the node is executed as a service request value are
stored at 'a' and 'b'
srv.request.a = atoll(argv[1]);
srv.request.b = atoll(argv[2]);
// Request the service. If the request is accepted, display the response value
```

First of all, we need to study the space conditions across aerospace (mechanics,vectors,arrays) And second, we need to develop more efficient servers and communications on the terminal side of linux, Rodos and **vxworks** across das u-boot. The space conditions are so hard to measure so, I come across some investigations, and I conclude that the easiest way to figure out what is going on in the outer space and parallel dimensions is to study the magnitudes simplifying the measurements and trajectories in arrays and matrices with Scipy and another classical programming because stateless functions are so helpful to check any dimensional state condition for the aerospace research.

```
    long* ebp;
68
    __asm__ __volatile__ ("mov %%ebp, %0":"=r"(ebp));
69
70
  /* instruction pointer of interrupted programm is at ebp[17]
71
  * see System V abi and linux src : include/asm-x86/sigcontext32.h
72
  */
73
74
  /* we replace the instruction pointer in the saved context and continue with
75
  * our own context saving procedure
76
  */
77
    __interruptedInstructionPointer__ = ebp[17];
```

```
78
    ebp[17]=(long)__asmSaveContext;
79
}
```

Vxworks is a platform where we can find important pieces of code and bootable functionality as well as linux armx86, arm x32 bytes to perform some parallel tasks and threads. I want to present some code to perform some transactional operations for satellites or spatial objects based on linux systems and kernel drivers. One of them might be RODOS and the ROS turtle servers are perfect to install some robots inside other bigger spatial objects.

Another important aspect is the Dimensions in the atmosphere and the trajectory collisions, robot kinematics to build important aspects of robotic systems for aerospace and some CPU functions such timers for satellites, simple and concurrent servers and pentesting techniques for linux kernels with important functionalities for spatial research. I pretend to unify the concrete system of programming with the conditions in outer space based on the enumerative higher order dimensions and trajectory systems for spatial objects.

Signals in the space to catch another materials in hard conditions with terminals:

```
char signalHandlerStack[SIGNAL_HANDLER_STACK_SIZE];
85
void Timer::init() {
86
    static int once = 0;
87
    if(once == 0) {
88
        int              retval;
89
        struct sigaction action;
90
        stack_t          signalStack;
91
92
        once = 1;
93
94
        /* callback function for signal handling */
95
        action.sa_handler = timerSignalHandler;
```

ROBOTICS

Kinematics and applied mechanics to study the degrees of freedom of the robot could vary in different scenarios if we have some variations in the temperature of materials and some pressure magnitudes:

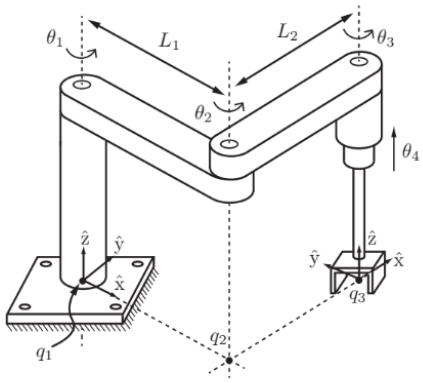


Figure 5.7: Space Jacobian for a spatial RRRP chain.

- Since the final joint is prismatic, $\omega_{s4} = (0, 0, 0)$, and the joint-axis direction is given by $v_{s4} = (0, 0, 1)$.

The space Jacobian is therefore

$$J_s(\theta) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & L_1 s_1 & L_1 s_1 + L_2 s_{12} & 0 \\ 0 & -L_1 c_1 & -L_1 c_1 - L_2 c_{12} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Each of the ROS, CPU and robot functionality needs to be tested at some points with some framework but the slow method of testing matrices with scipy and linux terminals is the best choice to test materials and magnitudes such as pressure and higher order dimensions. Design is less important than testing things such as materials and some spatial forces around the object to measure before the starting of the project.

