mav$_{t}rajectory_{g}eneration$

Nov 20, 2018

Table of Contents

1 Time Allocation **3**

1.1 Benchmark 3
1.2 Richter et al. 4
1.3 Burri et al. 4
1.4 Mellinger and Kumar 4
1.5 Segment violation 4

2 Library API **5**

2.1 Class Hierarchy 5
2.2 File Hierarchy 5
2.3 Full API 5
2.3.1 Namespaces 5
2.3.2 Classes and Structs 10
2.3.3 Enums 34
2.3.4 Functions 35
2.3.5 Variables 48
2.3.6 Typedefs 50
2.3.7 Directories 50
2.3.8 Files 53

3 Bibliography **145**

This repository contains tools for polynomial trajectory generation and optimization based on methods described in [1]. These techniques are especially suitable for rotary-wing micro aerial vehicles (MAVs). This README provides a brief overview of our trajectory generation utilities with some examples.

Authors: Markus Achtelik, Michael Burri, Helen Oleynikova, Rik Bähnemann, Marija Popović **Maintainer**: Rik Bähnemann, brik@ethz.ch **Affiliation**: Autonomous Systems Lab, ETH Zurich

CHAPTER 1

Time Allocation

Optimization for:

- Time only (*Burri*, *Richter*, *Mellinger* and *Segment violation*)
- Time and free constraints (*Burri* and *Richter*)

Methods implemented:

1. C. Richter, A. Bry, and N. Roy, "**Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,**" in *International Journal of Robotics Research*, Springer, 2016.
2. M. Burri, H. Oleynikova, M. Achtelik, and R. Siegwart, "**Real-Time Visual-Inertial Mapping, Re-localization and Planning Onboard MAVs in Previously Unknown Environments**". In *IEEE Int. Conf. on Intelligent Robots and Systems* (IROS), September 2015.
3. D. Mellinger and V. Kumar, "**Minimum Snap Trajectory Generation and Control for Quadrotors**"
4. Segment violation

1.1 Benchmark

- trajectory time
- computation time
- relative violation of velocity
- maximum distance between trajectory and straight line
- area between trajectory and straight line

Additionally:

- comparison of convergence time and quality of default and custom initial step
- comparison magic fabian vs. trapezoidal for initial time segments

1.2 Richter et al.

Paper: **Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments**Published in: *International Journal of Robotics Research*, SpringerYear: 2016

Usable for optimization of:

- Time only (`NonlinearOptimizationParameters::kRichterTime`):
- Time and free derivatives (`NonlinearOptimizationParameters::kRichterTimeAndConstraints`):

1.3 Burri et al.

Paper: **Real-Time Visual-Inertial Mapping, Re-localization and Planning Onboard MAVs in Previously Unknown Environments**Published in: *IEEE Int. Conf. on Intelligent Robots and Systems* (IROS)Year: 2015

Usable for optimization of:

- Time only (`NonlinearOptimizationParameters::kSquaredTime`)
- Time and free derivatives (`NonlinearOptimizationParameters::kSquaredTimeAndConstraints`):

1.4 Mellinger and Kumar

Paper: **Minimum Snap Trajectory Generation and Control for Quadrotors**Published in: *IEEE International Conference on Robotics and Automation* (ICRA)Year: 2011

Usable for optimization of:

- Time only (`NonlinearOptimizationParameters::kMellingerOuterLoop`):

1.5 Segment violation

Usable for optimization of:

- Time only:

CHAPTER 2

Library API

2.1 Class Hierarchy

2.2 File Hierarchy

2.3 Full API

2.3.1 Namespaces

Namespace mav_trajectory_generation

Contents

- *Namespaces*
- *Classes*
- *Enums*
- *Functions*
- *Variables*

Namespaces

- *Namespace mav_trajectory_generation::derivative_order*
- *Namespace mav_trajectory_generation::timing*

Classes

- *Template Struct ConvolutionDimension*
- *Struct Extremum*
- *Struct NonlinearOptimizationParameters*
- *Struct OptimizationInfo*
- *Struct PolynomialOptimizationNonLinear::ConstraintData*
- *Class FeasibilityAnalytic*
- *Class FeasibilityAnalytic::Settings*
- *Class FeasibilityBase*
- *Class FeasibilityRecursive*
- *Class FeasibilityRecursive::Settings*
- *Class FeasibilitySampling*
- *Class FeasibilitySampling::Settings*
- *Class HalfPlane*
- *Class InputConstraints*
- *Class Polynomial*
- *Template Class PolynomialOptimizationNonLinear*
- *Class Trajectory*
- *Class Vertex*

Enums

- *Enum InputConstraintType*
- *Enum InputFeasibilityResult*

Functions

- *Template Function mav_trajectory_generation::checkMatrices*
- *Function mav_trajectory_generation::computeBaseCoefficients*
- *Function mav_trajectory_generation::computeCostNumeric*
- *Function mav_trajectory_generation::computeTimeVelocityRamp*
- *Template Function mav_trajectory_generation::convolve*
- *Function mav_trajectory_generation::createRandomDouble*
- *Function mav_trajectory_generation::createRandomVertices*
- *Function mav_trajectory_generation::createRandomVertices1D*
- *Function mav_trajectory_generation::createSquareVertices*
- *Function mav_trajectory_generation::drawMavSampledTrajectory*

- *Function mav_trajectory_generation::drawMavSampledTrajectoryWithMavMarker*
- *Function mav_trajectory_generation::drawMavTrajectory*
- *Function mav_trajectory_generation::drawMavTrajectoryWithMavMarker*
- *Function mav_trajectory_generation::drawVertices*
- *Function mav_trajectory_generation::estimateSegmentTimes*
- *Function mav_trajectory_generation::estimateSegmentTimesNfabian*
- *Function mav_trajectory_generation::estimateSegmentTimesVelocityRamp*
- *Function mav_trajectory_generation::findLastNonZeroCoeff*
- *Function mav_trajectory_generation::findRootsJenkinsTraub*
- *Function mav_trajectory_generation::getHighestDerivativeFromN*
- *Function mav_trajectory_generation::getInputFeasibilityResultName*
- *Function mav_trajectory_generation::getMaximumMagnitude*
- *Function mav_trajectory_generation::operator<<(std::ostream&, const Vertex&)*
- *Function mav_trajectory_generation::operator<<(std::ostream&, const Extremum&)*
- *Function mav_trajectory_generation::operator<<(std::ostream&, const OptimizationInfo&)*
- *Function mav_trajectory_generation::operator<<(std::ostream&, const std::vector<Vertex>&)*
- *Function mav_trajectory_generation::orientationDerivativeToInt*
- *Function mav_trajectory_generation::orintationDerivativeToString*
- *Function mav_trajectory_generation::polynomialTrajectoryMsgToTrajectory*
- *Function mav_trajectory_generation::positionDerivativeToInt*
- *Function mav_trajectory_generation::positionDerivativeToString*
- *Function mav_trajectory_generation::sampledTrajectoryStatesToFile*
- *Template Function mav_trajectory_generation::sampleFlatStateAtTime*
- *Function mav_trajectory_generation::sampleSegmentAtTime*
- *Function mav_trajectory_generation::sampleTrajectoryAtTime*
- *Function mav_trajectory_generation::sampleTrajectoryInRange*
- *Function mav_trajectory_generation::sampleTrajectoryStartDuration*
- *Function mav_trajectory_generation::sampleWholeTrajectory*
- *Function mav_trajectory_generation::segmentsFromFile*
- *Function mav_trajectory_generation::segmentsToFile*
- *Template Function mav_trajectory_generation::sgn*
- *Function mav_trajectory_generation::trajectoryFromFile*
- *Function mav_trajectory_generation::trajectoryToFile*
- *Function mav_trajectory_generation::trajectoryToPolynomialTrajectoryMsg*

Variables

- *Variable mav_trajectory_generation::kOptimizationTimeLowerBound*

Namespace mav_trajectory_generation::derivative_order

Contents

- *Variables*

Variables

- *Variable mav_trajectory_generation::derivative_order::ACCELERATION*
- *Variable mav_trajectory_generation::derivative_order::ANGULAR_ACCELERATION*
- *Variable mav_trajectory_generation::derivative_order::ANGULAR_VELOCITY*
- *Variable mav_trajectory_generation::derivative_order::INVALID*
- *Variable mav_trajectory_generation::derivative_order::JERK*
- *Variable mav_trajectory_generation::derivative_order::ORIENTATION*
- *Variable mav_trajectory_generation::derivative_order::POSITION*
- *Variable mav_trajectory_generation::derivative_order::SNAP*
- *Variable mav_trajectory_generation::derivative_order::VELOCITY*

Namespace mav_trajectory_generation::timing

Contents

- *Classes*
- *Typedefs*

Classes

- *Struct TimerMapValue*
- *Template Class Accumulator*
- *Class DummyTimer*
- *Class MiniTimer*
- *Class Timer*
- *Class Timing*

Typedefs

- *Typedef mav_trajectory_generation::timing::DebugTimer*

Namespace mav_visualization

Contents

- *Classes*
- *Functions*
- *Typedefs*

Classes

- *Class Color*
- *Class HexacopterMarker*
- *Class LeicaMarker*
- *Class MarkerGroup*

Functions

- *Function mav_visualization::createPoint*
- *Function mav_visualization::drawArrowPoints*
- *Function mav_visualization::drawArrowPositionOrientation*
- *Function mav_visualization::drawAxes*
- *Function mav_visualization::drawAxesArrows*
- *Function mav_visualization::drawCovariance3D*

Typedefs

- *Typedef mav_visualization::MarkerVector*

Namespace nlopt

Contents

- *Functions*

Functions

- *Function nlopt::returnValueToString*

2.3.2 Classes and Structs

Template Struct ConvolutionDimension

- Defined in *File convolution.h*

Struct Documentation

template <int *D*, int *K*>
struct ConvolutionDimension

Public Types

enum [anonymous]
Values:

length = D + K - 1

Struct Extremum

- Defined in *File extremum.h*

Struct Documentation

struct Extremum
Container holding the properties of an extremum (time, value, segment where it occurred).

Public Functions

Extremum()

Extremum(double *_time*, double *_value*, int *_segment_idx*)

bool **operator<**(**const** *Extremum* &*rhs*) **const**

bool **operator>**(**const** *Extremum* &*rhs*) **const**

Public Members

double **time**
Time where the extremum occurs, relative to the segment start.

double **value**
Value of the extremum at time.

int **`segment_idx`**
Index of the segment where the extremum occurs.

Struct NonlinearOptimizationParameters

- Defined in *File polynomial_optimization_nonlinear.h*

Struct Documentation

`struct NonlinearOptimizationParameters`
Class holding all important parameters for nonlinear optimization.

Public Types

`enum TimeAllocMethod`
Values:

`kSquaredTime`

`kRichterTime`

`kMellingerOuterLoop`

`kSquaredTimeAndConstraints`

`kRichterTimeAndConstraints`

`kUnknown`

Public Members

double **`f_abs`** = -1
Default parameters should be reasonable enough to use without further fine-tuning.

Stopping criteria, if objective function changes less than absolute value. Disabled if negative.

double **`f_rel`** = 0.05
Stopping criteria, if objective function changes less than relative value.

Disabled if negative.

double **`x_rel`** = -1
Stopping criteria, if state changes less than relative value.

Disabled if negative.

double **`x_abs`** = -1
Stopping criteria, if state changes less than absolute value.

Disabled if negative.

double **`initial_stepsize_rel`** = 0.1
Determines a fraction of the initial guess as initial step size.

Heuristic value if negative.

double **`equality_constraint_tolerance`** = 1.0e-3
Absolute tolerance, within an equality constraint is considered as met.

double **`inequality_constraint_tolerance`** = 0.1
Absolute tolerance, within an inequality constraint is considered as met.

int **`max_iterations`** = 3000
Maximum number of iterations. Disabled if negative.

double **`time_penalty`** = 500.0
Penalty for the segment time.

nlopt::algorithm **`algorithm`** = nlopt::LN_BOBYQA
Optimization algorithm used by nlopt, see http:///ab-initio.mit.edu/wiki/index.php/NLopt_Algorithms Previous value was nlopt::LN_SBPLX, but we found that BOBYQA has slightly better convergence and lower run-time in the unit tests.

int **`random_seed`** = 0
Random seed, if an optimization algorithm involving random numbers is used (e.g.

nlopt::GN_ISRES). If set to a value < 0, a "random" (getTimeofday) value for the seed is chosen.

bool **`use_soft_constraints`** = true
Decide whether to use soft constraints.

double **`soft_constraint_weight`** = 100.0
Weights the relative violation of a soft constraint.

mav_trajectory_generation::*NonlinearOptimizationParameters*::*TimeAllocMethod* **`time_alloc_method`** = *kSquaredTimeAnd*

bool **`print_debug_info`** = false

bool **`print_debug_info_time_allocation`** = false

Struct OptimizationInfo

- Defined in *File polynomial_optimization_nonlinear.h*

Struct Documentation

`struct OptimizationInfo`

Public Members

int **`n_iterations`** = 0

int **`stopping_reason`** = nlopt::FAILURE

double **`cost_trajectory`** = 0.0

double **`cost_time`** = 0.0

double **`cost_soft_constraints`** = 0.0

double **`optimization_time`** = 0.0

std::map<int, *Extremum*> **`maxima`**

Struct PolynomialOptimizationNonLinear::ConstraintData

- Defined in *File polynomial_optimization_nonlinear.h*

Nested Relationships

This struct is a nested type of *Template Class PolynomialOptimizationNonLinear*.

Struct Documentation

`struct ConstraintData`
Holds the data for constraint evaluation, since these methods are static.

Public Members

template<>
PolynomialOptimizationNonLinear<N> ***`this_object`**

template<>
int **`derivative`**

template<>
double **`value`**

Struct TimerMapValue

- Defined in *File timing.h*

Struct Documentation

`struct TimerMapValue`

Public Functions

`TimerMapValue``()`

Public Members

Accumulator<double, double, 50> **`acc_`**
Create an accumulator with specified window size.

Class FeasibilityAnalytic

- Defined in *File feasibility_analytic.h*

Nested Relationships

Nested Types

- *Class FeasibilityAnalytic::Settings*

Inheritance Relationships

Base Type

- `public mav_trajectory_generation::FeasibilityBase` (*Class FeasibilityBase*)

Class Documentation

`class FeasibilityAnalytic` : **`public`** mav_trajectory_generation::*FeasibilityBase*

Analytic input feasibility checks.

Public Types

`typedef` std::vector<Eigen::VectorXcd, Eigen::aligned_allocator<Eigen::VectorXcd>> **`Roots`**

Public Functions

`FeasibilityAnalytic``()`

`FeasibilityAnalytic``(`**`const`** *Settings* &*settings*`)`

`FeasibilityAnalytic``(`**`const`** *InputConstraints* &*input_constraints*`)`

`FeasibilityAnalytic``(`**`const`** *Settings* &*settings*, **`const`** *InputConstraints* &*input_constraints*`)`

`virtual` *InputFeasibilityResult* **`checkInputFeasibility`**`(`**`const`** Segment &*segment*`)` **`const`**

Checks a segment for input feasibility.

Public Members

Settings **`settings_`**

The user settings.

`class Settings`

Public Functions

`Settings``()`

void **`setMinSectionTimeS`**`(`double *min_section_time_s*`)`

double **`getMinSectionTimeS`**`()` **`const`**

Class FeasibilityAnalytic::Settings

- Defined in *File feasibility_analytic.h*

Nested Relationships

This class is a nested type of *Class FeasibilityAnalytic*.

Class Documentation

class Settings

Public Functions

Settings()

void **setMinSectionTimeS**(double *min_section_time_s*)

double **getMinSectionTimeS**() **const**

Class FeasibilityBase

- Defined in *File feasibility_base.h*

Inheritance Relationships

Derived Types

- `public mav_trajectory_generation::FeasibilityAnalytic` (*Class FeasibilityAnalytic*)
- `public mav_trajectory_generation::FeasibilityRecursive` (*Class FeasibilityRecursive*)
- `public mav_trajectory_generation::FeasibilitySampling` (*Class FeasibilitySampling*)

Class Documentation

class FeasibilityBase

A base class for different implementations for dynamic and position feasibility checks.

Subclassed by *mav_trajectory_generation::FeasibilityAnalytic*, *mav_trajectory_generation::FeasibilityRecursive*, *mav_trajectory_generation::FeasibilitySampling*

Public Functions

FeasibilityBase()

Default input constraints, no half plane constraints.

FeasibilityBase(**const** *InputConstraints* &*input_constraints*)

User input constraints, no half plane constraints.

InputFeasibilityResult **checkInputFeasibilityTrajectory**(**const** *Trajectory* &*trajectory*) **const**

Checks a trajectory for input feasibility.

virtual *InputFeasibilityResult* **checkInputFeasibility**(**const** Segment &*segment*) **const**
Checks a segment for input feasibility.

InputConstraints **getInputConstraints**() **const**

bool **checkHalfPlaneFeasibility**(**const** *Trajectory* &*trajectory*) **const**
Checks if a trajectory stays within a set of half planes.

bool **checkHalfPlaneFeasibility**(**const** Segment &*segment*) **const**
Checks if a segment stays within a set of half planes.

This check computes the extrema for each axis and checks whether these lie in the positive half space as in https://github.com/markwmuller/RapidQuadrocopterTrajectories/blob/master/C%2B%2B/RapidTrajectoryGenerator.cpp#L149

Public Members

InputConstraints **input_constraints_**
Input constraints.

HalfPlane::*Vector* **half_plane_constraints_**
Half plane constraints, e.g., the ground plane or a box.

Eigen::Vector3d **gravity_**

Class FeasibilityRecursive

- Defined in *File feasibility_recursive.h*

Nested Relationships

Nested Types

- *Class FeasibilityRecursive::Settings*

Inheritance Relationships

Base Type

- `public mav_trajectory_generation::FeasibilityBase` (*Class FeasibilityBase*)

Class Documentation

class FeasibilityRecursive : **public** mav_trajectory_generation::*FeasibilityBase*
Recursive input feasibility checks.

This implementation is based on [1] and extended to test yaw rates and higher order polynomials. The general idea is to check a segment for lower and upper bounds that can be found easily by evaluating the single axis minima and maxima. We extend checking for maximum velocity constraints and yaw rate and acceleration constraints.

[1] Mueller, Mark W., Markus Hehn, and Raffaello D'Andrea. "A Computationally Efficient Motion Primitive for Quadrocopter *Trajectory* Generation." Robotics, IEEE Transactions on 31.6 (2015): 1294-1310.

See also https://github.com/markwmuller/RapidQuadrocopterTrajectories

Public Types

typedef std::vector<Eigen::VectorXcd, Eigen::aligned_allocator<Eigen::VectorXcd>> **Roots**

Public Functions

FeasibilityRecursive()

FeasibilityRecursive(**const** *Settings* &*settings*)

FeasibilityRecursive(**const** *InputConstraints* &*input_constraints*)

FeasibilityRecursive(**const** *Settings* &*settings*, **const** *InputConstraints* &*input_constraints*)

virtual *InputFeasibilityResult* **checkInputFeasibility**(**const** Segment &*segment*) **const**

Checks a segment for input feasibility.

Public Members

Settings **settings_**

The user settings.

class Settings

Public Functions

Settings()

void **setMinSectionTimeS**(double *min_section_time_s*)

double **getMinSectionTimeS**() **const**

Class FeasibilityRecursive::Settings

- Defined in *File feasibility_recursive.h*

Nested Relationships

This class is a nested type of *Class FeasibilityRecursive*.

Class Documentation

class Settings

Public Functions

Settings()

void **setMinSectionTimeS**(double *min_section_time_s*)

double **getMinSectionTimeS**() **const**

Class FeasibilitySampling

- Defined in *File feasibility_sampling.h*

Nested Relationships

Nested Types

- *Class FeasibilitySampling::Settings*

Inheritance Relationships

Base Type

- `public mav_trajectory_generation::FeasibilityBase` (*Class FeasibilityBase*)

Class Documentation

class FeasibilitySampling : **public** mav_trajectory_generation::*FeasibilityBase*
Sampling based input feasibility checks.

Public Functions

FeasibilitySampling()

FeasibilitySampling(**const** *Settings* &*settings*)

FeasibilitySampling(**const** *InputConstraints* &*input_constraints*)

FeasibilitySampling(**const** *Settings* &*settings*, **const** *InputConstraints* &*input_constraints*)

virtual *InputFeasibilityResult* **checkInputFeasibility**(**const** Segment &*segment*) **const**
Checks a segment for input feasibility.

Public Members

Settings **settings_**
The user settings.

class Settings

Public Functions

`Settings`()

void **`setSamplingIntervalS`**(double *sampling_interval_s*)

double **`getSamplingIntervalS`**() **`const`**

Class FeasibilitySampling::Settings

- Defined in *File feasibility_sampling.h*

Nested Relationships

This class is a nested type of *Class FeasibilitySampling*.

Class Documentation

`class Settings`

Public Functions

`Settings`()

void **`setSamplingIntervalS`**(double *sampling_interval_s*)

double **`getSamplingIntervalS`**() **`const`**

Class HalfPlane

- Defined in *File feasibility_base.h*

Class Documentation

`class HalfPlane`

A half plane is defined through a point and a normal.

Public Types

`typedef` std::vector<*HalfPlane*, Eigen::aligned_allocator<*HalfPlane*>> **`Vector`**

Public Functions

`HalfPlane`(**`const`** Eigen::Vector3d &*point*, **`const`** Eigen::Vector3d &*normal*)

`HalfPlane`(**`const`** Eigen::Vector3d &*a*, **`const`** Eigen::Vector3d &*b*, **`const`** Eigen::Vector3d &*c*)

Define the half plane from 3 counter-clockwise points (seen from above).

Public Members

Eigen::Vector3d **point**

Eigen::Vector3d **normal**

Public Static Functions

static *HalfPlane*::*Vector* **createBoundingBox** (**const** Eigen::Vector3d &*point*, **const** Eigen::Vector3d &*bounding_box_size*)

Create 6 half planes that form a box with inward-facing normals.

point = center of the box, bounding_box_size = x, y, z edge length.

Class InputConstraints

- Defined in *File input_constraints.h*

Class Documentation

class InputConstraints

Dynamic constraints of the MAV.

Public Functions

InputConstraints ()

Empty constraints object.

void **addConstraint** (int *constraint_type*, double *value*)

Set a constraint given type and value.

void **setDefaultValues** ()

Sets all constraints to reasonable default values.

bool **getConstraint** (int *constraint_type*, double **value*) **const**

Return a constraint. Returns false if constraint is not set.

bool **hasConstraint** (int *constraint_type*) **const**

Check if a specific constraint type is set.

bool **removeConstraint** (int *constraint_type*)

Remove a specific constraint type.

Returns false if constraint was not set.

Class Polynomial

- Defined in *File polynomial.h*

Class Documentation

`class Polynomial`

Implementation of polynomials of order N-1.

Order must be known at compile time. *Polynomial* coefficients are stored with increasing powers, i.e. c_0 + c_1*t … c_{N-1} * t^{N-1} where N = number of coefficients of the polynomial.

Public Types

`typedef` std::vector<*Polynomial*> **`Vector`**

Public Functions

`Polynomial`(int *N*)

`Polynomial`(int *N*, **`const`** Eigen::VectorXd &*coeffs*)

Assigns arbitrary coefficients to a polynomial.

`Polynomial`(**`const`** Eigen::VectorXd &*coeffs*)

int **`N`**() **`const`**

bool **`operator==`**(**`const`** *Polynomial* &*rhs*) **`const`**

bool **`operator!=`**(**`const`** *Polynomial* &*rhs*) **`const`**

Polynomial **`operator+`**(**`const`** *Polynomial* &*rhs*) **`const`**

Polynomial &**`operator+=`**(**`const`** *Polynomial* &*rhs*)

Polynomial **`operator*`**(**`const`** *Polynomial* &*rhs*) **`const`**

The product of two polynomials is the convolution of their coefficients.

Polynomial **`operator*`**(**`const`** double &*rhs*) **`const`**

The product of a polynomial with a scalar.

Note that polynomials are in general not homogeneous, i.e., f(a*t) != a*f(t)

void **`setCoefficients`**(**`const`** Eigen::VectorXd &*coeffs*)

Sets up the internal representation from coefficients.

Coefficients are stored in increasing order with the power of t, i.e. c1 + c2*t + c3*t^2 ==> coeffs = [c1 c2 c3]

Eigen::VectorXd **`getCoefficients`**(int *derivative* = 0) **`const`**

Returns the coefficients for the specified derivative of the polynomial as a ROW vector.

void **`evaluate`**(double *t*, Eigen::VectorXd **result*) **`const`**

Evaluates the polynomial at time t and writes the result.

Fills in all derivatives up to result.size()-1 (that is, if result is a 3-vector, then will fill in derivatives 0, 1, and 2).

double **`evaluate`**(double *t*, int *derivative*) **`const`**

Evaluates the specified derivative of the polynomial at time t and returns the result (only one value).

bool **getRoots** (int *derivative*, Eigen::VectorXcd **roots*) **const**
Uses Jenkins-Traub to get all the roots of the polynomial at a certain derivative.

bool **computeMinMaxCandidates** (double *t_start*, double *t_end*, int *derivative*, std::vector<double> **candidates*) **const**
Finds all candidates for the minimum and maximum between t_start and t_end by computing the roots of the derivative polynomial.

bool **selectMinMaxFromRoots** (double *t_start*, double *t_end*, int *derivative*, **const** Eigen::VectorXcd &*roots_derivative_of_derivative*, std::pair<double, double> **minimum*, std::pair<double, double> **maximum*) **const**
Evaluates the minimum and maximum of a polynomial between time t_start and t_end given the roots of the derivative.

