

Disclaimer

This isn't your usual C++ talk! It contains

- Math
- Physics
- Mentions your math and physics teacher
- C++ Code



All of this is for the greater good!

- Comprehensive solution for physical units in matrices
- Even stronger type-safety in C++

SW developer at Bosch (since 2008)

- Focus on solving real-world problems using C++
- Object tracking framework for self-driving car projects
- Author and maintainer of *type_safe_matrix* library



Daniel Withopf

PHYSICAL UNITS FOR MATRICES:

HOW HARD CAN IT BE?

PHYSICAL UNITS FOR MATRICES:

HOW HARD ~~T~~* CAN IT BE?

*GEORGE W. HART, MULTIDIMENSIONAL ANALYSIS

Background

C++ in the automotive industry / at Bosch

- Development processes with lots of code reviews
- Code should be written for the reader
 - C++ types that express the content
 - Leverage C++'s type system to catch problems early
- (in-house) physical units library (for scalars) in use for over 15 years
- (in-house) linear algebra library for vector and matrix arithmetic

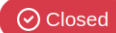

=> Missing: general solution for physical units in linear algebra types



Background

C++ open-source libraries and recent proposals for the standard library

- Linear algebra: Eigen, blaze and <http://wg21.link/P1385>
- Physical units: <https://github.com/mpusz/units> (<http://wg21.link/P1935>), boost units

handling of vector quantities and linear algebra libraries #23

 Closed  opened this issue on 6 Oct 2019 · 20 comments

  commented on 6 Oct 2019 ...

I'm curious to your thoughts on how this applies beyond scalar quantities. How might this be combined with a linear algebra library like eigen?

```
fs_vector<si::length<si::metre>, 3> v = { 1 * m, 2 * m, 3 * m };  
fs_vector<si::length<si::metre>, 3> u = { 3 * m, 2 * m, 1 * m };  
std::cout << "v + u = " << v + u << "\n"; // [4m, 4m, 4m]
```

=> Missing: general solution for physical units in linear algebra types

Further motivation

P1417: Historical lessons for C++ linear algebra library standardization

“Linear algebra libraries all must figure out what to do if users attempt to perform operations on objects with incompatible dimensions. [...] This gets more complicated, though, if we generalize “compatible dimensions” to the mathematical idea of a “vector space.” Two vectors might have the same dimensions, but it still might not make sense to add them together. For example, I can’t add a coordinate in 3-D Euclidean space to a quadratic polynomial with real coefficients, just like I can’t add meters to seconds.”
[Sec 3.6]

“Expression templates may hinder use of auto [...] because the type [...] may be some expression type that may hold references to concrete linear algebra objects. Returning the expression may result in dangling references” [Sec 3.1]

<http://wg21.link/P1417>

Non-expressive code examples

Can you tell what this line of code does?

```
measurement_vector[2] = other_vector[3];
```

- What do the vector entries describe?
- Is this an out-of-bounds access?

2nd try:

```
measurement_vector[VELOCITY_X] = other_vector[POSITION_X];
```

- Have the right index constants for the vector type been used?
- Is assigning a position to a velocity really intended?

3rd try:

```
measurement_vector[VELOCITY_X] = other_vector[VELOCITY_X];
```

- In which coordinate frame are measurement_vector and other_vector?

What we ideally want

My linear algebra library wishlist:

- Protection against out-of-bounds access (compile-time please!)
- Expressive (and enforced) names for vector / matrix entries
- Support of *non-uniform* physical units in vectors / matrices
- Compatibility check for physical units during all matrix operations (compile-time)
- Coordinate frame annotation for vectors and transformations

Named index structs

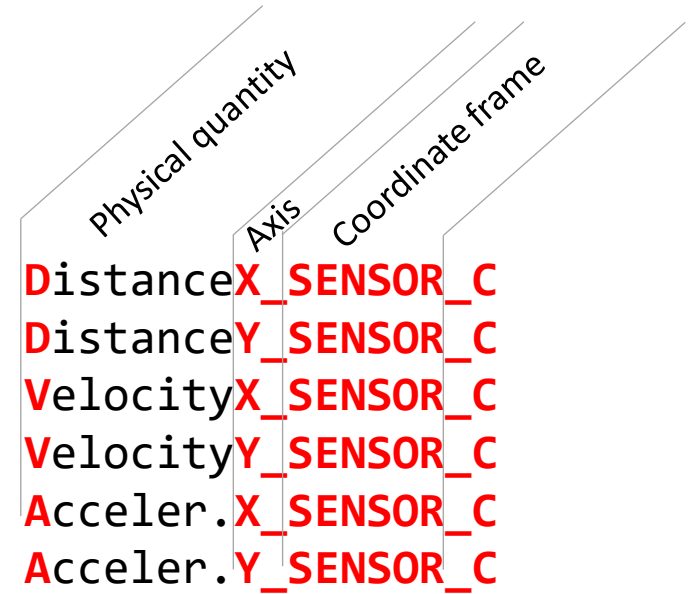
Identify each entry with a unique name: **DistanceX_C**

```
struct DX_C :  
  CartesianIdxType<VehicleFrontAxleCoords,  
    tsm::CartesianXAxis,  
    si::Metre>{};
```