```
include_directories(${catkin_INCLUDE_DIRS} ${Boost_INCLUDE_DIRS})

Chapter 7 _ Basic ROS Programming 175

add_executable(action_server src/action_server.cpp)
add_dependencies(action_server ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(action_server ${catkin_LIBRARIES})
add_executable(action_client src/action_client.cpp)
add_dependencies(action_client ${${PROJECT_NAME}_EXPORTED_TARGETS}
${catkin_EXPORTED_TARGETS})
target_link_libraries(action_client ${catkin_LIBRARIES})

RODOS signal stack
static int once = 0;
87
    if(once == 0) {
88
        int               retval;
89
        struct sigaction action;
90
        stack_t           signalStack;
```

```
#! /bin/bash
export MCU=STM32L432xx
export BOARD=nucleo_l432kc
export TARGET_LIB=${BOARD}
export LINKER_SCRIPT="${RODOS_SRC}/bare-metal/stm32l4/scripts/stm32l432kc.ld"
source $RODOS_VARS_DIR/stm32l4.sh
```

Libraries for arch-rodos-linux operations

```
export TARGET_LIB=gecko  #used as name for the generated lib
export SUB_ARCH=efr32fg1p
export SUB_ARCH_DIR=EFR32FG1P
export SUB_ARCH_FLAGS="-DEFR32FG1P -DEFR32FG1P133F256GM48
-DEFR32_SERIES1_CONFIG1_MICRO"
export RAIL_LIB="rail_efr32xg1_gcc_release"
source $RODOS_VARS_DIR/efr32fg1p.sh

ROS COMMUNICATIONS
rosmsg: ROS Message Information

Command Description
rosmsg list Show a list of all messages
rosmsg show [MESSAGE_NAME] Show information of a specified message
rosmsg md5 [MESSAGE_NAME] Show the md5sum
rosmsg package [PACKAGE_NAME] Show a list of messages used in a specified
package
rosmsg packages Show a list of all packages that use messages
```

Furthermore, higher order dimension tunnels could be modified at a spatial level with some type of virtualized material that would allow access to high dimensions in space. The problem is that to generate these new hologram-type terminals, a material that has not yet been investigated is needed. We need to investigate other forms to build computers based in higher dimensional spaces instead of quantum computers. Other spatial materials..

Sensor example

```
sensor_msgs/Joy
sensor_msgs/JoyFeedback
sensor_msgs/JoyFeedbackArray
sensor_msgs/LaserEcho
zeroconf_msgs/DiscoveredService

#pragma once
10
11
12
#include <stdint.h>
13
#include <stddef.h>
```

```
14
15
16
namespace RODOS {
17
18
19
/** Computes a 16 bit checksum (len in bytes) adding bytes and rotating result
*/
20
uint16_t checkSum(const void *buf, size_t len) ;
21
22
23
/** computes a 16 bit crc in a non optimized way, CCSDS recommends 0 (some
times 0xffff) as initial value **/
24
uint16_t computeCrc(const void* buf, size_t len, uint16_t initialValue);
```

Practical ideas; Go to our terminals to prove some spatial properties in vxworks or das-u boot, scipy And some kernels for spatial investigations. We need to install new Robots, new CPUs,s and new linux kernels to build spaceships.

```
/** An optimized crc generator.
28
  * first it generates a luck up table to speed up the crc computation
29
  * use only one in the system. more is not required.
30
  * CCSDS recommends 0 (but Warning: some times 0xffff) as initial value
31
  */
class CRC {
34
    uint16_t lookUpTable[256];
public:

    CRC();
    uint16_t computeCRC(const void* buf, size_t len, uint16_t initialValue);
};
/** Delivers a 16 bit  hash value for a string.
42
 *  both bytes contain only printable characters

uint16_t hash(const char* str);
46
47
48
/// Computes the checkSum of an array of 32-bit words, according to the CCSDS
algorithm.
49
/// Warning: len ist not bytes! it is num of 32-bit words
50
```

```
uint32_t checkSumXor32(const uint32_t *buf, size_t len);
```

Buffer:

```
protected:
36
    /// Data area consisting of three independent parts to avoid access
conflicts.
37
    Type buffer[3];
38
    /// Pointer to buffer area that is currently written or will be written
next.
39
    Type* writer;
40
    /// Pointer to buffer where last data is stored.
41
    volatile Type* reader;
42
    /// Additional buffer, to avoid conflicts between simultaneous read-write
access.
43
    Type* intermediate;
44
    /** Flag to protect buffer. It indicates access to the current read buffer
45
     * in order to prevent data conflicts.
46
     */
47
    volatile bool readingNow;
48
    volatile bool newDataAvailable; ///< indicates whether new data is
available
49
50
public:
51
  uint64_t writeCnt;    ///< counter of writing actions
52
  uint64_t readCnt;     ///< counter of reading actions
53
54
  CommBuffer() {
55
    writer = &buffer[0];
56
    reader = &buffer[1];
57
    intermediate = &buffer[2];
58
    readingNow = false;
```

```
59
    newDataAvailable = false;
60
    writeCnt = readCnt = 0;
61
  }
```

buffer-satellites