Returns the minimum and maximum as pair<t, value>.

bool **computeMinMax** (double *t_start*, double *t_end*, int *derivative*, std::pair<double, double> **minimum*, std::pair<double, double> **maximum*) **const**
Computes the minimum and maximum of a polynomial between time t_start and t_end by computing the roots of the derivative polynomial.

Returns the minimum and maximum as pair<t, value>.

bool **selectMinMaxFromCandidates** (**const** std::vector<double> &*candidates*, int *derivative*, std::pair<double, double> **minimum*, std::pair<double, double> **maximum*) **const**
Selects the minimum and maximum of a polynomial among a candidate set.

Returns the minimum and maximum as pair<t, value>.

bool **getPolynomialWithAppendedCoefficients** (int *new_N*, Polynomial **new_polynomial*) **const**
Increase the number of coefficients of this polynomial up to the specified degree by appending zeros.

void **scalePolynomialInTime** (double *scaling_factor*)
Scales the polynomial in time with a scaling factor.

To stretch out by a factor of 10, pass scaling_factor (b) = 1/10. To shrink by a factor of 10, pass scalign factor = 10. p_out = a12*b^12*t^12

- a11*b^11*t^11… etc.

Public Static Functions

static bool **selectMinMaxCandidatesFromRoots** (double *t_start*, double *t_end*, **const** Eigen::VectorXcd &*roots_derivative_of_derivative*, std::vector<double> **candidates*)
Finds all candidates for the minimum and maximum between t_start and t_end by evaluating the roots of the polynomial's derivative.

static void **baseCoeffsWithTime** (int *N*, int *derivative*, double *t*, Eigen::VectorXd **coeffs*)
Computes the base coefficients with the according powers of t, as e.g.

needed for computation of (in)equality constraints. Output: coeffs = vector to write the coefficients to Input: polynomial derivative for which the coefficients have to be computed Input: t = time of evaluation

static Eigen::VectorXd **baseCoeffsWithTime** (int *N*, int *derivative*, double *t*)
: Convenience method to compute the base coefficents with time static void baseCoeffsWithTime(const Eigen::MatrixBase<Derived> & coeffs, int derivative, double t)

static Eigen::VectorXd **convolve** (**const** Eigen::VectorXd &*data*, **const** Eigen::VectorXd &*kernel*)
: Discrete convolution of two vectors.

  convolve(d, k)[m] = sum(d[m - n] * k[n])

static int **getConvolutionLength** (int *data_size*, int *kernel_size*)

Public Static Attributes

constexpr int **kMaxN** = 12
: Maximum degree of a polynomial for which the static derivative (basis coefficient) matrix should be evaluated for.

  kMaxN = max. number of coefficients.

constexpr int **kMaxConvolutionSize** = 2 * *kMaxN* - 2
: kMaxConvolutionSize = max.

  convolution size for N = 12, convolved with its derivative.

Eigen::MatrixXd **base_coefficients_**
: One static shared across all members of the class, computed up to order kMaxConvolutionSize.

Template Class PolynomialOptimizationNonLinear

- Defined in *File polynomial_optimization_nonlinear.h*

Nested Relationships

Nested Types

- *Struct PolynomialOptimizationNonLinear::ConstraintData*

Class Documentation

template <int *_N* = 10>
class PolynomialOptimizationNonLinear
: Implements a nonlinear optimization of the unconstrained optimization of paths consisting of polynomial segments as described in [1] [1]: *Polynomial Trajectory* Planning for Aggressive Quadrotor Flight in Dense Indoor Environments.

  Charles Richter, Adam Bry, and Nicholas Roy. In ISRR 2013 _N specifies the number of coefficients for the underlying polynomials.

Public Types

enum [anonymous]
Values:

N = _N

Public Functions

PolynomialOptimizationNonLinear(size_t *dimension*, **const** *NonlinearOptimizationParameters* &*parameters*)
Sets up the nonlinear optimization problem.

Input: dimension = Spatial dimension of the problem. Usually 1 or 3. Input: parameters = Parameters for the optimization problem. Input: optimize_time_only = Specifies whether the optimization is run over the segment times only. If true, only the segment times are optimization parameters, and the remaining free parameters are found by solving the linear optimization problem with the given segment times in every iteration. If false, both segment times and free derivatives become optimization variables. The latter case is theoretically correct, but may result in more iterations.

bool **setupFromVertices**(**const** *Vertex*::*Vector* &*vertices*, **const** std::vector<double> &*segment_times*, int *derivative_to_optimize* = PolynomialOptimization<*N*>::kHighestDerivativeToOptimize)
Sets up the optimization problem from a vector of *Vertex* objects and a vector of times between the vertices.

Input: vertices = Vector containing the vertices defining the support points and constraints of the path. Input: segment_times = Vector containing an initial guess of the time between two vertices. Thus, its size is size(vertices) - 1. Input: derivative_to_optimize = Specifies the derivative of which the cost is optimized.

bool **addMaximumMagnitudeConstraint**(int *derivative_order*, double *maximum_value*)
Adds a constraint for the maximum of magnitude to the optimization problem.

Input: derivative_order = Order of the derivative, for which the constraint should be checked. Usually velocity (=1) or acceleration (=2). maximum_value = Maximum magnitude of the specified derivative.

bool **solveLinear**()
Solves the linear optimization problem according to [1].

The solver is re-used for every dimension, which means:

- segment times are equal for each dimension.
- each dimension has the same type/set of constraints. Their values can of course differ.

int **optimize**()
Runs the optimization until one of the stopping criteria in *NonlinearOptimizationParameters* and the constraints are met.

void **getTrajectory**(*Trajectory* **trajectory*) **const**
Get the resulting trajectory out prefer this as the main method to get the results of the optimization, over getting the reference to the linear optimizer.

const PolynomialOptimization<*N*> &**getPolynomialOptimizationRef**() **const**
Returns a const reference to the underlying linear optimization object.

PolynomialOptimization<*N*> &**getPolynomialOptimizationRef**()
Returns a non-const reference to the underlying linear optimization object.

OptimizationInfo **getOptimizationInfo**() **const**

double **getCost**() **const**

Functions for optimization, but may be useful for diagnostics outside.

Gets the trajectory cost (same as the cost in the linear problem).

double **getTotalCostWithSoftConstraints**() **const**

Gets the cost including the soft constraints and time costs, should be the same cost function as used in the full optimization.

Returns the same metrics regardless of time estimation method set.

void **scaleSegmentTimesWithViolation**()

Template Class Accumulator

- Defined in *File timing.h*

Class Documentation

template <typename *T*, typename *Total*, int *N*>
class Accumulator

Public Functions

Accumulator()

void **Add**(T *sample*)

int **TotalSamples**() **const**

double **Sum**() **const**

double **Mean**() **const**

double **RollingMean**() **const**

double **Max**() **const**

double **Min**() **const**

double **LazyVariance**() **const**

Class DummyTimer

- Defined in *File timing.h*

Class Documentation

class DummyTimer

A class that has the timer interface but does nothing.

Swapping this in in place of the *Timer* class (say with a typedef) should allow one to disable timing. Because all of the functions are inline, they should just disappear.

Public Functions

DummyTimer(size_t, bool = false)

DummyTimer(std::string **const**&, bool = false)

~DummyTimer()

void **Start**()

void **Stop**()

bool **IsTiming**()

Class MiniTimer

- Defined in *File timing.h*

Class Documentation

class MiniTimer
Small timer for benchmarking.

Public Functions

MiniTimer()

void **start**()

double **stop**()

double **getTime**() **const**

Class Timer

- Defined in *File timing.h*

Class Documentation

class Timer

Public Functions

Timer(size_t *handle*, bool *constructStopped* = false)

Timer(std::string **const** &*tag*, bool *constructStopped* = false)

~Timer()

void **Start**()

void **`Stop`**()

bool **`IsTiming`**() **`const`**

Class Timing

- Defined in *File timing.h*

Class Documentation

`class Timing`

Public Types

`typedef` std::map<std::string, size_t> **`map_t`**

Public Static Functions

`static` size_t **`GetHandle`**(std::string **`const`** &*tag*)
Definition of static functions to query the timers.

`static` std::string **`GetTag`**(size_t *handle*)

`static` double **`GetTotalSeconds`**(size_t *handle*)

`static` double **`GetTotalSeconds`**(std::string **`const`** &*tag*)

`static` double **`GetMeanSeconds`**(size_t *handle*)

`static` double **`GetMeanSeconds`**(std::string **`const`** &*tag*)

`static` size_t **`GetNumSamples`**(size_t *handle*)

`static` size_t **`GetNumSamples`**(std::string **`const`** &*tag*)

`static` double **`GetVarianceSeconds`**(size_t *handle*)

`static` double **`GetVarianceSeconds`**(std::string **`const`** &*tag*)

`static` double **`GetMinSeconds`**(size_t *handle*)

`static` double **`GetMinSeconds`**(std::string **`const`** &*tag*)

`static` double **`GetMaxSeconds`**(size_t *handle*)

`static` double **`GetMaxSeconds`**(std::string **`const`** &*tag*)

`static` double **`GetHz`**(size_t *handle*)

`static` double **`GetHz`**(std::string **`const`** &*tag*)

`static` void **`Print`**(std::ostream &*out*)

`static` std::string **`Print`**()

static std::string **SecondsToTimeString**(double *seconds*)

static void **Reset**()

static const *map_t* &**GetTimers**()

Friends

friend mav_trajectory_generation::timing::Timing::Timer

Class Trajectory

- Defined in *File trajectory.h*

Class Documentation

class Trajectory

Holder class for trajectories of D dimensions, of K segments, and polynomial order N-1.

(N=12 -> 11th order polynomial, with 12 coefficients).

Public Functions

Trajectory()

~Trajectory()

bool **operator==**(**const** *Trajectory* &*rhs*) **const**

bool **operator!=**(**const** *Trajectory* &*rhs*) **const**

int **D**() **const**

int **N**() **const**

int **K**() **const**

bool **empty**() **const**

void **clear**()

void **setSegments**(**const** Segment::Vector &*segments*)

void **addSegments**(**const** Segment::Vector &*segments*)

void **getSegments**(Segment::Vector **segments*) **const**

const Segment::Vector &**segments**() **const**

double **getMinTime**() **const**

double **getMaxTime**() **const**

std::vector<double> **getSegmentTimes**() **const**

Trajectory **getTrajectoryWithSingleDimension**(int *dimension*) **const**
: Functions to create new trajectories by splitting (getting a NEW trajectory with a single dimension) or compositing (create a new trajectory with another trajectory appended).

bool **getTrajectoryWithAppendedDimension**(**const** *Trajectory* &*trajectory_to_append*, *Trajectory* **new_trajectory*) **const**

bool **addTrajectories**(**const** std::vector<*Trajectory*> &*trajectories*, *Trajectory* **merged*) **const**
: Add trajectories with same dimensions and coefficients to this trajectory.

Vertex **getVertexAtTime**(double *t*, int *max_derivative_order*) **const**
: Evaluate the vertex constraint at time t.

Vertex **getStartVertex**(int *max_derivative_order*) **const**
: Evaluate the vertex constraint at start time.

Vertex **getGoalVertex**(int *max_derivative_order*) **const**
: Evaluate the vertex constraint at goal time.

Eigen::VectorXd **evaluate**(double *t*, int *derivative_order* = *derivative_order*::POSITION) **const**
: Evaluation functions.

  Evaluate at a single time, and a single derivative. Return type of dimension D.

void **evaluateRange**(double *t_start*, double *t_end*, double *dt*, int *derivative_order*, std::vector<Eigen::VectorXd> **result*, std::vector<double> **sampling_times* = nullptr) **const**
: Evaluates the trajectory in a specified range and derivative.

  Outputs are a vector of the sampled values (size of VectorXd is D) by time and optionally the actual sampling times.

bool **computeMinMaxMagnitude**(int *derivative*, **const** std::vector<int> &*dimensions*, *Extremum* **minimum*, *Extremum* **maximum*) **const**
: Compute the analytic minimum and maximum of magnitude for a given derivative and dimensions, e.g., [0, 1, 2] for position or [3] for yaw.

  Returns false in case of extremum calculation failure.

bool **computeMaxVelocityAndAcceleration**(double **v_max*, double **a_max*) **const**
: Compute max velocity and max acceleration. Shorthand for the method above.

bool **scaleSegmentTimesToMeetConstraints**(double *v_max*, double *a_max*)
: This method SCALES the segment times evenly to ensure that the trajectory is feasible given the provided v_max and a_max.

  Does not change the shape of the trajectory, and only *increases* segment times.

Class Vertex

- Defined in *File vertex.h*

Class Documentation

class Vertex
: A vertex describes the properties of a support point of a path.

A vertex has a set of constraints, the derivative of position a value, that have to be matched during optimization procedures. In case of a multi-dimensional vertex (mostly 3D), the constraint for a derivative exists in all dimensions, but can have different values in each dimension. X———X————X vertex segment

Public Types

typedef std::vector<*Vertex*> **Vector**

typedef Eigen::VectorXd **ConstraintValue**

typedef std::pair<int, *ConstraintValue*> **Constraint**

typedef std::map<int, *ConstraintValue*> **Constraints**

Public Functions

Vertex (size_t *dimension*)
Constructs an empty vertex and sets time_to_next and derivative_to_optimize to zero.

int **D** () **const**

void **addConstraint** (int *derivative_order*, double *value*)
Adds a constraint for the specified derivative order with the given value.

If this is a multi-dimensional vertex, all dimensions are set to the same value.

void **addConstraint** (int *type*, **const** Eigen::VectorXd &*constraint*)
Adds a constraint for the derivative specified in type with the given values in the constraint vector.

The dimension has to match the derivative.

bool **removeConstraint** (int *type*)
Removes a constraint for the derivative specified in type.

Returns false if constraint was not set.

void **makeStartOrEnd** (**const** Eigen::VectorXd &*constraint*, int *up_to_derivative*)
Sets a constraint for position and sets all derivatives up to (including) up_to_derivative to zero.

Convenience method for beginning / end vertices. up_to_derivative should be set to getHighestDerivative-FromN(N), where N is the order of your polynomial.

void **makeStartOrEnd** (double *value*, int *up_to_derivative*)

bool **hasConstraint** (int *derivative_order*) **const**
Returns whether the vertex has a constraint for the specified derivative order.

bool **getConstraint** (int *derivative_order*, Eigen::VectorXd **constraint*) **const**
Passes the value of the constraint for derivative order to *value, and returns whether the constraint is set.

Constraints::const_iterator **cBegin** () **const**
Returns a const iterator to the first constraint.

Constraints::const_iterator **cEnd** () **const**
Returns a const iterator to the end of the constraints, i.e.

one after the last element.

size_t **getNumberOfConstraints**() **const**
Returns the number of constraints.

bool **isEqualTol**(**const** *Vertex* &*rhs*, double *tol*) **const**
Checks if both lhs and rhs are equal up to tol in case of double values.

bool **getSubdimension**(**const** std::vector<size_t> &*subdimensions*, int *max_derivative_order*, *Vertex* **subvertex*) **const**
Get subdimension vertex.

Class Color

- Defined in *File helpers.h*

Inheritance Relationships

Base Type

- `public ColorRGBA`

Class Documentation

class Color : **public** ColorRGBA

Public Functions

Color()

Color(double *red*, double *green*, double *blue*)

Color(double *red*, double *green*, double *blue*, double *alpha*)

Public Static Functions

static const *Color* **White**()

static const *Color* **Black**()

static const *Color* **Gray**()

static const *Color* **Red**()

static const *Color* **Green**()

static const *Color* **Blue**()

static const *Color* **Yellow**()

static const *Color* **Orange**()

static const *Color* **Purple**()

`static const` *Color* **`Chartreuse`**()

`static const` *Color* **`Teal`**()

`static const` *Color* **`Pink`**()

Class HexacopterMarker

- Defined in *File hexacopter_marker.h*

Inheritance Relationships

Base Type

- `public mav_visualization::MarkerGroup` (*Class MarkerGroup*)

Class Documentation

`class HexacopterMarker` : **`public`** mav_visualization::*MarkerGroup*

Public Functions

`HexacopterMarker`(bool *simple* = false)

Class LeicaMarker

- Defined in *File leica_marker.h*

Inheritance Relationships

Base Type

- `public mav_visualization::MarkerGroup` (*Class MarkerGroup*)

Class Documentation

`class LeicaMarker` : **`public`** mav_visualization::*MarkerGroup*

Public Functions

`LeicaMarker`()

Class MarkerGroup

- Defined in *File marker_group.h*

Inheritance Relationships

Derived Types

- `public mav_visualization::HexacopterMarker` (*Class HexacopterMarker*)
- `public mav_visualization::LeicaMarker` (*Class LeicaMarker*)

Class Documentation

class MarkerGroup

Subclassed by *mav_visualization::HexacopterMarker*, *mav_visualization::LeicaMarker*

Public Functions

MarkerGroup()

virtual ~MarkerGroup()

void **getMarkers**(visualization_msgs::MarkerArray &*marker_array*, **const** double &*scale* = 1, bool *append* = false) **const**

void **getMarkers**(*MarkerVector* &*markers*, **const** double &*scale* = 1, bool *append* = false) **const**

void **setNamespace**(**const** std::string &*ns*)

void **setHeader**(**const** std_msgs::Header &*header*)

void **setHeaderAndNamespace**(**const** std_msgs::Header &*header*, **const** std::string &*ns*)

void **setAction**(**const** int32_t &*action*)

void **setLifetime**(double *lifetime*)

void **setFrameLocked**(bool *locked*)

void **transform**(**const** Eigen::Vector3d &*t*, **const** Eigen::Quaterniond &*q*)

void **publish**(ros::Publisher &*pub*)

Protected Attributes

std::string **name_**

std::string **description_**

MarkerVector **markers_**

Protected Static Functions

static void **transformMarker**(visualization_msgs::Marker &*marker*, **const** Eigen::Vector3d &*t*, **const** Eigen::Quaterniond &*q*)

Class TrajectorySamplerNode

- Defined in *File trajectory_sampler_node.h*

Class Documentation

class TrajectorySamplerNode

Public Functions

TrajectorySamplerNode(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

~TrajectorySamplerNode()

2.3.3 Enums

Enum InputConstraintType

- Defined in *File input_constraints.h*

Enum Documentation

enum mav_trajectory_generation::**InputConstraintType**

Values:

kFMin = 0
: Minimum acceleration (normalized thrust) in [m/s/s].

kFMax
: Maximum acceleration (normalized thrust) in [m/s/s].

kVMax
: Maximum velocity in [m/s].

kOmegaXYMax
: Maximum roll/pitch rate in [rad/s].

kOmegaZMax
: Maximum yaw rate in [rad/s].

kOmegaZDotMax
: Maximum yaw acceleration in [rad/s/s].

Enum InputFeasibilityResult

Enum Documentation

enum mav_trajectory_generation::**InputFeasibilityResult**

Values:

kInputFeasible = 0
: The trajectory is input feasible.

`kInputIndeterminable`
Cannot determine whether the trajectory is feasible with respect to the inputs.

`kInputInfeasibleThrustHigh`
The trajectory is infeasible, failed max.

thrust test first.

`kInputInfeasibleThrustLow`
The trajectory is infeasible, failed min.

thrust test first.

`kInputInfeasibleVelocity`
The *Trajectory* is infeasible, failed max.

velocity test first.

`kInputInfeasibleRollPitchRates`
The trajectory is infeasible, failed max.

roll/pitch rates test first.

`kInputInfeasibleYawRates`
The trajectory is infeasible, faild max.

yaw rates test first.

`kInputInfeasibleYawAcc`
The trajectory is infeasible, failed max.

yaw acceleration test first.

2.3.4 Functions

Template Function mav_trajectory_generation::checkMatrices

- Defined in *File test_utils.h*

Function Documentation

```
template <class T1, class T2>
bool mav_trajectory_generation::checkMatrices(const Eigen::MatrixBase<T1> &m1,
                                              const Eigen::MatrixBase<T2> &m2,
                                              double tol)
```

Function mav_trajectory_generation::computeBaseCoefficients

- Defined in *File polynomial.h*

Function Documentation

`Eigen::MatrixXd mav_trajectory_generation::computeBaseCoefficients(int N)`
Static functions to compute base coefficients.

Computes the base coefficients of the derivatives of the polynomial, up to order N.

Function mav_trajectory_generation::computeCostNumeric

- Defined in *File test_utils.h*

Function Documentation

double `mav_trajectory_generation::`**`computeCostNumeric`**`(`**`const`** *Trajectory* &*trajectory*, size_t *derivative*, double *dt* = 0.001`)`

Function mav_trajectory_generation::computeTimeVelocityRamp

- Defined in *File vertex.h*

Function Documentation

double `mav_trajectory_generation::`**`computeTimeVelocityRamp`**`(`**`const`** Eigen::VectorXd &*start*, **`const`** Eigen::VectorXd &*goal*, double *v_max*, double *a_max*`)`

Template Function mav_trajectory_generation::convolve

- Defined in *File convolution.h*

Function Documentation

`template` <int *DataDimension_*, int *KernelDimension_*>
Eigen::Matrix<double, *ConvolutionDimension*<DataDimension_, KernelDimension_>::length, 1> `mav_trajectory_generatio`

Function mav_trajectory_generation::createRandomDouble

- Defined in *File test_utils.h*

Function Documentation

double `mav_trajectory_generation::`**`createRandomDouble`**(double *min*, double *max*)

Function mav_trajectory_generation::createRandomVertices

- Defined in *File vertex.h*

Function Documentation

Vertex::*Vector* `mav_trajectory_generation::`**`createRandomVertices`**(int *maximum_derivative*, size_t *n_segments*, **`const`** Eigen::VectorXd &*minimum_position*, **`const`** Eigen::VectorXd &*maximum_position*, size_t *seed* = 0)

Creates random vertices for position within minimum_position and maximum_position.

Vertices at the beginning and end have only fixed constraints with their derivative set to zero, while all vertices in between have position as fixed constraint and the derivatives are left free. Input: maximum_derivative = The maximum derivative to be set to zero for beginning and end. Input: n_segments = Number of segments of the resulting trajectory. Number of vertices is n_segments + 1. Input: minimum_position = Minimum position of the space to sample. Input: maximum_position = Maximum position of the space to sample. Input: seed = Initial seed for random number generation. Output: return = Vector containing n_segments + 1 vertices.

Function mav_trajectory_generation::createRandomVertices1D

- Defined in *File vertex.h*

Function Documentation

Vertex::*Vector* `mav_trajectory_generation::`**`createRandomVertices1D`**(int *maximum_derivative*, size_t *n_segments*, double *minimum_position*, double *maximum_position*, size_t *seed* = 0)

Conveninence function to create 1D vertices.

Function mav_trajectory_generation::createSquareVertices

- Defined in *File vertex.h*

Function Documentation

Vertex::*Vector* `mav_trajectory_generation::`**`createSquareVertices`**(int *maximum_derivative*, **`const`** Eigen::Vector3d &*center*, double *side_length*, int *rounds*)

Function mav_trajectory_generation::drawMavSampledTrajectory

- Defined in *File ros_visualization.h*

Function Documentation

void `mav_trajectory_generation::`**`drawMavSampledTrajectory`**(**`const`** mav_msgs::EigenTrajectoryPoint::Vector &*flat_states*, double *distance*, **`const`** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Draw an eigen trajectory with additional markers spaced by distance (0.0 to disable).

Function mav_trajectory_generation::drawMavSampledTrajectoryWithMavMarker

- Defined in *File ros_visualization.h*

Function Documentation

void `mav_trajectory_generation::`**`drawMavSampledTrajectoryWithMavMarker`**(**`const`** mav_msgs::EigenTrajectoryPoin &*flat_states*, double *distance*, **`const`** std::string &*frame_id*, **`const`** mav_visualization::*MarkerGrou* &*additional_marker*, visualization_msgs::MarkerArray **marker_array*)

Draw a eigen trajectory with additional marker.

Function mav_trajectory_generation::drawMavTrajectory

- Defined in *File ros_visualization.h*

Function Documentation

void mav_trajectory_generation::**drawMavTrajectory**(**const** *Trajectory* &*trajectory*, double *distance*, **const** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Draws the trajectory of the MAV, with additional markers spaced by distance.

If distance = 0.0, then these additional markers are disabled.

Function mav_trajectory_generation::drawMavTrajectoryWithMavMarker

- Defined in *File ros_visualization.h*

Function Documentation

void mav_trajectory_generation::**drawMavTrajectoryWithMavMarker**(**const** *Trajectory* &*trajectory*, double *distance*, **const** std::string &*frame_id*, **const** mav_visualization::*MarkerGroup* &*additional_marker*, visualization_msgs::MarkerArray **marker_array*)

Same as drawMavTrajectory, but also draws an additional marker at a set distance.

Function mav_trajectory_generation::drawVertices

- Defined in *File ros_visualization.h*

Function Documentation

void mav_trajectory_generation::**drawVertices**(**const** *Vertex*::*Vector* &*vertices*, **const** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Visualize original vertices.

Function mav_trajectory_generation::estimateSegmentTimes

- Defined in *File vertex.h*

Function Documentation

std::vector<double> `mav_trajectory_generation::`**`estimateSegmentTimes`**(**const** *Vertex*::*Vector* &*vertices*, double *v_max*, double *a_max*)

Makes a rough estimate based on v_max and a_max about the time required to get from one vertex to the next.

Uses the current preferred method.

Function mav_trajectory_generation::estimateSegmentTimesNfabian

- Defined in *File vertex.h*

Function Documentation

std::vector<double> `mav_trajectory_generation::`**`estimateSegmentTimesNfabian`**(**const** *Vertex*::*Vector* &*vertices*, double *v_max*, double *a_max*, double *magic_fabian_constant* = 6.5)

Makes a rough estimate based on v_max and a_max about the time required to get from one vertex to the next.

t_est = 2 * distance/v_max * (1 + magic_fabian_constant * v_max/a_max * exp(- distance/v_max * 2); magic_fabian_constant was determined to 6.5 in a student project . . .