coordinate frame
axis identifier
si unit template

```
struct DY_SENSOR_C :  
  CartesianIdxType<SensorCoords,  
    tsm::CartesianYAxis,  
    si::Metre>{};
```

```
struct VehicleFrontAxleCoords : public CoordinateSystem<si::Metre> {  
  using Moving = std::false_type; // is the frame moving wrt to the "fixed" earth frame?  
};
```



One type for almost everything

```
template<class Scalar, class RowIdxList, class ColIdxList, class MatrixTag>
class TypeSafeMatrix {
    ... // methods
private:
    Eigen::Matrix<Scalar, SizeOf<RowIdxList>::value, SizeOf<ColIdxList>::value> m_matrix;
};
```

```
template<class Scalar, class RowIdxList, class MatrixTag>
using TypeSafeVector = TypeSafeMatrix<Scalar,
                                     RowIdxList,
                                     tsm::TypeList<tsm::NoIdxType>,
                                     MatrixTag>;
```

```
template<class Scalar>
using PosVec3InVehicleFrame<Scalar> =
    tsm::TypeSafeVector<Scalar, tsm::TypeList<DX_C, DY_C, DZ_C>, tsm::VectorTag>;
```

Creating a vector and access to elements

```
PosVec3InVehicleFrame<double> pos_vehicle{  
    other_position.entry<DX_C>(), // copy 1 entry from other vector (***)  
    tsm::wrapCoeffSi<DY_C>(si::Metre<double>{1.}), // unit argument (**)  
    tsm::wrapCoeff<DZ_C>(3.); // plain scalar argument (*)
```

```
// full check, only the same index from the same vector type can be assigned  
pos_vehicle.assignEntry(other_position.entry<DX_C>()); // (***)
```

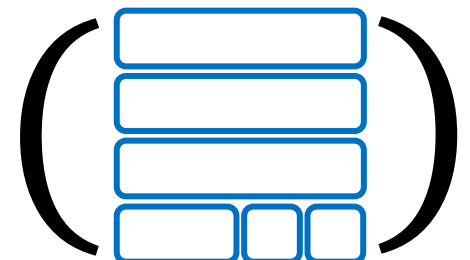
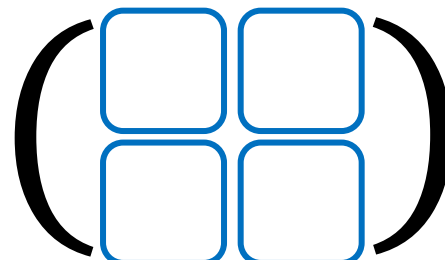
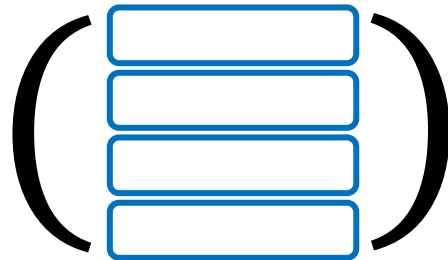
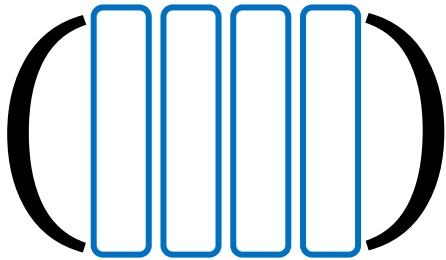
```
// unit check, allows assigning a y position from a different frame  
pos_sensor.coeffSiRef<DX_SENSOR_C>() = pos_vehicle.coeffSi<DY_C>(); // (**)
```

```
// no checks except for scalar type, allows assigning a velocity to a position  
pos_vehicle.at<DX_C>() = vel_sensor.at<VY_SENSOR_C>(); // (*)
```

