SAPIENZA
UNIVERSITÀ DI ROMA

Robot Programming
C++ Eigen

Giorgio Grisetti

Eigen

A template based header only math library for linear algebra

Supports fixed and dynamic matrices and vectors on base and user defined scalar types

Matrix<typename Scalar, int Rows, int Cols>

where

- Scalar can be {float, double, int, complex or user-defined>
- Rows can be an int or the constant Eigen::Dynamic
- Cols can be an int or the constant Eigen::Dynamic

It implements the standard operators and much more

https://eigen.tuxfamily.org

Eigen Basic Types

```
using Matrix3f = Eigen::Matrix<float, 3, 3>;
using Matrix2f = Eigen::Matrix<float, 2, 2>;
using Matrix2_3f = Eigen::Matrix<float, 2, 3>;
using Vector3f   = Eigen::Matrix<float, 3,1>;

int main() {
  Eigen::Matrix<float, 3,3> m1;
  Matrix3f m2;
  m2.setZero();

  m1 << 1,2,3,
        4,5,6,
        7,8,9;


  std::cerr << "m1: " << endl << m1 << endl;
  std::cerr << "m2: " << endl << m2 << endl;
...
```

eigen_00_basics_test.h

Example: Generic Point load/save

```
template <typename ContainerType_>
int loadPoints(ContainerType_& dest,
               std::istream& is) {
  using VectorType=
        typename ContainerType_::value_type;
  constexpr int dim=
        VectorType::RowsAtCompileTime;

  while (is.good()) {
    VectorType v;
    for (int i=0; i<dim; ++i) {
      is >> v(i);
    }
    if(! is.good())
      break;
    dest.push_back(v);
  }
  return dest.size();
}
```

```
template <typename ContainerType_>
int savePoints(std::ostream& os,
                  ContainerType_& src) {
  using VectorType =
    typename ContainerType_::value_type;
  constexpr int dim=
    VectorType::RowsAtCompileTime;

  for (const auto& v: src) {
    for (int i=0; i<dim; ++i) {
      os << v(i) << " ";
    }
    os << std::endl;
  }
  return src.size();
}
```

eigen_01_point_loading.h

Eigen: Members

The Eigen objects are machine optimized and when building in release they require to be aligned at certain address boundaries

When declaring a class that contains eigen objects we need to tell the compiler that we want our datatype to be aligned

just add the macro

`EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

in the public part of the class header

Giorgio shows example

Eigen: Containers

For the same reason we need to inform stl containers potentially holding Eigen objects, about the peculiarity of allocation.

To this end we need to pass a template argument that is the Eigen::aligned_allocator<T> when defining a container.

Example:

```
using Vector3fVector =
    std::vector<
        Vector3f,
        Eigen::aligned_allocator<Vector3f>
    >;
```

eigen_01_point_loading.h

Eigen: Isometries and Transforms

The Geometry package of Eigen provides the most common transforms

- Isometry
- Affine
- Similarity
- Projective

Transforms can be manipulated, multiplied and converted

- the method linear() accesses to the Rotation Matrix/Linear part
- the method translation() accesses to the translation/affine part

Transformations can be multiplied to points, to apply the corresponding change in reference system

Exercises

Write a function object that, given:

- a point **p** (static, with size known at compile time)
- a point **m**
- a point **n**

returns true if

$$(\mathbf{p}-\mathbf{m})^T \mathbf{n} < 0$$

Write the function in the form of a predicate (see previous lesson's exercise)

Exercises

Write a function that given

- three angles alpha alpha_x, alpha_y, alpha_z
- and a translation t=(x,y,z)
- a file containing 2D data points

Loads all points in a structure (list or vector)

Applies the transform [R_x(alpha_x)*R_y(alpha_y)*R_z(alpha_z) | t] to each point in the pool

Sorts the points according to the user specified coordinate(x, y, z) and prints them to the screen.

Exercises

Write a function that given

- a container (list or vector) of structs in the form

  <index> Vector<Scalar, N>
- A range on that list specified as a pair of iterators (start, stop)

Computes

- the mean (mu) and
- the covariance matrix (Sigma)

of the points in the range

$$\mu = \frac{1}{N}\sum_i (\mathbf{p}_i)$$

$$\Sigma = \frac{1}{N-1}\sum_i (\mathbf{p}_i - \mu)\cdot(\mathbf{p}_i - \mu)^T$$