Function mav_trajectory_generation::estimateSegmentTimesVelocityRamp

- Defined in *File vertex.h*

Function Documentation

std::vector<double> `mav_trajectory_generation::`**`estimateSegmentTimesVelocityRamp`**(**const** *Vertex*::*Vector* &*vertices*, double *v_max*, double *a_max*, double *time_factor* = 1.0)

Calculate the velocity assuming instantaneous constant acceleration a_max and straight line rest-to-rest trajectories.

The time_factor [1..Inf] increases the allocated time making the segments slower and thus feasibility more likely. This method does not take into account the start and goal velocity and acceleration.

Function mav_trajectory_generation::findLastNonZeroCoeff

- Defined in *File rpoly_ak1.h*

Function Documentation

int `mav_trajectory_generation::`**`findLastNonZeroCoeff`**(**const** Eigen::VectorXd &*coefficients*)

Function mav_trajectory_generation::findRootsJenkinsTraub

- Defined in *File rpoly_ak1.h*

Function Documentation

bool `mav_trajectory_generation::`**`findRootsJenkinsTraub`**(**const** Eigen::VectorXd &*coefficients_increasing*, Eigen::VectorXcd **roots*)

Function mav_trajectory_generation::getHighestDerivativeFromN

- Defined in *File vertex.h*

Function Documentation

int `mav_trajectory_generation::`**`getHighestDerivativeFromN`**(int *N*)

Function mav_trajectory_generation::getInputFeasibilityResultName

- Defined in *File feasibility_base.h*

Function Documentation

std::string `mav_trajectory_generation::`**`getInputFeasibilityResultName`**(*InputFeasibilityResult fr*)

Function mav_trajectory_generation::getMaximumMagnitude

- Defined in *File test_utils.h*

Function Documentation

double `mav_trajectory_generation::`**`getMaximumMagnitude`**(**`const`** *Trajectory &trajectory*, size_t *derivative*, double *dt* = 0.01)

Function mav_trajectory_generation::operator<<(std::ostream&, const Extremum&)

Function Documentation

std::ostream &`mav_trajectory_generation::`**`operator<<`**(std::ostream *&stream*, **`const`** *Extremum &e*)

Function mav_trajectory_generation::operator<<(std::ostream&, const OptimizationInfo&)

Function Documentation

std::ostream &`mav_trajectory_generation::`**`operator<<`**(std::ostream *&stream*, **`const`** *OptimizationInfo &val*)

Function mav_trajectory_generation::operator<<(std::ostream&, const Vertex&)

Function Documentation

std::ostream &`mav_trajectory_generation::`**`operator<<`**(std::ostream *&stream*, **`const`** *Vertex &v*)

Function mav_trajectory_generation::operator<<(std::ostream&, const std::vector<Vertex>&)

Function Documentation

std::ostream &`mav_trajectory_generation::`**`operator<<`**(std::ostream *&stream*, **`const`** std::vector<*Vertex*> *&vertices*)

Function mav_trajectory_generation::orientationDerivativeToInt

- Defined in *File motion_defines.h*

Function Documentation

int `mav_trajectory_generation::`**`orientationDerivativeToInt`**(**const** std::string &*string*)

Function mav_trajectory_generation::orintationDerivativeToString

- Defined in *File motion_defines.h*

Function Documentation

std::string `mav_trajectory_generation::`**`orintationDerivativeToString`**(int *derivative*)

Function mav_trajectory_generation::polynomialTrajectoryMsgToTrajectory

- Defined in *File ros_conversions.h*

Function Documentation

bool `mav_trajectory_generation::`**`polynomialTrajectoryMsgToTrajectory`**(**const** mav_planning_msgs::PolynomialTr &*msg*, *Trajectory* **trajectory*)

Converts a ROS polynomial trajectory msg into a *Trajectory*.

Function mav_trajectory_generation::positionDerivativeToInt

- Defined in *File motion_defines.h*

Function Documentation

int `mav_trajectory_generation::`**`positionDerivativeToInt`**(**const** std::string &*string*)

Function mav_trajectory_generation::positionDerivativeToString

Function Documentation

std::string `mav_trajectory_generation::`**`positionDerivativeToString`**(int *derivative*)

Function mav_trajectory_generation::sampledTrajectoryStatesToFile

- Defined in *File io.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampledTrajectoryStatesToFile`**(**`const`** std::string &*filename*, **`const`** *Trajectory* &*trajectory*)

Template Function mav_trajectory_generation::sampleFlatStateAtTime

- Defined in *File trajectory_sampling.h*

Function Documentation

`template <class` T>
bool `mav_trajectory_generation::`**`sampleFlatStateAtTime`**(**`const`** T &*type*, double *sample_time*, mav_msgs::EigenTrajectoryPoint **state*)

Function mav_trajectory_generation::sampleSegmentAtTime

- Defined in *File trajectory_sampling.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampleSegmentAtTime`**(**`const`** Segment &*segment*, double *sample_time*, mav_msgs::EigenTrajectoryPoint **state*)

Function mav_trajectory_generation::sampleTrajectoryAtTime

- Defined in *File trajectory_sampling.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampleTrajectoryAtTime`**(**`const`** *Trajectory* &*trajectory*, double *sample_time*, mav_msgs::EigenTrajectoryPoint **state*)

Function mav_trajectory_generation::sampleTrajectoryInRange

- Defined in *File trajectory_sampling.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampleTrajectoryInRange`**(**const** *Trajectory* &*trajectory*, double *min_time*, double *max_time*, double *sampling_interval*, mav_msgs::EigenTrajectoryPointVector **states*)

Function mav_trajectory_generation::sampleTrajectoryStartDuration

- Defined in *File trajectory_sampling.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampleTrajectoryStartDuration`**(**const** *Trajectory* &*trajectory*, double *start_time*, double *duration*, double *sampling_interval*, mav_msgs::EigenTrajectoryPointVector **states*)

Function mav_trajectory_generation::sampleWholeTrajectory

- Defined in *File trajectory_sampling.h*

Function Documentation

bool `mav_trajectory_generation::`**`sampleWholeTrajectory`**(**const** *Trajectory* &*trajectory*, double *sampling_interval*, mav_msgs::EigenTrajectoryPoint::Vector **states*)

Function mav_trajectory_generation::segmentsFromFile

- Defined in *File io.h*

Function Documentation

bool `mav_trajectory_generation::`**`segmentsFromFile`**(**const** std::string &*filename*, mav_trajectory_generation::Segment::Vector **segments*)

Function mav_trajectory_generation::segmentsToFile

- Defined in *File io.h*

Function Documentation

bool mav_trajectory_generation::**segmentsToFile**(**const** std::string *&filename*, **const** mav_trajectory_generation::Segment::Vector *&segments*)

Template Function mav_trajectory_generation::sgn

- Defined in *File convolution.h*

Function Documentation

template <**typename** T>
int mav_trajectory_generation::**sgn**(T *val*)

Function mav_trajectory_generation::trajectoryFromFile

- Defined in *File io.h*

Function Documentation

bool mav_trajectory_generation::**trajectoryFromFile**(**const** std::string *&filename*, mav_trajectory_generation::*Trajectory* **trajectory*)

Function mav_trajectory_generation::trajectoryToFile

- Defined in *File io.h*

Function Documentation

bool mav_trajectory_generation::**trajectoryToFile**(**const** std::string *&filename*, **const** mav_trajectory_generation::*Trajectory* *&trajectory*)

Function mav_trajectory_generation::trajectoryToPolynomialTrajectoryMsg

- Defined in *File ros_conversions.h*

Function Documentation

bool mav_trajectory_generation::**trajectoryToPolynomialTrajectoryMsg**(**const** *Trajectory* *&trajectory*, mav_planning_msgs::PolynomialTr **msg*)

Converts a trajectory into a ROS polynomial trajectory msg.

Function mav_visualization::createPoint

- Defined in *File helpers.h*

Function Documentation

geometry_msgs::Point `mav_visualization::`**`createPoint`**`(`double *x*, double *y*, double *z*`)`
helper function to create a geometry_msgs::Point

Function mav_visualization::drawArrowPoints

- Defined in *File helpers.h*

Function Documentation

void `mav_visualization::`**`drawArrowPoints`**`(`**`const`** Eigen::Vector3d &*p1*, **`const`** Eigen::Vector3d &*p2*, **`const`** std_msgs::ColorRGBA &*color*, double *diameter*, visualization_msgs::Marker **marker*`)`

Function mav_visualization::drawArrowPositionOrientation

- Defined in *File helpers.h*

Function Documentation

void `mav_visualization::`**`drawArrowPositionOrientation`**`(`**`const`** Eigen::Vector3d &*p*, **`const`** Eigen::Quaterniond &*q*, **`const`** std_msgs::ColorRGBA &*color*, double *length*, double *diameter*, visualization_msgs::Marker **marker*`)`

Function mav_visualization::drawAxes

- Defined in *File helpers.h*

Function Documentation

void `mav_visualization::`**`drawAxes`**`(`**`const`** Eigen::Vector3d &*p*, **`const`** Eigen::Quaterniond &*q*, double *scale*, double *line_width*, visualization_msgs::Marker **marker*`)`

Function mav_visualization::drawAxesArrows

- Defined in *File helpers.h*

Function Documentation

void `mav_visualization::`**`drawAxesArrows`**(**`const`** Eigen::Vector3d &*p*, **`const`** Eigen::Quaterniond &*q*, double *scale*, double *diameter*, visualization_msgs::MarkerArray **marker_array*)

Function mav_visualization::drawCovariance3D

- Defined in *File helpers.h*

Function Documentation

void `mav_visualization::`**`drawCovariance3D`**(**`const`** Eigen::Vector3d &*mu*, **`const`** Eigen::Matrix3d &*cov*, **`const`** std_msgs::ColorRGBA &*color*, double *n_sigma*, visualization_msgs::Marker **marker*)

Function nlopt::returnValueToString

Function Documentation

std::string `nlopt::`**`returnValueToString`**(int *return_value*)
Convenience function that turns nlopt's return values into something readable.

2.3.5 Variables

Variable mav_trajectory_generation::derivative_order::ACCELERATION

Variable Documentation

`constexpr` int `mav_trajectory_generation::derivative_order::`**`ACCELERATION`** = 2

Variable mav_trajectory_generation::derivative_order::ANGULAR_ACCELERATION

- Defined in *File motion_defines.h*

Variable Documentation

`constexpr` int `mav_trajectory_generation::derivative_order::`**`ANGULAR_ACCELERATION`** = 2

Variable mav_trajectory_generation::derivative_order::ANGULAR_VELOCITY

- Defined in *File motion_defines.h*

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`ANGULAR_VELOCITY`** = 1

Variable mav_trajectory_generation::derivative_order::INVALID

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`INVALID`** = -1

Variable mav_trajectory_generation::derivative_order::JERK

- Defined in *File motion_defines.h*

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`JERK`** = 3

Variable mav_trajectory_generation::derivative_order::ORIENTATION

- Defined in *File motion_defines.h*

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`ORIENTATION`** = 0

Variable mav_trajectory_generation::derivative_order::POSITION

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`POSITION`** = 0

Variable mav_trajectory_generation::derivative_order::SNAP

- Defined in *File motion_defines.h*

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`SNAP`** = 4

Variable mav_trajectory_generation::derivative_order::VELOCITY

Variable Documentation

constexpr int `mav_trajectory_generation::derivative_order::`**`VELOCITY`** = 1

Variable mav_trajectory_generation::kOptimizationTimeLowerBound

Variable Documentation

constexpr double `mav_trajectory_generation::`**`kOptimizationTimeLowerBound`** = 0.1

2.3.6 Typedefs

Typedef mav_trajectory_generation::timing::DebugTimer

- Defined in *File timing.h*

Typedef Documentation

typedef *Timer* `mav_trajectory_generation::timing::`**`DebugTimer`**

Typedef mav_visualization::MarkerVector

- Defined in *File marker_group.h*

Typedef Documentation

typedef std::vector<visualization_msgs::Marker> `mav_visualization::`**`MarkerVector`**

2.3.7 Directories

Directory include

Parent directory (`mav_trajectory_generation`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include`

Subdirectories

- *Directory generation*

Directory generation

Parent directory (`/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include/mav_trajectory_generation`

Subdirectories

- *Directory impl*
- *Directory rpoly*

Directory impl

Parent directory (`/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include/mav_trajectory_generation`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include/mav_trajectory_generation/impl`

Directory rpoly

Parent directory (`/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include/mav_trajectory_generation`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation/include/mav_trajectory_generation/rpoly`

Directory include

Parent directory (`mav_trajectory_generation_ros`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation_ros/include`

Subdirectories

- *Directory ros*

Directory ros

Parent directory (`/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation_ros/include`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_trajectory_generation_ros/include/mav_trajectory_generation_ros`

Directory include

Parent directory (`mav_visualization`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_visualization/include`

Subdirectories

- *Directory visualization*

Directory visualization

Parent directory (`/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_visualization/include`)

Directory path: `/home/docs/checkouts/readthedocs.org/user_builds/mav-trajectory-generation/checkouts/latest/mav_visualization/include/mav_visualization`

Directory generation

Directory path: `mav_trajectory_generation`

Subdirectories

- *Directory include*

Directory ros

Directory path: `mav_trajectory_generation_ros`

Subdirectories

- *Directory include*

Directory visualization

Directory path: `mav_visualization`

Subdirectories

- *Directory include*

2.3.8 Files

File convolution.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/convolution.h)*
- *Included By*
- *Namespaces*
- *Classes*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/convolution.h)

Program Listing for File convolution.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/convolution.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_CONVOLUTION_H_
#define MAV_TRAJECTORY_GENERATION_CONVOLUTION_H_

namespace mav_trajectory_generation {

template <int D, int K>
struct ConvolutionDimension {
  enum { length = D + K - 1 };
};

template <int DataDimension_, int KernelDimension_>
```

(continues on next page)

(continued from previous page)

```
Eigen::Matrix<double,
              ConvolutionDimension<DataDimension_, KernelDimension_>::length, 1>
convolve(const Eigen::Matrix<double, DataDimension_, 1>& data,
         const Eigen::Matrix<double, KernelDimension_, 1>& kernel) {
  const int convolution_dimension =
      ConvolutionDimension<DataDimension_, KernelDimension_>::length;
  Eigen::Matrix<double, convolution_dimension, 1> convolved;
  convolved.setZero();
  Eigen::Matrix<double, KernelDimension_, 1> kernel_reverse(kernel.reverse());

  for (int output_idx = 0; output_idx < convolution_dimension; ++output_idx) {
    const int data_idx = output_idx - KernelDimension_ + 1;

    int lower_bound = std::max(0, -data_idx);
    int upper_bound = std::min(KernelDimension_, DataDimension_ - data_idx);

    for (int kernel_idx = lower_bound; kernel_idx < upper_bound; ++kernel_idx) {
      convolved[output_idx] +=
          kernel_reverse[kernel_idx] * data[data_idx + kernel_idx];
    }
  }

  return convolved;
}

template <typename T>
int sgn(T val) {
  return (T(0) < val) - (val < T(0));
}

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_CONVOLUTION_H_
```

Included By

- *File polynomial_optimization_linear_impl.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Template Struct ConvolutionDimension*

File extremum.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/extremum.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/extremum.h)

Program Listing for File extremum.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/extremum.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_EXTREMUM_H_
#define MAV_TRAJECTORY_GENERATION_EXTREMUM_H_

#include <iostream>

namespace mav_trajectory_generation {

struct Extremum {
 public:
  Extremum() : time(0.0), value(0.0), segment_idx(0) {}

  Extremum(double _time, double _value, int _segment_idx)
      : time(_time), value(_value), segment_idx(_segment_idx) {}

  bool operator<(const Extremum& rhs) const { return value < rhs.value; }
```

(continues on next page)

(continued from previous page)

```
  bool operator>(const Extremum& rhs) const { return value > rhs.value; }

  double
      time;
  double value;
  int segment_idx;
};

inline std::ostream& operator<<(std::ostream& stream, const Extremum& e) {
  stream << "time: " << e.time << ", value: " << e.value
         << ", segment idx: " << e.segment_idx << std::endl;
  return stream;
}

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_EXTREMUM_H_
```

Includes

- `iostream`

Included By

- *File polynomial_optimization_linear.h*
- *File segment.h*
- *File trajectory.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Struct Extremum*

File feasibility_analytic.h

Contents

- *Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_analytic.h`)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_analytic.h)

Program Listing for File feasibility_analytic.h

Return to documentation for file (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_analytic.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_ANALYTIC_H_
#define MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_ANALYTIC_H_

#include <cmath>

#include <mav_trajectory_generation/segment.h>

#include "mav_trajectory_generation_ros/feasibility_base.h"

namespace mav_trajectory_generation {

class FeasibilityAnalytic : public FeasibilityBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef std::vector<Eigen::VectorXcd,
                      Eigen::aligned_allocator<Eigen::VectorXcd>>
      Roots;

  class Settings {
   public:
    Settings();

    inline void setMinSectionTimeS(double min_section_time_s) {
      min_section_time_s_ = std::abs(min_section_time_s);
    }
    inline double getMinSectionTimeS() const { return min_section_time_s_; }

   private:
    double min_section_time_s_;
```

(continues on next page)

(continued from previous page)

```
  };

  FeasibilityAnalytic() {}
  FeasibilityAnalytic(const Settings& settings);
  FeasibilityAnalytic(const InputConstraints& input_constraints);
  FeasibilityAnalytic(const Settings& settings,
                      const InputConstraints& input_constraints);
  virtual InputFeasibilityResult checkInputFeasibility(
      const Segment& segment) const;

  Settings settings_;

 private:
  InputFeasibilityResult analyticThrustFeasibility(
      const Segment& segment, std::vector<Extremum>* thrust_candidates,
      Segment* thrust_segment) const;
  InputFeasibilityResult recursiveRollPitchFeasibility(
      const Segment& pos_segment, const Segment& thrust_segment,
      const std::vector<Extremum>& thrust_candidates,
      const std::vector<Extremum>& jerk_candidates, double t_1,
      double t_2) const;
};
}  // namespace mav_trajectory_generation

#endif
```

Includes

- `cmath`
- `mav_trajectory_generation/segment.h` (*File segment.h*)
- `mav_trajectory_generation_ros/feasibility_base.h` (*File feasibility_base.h*)

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class FeasibilityAnalytic*
- *Class FeasibilityAnalytic::Settings*

File feasibility_base.h

Contents

- *Definition* (*mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_base.h*)

- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_base.h)

Program Listing for File feasibility_base.h

Return to documentation for file (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_base.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_BASE_H_
#define MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_BASE_H_

#include <glog/logging.h>
#include <ros/ros.h>
#include <Eigen/Core>
#include <Eigen/StdVector>

#include <mav_trajectory_generation/trajectory.h>

#include "input_constraints.h"

namespace mav_trajectory_generation {
enum InputFeasibilityResult {
  kInputFeasible = 0,
  kInputIndeterminable,
  kInputInfeasibleThrustHigh,
  kInputInfeasibleThrustLow,
  kInputInfeasibleVelocity,
  kInputInfeasibleRollPitchRates,
```

(continues on next page)

(continued from previous page)

```
  kInputInfeasibleYawRates,
  kInputInfeasibleYawAcc,
};

// Human readable InputFeasibilityResult.
std::string getInputFeasibilityResultName(InputFeasibilityResult fr);

class HalfPlane {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef std::vector<HalfPlane, Eigen::aligned_allocator<HalfPlane>> Vector;
  HalfPlane(const Eigen::Vector3d& point, const Eigen::Vector3d& normal);
  HalfPlane(const Eigen::Vector3d& a, const Eigen::Vector3d& b,
            const Eigen::Vector3d& c);

  static HalfPlane::Vector createBoundingBox(
      const Eigen::Vector3d& point, const Eigen::Vector3d& bounding_box_size);
  Eigen::Vector3d point;
  Eigen::Vector3d normal;
};

class FeasibilityBase {
 public:
  FeasibilityBase();
  FeasibilityBase(const InputConstraints& input_constraints);

  InputFeasibilityResult checkInputFeasibilityTrajectory(
      const Trajectory& trajectory) const;
  inline virtual InputFeasibilityResult checkInputFeasibility(
      const Segment& segment) const {
    ROS_ERROR_STREAM("Input feasibility check not implemented.");
    return InputFeasibilityResult::kInputIndeterminable;
  }
  inline InputConstraints getInputConstraints() const {
    return input_constraints_;
  }

  bool checkHalfPlaneFeasibility(const Trajectory& trajectory) const;
  bool checkHalfPlaneFeasibility(const Segment& segment) const;

  InputConstraints input_constraints_;
  HalfPlane::Vector half_plane_constraints_;
  Eigen::Vector3d gravity_;
};
}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_BASE_H_
```

Includes

- `Eigen/Core`
- `Eigen/StdVector`
- `glog/logging.h`
- `input_constraints.h` (*File input_constraints.h*)

- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)
- `ros/ros.h`

Included By

- *File feasibility_analytic.h*
- *File feasibility_recursive.h*
- *File feasibility_sampling.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class FeasibilityBase*
- *Class HalfPlane*

File feasibility_recursive.h

Contents

- *Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_recursive.h`)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_recursive.h`)

Program Listing for File feasibility_recursive.h

Return to documentation for file (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_recursive.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
```

(continues on next page)

(continued from previous page)

```
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_RECURSIVE_H_
#define MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_RECURSIVE_H_

#include <cmath>

#include <mav_trajectory_generation/segment.h>

#include "mav_trajectory_generation_ros/feasibility_base.h"

namespace mav_trajectory_generation {

class FeasibilityRecursive : public FeasibilityBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef std::vector<Eigen::VectorXcd,
                      Eigen::aligned_allocator<Eigen::VectorXcd>>
      Roots;

  class Settings {
   public:
    Settings();

    inline void setMinSectionTimeS(double min_section_time_s) {
      min_section_time_s_ = std::abs(min_section_time_s);
    }
    inline double getMinSectionTimeS() const { return min_section_time_s_; }

   private:
    double min_section_time_s_;
  };

  FeasibilityRecursive() {}
  FeasibilityRecursive(const Settings& settings);
  FeasibilityRecursive(const InputConstraints& input_constraints);
  FeasibilityRecursive(const Settings& settings,
                       const InputConstraints& input_constraints);

  virtual InputFeasibilityResult checkInputFeasibility(
      const Segment& segment) const;

  Settings settings_;

 private:
  InputFeasibilityResult recursiveFeasibility(const Segment& segment,
                                              const Roots& roots_acc,
```

(continues on next page)

(continued from previous page)

```
                                                   const Roots& roots_jerk,
                                                   const Roots& roots_snap,
                                                   double t_1, double t_2) const;

  double evaluateThrust(const Segment& segment, double time) const;
};
}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_RECURSIVE_H_
```

Includes

- `cmath`
- `mav_trajectory_generation/segment.h` (*File segment.h*)
- `mav_trajectory_generation_ros/feasibility_base.h` (*File feasibility_base.h*)

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class FeasibilityRecursive*
- *Class FeasibilityRecursive::Settings*

File feasibility_sampling.h

Contents

- *Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_sampling.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_sampling.h`)

Program Listing for File feasibility_sampling.h

Return to documentation for file (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/feasibility_sampling.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_SAMPLING_H_
#define MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_SAMPLING_H_

#include <cmath>

#include <mav_trajectory_generation/segment.h>

#include "mav_trajectory_generation_ros/feasibility_base.h"

namespace mav_trajectory_generation {

class FeasibilitySampling : public FeasibilityBase {
 public:
  class Settings {
   public:
    Settings();

    inline void setSamplingIntervalS(double sampling_interval_s) {
      sampling_interval_s_ = std::abs(sampling_interval_s);
    }
    inline double getSamplingIntervalS() const { return sampling_interval_s_; }

   private:
    double sampling_interval_s_;
  };

  FeasibilitySampling() {}
  FeasibilitySampling(const Settings& settings);
  FeasibilitySampling(const InputConstraints& input_constraints);
  FeasibilitySampling(const Settings& settings,
                      const InputConstraints& input_constraints);
  virtual InputFeasibilityResult checkInputFeasibility(
      const Segment& segment) const;

  Settings settings_;
};
}  // namespace mav_trajectory_generation
```

(continues on next page)

(continued from previous page)

```
#endif  // MAV_TRAJECTORY_GENERATION_ROS_FEASIBILITY_SAMPLING_H_
```

Includes

- cmath
- mav_trajectory_generation/segment.h (*File segment.h*)
- mav_trajectory_generation_ros/feasibility_base.h (*File feasibility_base.h*)

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class FeasibilitySampling*
- *Class FeasibilitySampling::Settings*

File helpers.h

Contents

- *Definition (mav_visualization/include/mav_visualization/helpers.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (mav_visualization/include/mav_visualization/helpers.h)

Program Listing for File helpers.h

Return to documentation for file (mav_visualization/include/mav_visualization/helpers.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
```

(continues on next page)

(continued from previous page)

```
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_VISUALIZATION_HELPERS_H_
#define MAV_VISUALIZATION_HELPERS_H_