More ways to create a vector / matrix

```
DeltaPosVec3InSensorFrame<double> delta_pos_sensor{  
    si::Second<double>{2.0} * other_vel_sensor.head<2>(), // other 2d vector in same frame  
    other_delta_pos.entry<DZ_SENSOR_C>()};
```

```
tsm::TypeSafeMatrix<double, tsm::TypeList<DX_C, VX_C>, tsm::TypeList<DX_C, VX_C>,  
    tsm::CovarianceMatrixTag> covariance{  
    tsm::wrapCoeffSi<DX_C, DX_C>{...}, tsm::wrapCoeffSi<DX_C, VX_C>{...},  
    tsm::wrapCoeffSi<VX_C, DX_C>{...}, tsm::wrapCoeffSi<VX_C, VX_C>{...}  
};
```



Matrix element access

Declare a type-safe (jacobian) matrix

```
tsm::TypeSafeMatrix<double,  
    tsm::TypeList<MEAS_DR, // radial dist[m]  
                 MEAS_VR, // radial vel[m/s]  
                 MEAS_ANGLE>, // angle [rad]  
    tsm::TypeList<DX_C, // distance x [m]  
                 DY_C, // distance y [m]  
                 DZ_C>, // distance z [m]  
    tsm::JacobianMatrixTag> jacobian{...};
```

Jacobian matrix	DX_C [m]	DY_C [m]	DZ_C [m]
MEAS_DR [m]			
MEAS_VR [m/s]			
MEAS_ANGLE [rad]			

Matrix element access

Which physical unit should be returned?

```
??? value = jacobian.coeffSi<MEAS_VR, DY_C>();
```

Jacobian matrix	DX_C [m]	DY_C [m]	DZ_C [m]
MEAS_DR [m]			
MEAS_VR [m/s]		???	
MEAS_ANGLE [rad]			

Column exponent = -1

Row exponent = 1

Jacobian matrix	DX_C [m]	DY_C [m]	DZ_C [m]
[m]			
[m/s]		[m/s]^row_exp * [m]^col_exp	
[rad]			

Taxonomy of vector and matrix types

Matrix / vector type	Row exponent	Column exponent	Nr of columns
Covariance matrix	1	1	
Jacobian matrix	1	-1	
Information matrix	-1	-1	
Position vector (VectorTag)	1	0	1
Position vector collection	1	0	>1
Displacement vector (DeltaVectorTag)	1	0	1
Displacement vector collection	1	0	>1
Information vector	-1	0	1
Information vector collection	-1	0	>1

*Now we have unit-safe
element access and out-of-bounds protection*

Can we do more???

Is unit-safety enough?

Let's try being unit-safe for all operations


Should this operation be allowed?

$$\begin{pmatrix} DX_C[m] \\ DY_C[m] \end{pmatrix} + \begin{pmatrix} VX_C[m/s] \\ VY_C[m/s] \end{pmatrix}$$

Should this be allowed?

$$\begin{pmatrix} [m] \\ [m] \end{pmatrix} + \Delta \begin{pmatrix} [m] \\ [m] \end{pmatrix}$$

Really?

 **Index types do not match!**

$$\begin{pmatrix} DX_C \\ DY_C \end{pmatrix} + \Delta \begin{pmatrix} DY_C \\ DZ_C \end{pmatrix}$$

Unit-safety is not enough,
we need index-type safety!

Matrix multiplication

What happens when e.g. transforming a covariance matrix to another frame?

```
cov_sensor = jacobian * cov_vehicle * jacobian.transpose();
```

$$\begin{array}{c}
 \mathbf{1} \\
 \mathbf{Jac} \\
 \begin{array}{c}
 DX_{SEN} \\
 DX_{SEN} \\
 VX_{SEN} \\
 VY_{SEN}
 \end{array}
 \end{array}
 \begin{array}{c|c}
 & \begin{array}{c}
 \mathbf{-1} \\
 DX_C \quad DY_C \quad VX_C \quad VY_C
 \end{array} \\
 \hline
 \begin{pmatrix}
 1.0 & 0 & 0 & 0 \\
 0 & 1.0 & 0 & 0 \\
 0 & 0 & 1.0 & 0 \\
 0 & 0 & 0 & 1.0
 \end{pmatrix}
 \end{array}
 \begin{array}{c}
 * \\
 \mathbf{1} \\
 \mathbf{1}
 \end{array}
 \begin{array}{c|c}
 & \begin{array}{c}
 \mathbf{1} \\
 DX_C \quad DY_C \quad VX_C \quad VY_C
 \end{array} \\
 \hline
 \begin{pmatrix}
 3.1 & 0 & 0 & 0 \\
 0 & 2.4 & 0 & 0 \\
 0 & 0 & 8.5 & 0 \\
 0 & 0 & 0 & 6.4
 \end{pmatrix}
 \end{array}
 \begin{array}{c}
 * \\
 \mathbf{-1} \\
 \mathbf{1}
 \end{array}
 \begin{array}{c|c}
 & \begin{array}{c}
 \mathbf{1} \\
 DX_{SEN} \quad DY_S \quad VX_S \quad VY_{SEN}
 \end{array} \\
 \hline
 \begin{pmatrix}
 1.0 & 0 & 0 & 0 \\
 0 & 1.0 & 0 & 0 \\
 0 & 0 & 1.0 & 0 \\
 0 & 0 & 0 & 1.0
 \end{pmatrix}
 \end{array}
 \end{array}$$

