Eigen Library Quick Tutorial

Multiplication:

```
Eigen::Matrix3d A = Eigen::Matrix3d::Identity();
Eigen::Vector3d a(0.5, 3, -0.4);
Eigen::Vector3d Aa = A * a;
std::cout << "The multiplication of A * a is " << std::endl << Aa << std::endl;

Eigen::MatrixXd B = Eigen::MatrixXd::Identity(6, 5);
Eigen::VectorXd b(5);
b << 1, 4, 6, -2, 0.4;
Eigen::VectorXd Bb = B * b;
std::cout << "The multiplication of B * b is " << std::endl << Bb << std::endl;
```

Transpose and inverse:

```
Eigen::MatrixXd A(3, 2);
A << 1, 2,
2, 3,
3, 4;
Eigen::MatrixXd B = A.transpose();// the transpose of A is a 2x3 matrix
Eigen::MatrixXd C = (B * A).inverse();// computer the inverse of BA, which is a 2x2 matrix
```

Dot product and cross product:

```
Eigen::Vector3d v(1, 2, 3);
Eigen::Vector3d w(0, 1, 2);

double vDotw = v.dot(w); // dot product of two vectors
Eigen::Vector3d vCrossw = v.cross(w); // cross product of two vectors
```

Accessing matrix:

```
Eigen::MatrixXd A = Eigen::MatrixXd::Random(7, 9);
std::cout << "The element at fourth row and 7the column is " << A(3, 6) << std::endl;

Eigen::MatrixXd B = A.block(1, 2, 3, 3);
std::cout << "Take sub-matrix whose upper left corner is A(1, 2)" << std::endl << B << std::endl;

Eigen::VectorXd a = A.col(1); // take the second column of A
Eigen::VectorXd b = B.row(0); // take the first row of B

Eigen::VectorXd c = a.head(3);// take the first three elements of a
Eigen::VectorXd d = b.tail(2);// take the last two elements of b
```

Quaternion:

```
Eigen::Quaterniond q(2, 0, 1, -3);
std::cout << "This quaternion consists of a scalar " << q.w() << " and a vector " << std::endl << q.vec() << std::endl;

q.normalize();
std::cout << "To represent rotation, we need to normalize it such that its length is " << q.norm() << std::endl;

Eigen::Vector3d v(1, 2, -1);
Eigen::Quaterniond p;
p.w() = 0;
p.vec() = v;
Eigen::Quaterniond rotatedP = q * p * q.inverse();
Eigen::Vector3d rotatedV = rotatedP.vec();
```

```
  std::cout << "We can now use it to rotate a vector " << std::endl << v << " to " << std::endl << rotatedV << std::endl;

  Eigen::Matrix3d R = q.toRotationMatrix(); // convert a quaternion to a 3x3 rotation matrix
  std::cout << "Compare with the result using an rotation matrix " << std::endl << R * v << std::endl;

  Eigen::Quaterniond a = Eigen::Quterniond::Identity();
  Eigen::Quaterniond b = Eigen::Quterniond::Identity();
  Eigen::Quaterniond c; // Adding two quaternion as two 4x1 vectors is not supported by the Elgen API. That is, c = a + b
is not allowed. We have to do this in a hard way
  c.w() = a.w() + b.w();
  c.x() = a.x() + b.x();
  c.y() = a.y() + b.y();
  c.z() = a.z() + b.z();
```