#include <Eigen/Eigenvalues>

#include <eigen_conversions/eigen_msg.h>
#include <std_msgs/ColorRGBA.h>
#include <visualization_msgs/MarkerArray.h>

namespace mav_visualization {

class Color : public std_msgs::ColorRGBA {
 public:
  Color() : std_msgs::ColorRGBA() {}
  Color(double red, double green, double blue) : Color(red, green, blue, 1.0) {}
  Color(double red, double green, double blue, double alpha) : Color() {
    r = red;
    g = green;
    b = blue;
    a = alpha;
  }

  static const Color White() { return Color(1.0, 1.0, 1.0); }
  static const Color Black() { return Color(0.0, 0.0, 0.0); }
  static const Color Gray() { return Color(0.5, 0.5, 0.5); }
  static const Color Red() { return Color(1.0, 0.0, 0.0); }
  static const Color Green() { return Color(0.0, 1.0, 0.0); }
  static const Color Blue() { return Color(0.0, 0.0, 1.0); }
  static const Color Yellow() { return Color(1.0, 1.0, 0.0); }
  static const Color Orange() { return Color(1.0, 0.5, 0.0); }
  static const Color Purple() { return Color(0.5, 0.0, 1.0); }
  static const Color Chartreuse() { return Color(0.5, 1.0, 0.0); }
  static const Color Teal() { return Color(0.0, 1.0, 1.0); }
  static const Color Pink() { return Color(1.0, 0.0, 0.5); }
};

inline geometry_msgs::Point createPoint(double x, double y, double z) {
  geometry_msgs::Point p;
  p.x = x;
  p.y = y;
  p.z = z;
  return p;
}

// Draws a covariance ellipsoid
// Input: mu = static 3 element vector, specifying the ellipsoid center
// Input: cov = static 3x3 covariance matrix
// Input: color = RGBA color of the ellipsoid
// Input: n_sigma = confidence area / scale of the ellipsoid
// Output: marker = The marker in which the ellipsoid should be drawn
```

(continues on next page)

(continued from previous page)

```
inline void drawCovariance3D(const Eigen::Vector3d& mu,
                             const Eigen::Matrix3d& cov,
                             const std_msgs::ColorRGBA& color, double n_sigma,
                             visualization_msgs::Marker* marker) {
  // TODO(helenol): What does this do???? Does anyone know?
  const Eigen::Matrix3d changed_covariance = (cov + cov.transpose()) * 0.5;
  Eigen::SelfAdjointEigenSolver<Eigen::Matrix3d> solver(
      changed_covariance, Eigen::ComputeEigenvectors);
  Eigen::Matrix3d V = solver.eigenvectors();
  // make sure it's a rotation matrix
  V.col(2) = V.col(0).cross(V.col(1));
  const Eigen::Vector3d sigma = solver.eigenvalues().cwiseSqrt() * n_sigma;

  tf::pointEigenToMsg(mu, marker->pose.position);
  tf::quaternionEigenToMsg(Eigen::Quaterniond(V), marker->pose.orientation);
  tf::vectorEigenToMsg(sigma * 2.0, marker->scale);  // diameter, not half axis
  marker->type = visualization_msgs::Marker::SPHERE;
  marker->color = color;
  marker->action = visualization_msgs::Marker::ADD;
}

inline void drawAxes(const Eigen::Vector3d& p, const Eigen::Quaterniond& q,
                     double scale, double line_width,
                     visualization_msgs::Marker* marker) {
  marker->colors.resize(6);
  marker->points.resize(6);
  marker->points[0] = createPoint(0, 0, 0);
  marker->points[1] = createPoint(1 * scale, 0, 0);
  marker->points[2] = createPoint(0, 0, 0);
  marker->points[3] = createPoint(0, 1 * scale, 0);
  marker->points[4] = createPoint(0, 0, 0);
  marker->points[5] = createPoint(0, 0, 1 * scale);

  marker->color = Color::Black();
  marker->colors[0] = Color::Red();
  marker->colors[1] = Color::Red();
  marker->colors[2] = Color::Green();
  marker->colors[3] = Color::Green();
  marker->colors[4] = Color::Blue();
  marker->colors[5] = Color::Blue();

  marker->scale.x = line_width;  // rest is unused
  marker->type = visualization_msgs::Marker::LINE_LIST;
  marker->action = visualization_msgs::Marker::ADD;

  tf::pointEigenToMsg(p, marker->pose.position);
  tf::quaternionEigenToMsg(q, marker->pose.orientation);
}

inline void drawArrowPositionOrientation(const Eigen::Vector3d& p,
                                         const Eigen::Quaterniond& q,
                                         const std_msgs::ColorRGBA& color,
                                         double length, double diameter,
                                         visualization_msgs::Marker* marker) {
  marker->type = visualization_msgs::Marker::ARROW;
  marker->action = visualization_msgs::Marker::ADD;
  marker->color = color;
```

(continues on next page)

(continued from previous page)

```
  tf::pointEigenToMsg(p, marker->pose.position);
  tf::quaternionEigenToMsg(q, marker->pose.orientation);

  marker->scale.x = length;
  marker->scale.y = diameter;
  marker->scale.z = diameter;
}

inline void drawArrowPoints(const Eigen::Vector3d& p1,
                            const Eigen::Vector3d& p2,
                            const std_msgs::ColorRGBA& color, double diameter,
                            visualization_msgs::Marker* marker) {
  marker->type = visualization_msgs::Marker::ARROW;
  marker->action = visualization_msgs::Marker::ADD;
  marker->color = color;

  marker->points.resize(2);
  tf::pointEigenToMsg(p1, marker->points[0]);
  tf::pointEigenToMsg(p2, marker->points[1]);

  marker->scale.x = diameter * 0.1;
  marker->scale.y = diameter * 2 * 0.1;
  marker->scale.z = 0;
}

inline void drawAxesArrows(const Eigen::Vector3d& p,
                           const Eigen::Quaterniond& q, double scale,
                           double diameter,
                           visualization_msgs::MarkerArray* marker_array) {
  marker_array->markers.resize(3);
  Eigen::Vector3d origin;
  origin.setZero();

  drawArrowPoints(origin + p, q * Eigen::Vector3d::UnitX() * scale + p,
                  Color::Red(), diameter, &marker_array->markers[0]);
  drawArrowPoints(origin + p, q * Eigen::Vector3d::UnitY() * scale + p,
                  Color::Green(), diameter, &marker_array->markers[1]);
  drawArrowPoints(origin + p, q * Eigen::Vector3d::UnitZ() * scale + p,
                  Color::Blue(), diameter, &marker_array->markers[2]);
}

}  // namespace mav_visualization

#endif  // MAV_VISUALIZATION_HELPERS_H_
```

Includes

- `Eigen/Eigenvalues`
- `eigen_conversions/eigen_msg.h`
- `std_msgs/ColorRGBA.h`
- `visualization_msgs/MarkerArray.h`

Namespaces

- *Namespace mav_visualization*

Classes

- *Class Color*

File hexacopter_marker.h

Contents

- *Definition* (`mav_visualization/include/mav_visualization/hexacopter_marker.h`)
- *Includes*
- *Namespaces*
- *Classes*

Definition (`mav_visualization/include/mav_visualization/hexacopter_marker.h`)

Program Listing for File hexacopter_marker.h

Return to documentation for file (`mav_visualization/include/mav_visualization/hexacopter_marker.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_VISUALIZATION_HEXACOPTER_MARKER_H_
#define MAV_VISUALIZATION_HEXACOPTER_MARKER_H_

#include "mav_visualization/marker_group.h"

namespace mav_visualization {

class HexacopterMarker : public MarkerGroup {
 public:
```

(continues on next page)

(continued from previous page)

```
  HexacopterMarker(bool simple = false);

 private:
  void createHexacopter(bool simple = false);
};

}  // namespace mav_visualization

#endif  // MAV_VISUALIZATION_HEXACOPTER_MARKER_H_
```

Includes

- `mav_visualization/marker_group.h` (*File marker_group.h*)

Namespaces

- *Namespace mav_visualization*

Classes

- *Class HexacopterMarker*

File input_constraints.h

Contents

- *Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/input_constraints.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/input_constraints.h`)

Program Listing for File input_constraints.h

Return to documentation for file (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/input_constraints.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
```

(continues on next page)

(continued from previous page)

```
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_INPUT_CONSTRAINTS_H_
#define MAV_TRAJECTORY_GENERATION_ROS_INPUT_CONSTRAINTS_H_

#include <map>

namespace mav_trajectory_generation {

enum InputConstraintType {
  kFMin = 0,
  kFMax,
  kVMax,
  kOmegaXYMax,
  kOmegaZMax,
  kOmegaZDotMax
};

class InputConstraints {
 public:
  InputConstraints() {}

  void addConstraint(int constraint_type, double value);

  void setDefaultValues();

  bool getConstraint(int constraint_type, double* value) const;

  bool hasConstraint(int constraint_type) const;

  bool removeConstraint(int constraint_type);

 private:
  std::map<int, double> constraints_;
};
}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_ROS_INPUT_CONSTRAINTS_H_
```

Includes

- `map`

Included By

- *File feasibility_base.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class InputConstraints*

File io.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/io.h)*
- *Includes*
- *Namespaces*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/io.h`)

Program Listing for File io.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/io.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
```

(continues on next page)

(continued from previous page)

```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_YAML_IO_H_
#define MAV_TRAJECTORY_GENERATION_YAML_IO_H_

#include "mav_trajectory_generation/segment.h"
#include "mav_trajectory_generation/trajectory.h"

namespace mav_trajectory_generation {

bool segmentsToFile(const std::string& filename,
                    const mav_trajectory_generation::Segment::Vector& segments);

inline bool trajectoryToFile(
    const std::string& filename,
    const mav_trajectory_generation::Trajectory& trajectory) {
  mav_trajectory_generation::Segment::Vector segments;
  trajectory.getSegments(&segments);
  return segmentsToFile(filename, segments);
}

bool segmentsFromFile(const std::string& filename,
                      mav_trajectory_generation::Segment::Vector* segments);

inline bool trajectoryFromFile(
    const std::string& filename,
    mav_trajectory_generation::Trajectory* trajectory) {
  mav_trajectory_generation::Segment::Vector segments;
  bool success = segmentsFromFile(filename, &segments);
  trajectory->setSegments(segments);
  return success;
}

bool sampledTrajectoryStatesToFile(const std::string& filename,
                                   const Trajectory& trajectory);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_YAML_IO_H_
```

Includes

- `mav_trajectory_generation/segment.h` (*File segment.h*)
- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)

Namespaces

- *Namespace mav_trajectory_generation*

File leica_marker.h

Contents

- *Definition (mav_visualization/include/mav_visualization/leica_marker.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (`mav_visualization/include/mav_visualization/leica_marker.h`)

Program Listing for File leica_marker.h

Return to documentation for file (`mav_visualization/include/mav_visualization/leica_marker.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_VISUALIZATION_LEICA_MARKER_H_
#define MAV_VISUALIZATION_LEICA_MARKER_H_

#include "mav_visualization/marker_group.h"

namespace mav_visualization {

class LeicaMarker : public MarkerGroup {
 public:
  LeicaMarker();

 private:
  void createLeica();
};

}  // namespace mav_visualization

#endif  // MAV_VISUALIZATION_LEICA_MARKER_H_
```

Includes

- `mav_visualization/marker_group.h` (*File marker_group.h*)

Namespaces

- *Namespace mav_visualization*

Classes

- *Class LeicaMarker*

File marker_group.h

Contents

- *Definition (`mav_visualization/include/mav_visualization/marker_group.h`)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_visualization/include/mav_visualization/marker_group.h`)

Program Listing for File marker_group.h

Return to documentation for file (`mav_visualization/include/mav_visualization/marker_group.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_VISUALIZATION_MARKER_GROUP_H_
#define MAV_VISUALIZATION_MARKER_GROUP_H_
```

(continues on next page)

(continued from previous page)

```
#include <ros/ros.h>
#include <visualization_msgs/MarkerArray.h>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <string>
#include <vector>

namespace mav_visualization {

typedef std::vector<visualization_msgs::Marker> MarkerVector;

class MarkerGroup {
 public:
  MarkerGroup();
  virtual ~MarkerGroup();
  void getMarkers(visualization_msgs::MarkerArray& marker_array,
                  const double& scale = 1, bool append = false) const;
  void getMarkers(MarkerVector& markers, const double& scale = 1,
                  bool append = false) const;
  void setNamespace(const std::string& ns);
  void setHeader(const std_msgs::Header& header);
  void setHeaderAndNamespace(const std_msgs::Header& header,
                             const std::string& ns);
  void setAction(const int32_t& action);
  void setLifetime(double lifetime);
  void setFrameLocked(bool locked);
  void transform(const Eigen::Vector3d& t, const Eigen::Quaterniond& q);
  void publish(ros::Publisher& pub);

 protected:
  std::string name_;
  std::string description_;
  MarkerVector markers_;
  static void transformMarker(visualization_msgs::Marker& marker,
                              const Eigen::Vector3d& t,
                              const Eigen::Quaterniond& q);
};

}  // namespace mav_visualization

#endif  // MAV_VISUALIZATION_MARKER_GROUP_H_
```

Includes

- `Eigen/Core`
- `Eigen/Geometry`
- `ros/ros.h`
- `string`
- `vector`
- `visualization_msgs/MarkerArray.h`

Included By

- *File ros_visualization.h*
- *File hexacopter_marker.h*
- *File leica_marker.h*

Namespaces

- *Namespace mav_visualization*

Classes

- *Class MarkerGroup*

File motion_defines.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/motion_defines.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/motion_defines.h`)

Program Listing for File motion_defines.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/motion_defines.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

(continues on next page)

(continued from previous page)

```
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_MOTION_DEFINES_H_
#define MAV_TRAJECTORY_MOTION_DEFINES_H_

#include <string>

namespace mav_trajectory_generation {

namespace derivative_order {
static constexpr int POSITION = 0;
static constexpr int VELOCITY = 1;
static constexpr int ACCELERATION = 2;
static constexpr int JERK = 3;
static constexpr int SNAP = 4;

static constexpr int ORIENTATION = 0;
static constexpr int ANGULAR_VELOCITY = 1;
static constexpr int ANGULAR_ACCELERATION = 2;

static constexpr int INVALID = -1;
}

std::string positionDerivativeToString(int derivative);
int positionDerivativeToInt(const std::string& string);

std::string orintationDerivativeToString(int derivative);
int orientationDerivativeToInt(const std::string& string);

}  // namespace mav_trajectory_generation

#endif
```

Includes

- `string`

Included By

- *File polynomial_optimization_linear.h*
- *File segment.h*
- *File vertex.h*

Namespaces

- *Namespace mav_trajectory_generation*
- *Namespace mav_trajectory_generation::derivative_order*

File polynomial.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/polynomial.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/polynomial.h`)

Program Listing for File polynomial.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/polynomial.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_POLYNOMIAL_H_
#define MAV_TRAJECTORY_GENERATION_POLYNOMIAL_H_

#include <glog/logging.h>
#include <Eigen/Eigen>
#include <Eigen/SVD>
#include <utility>
#include <vector>

namespace mav_trajectory_generation {

class Polynomial {
```

(continues on next page)

(continued from previous page)

```
public:
 typedef std::vector<Polynomial> Vector;

 static constexpr int kMaxN = 12;
 static constexpr int kMaxConvolutionSize = 2 * kMaxN - 2;
 static Eigen::MatrixXd base_coefficients_;

 Polynomial(int N) : N_(N), coefficients_(N) { coefficients_.setZero(); }

 Polynomial(int N, const Eigen::VectorXd& coeffs)
     : N_(N), coefficients_(coeffs) {
   CHECK_EQ(N_, coeffs.size()) << "Number of coefficients has to match.";
 }

 Polynomial(const Eigen::VectorXd& coeffs)
     : N_(coeffs.size()), coefficients_(coeffs) {}
 int N() const { return N_; }

 inline bool operator==(const Polynomial& rhs) const {
   return coefficients_ == rhs.coefficients_;
 }
 inline bool operator!=(const Polynomial& rhs) const {
   return !operator==(rhs);
 }
 inline Polynomial operator+(const Polynomial& rhs) const {
   return Polynomial(coefficients_ + rhs.coefficients_);
 }
 inline Polynomial& operator+=(const Polynomial& rhs) {
   this->coefficients_ += rhs.coefficients_;
   return *this;
 }
 inline Polynomial operator*(const Polynomial& rhs) const {
   return Polynomial(convolve(coefficients_, rhs.coefficients_));
 }
 inline Polynomial operator*(const double& rhs) const {
   return Polynomial(coefficients_ * rhs);
 }

 void setCoefficients(const Eigen::VectorXd& coeffs) {
   CHECK_EQ(N_, coeffs.size()) << "Number of coefficients has to match.";
   coefficients_ = coeffs;
 }

 Eigen::VectorXd getCoefficients(int derivative = 0) const {
   CHECK_LE(derivative, N_);
   if (derivative == 0) {
     return coefficients_;
   } else {
     Eigen::VectorXd result(N_);
     result.setZero();
     result.head(N_ - derivative) =
         coefficients_.tail(N_ - derivative)
             .cwiseProduct(
                 base_coefficients_
                     .block(derivative, derivative, 1, N_ - derivative)
                     .transpose());
     return result;
```

(continues on next page)

(continued from previous page)

```
    }
  }

  void evaluate(double t, Eigen::VectorXd* result) const {
    CHECK_LE(result->size(), N_);
    const int max_deg = result->size();

    const int tmp = N_ - 1;
    for (int i = 0; i < max_deg; i++) {
      Eigen::RowVectorXd row = base_coefficients_.block(i, 0, 1, N_);
      double acc = row[tmp] * coefficients_[tmp];
      for (int j = tmp - 1; j >= i; --j) {
        acc *= t;
        acc += row[j] * coefficients_[j];
      }
      (*result)[i] = acc;
    }
  }

  double evaluate(double t, int derivative) const {
    if (derivative >= N_) {
      return 0.0;
    }
    double result;
    const int tmp = N_ - 1;
    Eigen::RowVectorXd row = base_coefficients_.block(derivative, 0, 1, N_);
    result = row[tmp] * coefficients_[tmp];
    for (int j = tmp - 1; j >= derivative; --j) {
      result *= t;
      result += row[j] * coefficients_[j];
    }
    return result;
  }

  bool getRoots(int derivative, Eigen::VectorXcd* roots) const;

  static bool selectMinMaxCandidatesFromRoots(
      double t_start, double t_end,
      const Eigen::VectorXcd& roots_derivative_of_derivative,
      std::vector<double>* candidates);

  bool computeMinMaxCandidates(double t_start, double t_end, int derivative,
                               std::vector<double>* candidates) const;

  bool selectMinMaxFromRoots(
      double t_start, double t_end, int derivative,
      const Eigen::VectorXcd& roots_derivative_of_derivative,
      std::pair<double, double>* minimum,
      std::pair<double, double>* maximum) const;

  bool computeMinMax(double t_start, double t_end, int derivative,
                     std::pair<double, double>* minimum,
                     std::pair<double, double>* maximum) const;

  bool selectMinMaxFromCandidates(const std::vector<double>& candidates,
                                  int derivative,
                                  std::pair<double, double>* minimum,
```

(continues on next page)

(continued from previous page)

```
                                   std::pair<double, double>* maximum) const;

  bool getPolynomialWithAppendedCoefficients(int new_N,
                                              Polynomial* new_polynomial) const;

  static void baseCoeffsWithTime(int N, int derivative, double t,
                                 Eigen::VectorXd* coeffs) {
    CHECK_LT(derivative, N);
    CHECK_GE(derivative, 0);

    coeffs->resize(N, 1);
    coeffs->setZero();
    (*coeffs)[derivative] = base_coefficients_(derivative, derivative);

    if (std::abs(t) < std::numeric_limits<double>::epsilon()) return;

    double t_power = t;
    for (int j = derivative + 1; j < N; j++) {
      (*coeffs)[j] = base_coefficients_(derivative, j) * t_power;
      t_power = t_power * t;
    }
  }

  static Eigen::VectorXd baseCoeffsWithTime(int N, int derivative, double t) {
    Eigen::VectorXd c(N);
    baseCoeffsWithTime(N, derivative, t, &c);
    return c;
  }

  static Eigen::VectorXd convolve(const Eigen::VectorXd& data,
                                  const Eigen::VectorXd& kernel);

  static inline int getConvolutionLength(int data_size, int kernel_size) {
    return data_size + kernel_size - 1;
  }

  void scalePolynomialInTime(double scaling_factor);

 private:
  int N_;
  Eigen::VectorXd coefficients_;
};


Eigen::MatrixXd computeBaseCoefficients(int N);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_POLYNOMIAL_H_
```

Includes

- `Eigen/Eigen`
- `Eigen/SVD`

- glog/logging.h
- utility
- vector

Included By

- *File polynomial_optimization_linear.h*
- *File segment.h*
- *File vertex.h*
- *File trajectory_sampler_node.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class Polynomial*

File polynomial_optimization_linear.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_linear.h)*
- *Includes*
- *Included By*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_linear.h)

Program Listing for File polynomial_optimization_linear.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_linear.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
```

(continues on next page)

(continued from previous page)

```
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_LINEAR_H_
#define MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_LINEAR_H_

#include <glog/logging.h>
#include <Eigen/Sparse>
#include <tuple>

#include "mav_trajectory_generation/extremum.h"
#include "mav_trajectory_generation/motion_defines.h"
#include "mav_trajectory_generation/polynomial.h"
#include "mav_trajectory_generation/segment.h"
#include "mav_trajectory_generation/trajectory.h"
#include "mav_trajectory_generation/vertex.h"

namespace mav_trajectory_generation {

template <int _N = 10>
class PolynomialOptimization {
  static_assert(_N % 2 == 0, "The number of coefficients has to be even.");

 public:
  enum { N = _N };
  static constexpr int kHighestDerivativeToOptimize = N / 2 - 1;
  typedef Eigen::Matrix<double, N, N> SquareMatrix;
  typedef std::vector<SquareMatrix, Eigen::aligned_allocator<SquareMatrix> >
      SquareMatrixVector;

  PolynomialOptimization(size_t dimension);

  bool setupFromVertices(
      const Vertex::Vector& vertices, const std::vector<double>& segment_times,
      int derivative_to_optimize = kHighestDerivativeToOptimize);

  bool setupFromPositons(const std::vector<double>& positions,
                         const std::vector<double>& times);

  static void invertMappingMatrix(const SquareMatrix& mapping_matrix,
                                  SquareMatrix* inverse_mapping_matrix);

  static void setupMappingMatrix(double segment_time, SquareMatrix* A);

  double computeCost() const;

  void updateSegmentTimes(const std::vector<double>& segment_times);
```

(continues on next page)

(continued from previous page)

```
bool solveLinear();

void getTrajectory(Trajectory* trajectory) const {
  CHECK_NOTNULL(trajectory);
  trajectory->setSegments(segments_);
}

template <int Derivative>
static bool computeSegmentMaximumMagnitudeCandidates(
    const Segment& segment, double t_start, double t_stop,
    std::vector<double>* candidates);

static bool computeSegmentMaximumMagnitudeCandidates(
    int derivative, const Segment& segment, double t_start, double t_stop,
    std::vector<double>* candidates);

template <int Derivative>
static void computeSegmentMaximumMagnitudeCandidatesBySampling(
    const Segment& segment, double t_start, double t_stop,
    double sampling_interval, std::vector<double>* candidates);

template <int Derivative>
Extremum computeMaximumOfMagnitude(std::vector<Extremum>* candidates) const;

Extremum computeMaximumOfMagnitude(int derivative,
                                   std::vector<Extremum>* candidates) const;

void getVertices(Vertex::Vector* vertices) const {
  CHECK_NOTNULL(vertices);
  *vertices = vertices_;
}

void getSegments(Segment::Vector* segments) const {
  CHECK_NOTNULL(segments);
  *segments = segments_;
}

void getSegmentTimes(std::vector<double>* segment_times) const {
  CHECK(segment_times != nullptr);
  *segment_times = segment_times_;
}

void getFreeConstraints(
    std::vector<Eigen::VectorXd>* free_constraints) const {
  CHECK(free_constraints != nullptr);
  *free_constraints = free_constraints_compact_;
}

void setFreeConstraints(const std::vector<Eigen::VectorXd>& free_constraints);

void getFixedConstraints(
    std::vector<Eigen::VectorXd>* fixed_constraints) const {
  CHECK(fixed_constraints != nullptr);
  *fixed_constraints = fixed_constraints_compact_;
}
```

(continues on next page)

(continued from previous page)

```
  static void computeQuadraticCostJacobian(int derivative, double t,
                                           SquareMatrix* cost_jacobian);

  size_t getDimension() const { return dimension_; }
  size_t getNumberSegments() const { return n_segments_; }
  size_t getNumberAllConstraints() const { return n_all_constraints_; }
  size_t getNumberFixedConstraints() const { return n_fixed_constraints_; }
  size_t getNumberFreeConstraints() const { return n_free_constraints_; }
  int getDerivativeToOptimize() const { return derivative_to_optimize_; }

  void getAInverse(Eigen::MatrixXd* A_inv) const;
  void getM(Eigen::MatrixXd* M) const;
  void getR(Eigen::MatrixXd* R) const;
  void getA(Eigen::MatrixXd* A) const;
  void getMpinv(Eigen::MatrixXd* M_pinv) const;

  void printReorderingMatrix(std::ostream& stream) const;

 private:
  void constructR(Eigen::SparseMatrix<double>* R) const;

  void setupConstraintReorderingMatrix();

  void updateSegmentsFromCompactConstraints();

  Eigen::SparseMatrix<double> constraint_reordering_;

  Vertex::Vector vertices_;

  Segment::Vector segments_;

  SquareMatrixVector inverse_mapping_matrices_;

  SquareMatrixVector cost_matrices_;

  std::vector<Eigen::VectorXd> fixed_constraints_compact_;

  std::vector<Eigen::VectorXd> free_constraints_compact_;

  std::vector<double> segment_times_;

  size_t dimension_;

  int derivative_to_optimize_;
  size_t n_vertices_;
  size_t n_segments_;

  size_t n_all_constraints_;
  size_t n_fixed_constraints_;
  size_t n_free_constraints_;
};

struct Constraint {
  inline bool operator<(const Constraint& rhs) const {
    if (vertex_idx < rhs.vertex_idx) return true;
    if (rhs.vertex_idx < vertex_idx) return false;
```