Important classes

Building blocks:

```
template<class Derived> class MatrixBase{};
```

```
template<class Promotion, // infers resulting TypeSafeMatrix  
        class LinalgExpression> // Eigen expression  
class MatrixExpression : public  
    MatrixBase<MatrixExpression<Promotion, LinalgExpression>>{};
```

```
template<class Scalar, class RowIdxList, class ColIdxList, class MatrixTag>  
class TypeSafeMatrix : public MatrixBase<  
    TypeSafeMatrix<ScalarT, RowIdxList, ColIdxList, MatrixTagT>>{};
```

What operator+ looks like

```
template<typename OtherLeaf_T>
auto operator+(const MatrixBase<OtherLeaf_T>& other) const
    -> detail::MatrixExpression<detail::Promotion2dAddition<Leaf_T, OtherLeaf_T>,
                               decltype(this->underlying() + other.underlying())> {...}
```

```
template<typename Expr1, typename Expr2>
struct Promotion2dAddition
: RequiresIdenticalScalarType<Expr1, Expr2>, // inject Scalar type
  RequiresIdenticalRowIndices<Expr1, Expr2>, // inject RowIdxList type
  RequiresIdenticalColIndices<Expr1, Expr2>, // inject ColIdxList type
  RequiresIdenticalRowUnitExponent<Expr1, Expr2>,
  RequiresIdenticalColUnitExponent<Expr1, Expr2>,
  RequiresMatrixTagsAdditionCompatible<typename Expr1::MatrixTag, // inject MatrixTag type
                                        typename Expr2::MatrixTag> {};
```

```
template<typename Expr1, typename Expr2>
struct RequiresIdenticalRowIndices {
    static_assert(detail::TypeIdentityChecker<typename Expr1::RowIdxList,
                                             typename Expr2::RowIdxList>::value,
                  "Row index types are not equal as required");
    using RowIdxList = typename Expr1::RowIdxList;
};
```

Compiler error messages

```
auto res = tsm::PosVector3InVehicleFrame{} + tsm::DeltaPosVector2InVehicleFrame{};
```

```
../type_safe_matrix/typelist_operations.h: In instantiation of 'class  
detail::TypeIdentityChecker<TypeList<DX_C, DY_C, DZ_C>, TypeList<DX_C, DY_C> >':  
../type_safe_matrix/promotion_precondition_checks.h:216:35:   required from 'class  
detail::RequiresIdenticalRowIndices<TypeSafeMatrix<double, TypeList<DX_C, DY_C, DZ_C>,  
TypeList<NoIdx>, VectorTag>, TypeSafeMatrix<double, TypeList<DX_C, DY_C>, TypeList<NoIdx>,  
DeltaVectorTag> >'  
../type_safe_matrix/typed_matrix_promotions.h:197:7:   required from 'class  
detail::Promotion2dAddition<TypeSafeMatrix<double, TypeList<DX_C, DY_C, DZ_C>,  
TypeList<NoIdx>, VectorTag>, TypeSafeMatrix<double, TypeList<DX_C, DY_C>, TypeList<NoIdx>,  
DeltaVectorTag> >'  
.....  
../type_safe_matrix/test/usage_examples.cpp:505:67:   required from here  
../type_safe_matrix/typelist_operations.h:66:3: error: static assertion failed: actual type  
(1st template arg of TypeIdentityChecker) does not match desired type (2nd arg);
```

Compiler error message with C++20 concepts

```
auto res = tsm::PosVector3InSensorFrame{} + tsm::DeltaPosVector2InSensorFrame{};
```

error: no match for 'operator+' (operand types are 'Vector3' {aka 'TypeSafeMatrix<double, TypeList<DX_C, DY_C, DZ_C>, TypeList<NoIdxType>, VectorTag>'} and 'Vector2' {aka 'TypeSafeMatrix<double, TypeList<DX_C, DY_C>, TypeList<NoIdxType>, VectorTag>'})

```
44 |     auto res = Vector3{} + Vector2{};
    |                  ^
    |                  TypeSafeMatrix<[...],TypeList<DX_C, DY_C>,[...],[...]>
    |                  TypeSafeMatrix<[...],TypeList<DX_C, DY_C, DZ_C>,[...],[...]>
```

```
33 |     TypeSafeMatrix operator+(const OtherT& other) requires Addable<TypeSafeMatrix,
    |     OtherT>
    |                   ^~~~~~
```