(continues on next page)

(continued from previous page)

```
    if (constraint_idx < rhs.constraint_idx) return true;
    if (rhs.constraint_idx < constraint_idx) return false;
    return false;
  }

  inline bool operator==(const Constraint& rhs) const {
    return vertex_idx == rhs.vertex_idx && constraint_idx == rhs.constraint_idx;
  }

  size_t vertex_idx;
  size_t constraint_idx;
  Vertex::ConstraintValue value;
};

}  // namespace mav_trajectory_generation

#include "mav_trajectory_generation/impl/polynomial_optimization_linear_impl.h"

#endif  // MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_LINEAR_H_
```

Includes

- `Eigen/Sparse`
- `glog/logging.h`
- `mav_trajectory_generation/extremum.h` (*File extremum.h*)
- `mav_trajectory_generation/impl/polynomial_optimization_linear_impl.h` (*File polynomial_optimization_linear_impl.h*)
- `mav_trajectory_generation/motion_defines.h` (*File motion_defines.h*)
- `mav_trajectory_generation/polynomial.h` (*File polynomial.h*)
- `mav_trajectory_generation/segment.h` (*File segment.h*)
- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)
- `mav_trajectory_generation/vertex.h` (*File vertex.h*)
- `tuple`

Included By

- *File polynomial_optimization_nonlinear_impl.h*
- *File polynomial_optimization_nonlinear.h*

File polynomial_optimization_linear_impl.h

Contents

- *Definition* (*mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_linear_impl.h*)
- *Includes*
- *Included By*
- *Namespaces*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_linear_impl.h)

Program Listing for File polynomial_optimization_linear_impl.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_linear_impl.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_LINEAR_IMPL_H_
#define MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_LINEAR_IMPL_H_

#include <glog/logging.h>
#include <Eigen/Sparse>
#include <set>
#include <tuple>

// fixes error due to std::iota (has been introduced in c++ standard lately
// and may cause compilation errors depending on compiler)
#if __cplusplus <= 199711L
  #include <algorithm>
#else
  #include <numeric>
#endif

#include "mav_trajectory_generation/convolution.h"
```

(continues on next page)

(continued from previous page)

```
namespace mav_trajectory_generation {

template <int _N>
PolynomialOptimization<_N>::PolynomialOptimization(size_t dimension)
    : dimension_(dimension),
      derivative_to_optimize_(derivative_order::INVALID),
      n_vertices_(0),
      n_segments_(0),
      n_all_constraints_(0),
      n_fixed_constraints_(0),
      n_free_constraints_(0) {
  fixed_constraints_compact_.resize(dimension_);
  free_constraints_compact_.resize(dimension_);
}

template <int _N>
bool PolynomialOptimization<_N>::setupFromVertices(
    const Vertex::Vector& vertices, const std::vector<double>& times,
    int derivative_to_optimize) {
  CHECK(derivative_to_optimize >= 0 &&
        derivative_to_optimize <= kHighestDerivativeToOptimize)
      << "You tried to optimize the " << derivative_to_optimize
      << "th derivative of position on a " << N
      << "th order polynomial. This is not possible, you either need a higher "
         "order polynomial or a smaller derivative to optimize.";

  derivative_to_optimize_ = derivative_to_optimize;
  vertices_ = vertices;
  segment_times_ = times;

  n_vertices_ = vertices.size();
  n_segments_ = n_vertices_ - 1;

  segments_.resize(n_segments_, Segment(N, dimension_));

  CHECK(n_vertices_ == times.size() + 1)
      << "Size of times must be one less than positions.";

  inverse_mapping_matrices_.resize(n_segments_);
  cost_matrices_.resize(n_segments_);

  // Iterate through all vertices and remove invalid constraints (order too
  // high).
  for (size_t vertex_idx = 0; vertex_idx < n_vertices_; ++vertex_idx) {
    Vertex& vertex = vertices_[vertex_idx];

    // Check if we have valid constraints.
    bool vertex_valid = true;
    Vertex vertex_tmp(dimension_);
    for (Vertex::Constraints::const_iterator it = vertex.cBegin();
         it != vertex.cEnd(); ++it) {
      if (it->first > kHighestDerivativeToOptimize) {
        vertex_valid = false;
        LOG(WARNING) << "Invalid constraint on vertex " << vertex_idx
                     << ": maximum possible derivative is "
                     << kHighestDerivativeToOptimize << ", but was set to "
```

(continues on next page)

(continued from previous page)

```
                     << it->first << ". Ignoring constraint";
      } else {
        vertex_tmp.addConstraint(it->first, it->second);
      }
    }
    if (!vertex_valid) {
      vertex = vertex_tmp;
    }
  }
  updateSegmentTimes(times);
  setupConstraintReorderingMatrix();
  return true;
}

template <int _N>
void PolynomialOptimization<_N>::setupMappingMatrix(double segment_time,
                                                    SquareMatrix* A) {
  // The sum of fixed/free variables has to be equal on both ends of the
  // segment.
  // Thus, A is created as [A(t=0); A(t=segment_time)].
  for (int i = 0; i < N / 2; ++i) {
    A->row(i) = Polynomial::baseCoeffsWithTime(N, i, 0.0);
    A->row(i + N / 2) = Polynomial::baseCoeffsWithTime(N, i, segment_time);
  }
}

template <int _N>
double PolynomialOptimization<_N>::computeCost() const {
  CHECK(n_segments_ == segments_.size() &&
        n_segments_ == cost_matrices_.size());
  double cost = 0;
  for (size_t segment_idx = 0; segment_idx < n_segments_; ++segment_idx) {
    const SquareMatrix& Q = cost_matrices_[segment_idx];
    const Segment& segment = segments_[segment_idx];
    for (size_t dimension_idx = 0; dimension_idx < dimension_;
         ++dimension_idx) {
      const Eigen::VectorXd c =
          segment[dimension_idx].getCoefficients(derivative_order::POSITION);
      const double partial_cost = c.transpose() * Q * c;
      cost += partial_cost;
    }
  }
  return 0.5 * cost;  // cost = 0.5 * c^T * Q * c
}

template <int _N>
void PolynomialOptimization<_N>::invertMappingMatrix(
    const SquareMatrix& mapping_matrix, SquareMatrix* inverse_mapping_matrix) {
  // The mapping matrix has the following structure:
  // [ x 0 0 0 0 0 ]
  // [ 0 x 0 0 0 0 ]
  // [ 0 0 x 0 0 0 ]
  // [ x x x x x x ]
  // [ 0 x x x x x ]
  // [ 0 0 x x x x ]
  // ==>
  // [ A_diag B=0 ]
```

(continues on next page)

(continued from previous page)

```
  // [ C       D    ]
  // We make use of the Schur-complement, so the inverse is:
  // [ inv(A_diag)               0       ]
  // [ -inv(D) * C * inv(A_diag) inv(D) ]
  const int half_n = N / 2;

  // "template" keyword required below as half_n is dependent on the template
  // parameter.
  const Eigen::Matrix<double, half_n, 1> A_diag =
      mapping_matrix.template block<half_n, half_n>(0, 0).diagonal();
  const Eigen::Matrix<double, half_n, half_n> A_inv =
      A_diag.cwiseInverse().asDiagonal();

  const Eigen::Matrix<double, half_n, half_n> C =
      mapping_matrix.template block<half_n, half_n>(half_n, 0);

  const Eigen::Matrix<double, half_n, half_n> D_inv =
      mapping_matrix.template block<half_n, half_n>(half_n, half_n).inverse();

  inverse_mapping_matrix->template block<half_n, half_n>(0, 0) = A_inv;
  inverse_mapping_matrix->template block<half_n, half_n>(0, half_n).setZero();
  inverse_mapping_matrix->template block<half_n, half_n>(half_n, 0) =
      -D_inv * C * A_inv;
  inverse_mapping_matrix->template block<half_n, half_n>(half_n, half_n) =
      D_inv;
}

template <int _N>
void PolynomialOptimization<_N>::setupConstraintReorderingMatrix() {
  typedef Eigen::Triplet<double> Triplet;
  std::vector<Triplet> reordering_list;

  const size_t n_vertices = vertices_.size();

  std::vector<Constraint> all_constraints;
  std::set<Constraint> fixed_constraints;
  std::set<Constraint> free_constraints;

  all_constraints.reserve(
      n_vertices_ * N /
      2);  // Will have exactly this number of elements in the end.

  for (size_t vertex_idx = 0; vertex_idx < n_vertices; ++vertex_idx) {
    const Vertex& vertex = vertices_[vertex_idx];

    // Extract constraints and sort them to fixed and free. For the start and
    // end Vertex, we need to do this once, while we need to do it twice for the
    // other vertices, since constraints are shared and enforce continuity.
    int n_constraint_occurence = 2;
    if (vertex_idx == 0 || vertex_idx == (n_segments_))
      n_constraint_occurence = 1;
    for (int co = 0; co < n_constraint_occurence; ++co) {
      for (size_t constraint_idx = 0; constraint_idx < N / 2;
           ++constraint_idx) {
        Constraint constraint;
        constraint.vertex_idx = vertex_idx;
        constraint.constraint_idx = constraint_idx;
```

(continues on next page)

(continued from previous page)

```
        bool has_constraint =
            vertex.getConstraint(constraint_idx, &(constraint.value));
        if (has_constraint) {
          all_constraints.push_back(constraint);
          fixed_constraints.insert(constraint);
        } else {
          constraint.value = Vertex::ConstraintValue::Constant(dimension_, 0);
          all_constraints.push_back(constraint);
          free_constraints.insert(constraint);
        }
      }
    }
  }

  n_all_constraints_ = all_constraints.size();
  n_fixed_constraints_ = fixed_constraints.size();
  n_free_constraints_ = free_constraints.size();

  reordering_list.reserve(n_all_constraints_);
  constraint_reordering_ = Eigen::SparseMatrix<double>(
      n_all_constraints_, n_fixed_constraints_ + n_free_constraints_);

  for (Eigen::VectorXd& df : fixed_constraints_compact_)
    df.resize(n_fixed_constraints_, Eigen::NoChange);

  int row = 0;
  int col = 0;
  for (const Constraint& ca : all_constraints) {
    for (const Constraint& cf : fixed_constraints) {
      if (ca == cf) {
        reordering_list.emplace_back(Triplet(row, col, 1.0));
        for (size_t d = 0; d < dimension_; ++d) {
          Eigen::VectorXd& df = fixed_constraints_compact_[d];
          const Eigen::VectorXd constraint_all_dimensions = cf.value;
          df[col] = constraint_all_dimensions[d];
        }
      }
      ++col;
    }
    for (const Constraint& cp : free_constraints) {
      if (ca == cp) reordering_list.emplace_back(Triplet(row, col, 1.0));
      ++col;
    }
    col = 0;
    ++row;
  }

  constraint_reordering_.setFromTriplets(reordering_list.begin(),
                                         reordering_list.end());
}

template <int _N>
void PolynomialOptimization<_N>::updateSegmentsFromCompactConstraints() {
  const size_t n_all_constraints = n_fixed_constraints_ + n_free_constraints_;

  for (size_t dimension_idx = 0; dimension_idx < dimension_; ++dimension_idx) {
    const Eigen::VectorXd& df = fixed_constraints_compact_[dimension_idx];
```

(continues on next page)

(continued from previous page)

```
    const Eigen::VectorXd& dp_opt = free_constraints_compact_[dimension_idx];

    Eigen::VectorXd d_all(n_all_constraints);
    d_all << df, dp_opt;

    for (size_t i = 0; i < n_segments_; ++i) {
      const Eigen::Matrix<double, N, 1> new_d =
          constraint_reordering_.block(i * N, 0, N, n_all_constraints) * d_all;
      const Eigen::Matrix<double, N, 1> coeffs =
          inverse_mapping_matrices_[i] * new_d;
      Segment& segment = segments_[i];
      segment.setTime(segment_times_[i]);
      segment[dimension_idx] = Polynomial(N, coeffs);
    }
  }
}

template <int _N>
void PolynomialOptimization<_N>::updateSegmentTimes(
    const std::vector<double>& segment_times) {
  const size_t n_segment_times = segment_times.size();
  CHECK(n_segment_times == n_segments_)
      << "Number of segment times (" << n_segment_times
      << ") does not match number of segments (" << n_segments_ << ")";

  segment_times_ = segment_times;

  for (size_t i = 0; i < n_segments_; ++i) {
    const double segment_time = segment_times[i];
    CHECK_GT(segment_time, 0) << "Segment times need to be greater than zero";

    computeQuadraticCostJacobian(derivative_to_optimize_, segment_time,
                                 &cost_matrices_[i]);
    SquareMatrix A;
    setupMappingMatrix(segment_time, &A);
    invertMappingMatrix(A, &inverse_mapping_matrices_[i]);
  };
}

template <int _N>
void PolynomialOptimization<_N>::constructR(
    Eigen::SparseMatrix<double>* R) const {
  CHECK_NOTNULL(R);
  typedef Eigen::Triplet<double> Triplet;
  std::vector<Triplet> cost_unconstrained_triplets;
  cost_unconstrained_triplets.reserve(N * N * n_segments_);

  for (size_t i = 0; i < n_segments_; ++i) {
    const SquareMatrix& Ai = inverse_mapping_matrices_[i];
    const SquareMatrix& Q = cost_matrices_[i];
    const SquareMatrix H = Ai.transpose() * Q * Ai;
    const int start_pos = i * N;
    for (int row = 0; row < N; ++row) {
      for (int col = 0; col < N; ++col) {
        cost_unconstrained_triplets.emplace_back(
            Triplet(start_pos + row, start_pos + col, H(row, col)));
      }
```

(continues on next page)

(continued from previous page)

```
    }
  }
  Eigen::SparseMatrix<double> cost_unconstrained(N * n_segments_,
                                                 N * n_segments_);
  cost_unconstrained.setFromTriplets(cost_unconstrained_triplets.begin(),
                                     cost_unconstrained_triplets.end());

  // [1]: R = C^T * H * C. C: constraint_reodering_ ; H: cost_unconstrained,
  // assembled from the block-H above.
  *R = constraint_reordering_.transpose() * cost_unconstrained *
       constraint_reordering_;
}

template <int _N>
bool PolynomialOptimization<_N>::solveLinear() {
  CHECK(derivative_to_optimize_ >= 0 &&
        derivative_to_optimize_ <= kHighestDerivativeToOptimize);
  // Catch the fully constrained case:
  if (n_free_constraints_ == 0) {
    LOG(WARNING)
        << "No free constraints set in the vertices. Polynomial can "
           "not be optimized. Outputting fully constrained polynomial.";
    updateSegmentsFromCompactConstraints();
    return true;
  }

  // TODO(acmarkus): figure out if sparse becomes less efficient for small
  // problems, and switch back to dense in case.

  // Compute cost matrix for the unconstrained optimization problem.
  // Block-wise H = A^{-T}QA^{-1} according to [1]
  Eigen::SparseMatrix<double> R;
  constructR(&R);

  // Extract block matrices and prepare solver.
  Eigen::SparseMatrix<double> Rpf = R.block(
      n_fixed_constraints_, 0, n_free_constraints_, n_fixed_constraints_);
  Eigen::SparseMatrix<double> Rpp =
      R.block(n_fixed_constraints_, n_fixed_constraints_, n_free_constraints_,
              n_free_constraints_);
  Eigen::SparseQR<Eigen::SparseMatrix<double>, Eigen::COLAMDOrdering<int>>
      solver;
  solver.compute(Rpp);

  // Compute dp_opt for every dimension.
  for (size_t dimension_idx = 0; dimension_idx < dimension_; ++dimension_idx) {
    Eigen::VectorXd df =
        -Rpf * fixed_constraints_compact_[dimension_idx];  // Rpf = Rfp^T
    free_constraints_compact_[dimension_idx] =
        solver.solve(df);  // dp = -Rpp^-1 * Rpf * df
  }

  updateSegmentsFromCompactConstraints();
  return true;
}

template <int _N>
```

(continues on next page)

(continued from previous page)

```
void PolynomialOptimization<_N>::printReorderingMatrix(
    std::ostream& stream) const {
  stream << "Mapping matrix:\n" << constraint_reordering_ << std::endl;
}

template <int _N>
template <int Derivative>
bool PolynomialOptimization<_N>::computeSegmentMaximumMagnitudeCandidates(
    const Segment& segment, double t_start, double t_stop,
    std::vector<double>* candidates) {
  return computeSegmentMaximumMagnitudeCandidates(Derivative, segment, t_start,
                                                  t_stop, candidates);
}

template <int _N>
bool PolynomialOptimization<_N>::computeSegmentMaximumMagnitudeCandidates(
    int derivative, const Segment& segment, double t_start, double t_stop,
    std::vector<double>* candidates) {
  CHECK(candidates);
  CHECK(N - derivative - 1 > 0) << "N-Derivative-1 has to be greater 0";

  // Use the implementation of this in the segment (template-free) as it's
  // actually faster.
  std::vector<int> dimensions(segment.D());
  std::iota(dimensions.begin(), dimensions.end(), 0);
  return segment.computeMinMaxMagnitudeCandidateTimes(
      derivative, t_start, t_stop, dimensions, candidates);
}

template <int _N>
template <int Derivative>
void PolynomialOptimization<_N>::
    computeSegmentMaximumMagnitudeCandidatesBySampling(
        const Segment& segment, double t_start, double t_stop, double dt,
        std::vector<double>* candidates) {
  Eigen::VectorXd value_old, value_start;
  value_start = segment.evaluate(t_start - dt, Derivative);

  value_old = segment.evaluate(t_start, Derivative);

  // Determine initial direction from t_start -dt to t_start.
  // t_start may be an extremum, especially for start and end vertices!
  double direction = value_old.norm() - value_start.norm();

  // Continue with direction from t_start to t_start + dt until t_stop + dt.
  // Again, there may be an extremum at t_stop (e.g. end vertex).
  for (double t = t_start + dt; t < t_stop + dt; t += dt) {
    Eigen::VectorXd value_new;
    value_new = segment.evaluate(t, Derivative);

    double direction_new = value_new.norm() - value_old.norm();

    if (std::signbit(direction) != std::signbit(direction_new)) {
      Eigen::VectorXd value_deriv = segment.evaluate(t - dt, Derivative + 1);
      if (value_deriv.norm() < 1e-2) {
        candidates->push_back(t - dt);  // extremum was at last dt
      }
```

(continues on next page)

(continued from previous page)

```
    }

    value_old = value_new;
    direction = direction_new;
  }
}

template <int _N>
template <int Derivative>
Extremum PolynomialOptimization<_N>::computeMaximumOfMagnitude(
    std::vector<Extremum>* candidates) const {
  return computeMaximumOfMagnitude(Derivative, candidates);
}

template <int _N>
Extremum PolynomialOptimization<_N>::computeMaximumOfMagnitude(
    int derivative, std::vector<Extremum>* candidates) const {
  if (candidates != nullptr) candidates->clear();

  int segment_idx = 0;
  Extremum extremum;
  for (const Segment& s : segments_) {
    std::vector<double> extrema_times;
    extrema_times.reserve(N - 1);
    // Add the beginning as well. Call below appends its extrema.
    extrema_times.push_back(0.0);
    computeSegmentMaximumMagnitudeCandidates(derivative, s, 0.0, s.getTime(),
                                             &extrema_times);

    for (double t : extrema_times) {
      const Extremum candidate(t, s.evaluate(t, derivative).norm(),
                               segment_idx);
      if (extremum < candidate) extremum = candidate;
      if (candidates != nullptr) candidates->emplace_back(candidate);
    }
    ++segment_idx;
  }
  // Check last time at last segment.
  const Extremum candidate(
      segments_.back().getTime(),
      segments_.back().evaluate(segments_.back().getTime(), derivative).norm(),
      n_segments_ - 1);
  if (extremum < candidate) extremum = candidate;
  if (candidates != nullptr) candidates->emplace_back(candidate);

  return extremum;
}

template <int _N>
void PolynomialOptimization<_N>::setFreeConstraints(
    const std::vector<Eigen::VectorXd>& free_constraints) {
  CHECK(free_constraints.size() == dimension_);
  for (const Eigen::VectorXd& v : free_constraints)
    CHECK(static_cast<size_t>(v.size()) == n_free_constraints_);

  free_constraints_compact_ = free_constraints;
  updateSegmentsFromCompactConstraints();
```

(continues on next page)

(continued from previous page)

```
}

template <int _N>
void PolynomialOptimization<_N>::getAInverse(Eigen::MatrixXd* A_inv) const {
  CHECK_NOTNULL(A_inv);

  A_inv->resize(N * n_segments_, N * n_segments_);
  A_inv->setZero();

  for (size_t i = 0; i < n_segments_; ++i) {
    (*A_inv).block<N, N>(N * i, N * i) = inverse_mapping_matrices_[i];
  }
}

template <int _N>
void PolynomialOptimization<_N>::getM(Eigen::MatrixXd* M) const {
  CHECK_NOTNULL(M);
  *M = constraint_reordering_;
}

template <int _N>
void PolynomialOptimization<_N>::getR(Eigen::MatrixXd* R) const {
  CHECK_NOTNULL(R);

  Eigen::SparseMatrix<double> R_sparse;
  constructR(&R_sparse);

  *R = R_sparse;
}

template <int _N>
void PolynomialOptimization<_N>::getA(Eigen::MatrixXd* A) const {
  CHECK_NOTNULL(A);
  A->resize(N * n_segments_, N * n_segments_);
  A->setZero();

  // Create a mapping matrix per segment and append them together.
  for (size_t i = 0; i < n_segments_; ++i) {
    const double segment_time = segment_times_[i];
    CHECK_GT(segment_time, 0) << "Segment times need to be greater than zero";

    SquareMatrix A_segment;
    setupMappingMatrix(segment_time, &A_segment);

    (*A).block<N, N>(N * i, N * i) = A_segment;
  }
}

template <int _N>
void PolynomialOptimization<_N>::getMpinv(Eigen::MatrixXd* M_pinv) const {
  CHECK_NOTNULL(M_pinv);

  // Pseudoinverse implementation by @SebastianInd.
  *M_pinv = constraint_reordering_.transpose();
  for (int M_row = 0; M_row < M_pinv->rows(); M_row++) {
    M_pinv->row(M_row) = M_pinv->row(M_row) / M_pinv->row(M_row).sum();
  }
```

(continues on next page)

(continued from previous page)

```
}

template <int _N>
void PolynomialOptimization<_N>::computeQuadraticCostJacobian(
    int derivative, double t, SquareMatrix* cost_jacobian) {
  CHECK_LT(derivative, N);

  cost_jacobian->setZero();
  for (int col = 0; col < N - derivative; col++) {
    for (int row = 0; row < N - derivative; row++) {
      double exponent = (N - 1 - derivative) * 2 + 1 - row - col;

      (*cost_jacobian)(N - 1 - row, N - 1 - col) =
          Polynomial::base_coefficients_(derivative, N - 1 - row) *
          Polynomial::base_coefficients_(derivative, N - 1 - col) *
          pow(t, exponent) * 2.0 / exponent;
    }
  }
}

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_LINEAR_IMPL_H_
```

Includes

- `Eigen/Sparse`
- `algorithm`
- `glog/logging.h`
- `mav_trajectory_generation/convolution.h` (*File convolution.h*)
- `set`
- `tuple`

Included By

- *File polynomial_optimization_linear.h*

Namespaces

- *Namespace mav_trajectory_generation*

File polynomial_optimization_nonlinear.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_nonlinear.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_nonlinear.h)

Program Listing for File polynomial_optimization_nonlinear.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/polynomial_optimization_nonlinear.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_NONLINEAR_H_
#define MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_NONLINEAR_H_

#include <memory>
#include <nlopt.hpp>

#include "mav_trajectory_generation/polynomial_optimization_linear.h"

namespace mav_trajectory_generation {

constexpr double kOptimizationTimeLowerBound = 0.1;

struct NonlinearOptimizationParameters {

  double f_abs = -1;

  double f_rel = 0.05;

  double x_rel = -1;
```

(continues on next page)

(continued from previous page)

```
  double x_abs = -1;

  double initial_stepsize_rel = 0.1;

  double equality_constraint_tolerance = 1.0e-3;

  double inequality_constraint_tolerance = 0.1;

  int max_iterations = 3000;

  double time_penalty = 500.0;

  nlopt::algorithm algorithm = nlopt::LN_BOBYQA;

  int random_seed = 0;

  bool use_soft_constraints = true;

  double soft_constraint_weight = 100.0;

  enum TimeAllocMethod {
    kSquaredTime,
    kRichterTime,
    kMellingerOuterLoop,
    kSquaredTimeAndConstraints,
    kRichterTimeAndConstraints,
    kUnknown
  } time_alloc_method = kSquaredTimeAndConstraints;

  bool print_debug_info = false;
  bool print_debug_info_time_allocation = false;
};

struct OptimizationInfo {
  int n_iterations = 0;
  int stopping_reason = nlopt::FAILURE;
  double cost_trajectory = 0.0;
  double cost_time = 0.0;
  double cost_soft_constraints = 0.0;
  double optimization_time = 0.0;
  std::map<int, Extremum> maxima;
};

std::ostream& operator<<(std::ostream& stream, const OptimizationInfo& val);

template <int _N = 10>
class PolynomialOptimizationNonLinear {
  static_assert(_N % 2 == 0, "The number of coefficients has to be even.");

 public:
  enum { N = _N };

  PolynomialOptimizationNonLinear(
      size_t dimension, const NonlinearOptimizationParameters& parameters);

  bool setupFromVertices(
```

(continues on next page)