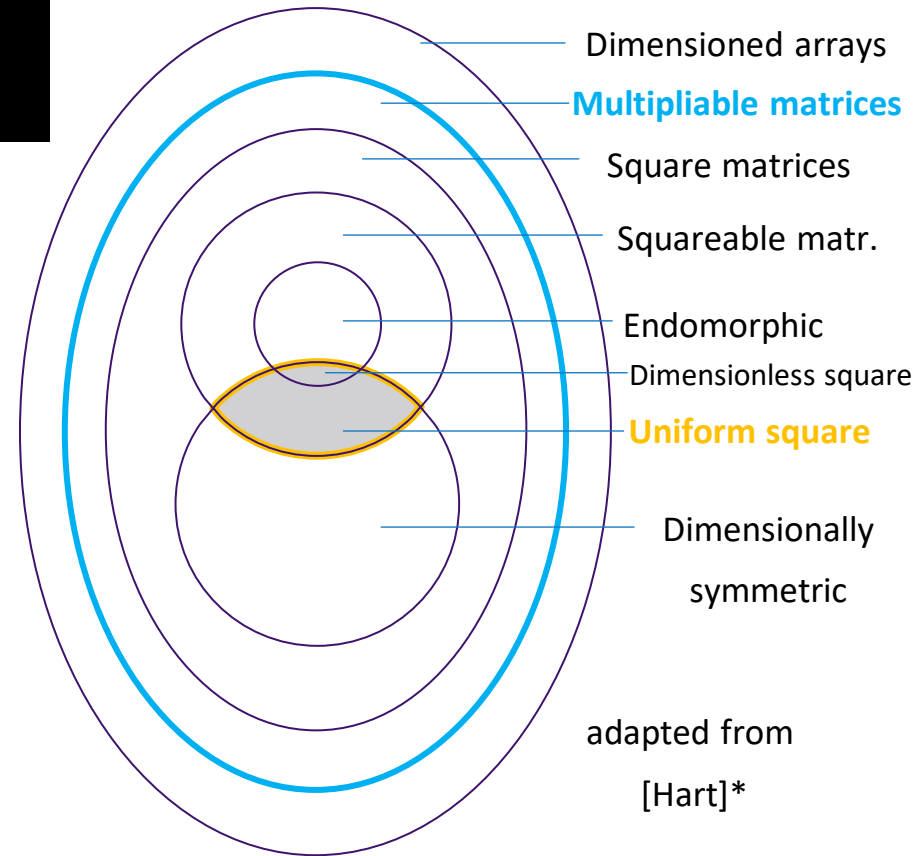
<source>:33:19: note: constraints not satisfied

required for the satisfaction of '**Addable**<TypeSafeMatrix<ScalarT, RowIdxListT, ColIdxListT, MatrixTagT>, OtherT>'

note: nested requirement '**is_same_v**<typename T1::RowIdxListType, typename T2::RowIdxListType>' is not satisfied

Uniform matrices vs. *Index-oriented design (TypeSafeMatrix)*

```
fs_vector<si::length<si::metre>, 3> v = {1*m, 2*m, 3*m};  
fs_vector<si::length<si::metre>, 3> u = {3*m, 2*m, 1*m};  
std::cout << "v + u = " << v + u << "\n"; // [4m,4m,4m]
```



*George W. Hart: *Multidimensional Analysis*, Springer

Uniform matrices vs. *Index-oriented design (TypeSafeMatrix)*

What can be added / multiplied?