(continued from previous page)

```
      const Vertex::Vector& vertices, const std::vector<double>& segment_times,
      int derivative_to_optimize =
          PolynomialOptimization<N>::kHighestDerivativeToOptimize);

  bool addMaximumMagnitudeConstraint(int derivative_order,
                                     double maximum_value);

  bool solveLinear();

  int optimize();

  void getTrajectory(Trajectory* trajectory) const {
    poly_opt_.getTrajectory(trajectory);
  }

  const PolynomialOptimization<N>& getPolynomialOptimizationRef() const {
    return poly_opt_;
  }

  PolynomialOptimization<N>& getPolynomialOptimizationRef() {
    return poly_opt_;
  }

  OptimizationInfo getOptimizationInfo() const { return optimization_info_; }

  double getCost() const;

  double getTotalCostWithSoftConstraints() const;

  void scaleSegmentTimesWithViolation();

 private:
  struct ConstraintData {
    PolynomialOptimizationNonLinear<N>* this_object;
    int derivative;
    double value;
  };

  static double objectiveFunctionTime(const std::vector<double>& segment_times,
                                      std::vector<double>& gradient,
                                      void* data);

  static double objectiveFunctionTimeMellingerOuterLoop(
      const std::vector<double>& segment_times, std::vector<double>& gradient,
      void* data);

  static double objectiveFunctionTimeAndConstraints(
      const std::vector<double>& optimization_variables,
      std::vector<double>& gradient, void* data);

  static double evaluateMaximumMagnitudeConstraint(
      const std::vector<double>& optimization_variables,
      std::vector<double>& gradient, void* data);

  int optimizeTime();
  int optimizeTimeMellingerOuterLoop();
```

(continues on next page)

(continued from previous page)

```
  int optimizeTimeAndFreeConstraints();

  double evaluateMaximumMagnitudeAsSoftConstraint(
      const std::vector<std::shared_ptr<ConstraintData> >&
          inequality_constraints,
      double weight, double maximum_cost = 1.0e12) const;

  void setFreeEndpointDerivativeHardConstraints(
      const Vertex::Vector& vertices, std::vector<double>* lower_bounds,
      std::vector<double>* upper_bounds);

  double getCostAndGradientMellinger(std::vector<double>* gradients);

  static double computeTotalTrajectoryTime(
      const std::vector<double>& segment_times);

  std::shared_ptr<nlopt::opt> nlopt_;

  PolynomialOptimization<N> poly_opt_;

  NonlinearOptimizationParameters optimization_parameters_;

  std::vector<std::shared_ptr<ConstraintData> > inequality_constraints_;

  OptimizationInfo optimization_info_;
};

}  // namespace mav_trajectory_generation

namespace nlopt {
std::string returnValueToString(int return_value);
}  // namespace nlopt

#endif  // MAV_TRAJECTORY_GENERATION_POLYNOMIAL_OPTIMIZATION_NONLINEAR_H_

#include "mav_trajectory_generation/impl/polynomial_optimization_nonlinear_impl.h"
```

Includes

- `mav_trajectory_generation/impl/polynomial_optimization_nonlinear_impl.h` (*File polynomial_optimization_nonlinear_impl.h*)
- `mav_trajectory_generation/polynomial_optimization_linear.h` (*File polynomial_optimization_linear.h*)
- `memory`
- `nlopt.hpp`

Namespaces

- *Namespace mav_trajectory_generation*
- *Namespace nlopt*

Classes

- *Struct NonlinearOptimizationParameters*
- *Struct OptimizationInfo*
- *Struct PolynomialOptimizationNonLinear::ConstraintData*
- *Template Class PolynomialOptimizationNonLinear*

File polynomial_optimization_nonlinear_impl.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_nonlinear_impl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_nonlinear_impl.h`)

Program Listing for File polynomial_optimization_nonlinear_impl.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/impl/polynomial_optimization_nonlinear_impl.h`)

```
/*
* Copyright (c) 2015, Markus Achtelik, ASL, ETH Zurich, Switzerland
* You can contact the author at <markus dot achtelik at mavt dot ethz dot ch>
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

#ifndef MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_NONLINEAR_IMPL_H_
#define MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_NONLINEAR_IMPL_H_

#include <chrono>
#include <numeric>
```

(continues on next page)

(continued from previous page)

```
#include "mav_trajectory_generation/polynomial_optimization_linear.h"
#include "mav_trajectory_generation/timing.h"

namespace mav_trajectory_generation {

inline std::ostream& operator<<(std::ostream& stream,
                                const OptimizationInfo& val) {
  stream << "--- optimization info ---" << std::endl;
  stream << "  optimization time:     " << val.optimization_time << std::endl;
  stream << "  n_iterations:          " << val.n_iterations << std::endl;
  stream << "  stopping reason:       "
         << nlopt::returnValueToString(val.stopping_reason) << std::endl;
  stream << "  cost trajectory:       " << val.cost_trajectory << std::endl;
  stream << "  cost time:             " << val.cost_time << std::endl;
  stream << "  cost soft constraints: " << val.cost_soft_constraints
         << std::endl;
  stream << "  maxima: " << std::endl;
  for (const std::pair<int, Extremum>& m : val.maxima) {
    stream << "    " << positionDerivativeToString(m.first) << ": "
           << m.second.value << " in segment " << m.second.segment_idx
           << " and segment time " << m.second.time << std::endl;
  }
  return stream;
}

template <int _N>
PolynomialOptimizationNonLinear<_N>::PolynomialOptimizationNonLinear(
    size_t dimension, const NonlinearOptimizationParameters& parameters)
    : poly_opt_(dimension), optimization_parameters_(parameters) {}

template <int _N>
bool PolynomialOptimizationNonLinear<_N>::setupFromVertices(
    const Vertex::Vector& vertices, const std::vector<double>& segment_times,
    int derivative_to_optimize) {
  bool ret = poly_opt_.setupFromVertices(vertices, segment_times,
                                         derivative_to_optimize);

  size_t n_optimization_parameters;
  switch (optimization_parameters_.time_alloc_method) {
    case NonlinearOptimizationParameters::kSquaredTime:
    case NonlinearOptimizationParameters::kRichterTime:
    case NonlinearOptimizationParameters::kMellingerOuterLoop:
      n_optimization_parameters = segment_times.size();
      break;
    default:
      n_optimization_parameters =
          segment_times.size() +
          poly_opt_.getNumberFreeConstraints() * poly_opt_.getDimension();
      break;
  }

  nlopt_.reset(new nlopt::opt(optimization_parameters_.algorithm,
                              n_optimization_parameters));
  nlopt_->set_ftol_rel(optimization_parameters_.f_rel);
  nlopt_->set_ftol_abs(optimization_parameters_.f_abs);
  nlopt_->set_xtol_rel(optimization_parameters_.x_rel);
  nlopt_->set_xtol_abs(optimization_parameters_.x_abs);
```

(continues on next page)

(continued from previous page)

```
  nlopt_->set_maxeval(optimization_parameters_.max_iterations);

  if (optimization_parameters_.random_seed < 0)
    nlopt_srand_time();
  else
    nlopt_srand(optimization_parameters_.random_seed);

  return ret;
}

template <int _N>
bool PolynomialOptimizationNonLinear<_N>::solveLinear() {
  return poly_opt_.solveLinear();
}

template <int _N>
int PolynomialOptimizationNonLinear<_N>::optimize() {
  optimization_info_ = OptimizationInfo();
  int result = nlopt::FAILURE;

  const std::chrono::high_resolution_clock::time_point t_start =
      std::chrono::high_resolution_clock::now();

  switch (optimization_parameters_.time_alloc_method) {
    case NonlinearOptimizationParameters::kSquaredTime:
    case NonlinearOptimizationParameters::kRichterTime:
      result = optimizeTime();
      break;
    case NonlinearOptimizationParameters::kSquaredTimeAndConstraints:
    case NonlinearOptimizationParameters::kRichterTimeAndConstraints:
      result = optimizeTimeAndFreeConstraints();
      break;
    case NonlinearOptimizationParameters::kMellingerOuterLoop:
      result = optimizeTimeMellingerOuterLoop();
      break;
    default:
      break;
  }

  const std::chrono::high_resolution_clock::time_point t_stop =
      std::chrono::high_resolution_clock::now();
  optimization_info_.optimization_time =
      std::chrono::duration_cast<std::chrono::duration<double> >(t_stop -
                                                                 t_start)
          .count();

  optimization_info_.stopping_reason = result;

  return result;
}

template <int _N>
int PolynomialOptimizationNonLinear<_N>::optimizeTime() {
  std::vector<double> initial_step, segment_times;

  poly_opt_.getSegmentTimes(&segment_times);
  const size_t n_segments = segment_times.size();
```

(continues on next page)

(continued from previous page)

```
  initial_step.reserve(n_segments);
  for (double t : segment_times) {
    initial_step.push_back(optimization_parameters_.initial_stepsize_rel * t);
  }

  try {
    // Set a lower bound on the segment time per segment to avoid numerical
    // issues.
    nlopt_->set_initial_step(initial_step);
    nlopt_->set_upper_bounds(std::numeric_limits<double>::max());
    nlopt_->set_lower_bounds(kOptimizationTimeLowerBound);
    nlopt_->set_min_objective(
        &PolynomialOptimizationNonLinear<N>::objectiveFunctionTime, this);
  } catch (std::exception& e) {
    LOG(ERROR) << "error while setting up nlopt: " << e.what() << std::endl;
    return nlopt::FAILURE;
  }

  double final_cost = std::numeric_limits<double>::max();
  int result;

  try {
    result = nlopt_->optimize(segment_times, final_cost);
  } catch (std::exception& e) {
    LOG(ERROR) << "error while running nlopt: " << e.what() << std::endl;
    return nlopt::FAILURE;
  }

  return result;
}

template <int _N>
int PolynomialOptimizationNonLinear<_N>::optimizeTimeMellingerOuterLoop() {
  std::vector<double> segment_times;
  poly_opt_.getSegmentTimes(&segment_times);

  // Save original segment times
  std::vector<double> original_segment_times = segment_times;

  if (optimization_parameters_.print_debug_info_time_allocation) {
    std::cout << "Segment times: ";
    for (const double seg_time : segment_times) {
      std::cout << seg_time << " ";
    }
    std::cout << std::endl;
  }

  try {
    // Set a lower bound on the segment time per segment to avoid numerical
    // issues.
    nlopt_->set_upper_bounds(std::numeric_limits<double>::max());
    nlopt_->set_lower_bounds(kOptimizationTimeLowerBound);
    nlopt_->set_min_objective(&PolynomialOptimizationNonLinear<
                                  N>::objectiveFunctionTimeMellingerOuterLoop,
                              this);
  } catch (std::exception& e) {
```

(continues on next page)

(continued from previous page)

```
    LOG(ERROR) << "error while setting up nlopt: " << e.what() << std::endl;
    return nlopt::FAILURE;
  }

  double final_cost = std::numeric_limits<double>::max();
  int result = nlopt::FAILURE;

  try {
    result = nlopt_->optimize(segment_times, final_cost);
  } catch (std::exception& e) {
    LOG(ERROR) << "error while running nlopt: " << e.what()
               << ". This likely means the optimization aborted early."
               << std::endl;
    if (final_cost == std::numeric_limits<double>::max()) {
      return nlopt::FAILURE;
    }

    if (optimization_parameters_.print_debug_info_time_allocation) {
      std::cout << "Segment times after opt: ";
      for (const double seg_time : segment_times) {
        std::cout << seg_time << " ";
      }
      std::cout << std::endl;
      std::cout << "Final cost: " << final_cost << std::endl;
      std::cout << "Nlopt result: " << result << std::endl;
    }
  }

  // Scaling of segment times
  std::vector<double> relative_segment_times;
  poly_opt_.getSegmentTimes(&relative_segment_times);
  scaleSegmentTimesWithViolation();
  std::vector<double> scaled_segment_times;
  poly_opt_.getSegmentTimes(&scaled_segment_times);

  // Print all parameter after scaling
  if (optimization_parameters_.print_debug_info_time_allocation) {
    std::cout << "[MEL          Original]: ";
    std::for_each(original_segment_times.cbegin(),
                  original_segment_times.cend(),
                  [](double c) { std::cout << c << " "; });
    std::cout << std::endl;
    std::cout << "[MEL RELATIVE Solution]: ";
    std::for_each(relative_segment_times.cbegin(),
                  relative_segment_times.cend(),
                  [](double c) { std::cout << c << " "; });
    std::cout << std::endl;
    std::cout << "[MEL          Solution]: ";
    std::for_each(scaled_segment_times.cbegin(), scaled_segment_times.cend(),
                  [](double c) { std::cout << c << " "; });
    std::cout << std::endl;
    std::cout << "[MEL   Trajectory Time] Before: "
              << std::accumulate(original_segment_times.begin(),
                                 original_segment_times.end(), 0.0)
              << " | After Rel Change: "
              << std::accumulate(relative_segment_times.begin(),
                                 relative_segment_times.end(), 0.0)
```

(continues on next page)

(continued from previous page)

```
              << " | After Scaling: "
              << std::accumulate(scaled_segment_times.begin(),
                                 scaled_segment_times.end(), 0.0)
              << std::endl;
  }

  return result;
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::getCost() const {
  return poly_opt_.computeCost();
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::getTotalCostWithSoftConstraints()
    const {
  double cost_trajectory = poly_opt_.computeCost();

  // Use consistent cost metrics regardless of method set, to compare between
  // methods.
  std::vector<double> segment_times;
  poly_opt_.getSegmentTimes(&segment_times);
  double total_time =
      std::accumulate(segment_times.begin(), segment_times.end(), 0.0);
  double cost_time =
      total_time * total_time * optimization_parameters_.time_penalty;
  double cost_constraints = evaluateMaximumMagnitudeAsSoftConstraint(
      inequality_constraints_, optimization_parameters_.soft_constraint_weight,
      1e9);

  return cost_trajectory + cost_time + cost_constraints;
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::getCostAndGradientMellinger(
    std::vector<double>* gradients) {
  // Weighting terms for different costs
  // Retrieve the current segment times
  std::vector<double> segment_times;
  poly_opt_.getSegmentTimes(&segment_times);
  const double J_d = poly_opt_.computeCost();

  if (poly_opt_.getNumberSegments() == 1) {
    if (gradients != NULL) {
      gradients->clear();
      gradients->resize(poly_opt_.getNumberSegments(), 0.0);
    }

    return J_d;
  }

  if (gradients != NULL) {
    const size_t n_segments = poly_opt_.getNumberSegments();

    gradients->clear();
    gradients->resize(n_segments);
```

(continues on next page)

(continued from previous page)

```
    // Initialize changed segment times for numerical derivative
    std::vector<double> segment_times_bigger(n_segments);
    const double increment_time = 0.1;
    for (size_t n = 0; n < n_segments; ++n) {
      // Now the same with an increased segment time
      // Calculate cost with higher segment time
      segment_times_bigger = segment_times;
      // Deduct h*(-1/(m-2)) according to paper Mellinger "Minimum snap
      // trajectory generation and control for quadrotors"
      double const_traj_time_corr = increment_time / (n_segments - 1.0);
      for (size_t i = 0; i < segment_times_bigger.size(); ++i) {
        if (i == n) {
          segment_times_bigger[i] += increment_time;
        } else {
          segment_times_bigger[i] -= const_traj_time_corr;
        }
      }

      // TODO: add case if segment_time is at threshold 0.1s
      // 1) How many segments > 0.1s
      // 2) trajectory time correction only on those
      // for (int j = 0; j < segment_times_bigger.size(); ++j) {
      //   double thresh_corr = 0.0;
      //   if (segment_times_bigger[j] < 0.1) {
      //     thresh_corr = 0.1-segment_times_bigger[j];
      //   }
      // }

      // Check and make sure that segment times are >
      // kOptimizationTimeLowerBound
      for (double& t : segment_times_bigger) {
        t = std::max(kOptimizationTimeLowerBound, t);
      }

      // Update the segment times. This changes the polynomial coefficients.
      poly_opt_.updateSegmentTimes(segment_times_bigger);
      poly_opt_.solveLinear();

      // Calculate cost and gradient with new segment time
      const double J_d_bigger = poly_opt_.computeCost();
      const double dJd_dt = (J_d_bigger - J_d) / increment_time;

      // Calculate the gradient
      gradients->at(n) = dJd_dt;
    }

    // Set again the original segment times from before calculating the
    // numerical gradient
    poly_opt_.updateSegmentTimes(segment_times);
    poly_opt_.solveLinear();
  }

  // Compute cost without gradient
  return J_d;
}
```

(continues on next page)

(continued from previous page)

```
template <int _N>
void PolynomialOptimizationNonLinear<_N>::scaleSegmentTimesWithViolation() {
  // Get trajectory
  Trajectory traj;
  poly_opt_.getTrajectory(&traj);

  // Get constraints
  double v_max = 0.0;
  double a_max = 0.0;
  for (const auto& constraint : inequality_constraints_) {
    if (constraint->derivative == derivative_order::VELOCITY) {
      v_max = constraint->value;
    } else if (constraint->derivative == derivative_order::ACCELERATION) {
      a_max = constraint->value;
    }
  }

  if (optimization_parameters_.print_debug_info_time_allocation) {
    double v_max_actual, a_max_actual;
    traj.computeMaxVelocityAndAcceleration(&v_max_actual, &a_max_actual);
    std::cout << "[Time Scaling] Beginning:  v: max: " << v_max_actual << " / "
              << v_max << " a: max: " << a_max_actual << " / " << a_max
              << std::endl;
  }

  // Run the trajectory time scaling.
  traj.scaleSegmentTimesToMeetConstraints(v_max, a_max);

  std::vector<double> segment_times;
  segment_times = traj.getSegmentTimes();
  poly_opt_.updateSegmentTimes(segment_times);
  poly_opt_.solveLinear();

  if (optimization_parameters_.print_debug_info_time_allocation) {
    double v_max_actual, a_max_actual;
    traj.computeMaxVelocityAndAcceleration(&v_max_actual, &a_max_actual);
    std::cout << "[Time Scaling] End: v: max: " << v_max_actual << " / "
              << v_max << " a: max: " << a_max_actual << " / " << a_max
              << std::endl;
  }
}

template <int _N>
int PolynomialOptimizationNonLinear<_N>::optimizeTimeAndFreeConstraints() {
  std::vector<double> initial_step, initial_solution, segment_times,
      lower_bounds, upper_bounds;

  poly_opt_.getSegmentTimes(&segment_times);
  const size_t n_segments = segment_times.size();

  // compute initial solution
  poly_opt_.solveLinear();
  std::vector<Eigen::VectorXd> free_constraints;
  poly_opt_.getFreeConstraints(&free_constraints);
  if (free_constraints.size() == 0 || free_constraints.front().size() == 0) {
    LOG(WARNING)
        << "No free derivative variables, same as time-only optimization.";
```

(continues on next page)

(continued from previous page)

```
}

const size_t n_optimization_variables =
    n_segments + free_constraints.size() * free_constraints.front().size();

CHECK_GT(n_optimization_variables, 0u);

initial_solution.reserve(n_optimization_variables);
initial_step.reserve(n_optimization_variables);
lower_bounds.reserve(n_optimization_variables);
upper_bounds.reserve(n_optimization_variables);

// copy all constraints into one vector:
for (double t : segment_times) {
  initial_solution.push_back(t);
}

for (const Eigen::VectorXd& c : free_constraints) {
  for (int i = 0; i < c.size(); ++i) {
    initial_solution.push_back(c[i]);
  }
}

// Setup for getting bounds on the free endpoint derivatives
std::vector<double> lower_bounds_free, upper_bounds_free;
const size_t n_optimization_variables_free =
    free_constraints.size() * free_constraints.front().size();
lower_bounds_free.reserve(n_optimization_variables_free);
upper_bounds_free.reserve(n_optimization_variables_free);

// Get the lower and upper bounds constraints on the free endpoint
// derivatives
Vertex::Vector vertices;
poly_opt_.getVertices(&vertices);
setFreeEndpointDerivativeHardConstraints(vertices, &lower_bounds_free,
                                         &upper_bounds_free);

// Set segment time constraints
for (size_t l = 0; l < n_segments; ++l) {
  lower_bounds.push_back(kOptimizationTimeLowerBound);
  upper_bounds.push_back(std::numeric_limits<double>::max());
}
// Append free endpoint derivative constraints
lower_bounds.insert(std::end(lower_bounds), std::begin(lower_bounds_free),
                    std::end(lower_bounds_free));
upper_bounds.insert(std::end(upper_bounds), std::begin(upper_bounds_free),
                    std::end(upper_bounds_free));

for (size_t i = 0; i < initial_solution.size(); i++) {
  double x = initial_solution[i];
  const double abs_x = std::abs(x);
  // Initial step size cannot be 0.0 --> invalid arg
  if (abs_x <= std::numeric_limits<double>::lowest()) {
    initial_step.push_back(1e-13);
  } else {
    initial_step.push_back(optimization_parameters_.initial_stepsize_rel *
                           abs_x);
```

(continues on next page)

(continued from previous page)

```
    }

    // Check if initial solution isn't already out of bounds.
    if (x < lower_bounds[i]) {
      lower_bounds[i] = x;
    } else if (x > upper_bounds[i]) {
      upper_bounds[i] = x;
    }
  }

  // Make sure everything is the same size, otherwise NLOPT will have a bad
  // time.
  CHECK_EQ(lower_bounds.size(), upper_bounds.size());
  CHECK_EQ(initial_solution.size(), lower_bounds.size());
  CHECK_EQ(initial_solution.size(), initial_step.size());
  CHECK_EQ(initial_solution.size(), n_optimization_variables);

  try {
    nlopt_->set_initial_step(initial_step);
    nlopt_->set_lower_bounds(lower_bounds);
    nlopt_->set_upper_bounds(upper_bounds);
    nlopt_->set_min_objective(&PolynomialOptimizationNonLinear<
                                  N>::objectiveFunctionTimeAndConstraints,
                              this);
  } catch (std::exception& e) {
    LOG(ERROR) << "error while setting up nlopt: " << e.what() << std::endl;
    return nlopt::FAILURE;
  }

  double final_cost = std::numeric_limits<double>::max();
  int result;

  try {
    timing::Timer timer_solve("optimize_nonlinear_full_total_time");

    result = nlopt_->optimize(initial_solution, final_cost);
    timer_solve.Stop();
  } catch (std::exception& e) {
    LOG(ERROR) << "error while running nlopt: " << e.what() << std::endl;
    return nlopt::FAILURE;
  }

  return result;
}

template <int _N>
bool PolynomialOptimizationNonLinear<_N>::addMaximumMagnitudeConstraint(
    int derivative, double maximum_value) {
  CHECK_GE(derivative, 0);
  CHECK_GE(maximum_value, 0.0);

  std::shared_ptr<ConstraintData> constraint_data(new ConstraintData);
  constraint_data->derivative = derivative;
  constraint_data->value = maximum_value;
  constraint_data->this_object = this;

  // Store the shared_ptrs such that their data will be destroyed later.
```

(continues on next page)

(continued from previous page)

```
  inequality_constraints_.push_back(constraint_data);

  if (!optimization_parameters_.use_soft_constraints) {
    try {
      nlopt_->add_inequality_constraint(
          &PolynomialOptimizationNonLinear<
              N>::evaluateMaximumMagnitudeConstraint,
          constraint_data.get(),
          optimization_parameters_.inequality_constraint_tolerance);
    } catch (std::exception& e) {
      LOG(ERROR) << "ERROR while setting inequality constraint " << e.what()
                 << std::endl;
      return false;
    }
  }
  return true;
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::objectiveFunctionTime(
    const std::vector<double>& segment_times, std::vector<double>& gradient,
    void* data) {
  CHECK(gradient.empty())
      << "computing gradient not possible, choose a gradient free method";
  CHECK_NOTNULL(data);

  PolynomialOptimizationNonLinear<N>* optimization_data =
      static_cast<PolynomialOptimizationNonLinear<N>*>(data);  // wheee ...

  CHECK_EQ(segment_times.size(),
           optimization_data->poly_opt_.getNumberSegments());

  optimization_data->poly_opt_.updateSegmentTimes(segment_times);
  optimization_data->poly_opt_.solveLinear();
  double cost_trajectory = optimization_data->poly_opt_.computeCost();
  double cost_time = 0;
  double cost_constraints = 0;
  const double total_time = computeTotalTrajectoryTime(segment_times);

  switch (optimization_data->optimization_parameters_.time_alloc_method) {
    case NonlinearOptimizationParameters::kRichterTime:
      cost_time =
          total_time * optimization_data->optimization_parameters_.time_penalty;
      break;
    default:  // kSquaredTime
      cost_time = total_time * total_time *
                  optimization_data->optimization_parameters_.time_penalty;
      break;
  }

  if (optimization_data->optimization_parameters_.print_debug_info) {
    std::cout << "---- cost at iteration "
              << optimization_data->optimization_info_.n_iterations << "---- "
              << std::endl;
    std::cout << "  trajectory: " << cost_trajectory << std::endl;
    std::cout << "  time: " << cost_time << std::endl;
  }
```

(continues on next page)