Unit as value type

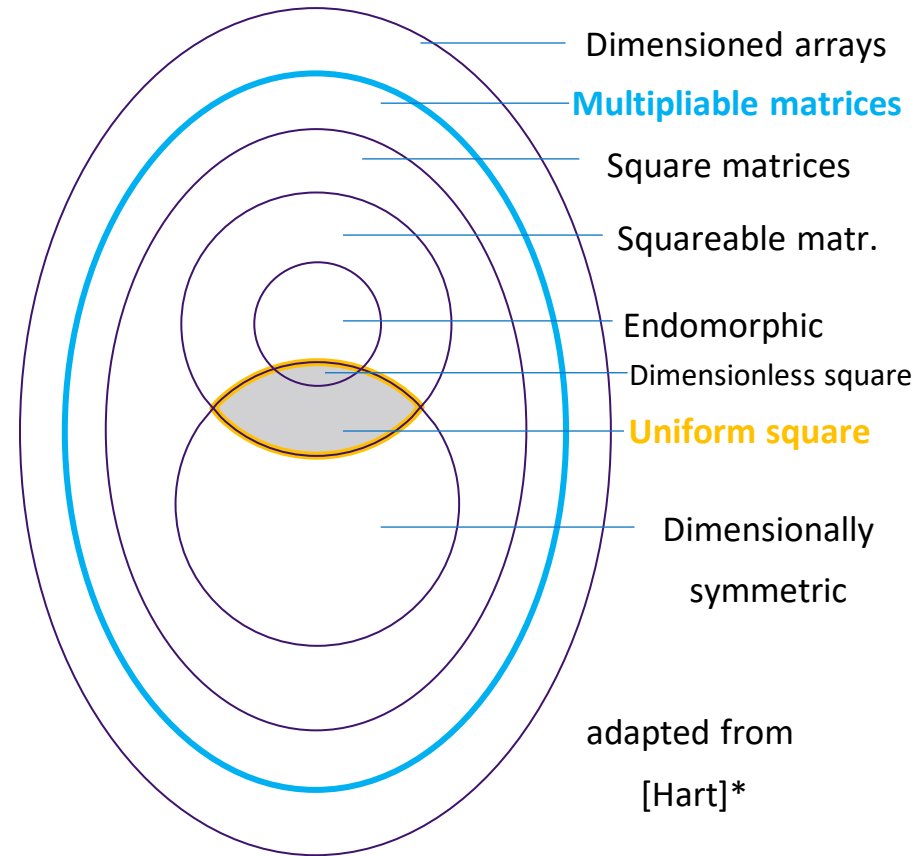
a + b	$\begin{pmatrix} DX \\ DY \end{pmatrix}$	$\begin{pmatrix} DY \\ DZ \end{pmatrix}$	$\Delta \begin{pmatrix} DX \\ DY \end{pmatrix}$	$\begin{pmatrix} VX \\ VY \end{pmatrix}$
$\begin{pmatrix} DX & DY \end{pmatrix}^T$	yes	yes	yes	no
$\begin{pmatrix} DY & DZ \end{pmatrix}^T$		yes	yes	no
$\Delta \begin{pmatrix} DX & DY \end{pmatrix}^T$			yes	no
$\begin{pmatrix} VX & VY \end{pmatrix}^T$				yes

A * B	jac	jac ^{AT}	cov	vector
jac	yes	yes	yes	yes
jac ^{AT}	yes	yes	yes	yes
cov	yes	yes	yes	yes
vector	yes	yes	yes	yes

TypeSafeMatrix

a + b	$\begin{pmatrix} DX \\ DY \end{pmatrix}$	$\begin{pmatrix} DY \\ DZ \end{pmatrix}$	$\Delta \begin{pmatrix} DX \\ DY \end{pmatrix}$	$\begin{pmatrix} VX \\ VY \end{pmatrix}$
$\begin{pmatrix} DX & DY \end{pmatrix}^T$	no	no	yes	no
$\begin{pmatrix} DY & DZ \end{pmatrix}^T$		no	no	no
$\Delta \begin{pmatrix} DX & DY \end{pmatrix}^T$			yes	no
$\begin{pmatrix} VX & VY \end{pmatrix}^T$				yes

A * B	jac	jac ^{AT}	cov	vector
jac	maybe	no	maybe	maybe
jac ^{AT}	no	maybe	no	no
cov	no	maybe	no	no
vector	no	no	no	no



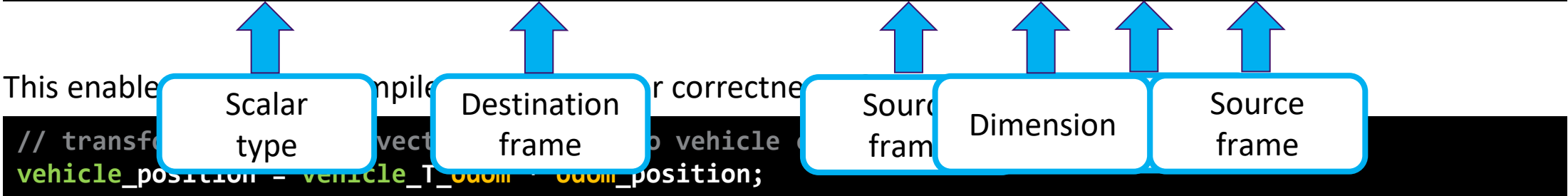
adapted from [Hart]*

*George W. Hart: *Multidimensional Analysis*, Springer

Transformation and rotation matrices

How to define an isometry (a transformation consisting of rotation and translation):

```
tsm::Isometry<double, tsm::VehicleFrontAxleCoords, tsm::OdomCoords, 3> vehicle_T_odom{...};
```