(continued from previous page)

```
  if (optimization_data->optimization_parameters_.use_soft_constraints) {
    cost_constraints =
        optimization_data->evaluateMaximumMagnitudeAsSoftConstraint(
            optimization_data->inequality_constraints_,
            optimization_data->optimization_parameters_.soft_constraint_weight);
  }

  if (optimization_data->optimization_parameters_.print_debug_info) {
    std::cout << "  sum: " << cost_trajectory + cost_time + cost_constraints
              << std::endl;
    std::cout << "  total time: " << total_time << std::endl;
  }

  optimization_data->optimization_info_.n_iterations++;
  optimization_data->optimization_info_.cost_trajectory = cost_trajectory;
  optimization_data->optimization_info_.cost_time = cost_time;
  optimization_data->optimization_info_.cost_soft_constraints =
      cost_constraints;

  return cost_trajectory + cost_time + cost_constraints;
}

template <int _N>
double
PolynomialOptimizationNonLinear<_N>::objectiveFunctionTimeMellingerOuterLoop(
    const std::vector<double>& segment_times, std::vector<double>& gradient,
    void* data) {
  CHECK(!gradient.empty())
      << "only with gradients possible, choose a gradient based method";
  CHECK_NOTNULL(data);

  PolynomialOptimizationNonLinear<N>* optimization_data =
      static_cast<PolynomialOptimizationNonLinear<N>*>(data);  // wheee ...

  CHECK_EQ(segment_times.size(),
           optimization_data->poly_opt_.getNumberSegments());

  optimization_data->poly_opt_.updateSegmentTimes(segment_times);
  optimization_data->poly_opt_.solveLinear();
  double cost_trajectory;
  if (!gradient.empty()) {
    cost_trajectory = optimization_data->getCostAndGradientMellinger(&gradient);
  } else {
    cost_trajectory = optimization_data->getCostAndGradientMellinger(NULL);
  }

  if (optimization_data->optimization_parameters_.print_debug_info) {
    std::cout << "---- cost at iteration "
              << optimization_data->optimization_info_.n_iterations << "---- "
              << std::endl;
    std::cout << "  segment times: ";
    for (double segment_time : segment_times) {
      std::cout << segment_time << " ";
    }
    std::cout << std::endl;
    std::cout << "  sum: " << cost_trajectory << std::endl;
```

(continues on next page)

(continued from previous page)

```cpp
  }

  optimization_data->optimization_info_.n_iterations++;
  optimization_data->optimization_info_.cost_trajectory = cost_trajectory;

  return cost_trajectory;
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::objectiveFunctionTimeAndConstraints(
    const std::vector<double>& x, std::vector<double>& gradient, void* data) {
  CHECK(gradient.empty())
      << "computing gradient not possible, choose a gradient-free method";
  CHECK_NOTNULL(data);

  PolynomialOptimizationNonLinear<N>* optimization_data =
      static_cast<PolynomialOptimizationNonLinear<N>*>(data);  // wheee ...

  const size_t n_segments = optimization_data->poly_opt_.getNumberSegments();
  const size_t n_free_constraints =
      optimization_data->poly_opt_.getNumberFreeConstraints();
  const size_t dim = optimization_data->poly_opt_.getDimension();

  CHECK_EQ(x.size(), n_segments + n_free_constraints * dim);

  std::vector<Eigen::VectorXd> free_constraints;
  free_constraints.resize(dim);
  std::vector<double> segment_times;
  segment_times.reserve(n_segments);

  for (size_t i = 0; i < n_segments; ++i) {
    segment_times.push_back(x[i]);
  }

  for (size_t d = 0; d < dim; ++d) {
    const size_t idx_start = n_segments + d * n_free_constraints;

    Eigen::VectorXd& free_constraints_dim = free_constraints[d];
    free_constraints_dim.resize(n_free_constraints, Eigen::NoChange);
    for (size_t i = 0; i < n_free_constraints; ++i) {
      free_constraints_dim[i] = x[idx_start + i];
    }
  }

  optimization_data->poly_opt_.updateSegmentTimes(segment_times);
  optimization_data->poly_opt_.setFreeConstraints(free_constraints);

  double cost_trajectory = optimization_data->poly_opt_.computeCost();
  double cost_time = 0;
  double cost_constraints = 0;

  const double total_time = computeTotalTrajectoryTime(segment_times);
  switch (optimization_data->optimization_parameters_.time_alloc_method) {
    case NonlinearOptimizationParameters::kRichterTimeAndConstraints:
      cost_time =
          total_time * optimization_data->optimization_parameters_.time_penalty;
      break;
```

(continues on next page)

(continued from previous page)

```
    default:  // kSquaredTimeAndConstraints
      cost_time = total_time * total_time *
                  optimization_data->optimization_parameters_.time_penalty;
      break;
  }

  if (optimization_data->optimization_parameters_.print_debug_info) {
    std::cout << "---- cost at iteration "
              << optimization_data->optimization_info_.n_iterations << "---- "
              << std::endl;
    std::cout << "  trajectory: " << cost_trajectory << std::endl;
    std::cout << "  time: " << cost_time << std::endl;
  }

  if (optimization_data->optimization_parameters_.use_soft_constraints) {
    cost_constraints =
        optimization_data->evaluateMaximumMagnitudeAsSoftConstraint(
            optimization_data->inequality_constraints_,
            optimization_data->optimization_parameters_.soft_constraint_weight);
  }

  if (optimization_data->optimization_parameters_.print_debug_info) {
    std::cout << "  sum: " << cost_trajectory + cost_time + cost_constraints
              << std::endl;
    std::cout << "  total time: " << total_time << std::endl;
  }

  optimization_data->optimization_info_.n_iterations++;
  optimization_data->optimization_info_.cost_trajectory = cost_trajectory;
  optimization_data->optimization_info_.cost_time = cost_time;
  optimization_data->optimization_info_.cost_soft_constraints =
      cost_constraints;

  return cost_trajectory + cost_time + cost_constraints;
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::evaluateMaximumMagnitudeConstraint(
    const std::vector<double>& segment_times, std::vector<double>& gradient,
    void* data) {
  CHECK(gradient.empty())
      << "computing gradient not possible, choose a gradient-free method";
  ConstraintData* constraint_data =
      static_cast<ConstraintData*>(data);  // wheeee ...
  PolynomialOptimizationNonLinear<N>* optimization_data =
      constraint_data->this_object;

  Extremum max;
  max = optimization_data->poly_opt_.computeMaximumOfMagnitude(
      constraint_data->derivative, nullptr);

  optimization_data->optimization_info_.maxima[constraint_data->derivative] =
      max;

  return max.value - constraint_data->value;
}
```

(continues on next page)

(continued from previous page)

```
template <int _N>
double
PolynomialOptimizationNonLinear<_N>::evaluateMaximumMagnitudeAsSoftConstraint(
    const std::vector<std::shared_ptr<ConstraintData> >& inequality_constraints,
    double weight, double maximum_cost) const {
  std::vector<double> dummy;
  double cost = 0;

  if (optimization_parameters_.print_debug_info)
    std::cout << "  soft_constraints: " << std::endl;

  for (std::shared_ptr<const ConstraintData> constraint :
       inequality_constraints_) {
    // need to call the c-style callback function here, thus the ugly cast to
    // void*.
    double abs_violation = evaluateMaximumMagnitudeConstraint(
        dummy, dummy, (void*)constraint.get());

    double relative_violation = abs_violation / constraint->value;
    const double current_cost =
        std::min(maximum_cost, exp(relative_violation * weight));
    cost += current_cost;
    if (optimization_parameters_.print_debug_info) {
      std::cout << "    derivative " << constraint->derivative
                << " abs violation: " << abs_violation
                << " : relative violation: " << relative_violation
                << " cost: " << current_cost << std::endl;
    }
  }
  return cost;
}

template <int _N>
void PolynomialOptimizationNonLinear<_N>::
    setFreeEndpointDerivativeHardConstraints(
        const Vertex::Vector& vertices, std::vector<double>* lower_bounds,
        std::vector<double>* upper_bounds) {
  CHECK_NOTNULL(lower_bounds);
  CHECK_NOTNULL(upper_bounds);
  CHECK(lower_bounds->empty()) << "Lower bounds not empty!";
  CHECK(upper_bounds->empty()) << "Upper bounds not empty!";

  const size_t n_free_constraints = poly_opt_.getNumberFreeConstraints();
  const size_t dim = poly_opt_.getDimension();
  const int derivative_to_optimize = poly_opt_.getDerivativeToOptimize();

  // Set all values to -inf/inf and reset only bounded opti param with values
  lower_bounds->resize(dim * n_free_constraints,
                       std::numeric_limits<double>::lowest());
  upper_bounds->resize(dim * n_free_constraints,
                       std::numeric_limits<double>::max());

  // Add higher order derivative constraints (v_max and a_max)
  // Check at each vertex which of the derivatives is a free derivative.
  // If it is a free derivative check if we have a constraint in
  // inequality_constraints_ and set the constraint as hard constraint in
  // lower_bounds and upper_bounds
```

(continues on next page)

(continued from previous page)

```
  for (const auto& constraint_data : inequality_constraints_) {
    unsigned int free_deriv_counter = 0;
    const int derivative_hc = constraint_data->derivative;
    const double value_hc = constraint_data->value;

    for (size_t v = 0; v < vertices.size(); ++v) {
      for (int deriv = 0; deriv <= derivative_to_optimize; ++deriv) {
        if (!vertices[v].hasConstraint(deriv)) {
          if (deriv == derivative_hc) {
            for (size_t k = 0; k < dim; ++k) {
              unsigned int start_idx = k * n_free_constraints;
              lower_bounds->at(start_idx + free_deriv_counter) =
                  -std::abs(value_hc);
              upper_bounds->at(start_idx + free_deriv_counter) =
                  std::abs(value_hc);
            }
          }
          free_deriv_counter++;
        }
      }
    }
  }
}

template <int _N>
double PolynomialOptimizationNonLinear<_N>::computeTotalTrajectoryTime(
    const std::vector<double>& segment_times) {
  double total_time = 0;
  for (double t : segment_times) total_time += t;
  return total_time;
}

}  // namespace mav_trajectory_generation

namespace nlopt {

inline std::string returnValueToString(int return_value) {
  switch (return_value) {
    case nlopt::SUCCESS:
      return std::string("SUCCESS");
    case nlopt::FAILURE:
      return std::string("FAILURE");
    case nlopt::INVALID_ARGS:
      return std::string("INVALID_ARGS");
    case nlopt::OUT_OF_MEMORY:
      return std::string("OUT_OF_MEMORY");
    case nlopt::ROUNDOFF_LIMITED:
      return std::string("ROUNDOFF_LIMITED");
    case nlopt::FORCED_STOP:
      return std::string("FORCED_STOP");
    case nlopt::STOPVAL_REACHED:
      return std::string("STOPVAL_REACHED");
    case nlopt::FTOL_REACHED:
      return std::string("FTOL_REACHED");
    case nlopt::XTOL_REACHED:
      return std::string("XTOL_REACHED");
    case nlopt::MAXEVAL_REACHED:
```

(continues on next page)

(continued from previous page)

```
      return std::string("MAXEVAL_REACHED");
    case nlopt::MAXTIME_REACHED:
      return std::string("MAXTIME_REACHED");
    default:
      return std::string("ERROR CODE UNKNOWN");
  }
}
}  // namespace nlopt

#endif  // MAV_TRAJECTORY_GENERATION_IMPL_POLYNOMIAL_OPTIMIZATION_NONLINEAR_IMPL_H_
```

Includes

- `chrono`
- `mav_trajectory_generation/polynomial_optimization_linear.h` (*File polynomial_optimization_linear.h*)
- `mav_trajectory_generation/timing.h` (*File timing.h*)
- `numeric`

Included By

- *File polynomial_optimization_nonlinear.h*

Namespaces

- *Namespace mav_trajectory_generation*
- *Namespace nlopt*

File ros_conversions.h

Contents

- *Definition (`mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_conversions.h`)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_conversions.h)

Program Listing for File ros_conversions.h

Return to documentation for file (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_conversions.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_ROS_CONVERSIONS_H_
#define MAV_TRAJECTORY_GENERATION_ROS_ROS_CONVERSIONS_H_

#include <mav_planning_msgs/PolynomialTrajectory4D.h>
#include <mav_planning_msgs/conversions.h>

#include <mav_trajectory_generation/trajectory.h>

namespace mav_trajectory_generation {

bool trajectoryToPolynomialTrajectoryMsg(
    const Trajectory& trajectory,
    mav_planning_msgs::PolynomialTrajectory4D* msg);

bool polynomialTrajectoryMsgToTrajectory(
    const mav_planning_msgs::PolynomialTrajectory4D& msg,
    Trajectory* trajectory);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_ROS_ROS_CONVERSIONS_H_
```

Includes

- mav_planning_msgs/PolynomialTrajectory4D.h
- mav_planning_msgs/conversions.h

- mav_trajectory_generation/trajectory.h (*File trajectory.h*)

Included By

- *File trajectory_sampler_node.h*

Namespaces

- *Namespace mav_trajectory_generation*

File ros_visualization.h

Contents

- *Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_visualization.h)*
- *Includes*
- *Namespaces*

Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_visualization.h)

Program Listing for File ros_visualization.h

Return to documentation for file (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/ros_visualization.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_ROS_ROS_VISUALIZATION_H_
```

(continues on next page)

(continued from previous page)

```
#define MAV_TRAJECTORY_GENERATION_ROS_ROS_VISUALIZATION_H_

#include <mav_msgs/eigen_mav_msgs.h>
#include <mav_visualization/marker_group.h>
#include <visualization_msgs/MarkerArray.h>

#include <mav_trajectory_generation/trajectory.h>
#include <mav_trajectory_generation/vertex.h>

namespace mav_trajectory_generation {

void drawMavTrajectory(const Trajectory& trajectory, double distance,
                       const std::string& frame_id,
                       visualization_msgs::MarkerArray* marker_array);

void drawMavSampledTrajectory(
    const mav_msgs::EigenTrajectoryPoint::Vector& flat_states, double distance,
    const std::string& frame_id, visualization_msgs::MarkerArray* marker_array);

void drawMavTrajectoryWithMavMarker(
    const Trajectory& trajectory, double distance, const std::string& frame_id,
    const mav_visualization::MarkerGroup& additional_marker,
    visualization_msgs::MarkerArray* marker_array);

void drawMavSampledTrajectoryWithMavMarker(
    const mav_msgs::EigenTrajectoryPoint::Vector& flat_states, double distance,
    const std::string& frame_id,
    const mav_visualization::MarkerGroup& additional_marker,
    visualization_msgs::MarkerArray* marker_array);

void drawVertices(const Vertex::Vector& vertices, const std::string& frame_id,
                  visualization_msgs::MarkerArray* marker_array);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_ROS_ROS_CONVERSIONS_H_
```

Includes

- `mav_msgs/eigen_mav_msgs.h`
- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)
- `mav_trajectory_generation/vertex.h` (*File vertex.h*)
- `mav_visualization/marker_group.h` (*File marker_group.h*)
- `visualization_msgs/MarkerArray.h`

Namespaces

- *Namespace mav_trajectory_generation*

File rpoly_ak1.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/rpoly/rpoly_ak1.h)*
- *Includes*
- *Namespaces*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/rpoly/rpoly_ak1.h`)

Program Listing for File rpoly_ak1.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/rpoly/rpoly_ak1.h`)

```
/*
 * Copyright (c) 2018, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_RPOLY_RPOLY_AK1_H_
#define MAV_TRAJECTORY_GENERATION_RPOLY_RPOLY_AK1_H_

#include <Eigen/Eigen>

namespace mav_trajectory_generation {

int findLastNonZeroCoeff(const Eigen::VectorXd& coefficients);

bool findRootsJenkinsTraub(const Eigen::VectorXd& coefficients_increasing,
                           Eigen::VectorXcd* roots);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_RPOLY_RPOLY_AK1_H_
```

Includes

- `Eigen/Eigen`

Namespaces

- *Namespace mav_trajectory_generation*

File segment.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/segment.h)*
- *Includes*
- *Included By*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/segment.h`)

Program Listing for File segment.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/segment.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_SEGMENT_H_
#define MAV_TRAJECTORY_GENERATION_SEGMENT_H_

#include <glog/logging.h>
#include <Eigen/Core>
```

(continues on next page)

(continued from previous page)

```cpp
#include <chrono>
#include <map>
#include <vector>

#include "mav_trajectory_generation/extremum.h"
#include "mav_trajectory_generation/motion_defines.h"
#include "mav_trajectory_generation/polynomial.h"

namespace mav_trajectory_generation {

constexpr double kNumNSecPerSec = 1.0e9;
constexpr double kNumSecPerNsec = 1.0e-9;

class Segment {
 public:
  typedef std::vector<Segment> Vector;

  Segment(int N, int D) : time_(0.0), N_(N), D_(D) {
    polynomials_.resize(D_, Polynomial(N_));
  }
  Segment(const Segment& segment) = default;

  bool operator==(const Segment& rhs) const;
  inline bool operator!=(const Segment& rhs) const { return !operator==(rhs); }

  int D() const { return D_; }
  int N() const { return N_; }
  double getTime() const { return time_; }
  uint64_t getTimeNSec() const {
    return static_cast<uint64_t>(kNumNSecPerSec * time_);
  }

  void setTime(double time_sec) { time_ = time_sec; }
  void setTimeNSec(uint64_t time_ns) { time_ = time_ns * kNumSecPerNsec; }

  Polynomial& operator[](size_t idx);

  const Polynomial& operator[](size_t idx) const;

  const Polynomial::Vector& getPolynomialsRef() const { return polynomials_; }

  Eigen::VectorXd evaluate(
      double t, int derivative_order = derivative_order::POSITION) const;

  bool computeMinMaxMagnitudeCandidateTimes(
      int derivative, double t_start, double t_end,
      const std::vector<int>& dimensions,
      std::vector<double>* candidate_times) const;

  bool computeMinMaxMagnitudeCandidates(
      int derivative, double t_start, double t_end,
      const std::vector<int>& dimensions,
      std::vector<Extremum>* candidates) const;

  bool selectMinMaxMagnitudeFromCandidates(
      int derivative, double t_start, double t_end,
      const std::vector<int>& dimensions,
```

(continues on next page)

(continued from previous page)

```
      const std::vector<Extremum>& candidates, Extremum* minimum,
      Extremum* maximum) const;

  bool getSegmentWithSingleDimension(int dimension, Segment* new_segment) const;
  bool getSegmentWithAppendedDimension(const Segment& segment_to_append,
                                       Segment* new_segment) const;

 protected:
  Polynomial::Vector polynomials_;
  double time_;

 private:
  int N_;
  int D_;
};

void printSegment(std::ostream& stream, const Segment& s, int derivative);

std::ostream& operator<<(std::ostream& stream, const Segment& s);

std::ostream& operator<<(std::ostream& stream,
                         const std::vector<Segment>& segments);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_SEGMENT_H_
```

Includes

- `Eigen/Core`
- `chrono`
- `glog/logging.h`
- `map`
- `mav_trajectory_generation/extremum.h` (*File extremum.h*)
- `mav_trajectory_generation/motion_defines.h` (*File motion_defines.h*)
- `mav_trajectory_generation/polynomial.h` (*File polynomial.h*)
- `vector`

Included By

- *File io.h*
- *File polynomial_optimization_linear.h*
- *File trajectory.h*
- *File feasibility_analytic.h*
- *File feasibility_recursive.h*
- *File feasibility_sampling.h*

File test_utils.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/test_utils.h)*
- *Includes*
- *Namespaces*

Definition (mav_trajectory_generation/include/mav_trajectory_generation/test_utils.h)

Program Listing for File test_utils.h

Return to documentation for file (mav_trajectory_generation/include/mav_trajectory_generation/test_utils.h)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_TEST_UTILS_H_
#define MAV_TRAJECTORY_GENERATION_TEST_UTILS_H_

#include <Eigen/Eigen>
#include <random>

#include "mav_trajectory_generation/trajectory.h"

namespace mav_trajectory_generation {

inline double createRandomDouble(double min, double max) {
  return (max - min) * (static_cast<double>(std::rand()) /
                        static_cast<double>(RAND_MAX)) +
         min;
}
```

(continues on next page)

(continued from previous page)

```
template <class T1, class T2>
bool checkMatrices(const Eigen::MatrixBase<T1>& m1,
                   const Eigen::MatrixBase<T2>& m2, double tol) {
  return (m1 - m2).cwiseAbs().maxCoeff() < tol;
}

double getMaximumMagnitude(const Trajectory& trajectory, size_t derivative,
                           double dt = 0.01) {
  double maximum = -1e9;

  for (double ts = 0; ts < trajectory.getMaxTime(); ts += dt) {
    double current_value = trajectory.evaluate(ts, derivative).norm();
    if (current_value > maximum) {
      maximum = current_value;
    }
  }
  return maximum;
}

double computeCostNumeric(const Trajectory& trajectory, size_t derivative,
                          double dt = 0.001) {
  double cost = 0;

  for (double ts = 0; ts < trajectory.getMaxTime(); ts += dt) {
    cost += trajectory.evaluate(ts, derivative).squaredNorm() * dt;
  }
  return cost;
}

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_TEST_UTILS_H_
```

Includes

- `Eigen/Eigen`
- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)
- `random`

Namespaces

- *Namespace mav_trajectory_generation*

File timing.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/timing.h)*

- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/timing.h`)

Program Listing for File timing.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/timing.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/* Adapted from Paul Furgale Schweizer Messer sm_timing */

#ifndef MAV_TRAJECTORY_GENERATION_TIMING_H_
#define MAV_TRAJECTORY_GENERATION_TIMING_H_

#include <algorithm>
#include <chrono>
#include <limits>
#include <map>
#include <string>
#include <vector>

namespace mav_trajectory_generation {
namespace timing {

template <typename T, typename Total, int N>
class Accumulator {
 public:
  Accumulator()
      : window_samples_(0),
        total_samples_(0),
```

(continues on next page)

(continued from previous page)

```
        window_sum_(0),
        sum_(0),
        min_(std::numeric_limits<T>::max()),
        max_(std::numeric_limits<T>::min()) {}

  void Add(T sample) {
    if (window_samples_ < N) {
      samples_[window_samples_++] = sample;
      window_sum_ += sample;
    } else {
      T& oldest = samples_[window_samples_++ % N];
      window_sum_ += sample - oldest;
      oldest = sample;
    }
    sum_ += sample;
    ++total_samples_;
    if (sample > max_) {
      max_ = sample;
    }
    if (sample < min_) {
      min_ = sample;
    }
  }

  int TotalSamples() const { return total_samples_; }

  double Sum() const { return sum_; }

  double Mean() const { return sum_ / total_samples_; }

  double RollingMean() const {
    return window_sum_ / std::min(window_samples_, N);
  }

  double Max() const { return max_; }

  double Min() const { return min_; }

  double LazyVariance() const {
    if (window_samples_ == 0) {
      return 0.0;
    }
    double var = 0;
    double mean = RollingMean();
    for (int i = 0; i < std::min(window_samples_, N); ++i) {
      var += (samples_[i] - mean) * (samples_[i] - mean);
    }
    var /= std::min(window_samples_, N);
    return var;
  }

 private:
  int window_samples_;
  int total_samples_;
  Total window_sum_;
  Total sum_;
  T min_;
```

(continues on next page)

(continued from previous page)

```
  T max_;
  T samples_[N];
};

struct TimerMapValue {
  TimerMapValue() {}

  Accumulator<double, double, 50> acc_;
};

class DummyTimer {
 public:
  DummyTimer(size_t /*handle*/, bool /*constructStopped*/ = false) {}
  DummyTimer(std::string const& /*tag*/, bool /*constructStopped*/ = false) {}
  ~DummyTimer() {}

  void Start() {}
  void Stop() {}
  bool IsTiming() { return false; }
};

class Timer {
 public:
  Timer(size_t handle, bool constructStopped = false);
  Timer(std::string const& tag, bool constructStopped = false);
  ~Timer();

  void Start();
  void Stop();
  bool IsTiming() const;

 private:
  std::chrono::time_point<std::chrono::system_clock> time_;

  bool timing_;
  size_t handle_;
};

class Timing {
 public:
  typedef std::map<std::string, size_t> map_t;
  friend class Timer;
  static size_t GetHandle(std::string const& tag);
  static std::string GetTag(size_t handle);
  static double GetTotalSeconds(size_t handle);
  static double GetTotalSeconds(std::string const& tag);
  static double GetMeanSeconds(size_t handle);
  static double GetMeanSeconds(std::string const& tag);
  static size_t GetNumSamples(size_t handle);
  static size_t GetNumSamples(std::string const& tag);
  static double GetVarianceSeconds(size_t handle);
  static double GetVarianceSeconds(std::string const& tag);
  static double GetMinSeconds(size_t handle);
  static double GetMinSeconds(std::string const& tag);
  static double GetMaxSeconds(size_t handle);
  static double GetMaxSeconds(std::string const& tag);
  static double GetHz(size_t handle);
```

(continues on next page)

(continued from previous page)

```
  static double GetHz(std::string const& tag);
  static void Print(std::ostream& out);
  static std::string Print();
  static std::string SecondsToTimeString(double seconds);
  static void Reset();
  static const map_t& GetTimers() { return Instance().tag_map_; }

 private:
  void AddTime(size_t handle, double seconds);

  static Timing& Instance();

  Timing();
  ~Timing();

  typedef std::vector<TimerMapValue> list_t;

  list_t timers_;
  map_t tag_map_;
  size_t max_tag_length_;
};

#if DISABLE_TIMING
typedef DummyTimer DebugTimer;
#else
typedef Timer DebugTimer;
#endif

class MiniTimer {
 public:
  MiniTimer() : start_(std::chrono::system_clock::now()) {}

  void start() { start_ = std::chrono::system_clock::now(); }

  double stop() {
    end_ = std::chrono::system_clock::now();
    return getTime();
  }

  double getTime() const {
    if (end_ < start_) {
      return 0.0;
    }
    std::chrono::duration<double> duration = end_ - start_;
    return duration.count();
  }

 private:
  std::chrono::time_point<std::chrono::system_clock> start_;
  std::chrono::time_point<std::chrono::system_clock> end_;
};