Valid or not?

```
vehicle_velocity = vehicle_T_odom * odom_velocity; Velocities should not be translated => Compile error
```

```
vehicle_velocity = vehicle_T_odom.linear() * odom_velocity;
```

```
vehicle_delta_vector = vehicle_T_odom.linear() * odom_delta_vector;
```

User-defined index structs

- Until now, we mainly used 2d- and 3d cartesian position and velocity vectors in different coordinate frames
- A TypeSafeMatrix is not restricted to this use-case, it can also be used to represent other quantities

```
struct SLOPE : tsm::NonCartesianIdxType<si::Metre> {}; // slope of a line fit
struct OFFSET : tsm::NonCartesianIdxType<si::Metre> {}; // offset of a line fit
```

- Operations that are only allowed for cartesian vectors:
 - Transformation to a different frame
 - norm(), cross(), dot()

Auto variables and expression templates

```
// method returning by value  
Eigen::Vector3d calculateOffset() {...}  
Eigen::Vector3d vectorA = Eigen::Vector3d::Ones();
```

```
double result = (vectorA + calculateOffset()).norm();
```

```
auto sum = vectorA + calculateOffset();  
double result = sum.norm();
```

```
const auto& sum = vectorA + calculateOffset();  
double result = sum.norm();
```

How to prevent usage of MatrixExpression

```
template<class T>
void staticAssertIfLvalueMatrixExpression() {
    static_assert(!IsTsmMatrixExpression<std::decay_t<T>> || std::is_rvalue_reference<T>::value);
}

template<class Leaf>
class MatrixBase {
    template<class Other>
    auto operator+(Other&& other) const& -> detail::MatrixExpression<...> {
        detail::staticAssertIfMatrixExpression<Leaf>();
        detail::staticAssertIfLvalueMatrixExpression<decltype(other)>();
        return {underlying() + std::forward<Other>(other).underlying()};
    }
    template<class Other>
    auto operator+(Other&& other) const&& -> detail::MatrixExpression<...> {
        detail::staticAssertIfLvalueMatrixExpression<decltype(other)>();
        return {std::move(*this).underlying() + std::forward<Other>(other).underlying()};
    }
};
```

Supported operations

<code>A.uncheckedConstMatrix()</code>	const access to underlying matrix (Eigen / blaze)
<code>A.uncheckedMutableMatrix()</code>	mutable access to underlying matrix (Eigen / blaze)
<code>A.asDeltaVector()</code>	convert the vector to a delta vector
<code>A.uncheckedAsNonDeltaVector()</code>	convert the vector to a non-delta vector
<code>A.narrowing_cast<Scalar>()</code>	convert (lossy) the matrix to use another scalar type (e.g. int32_t → int16_t)
<code>A.widening_cast<Scalar>()</code>	convert (lossless) the matrix to use another scalar type (e.g. int16_t → int32_t)
<code>A.coeffSi<ROW_IDX, COL_IDX>()</code>	const access to individual matrix element as si unit
<code>A.coeffSiRef<ROW_IDX, COL_IDX>()</code>	non-const access to individual matrix element as si unit
<code>A.at<ROW_IDX, COL_IDX>()</code>	const access to individual matrix element as scalar
<code>A.at<ROW_IDX, COL_IDX>()</code>	non-const access to individual matrix element as scalar
<code>A.row<ROW_IDX>()</code>	const access to a row in a matrix
<code>A.assignRow<ROW_IDX>(b)</code>	assign a row in a matrix
<code>A.col<COL_IDX>()</code>	access a column in a matrix
<code>A.assignCol<COL_IDX>(b)</code>	assign a column in a matrix
<code>A.block<RowTypeList, ColTypeList>()</code>	const access to a block in a matrix
<code>A.assignBlock<RowTypeList, ColTypeList>(B)</code>	assign a block in a matrix

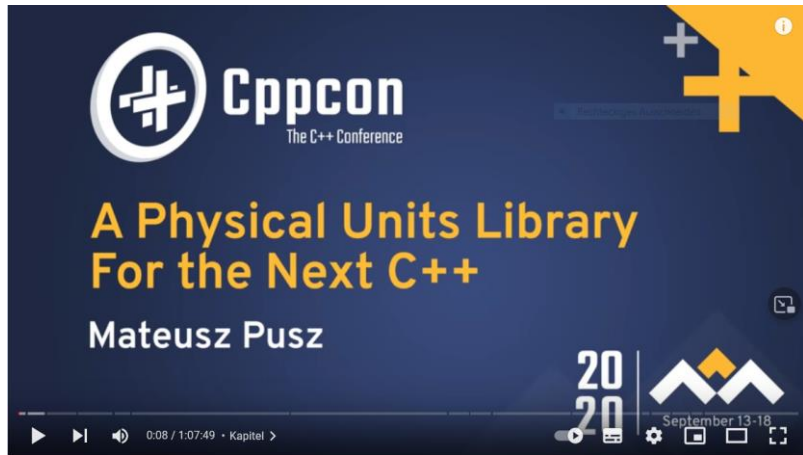
Supported operations

<code>A.transpose()</code>	return the transpose of the matrix
<code>A + - B</code>	Addition / subtraction of 2 matrices (have to be compatible)
<code>A * B</code>	multiplication of 2 matrices
<code>a.dot(b)</code>	scalar product of 2 cartesian delta-vectors
<code>A.determinant()</code>	determinant of a matrix (only available for dim < 4)
<code>a.norm()</code>	calculates the L2-norm (length) of a cartesian delta-vector
<code>a.squaredNorm()</code>	calculates the square L2-norm of a cartesian delta-vector
<code>a.cross(b)</code>	calculates the cross product of 2 cartesian delta-vectors
<code>a.head<n>()</code>	returns the first n elements of the vector
<code>A.inverse()</code>	Calculates matrix inverse (only for dim <= 4)
<code>a * si-unit; a / si-unit; a * scalar; a / scalar;</code>	divides a cartesian delta-vector by a si-unit / scalar
<code>A.setRowTo<ROW_IDX>(scalar)</code>	set a row to a value
<code>A.setColTo<COL_IDX>(scalar)</code>	set a column to a value
<code>A = MatrixType::Ones(); ::Zero(); Identity();</code>	Matrix expression where each entry is 1 / 0 / Identity matrix
<code>a = VectorType::Unit<ROW_IDX>();</code>	Set vector to the unit vector where only ROW_IDX is 1
<code>A.setIdentity(); A.setOnes(); A.setZero();</code>	set the matrix to identity / 1 / 0

What we get with TypeSafeMatrix

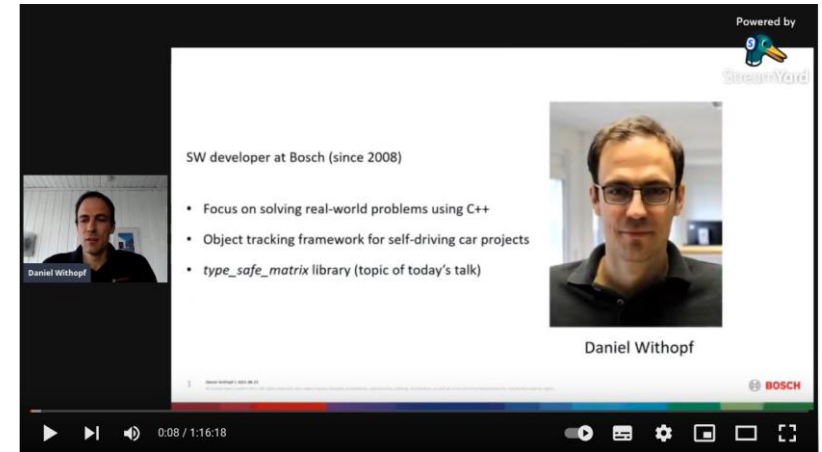
- Expressive (and enforced) names for vector / matrix entries
- Protection against out-of-bounds access (compile-time)
- Full support for physical units in vectors / matrices
- Compatibility check of index structs for all matrix operations (stronger condition than physical units-check)
- Include notion of coordinate frames
 - Possibility to specify a coordinate frame for a vector and source and dest coordinate frame for transformations
- Abstraction of underlying linalg library

Physical units library <http://wg21.link/P1385>



<https://www.youtube.com/watch?v=7dExYGSOJzo>

Earlier version of my talk, contains more details about object tracking



<https://www.youtube.com/watch?v=J6H9CwzynoQ>

We at Bosch are **hiring**

Join us to shape the future of automated driving

<https://lnkd.in/d9w6pBVh>

Example how Uber / Aurora use their units library



Thank you for listening, looking forward to your questions!