}  // namespace timing
}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_TIMING_H_
```

Includes

- `algorithm`
- `chrono`
- `limits`
- `map`
- `string`
- `vector`

Included By

- *File polynomial_optimization_nonlinear_impl.h*

Namespaces

- *Namespace mav_trajectory_generation*
- *Namespace mav_trajectory_generation::timing*

Classes

- *Struct TimerMapValue*
- *Template Class Accumulator*
- *Class DummyTimer*
- *Class MiniTimer*
- *Class Timer*
- *Class Timing*

File trajectory.h

Contents

- *Definition* (`mav_trajectory_generation/include/mav_trajectory_generation/trajectory.h`)
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/trajectory.h`)

Program Listing for File trajectory.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/trajectory.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_TRAJECTORY_H_
#define MAV_TRAJECTORY_GENERATION_TRAJECTORY_H_

#include "mav_trajectory_generation/extremum.h"
#include "mav_trajectory_generation/segment.h"
#include "mav_trajectory_generation/vertex.h"

namespace mav_trajectory_generation {

class Trajectory {
 public:
  Trajectory() : D_(0), N_(0), max_time_(0.0) {}
  ~Trajectory() {}

  bool operator==(const Trajectory& rhs) const;
  inline bool operator!=(const Trajectory& rhs) const {
    return !operator==(rhs);
  }

  int D() const { return D_; }
  int N() const { return N_; }
  int K() const { return segments_.size(); }

  bool empty() const { return segments_.empty(); }
  void clear() {
    segments_.clear();
    D_ = 0;
    N_ = 0;
    max_time_ = 0.0;
```

(continues on next page)

(continued from previous page)

```
  }

  void setSegments(const Segment::Vector& segments) {
    CHECK(!segments.empty());
    // Reset states.
    D_ = segments.front().D();
    N_ = segments.front().N();
    max_time_ = 0.0;
    segments_.clear();

    addSegments(segments);
  }

  void addSegments(const Segment::Vector& segments) {
    for (const Segment& segment : segments) {
      CHECK_EQ(segment.D(), D_);
      CHECK_EQ(segment.N(), N_);
      max_time_ += segment.getTime();
    }
    segments_.insert(segments_.end(), segments.begin(), segments.end());
  }

  void getSegments(Segment::Vector* segments) const {
    CHECK_NOTNULL(segments);
    *segments = segments_;
  }

  const Segment::Vector& segments() const { return segments_; }

  double getMinTime() const { return 0.0; }
  double getMaxTime() const { return max_time_; }
  std::vector<double> getSegmentTimes() const;

  Trajectory getTrajectoryWithSingleDimension(int dimension) const;
  bool getTrajectoryWithAppendedDimension(
      const Trajectory& trajectory_to_append, Trajectory* new_trajectory) const;

  bool addTrajectories(const std::vector<Trajectory>& trajectories,
                       Trajectory* merged) const;

  Vertex getVertexAtTime(double t, int max_derivative_order) const;
  Vertex getStartVertex(int max_derivative_order) const;
  Vertex getGoalVertex(int max_derivative_order) const;

  Eigen::VectorXd evaluate(
      double t, int derivative_order = derivative_order::POSITION) const;

  void evaluateRange(double t_start, double t_end, double dt,
                     int derivative_order, std::vector<Eigen::VectorXd>* result,
                     std::vector<double>* sampling_times = nullptr) const;

  bool computeMinMaxMagnitude(int derivative,
                              const std::vector<int>& dimensions,
                              Extremum* minimum, Extremum* maximum) const;

  bool computeMaxVelocityAndAcceleration(double* v_max, double* a_max) const;
```

(continues on next page)

(continued from previous page)

```
  bool scaleSegmentTimesToMeetConstraints(double v_max, double a_max);

 private:
  int D_;
  int N_;
  double max_time_;

  Segment::Vector segments_;
};

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_TRAJECTORY_H_
```

Includes

- `mav_trajectory_generation/extremum.h` (*File extremum.h*)
- `mav_trajectory_generation/segment.h` (*File segment.h*)
- `mav_trajectory_generation/vertex.h` (*File vertex.h*)

Included By

- *File io.h*
- *File polynomial_optimization_linear.h*
- *File test_utils.h*
- *File trajectory_sampling.h*
- *File feasibility_base.h*
- *File ros_conversions.h*
- *File ros_visualization.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class Trajectory*

File trajectory_sampler_node.h

Contents

- *Definition* (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/trajectory_sampler_node.h)
- *Includes*
- *Classes*

Definition (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/trajectory_sampler_node.h)

Program Listing for File trajectory_sampler_node.h

Return to documentation for file (mav_trajectory_generation_ros/include/mav_trajectory_generation_ros/trajectory_sampler_node.h)

```
/*
 * Copyright (c) 2017, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2017, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2017, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2017, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2017, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef TRAJECTORY_SAMPLER_NODE_H
#define TRAJECTORY_SAMPLER_NODE_H

#include <mav_msgs/conversions.h>
#include <mav_msgs/default_topics.h>
#include <mav_msgs/eigen_mav_msgs.h>
#include <mav_planning_msgs/PolynomialSegment4D.h>
#include <mav_planning_msgs/PolynomialTrajectory4D.h>
#include <ros/ros.h>
#include <std_srvs/Empty.h>
#include <trajectory_msgs/MultiDOFJointTrajectory.h>

#include <mav_trajectory_generation/polynomial.h>
#include <mav_trajectory_generation/trajectory_sampling.h>
#include <mav_trajectory_generation_ros/ros_conversions.h>

class TrajectorySamplerNode {
 public:
  TrajectorySamplerNode(const ros::NodeHandle& nh,
                        const ros::NodeHandle& nh_private);
```

(continues on next page)

(continued from previous page)

```
  ~TrajectorySamplerNode();

 private:
  void pathSegmentsCallback(
      const mav_planning_msgs::PolynomialTrajectory4D& segments_message);
  bool stopSamplingCallback(std_srvs::Empty::Request& request,
                            std_srvs::Empty::Response& response);
  void commandTimerCallback(const ros::TimerEvent&);

  ros::NodeHandle nh_;
  ros::NodeHandle nh_private_;

  ros::Timer publish_timer_;
  ros::Subscriber trajectory_sub_;
  ros::Publisher command_pub_;
  ros::ServiceServer stop_srv_;
  ros::Time start_time_;

  ros::ServiceClient position_hold_client_;

  bool publish_whole_trajectory_;
  double dt_;
  double current_sample_time_;

  mav_trajectory_generation::Trajectory trajectory_;
};

#endif  // TRAJECTORY_SAMPLER_NODE_H
```

Includes

- `mav_msgs/conversions.h`
- `mav_msgs/default_topics.h`
- `mav_msgs/eigen_mav_msgs.h`
- `mav_planning_msgs/PolynomialSegment4D.h`
- `mav_planning_msgs/PolynomialTrajectory4D.h`
- `mav_trajectory_generation/polynomial.h` (*File polynomial.h*)
- `mav_trajectory_generation/trajectory_sampling.h` (*File trajectory_sampling.h*)
- `mav_trajectory_generation_ros/ros_conversions.h` (*File ros_conversions.h*)
- `ros/ros.h`
- `std_srvs/Empty.h`
- `trajectory_msgs/MultiDOFJointTrajectory.h`

Classes

- *Class TrajectorySamplerNode*

File trajectory_sampling.h

Contents

- *Definition (mav_trajectory_generation/include/mav_trajectory_generation/trajectory_sampling.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/trajectory_sampling.h`)

Program Listing for File trajectory_sampling.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/trajectory_sampling.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_TRAJECTORY_SAMPLING_H_
#define MAV_TRAJECTORY_GENERATION_TRAJECTORY_SAMPLING_H_

#include <mav_msgs/eigen_mav_msgs.h>
#include "mav_trajectory_generation/trajectory.h"

namespace mav_trajectory_generation {

// All of these functions sample a trajectory at a time, or a range of times,
// into an EigenTrajectoryPoint (or vector). These support 3D or 4D
// trajectories. If the trajectories are 4D, the 4th dimension is assumed to
// be yaw.
// If no yaw is set, then it is simply left at its current value (0 by default).
```

(continues on next page)

(continued from previous page)

```
bool sampleTrajectoryAtTime(const Trajectory& trajectory, double sample_time,
                            mav_msgs::EigenTrajectoryPoint* state);

bool sampleTrajectoryInRange(const Trajectory& trajectory, double min_time,
                             double max_time, double sampling_interval,
                             mav_msgs::EigenTrajectoryPointVector* states);

bool sampleTrajectoryStartDuration(
    const Trajectory& trajectory, double start_time, double duration,
    double sampling_interval, mav_msgs::EigenTrajectoryPointVector* states);

bool sampleWholeTrajectory(const Trajectory& trajectory,
                           double sampling_interval,
                           mav_msgs::EigenTrajectoryPoint::Vector* states);

bool sampleSegmentAtTime(const Segment& segment, double sample_time,
                         mav_msgs::EigenTrajectoryPoint* state);

template<class T>
bool sampleFlatStateAtTime(const T& type, double sample_time,
                           mav_msgs::EigenTrajectoryPoint* state);

}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_TRAJECTORY_SAMPLING_H_
```

Includes

- `mav_msgs/eigen_mav_msgs.h`
- `mav_trajectory_generation/trajectory.h` (*File trajectory.h*)

Included By

- *File trajectory_sampler_node.h*

Namespaces

- *Namespace mav_trajectory_generation*

File vertex.h

Contents

- *Definition* (`mav_trajectory_generation/include/mav_trajectory_generation/vertex.h`)
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*

Definition (`mav_trajectory_generation/include/mav_trajectory_generation/vertex.h`)

Program Listing for File vertex.h

Return to documentation for file (`mav_trajectory_generation/include/mav_trajectory_generation/vertex.h`)

```
/*
 * Copyright (c) 2016, Markus Achtelik, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Michael Burri, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Helen Oleynikova, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Rik Bähnemann, ASL, ETH Zurich, Switzerland
 * Copyright (c) 2016, Marija Popovic, ASL, ETH Zurich, Switzerland
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http:///www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#ifndef MAV_TRAJECTORY_GENERATION_VERTEX_H_
#define MAV_TRAJECTORY_GENERATION_VERTEX_H_

#include <glog/logging.h>
#include <Eigen/Core>
#include <chrono>
#include <map>
#include <vector>

#include "mav_trajectory_generation/motion_defines.h"
#include "mav_trajectory_generation/polynomial.h"

namespace mav_trajectory_generation {

class Vertex {
 public:
  typedef std::vector<Vertex> Vector;
  typedef Eigen::VectorXd ConstraintValue;
  typedef std::pair<int, ConstraintValue> Constraint;
  typedef std::map<int, ConstraintValue> Constraints;

  Vertex(size_t dimension) : D_(dimension) {}
```

(continues on next page)

(continued from previous page)

```
  int D() const { return D_; }

  inline void addConstraint(int derivative_order, double value) {
    constraints_[derivative_order] = ConstraintValue::Constant(D_, value);
  }

  void addConstraint(int type, const Eigen::VectorXd& constraint);

  bool removeConstraint(int type);

  void makeStartOrEnd(const Eigen::VectorXd& constraint, int up_to_derivative);

  void makeStartOrEnd(double value, int up_to_derivative) {
    makeStartOrEnd(Eigen::VectorXd::Constant(D_, value), up_to_derivative);
  }

  bool hasConstraint(int derivative_order) const;

  bool getConstraint(int derivative_order, Eigen::VectorXd* constraint) const;

  typename Constraints::const_iterator cBegin() const {
    return constraints_.begin();
  }

  typename Constraints::const_iterator cEnd() const {
    return constraints_.end();
  }

  size_t getNumberOfConstraints() const { return constraints_.size(); }

  bool isEqualTol(const Vertex& rhs, double tol) const;

  bool getSubdimension(const std::vector<size_t>& subdimensions,
                       int max_derivative_order, Vertex* subvertex) const;

 private:
  int D_;
  Constraints constraints_;
};

std::ostream& operator<<(std::ostream& stream, const Vertex& v);

std::ostream& operator<<(std::ostream& stream,
                         const std::vector<Vertex>& vertices);

std::vector<double> estimateSegmentTimes(const Vertex::Vector& vertices,
                                         double v_max, double a_max);

std::vector<double> estimateSegmentTimesVelocityRamp(
    const Vertex::Vector& vertices, double v_max, double a_max,
    double time_factor = 1.0);

std::vector<double> estimateSegmentTimesNfabian(
    const Vertex::Vector& vertices, double v_max, double a_max,
    double magic_fabian_constant = 6.5);

double computeTimeVelocityRamp(const Eigen::VectorXd& start,
```

(continues on next page)

(continued from previous page)

```
                                const Eigen::VectorXd& goal, double v_max,
                                double a_max);

inline int getHighestDerivativeFromN(int N) { return N / 2 - 1; }

Vertex::Vector createRandomVertices(int maximum_derivative, size_t n_segments,
                                    const Eigen::VectorXd& minimum_position,
                                    const Eigen::VectorXd& maximum_position,
                                    size_t seed = 0);

Vertex::Vector createSquareVertices(int maximum_derivative,
                                    const Eigen::Vector3d& center,
                                    double side_length, int rounds);

Vertex::Vector createRandomVertices1D(int maximum_derivative, size_t n_segments,
                                      double minimum_position,
                                      double maximum_position, size_t seed = 0);
}  // namespace mav_trajectory_generation

#endif  // MAV_TRAJECTORY_GENERATION_VERTEX_H_
```

Includes

- `Eigen/Core`
- `chrono`
- `glog/logging.h`
- `map`
- `mav_trajectory_generation/motion_defines.h` (*File motion_defines.h*)
- `mav_trajectory_generation/polynomial.h` (*File polynomial.h*)
- `vector`

Included By

- *File polynomial_optimization_linear.h*
- *File trajectory.h*
- *File ros_visualization.h*

Namespaces

- *Namespace mav_trajectory_generation*

Classes

- *Class Vertex*

CHAPTER 3

Bibliography

This implementation is largely based on the work of C. Richter *et al*, who should be cited if this is used in a scientific publication (or the preceding conference papers): [1] C. Richter, A. Bry, and N. Roy, "**Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,**" in *International Journal of Robotics Research*, Springer, 2016.

```
@incollection{richter2016polynomial,
  title={Polynomial trajectory planning for aggressive quadrotor flight in dense␣
↪indoor environments},
  author={Richter, Charles and Bry, Adam and Roy, Nicholas},
  booktitle={Robotics Research},
  pages={649--666},
  year={2016},
  publisher={Springer}
}
```

Furthermore, the nonlinear optimization features our own extensions, described in:

Michael Burri, Helen Oleynikova, Markus Achtelik, and Roland Siegwart, "**Real-Time Visual-Inertial Mapping, Re-localization and Planning Onboard MAVs in Previously Unknown Environments**". In *IEEE Int. Conf. on Intelligent Robots and Systems* (IROS), September 2015.

```
@inproceedings{burri2015real-time,
  author={Burri, Michael and Oleynikova, Helen and  and Achtelik, Markus W. and␣
↪Siegwart, Roland},
  booktitle={Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International␣
↪Conference on},
  title={Real-Time Visual-Inertial Mapping, Re-localization and Planning Onboard MAVs␣
↪in Unknown Environments},
  year={2015},
  month={Sept}
}
```

Index

M

mav_trajectory_generation::checkMatrices (C++ function), 35
mav_trajectory_generation::computeBaseCoefficients (C++ function), 35
mav_trajectory_generation::computeCostNumeric (C++ function), 36
mav_trajectory_generation::computeTimeVelocityRamp (C++ function), 36
mav_trajectory_generation::ConvolutionDimension (C++ class), 10
mav_trajectory_generation::ConvolutionDimension::__anonymous0 (C++ type), 10
mav_trajectory_generation::ConvolutionDimension::length (C++ enumerator), 10
mav_trajectory_generation::convolve (C++ function), 36
mav_trajectory_generation::createRandomDouble (C++ function), 37
mav_trajectory_generation::createRandomVertices (C++ function), 37
mav_trajectory_generation::createRandomVertices1D (C++ function), 37
mav_trajectory_generation::createSquareVertices (C++ function), 38
mav_trajectory_generation::derivative_order::ACCELERATION (C++ member), 48
mav_trajectory_generation::derivative_order::ANGULAR_ACCELERATION (C++ member), 48
mav_trajectory_generation::derivative_order::ANGULAR_VELOCITY (C++ member), 49
mav_trajectory_generation::derivative_order::INVALID (C++ member), 49
mav_trajectory_generation::derivative_order::JERK (C++ member), 49
mav_trajectory_generation::derivative_order::ORIENTATION (C++ member), 49
mav_trajectory_generation::derivative_order::POSITION (C++ member), 49
mav_trajectory_generation::derivative_order::SNAP (C++ member), 49
mav_trajectory_generation::derivative_order::VELOCITY (C++ member), 49
mav_trajectory_generation::drawMavSampledTrajectory (C++ function), 38
mav_trajectory_generation::drawMavSampledTrajectoryWithMavMarker (C++ function), 38
mav_trajectory_generation::drawMavTrajectory (C++ function), 39
mav_trajectory_generation::drawMavTrajectoryWithMavMarker (C++ function), 39
mav_trajectory_generation::drawVertices (C++ function), 39
mav_trajectory_generation::estimateSegmentTimes (C++ function), 40
mav_trajectory_generation::estimateSegmentTimesNfabian (C++ function), 40
mav_trajectory_generation::estimateSegmentTimesVelocityRamp (C++ function), 41
mav_trajectory_generation::Extremum (C++ class), 10
mav_trajectory_generation::Extremum::Extremum (C++ function), 10
mav_trajectory_generation::Extremum::operator> (C++ function), 10
mav_trajectory_generation::Extremum::operator< (C++ function), 10
mav_trajectory_generation::Extremum::segment_idx (C++ member), 10
mav_trajectory_generation::Extremum::time (C++ member), 10
mav_trajectory_generation::Extremum::value (C++ member), 10
mav_trajectory_generation::FeasibilityAnalytic (C++ class), 14
mav_trajectory_generation::FeasibilityAnalytic::checkInputFeasibility (C++ function), 14
mav_trajectory_generation::FeasibilityAnalytic::FeasibilityAnalytic (C++ function), 14
mav_trajectory_generation::FeasibilityAnalytic::Roots (C++ type), 14

mav_trajectory_generation::FeasibilityAnalytic::Settings (C++ class), 14, 15
mav_trajectory_generation::FeasibilityAnalytic::Settings::getMinSectionTimeS (C++ function), 14, 15
mav_trajectory_generation::FeasibilityAnalytic::Settings::setMinSectionTimeS (C++ function), 14, 15
mav_trajectory_generation::FeasibilityAnalytic::Settings::Settings (C++ function), 14, 15
mav_trajectory_generation::FeasibilityAnalytic::settings_ (C++ member), 14
mav_trajectory_generation::FeasibilityBase (C++ class), 15
mav_trajectory_generation::FeasibilityBase::checkHalfPlaneFeasibility (C++ function), 16
mav_trajectory_generation::FeasibilityBase::checkInputFeasibility (C++ function), 15
mav_trajectory_generation::FeasibilityBase::checkInputFeasibilityTrajectory (C++ function), 15
mav_trajectory_generation::FeasibilityBase::FeasibilityBase (C++ function), 15
mav_trajectory_generation::FeasibilityBase::getInputConstraints (C++ function), 16
mav_trajectory_generation::FeasibilityBase::gravity_ (C++ member), 16
mav_trajectory_generation::FeasibilityBase::half_plane_constraints_ (C++ member), 16
mav_trajectory_generation::FeasibilityBase::input_constraints_ (C++ member), 16
mav_trajectory_generation::FeasibilityRecursive (C++ class), 16
mav_trajectory_generation::FeasibilityRecursive::checkInputFeasibility (C++ function), 17
mav_trajectory_generation::FeasibilityRecursive::FeasibilityRecursive (C++ function), 17
mav_trajectory_generation::FeasibilityRecursive::Roots (C++ type), 17
mav_trajectory_generation::FeasibilityRecursive::Settings (C++ class), 17
mav_trajectory_generation::FeasibilityRecursive::Settings::getMinSectionTimeS (C++ function), 17, 18
mav_trajectory_generation::FeasibilityRecursive::Settings::setMinSectionTimeS (C++ function), 17, 18
mav_trajectory_generation::FeasibilityRecursive::Settings::Settings (C++ function), 17, 18
mav_trajectory_generation::FeasibilityRecursive::settings_ (C++ member), 17
mav_trajectory_generation::FeasibilitySampling (C++ class), 18
mav_trajectory_generation::FeasibilitySampling::checkInputFeasibility (C++ function), 18
mav_trajectory_generation::FeasibilitySampling::FeasibilitySampling (C++ function), 18
mav_trajectory_generation::FeasibilitySampling::Settings (C++ class), 18, 19
mav_trajectory_generation::FeasibilitySampling::Settings::getSamplingInte (C++ function), 19
mav_trajectory_generation::FeasibilitySampling::Settings::setSamplingInte (C++ function), 19
mav_trajectory_generation::FeasibilitySampling::Settings::Settings (C++ function), 19
mav_trajectory_generation::FeasibilitySampling::settings_ (C++ member), 18
mav_trajectory_generation::findLastNonZeroCoeff (C++ function), 41
mav_trajectory_generation::findRootsJenkinsTraub (C++ function), 41
mav_trajectory_generation::getHighestDerivativeFromN (C++ function), 41
mav_trajectory_generation::getInputFeasibilityResultName (C++ function), 42
mav_trajectory_generation::getMaximumMagnitude (C++ function), 42
mav_trajectory_generation::HalfPlane (C++ class), 19
mav_trajectory_generation::HalfPlane::createBoundingBox (C++ function), 20
mav_trajectory_generation::HalfPlane::HalfPlane (C++ function), 19
mav_trajectory_generation::HalfPlane::normal (C++ member), 20
mav_trajectory_generation::HalfPlane::point (C++ member), 20
mav_trajectory_generation::HalfPlane::Vector (C++ type), 19
mav_trajectory_generation::InputConstraints (C++ class), 20
mav_trajectory_generation::InputConstraints::addConstraint (C++ function), 20
mav_trajectory_generation::InputConstraints::getConstraint (C++ function), 20
mav_trajectory_generation::InputConstraints::hasConstraint (C++ function), 20
mav_trajectory_generation::InputConstraints::InputConstraints (C++ function), 20
mav_trajectory_generation::InputConstraints::removeConstraint (C++ function), 20
mav_trajectory_generation::InputConstraints::setDefaultValues (C++ function), 20
mav_trajectory_generation::InputConstraintType (C++ type), 34
mav_trajectory_generation::InputFeasibilityResult (C++ type), 34
mav_trajectory_generation::kFMax (C++ enumerator), 34
mav_trajectory_generation::kFMin (C++ enumerator), 34
mav_trajectory_generation::kInputFeasible (C++ enumerator), 34
mav_trajectory_generation::kInputIndeterminable (C++ enumerator), 34

mav_trajectory_generation::kInputInfeasibleRollPitchRates (C++ enumerator), 35
mav_trajectory_generation::kInputInfeasibleThrustHigh (C++ enumerator), 35
mav_trajectory_generation::kInputInfeasibleThrustLow (C++ enumerator), 35
mav_trajectory_generation::kInputInfeasibleVelocity (C++ enumerator), 35
mav_trajectory_generation::kInputInfeasibleYawAcc (C++ enumerator), 35
mav_trajectory_generation::kInputInfeasibleYawRates (C++ enumerator), 35
mav_trajectory_generation::kOmegaXYMax (C++ enumerator), 34
mav_trajectory_generation::kOmegaZDotMax (C++ enumerator), 34
mav_trajectory_generation::kOmegaZMax (C++ enumerator), 34
mav_trajectory_generation::kOptimizationTimeLowerBound (C++ member), 50
mav_trajectory_generation::kVMax (C++ enumerator), 34
mav_trajectory_generation::NonlinearOptimizationParameters (C++ class), 11
mav_trajectory_generation::NonlinearOptimizationParameters::algorithm (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::equality_constraint_tolerance (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::f_abs (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::f_rel (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::inequality_constraint_tolerance (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::initial_stepsize_rel (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kMellingerOuterLoop (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kRichterTime (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kRichterTimeAndConstraints (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kSquaredTime (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kSquaredTimeAndConstraints (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::kUnknown (C++ enumerator), 11
mav_trajectory_generation::NonlinearOptimizationParameters::max_iterations (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::print_debug_info (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::print_debug_info_time_allocation (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::random_see (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::soft_constra (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::time_alloc_ (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::time_penalt (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::TimeAllocM (C++ type), 11
mav_trajectory_generation::NonlinearOptimizationParameters::use_soft_co (C++ member), 12
mav_trajectory_generation::NonlinearOptimizationParameters::x_abs (C++ member), 11
mav_trajectory_generation::NonlinearOptimizationParameters::x_rel (C++ member), 11
mav_trajectory_generation::operator<< (C++ function), 42
mav_trajectory_generation::OptimizationInfo (C++ class), 12
mav_trajectory_generation::OptimizationInfo::cost_soft_constraints (C++ member), 12
mav_trajectory_generation::OptimizationInfo::cost_time (C++ member), 12
mav_trajectory_generation::OptimizationInfo::cost_trajectory (C++ member), 12
mav_trajectory_generation::OptimizationInfo::maxima (C++ member), 12
mav_trajectory_generation::OptimizationInfo::n_iterations (C++ member), 12
mav_trajectory_generation::OptimizationInfo::optimization_time (C++ member), 12
mav_trajectory_generation::OptimizationInfo::stopping_reason (C++ member), 12
mav_trajectory_generation::orientationDerivativeToInt (C++ function), 43
mav_trajectory_generation::orintationDerivativeToString (C++ function), 43
mav_trajectory_generation::Polynomial (C++ class), 21
mav_trajectory_generation::Polynomial::base_coefficients_ (C++ member), 23
mav_trajectory_generation::Polynomial::baseCoeffsWithTime (C++ function), 22
mav_trajectory_generation::Polynomial::computeMinMax (C++ function), 22
mav_trajectory_generation::Polynomial::computeMinMaxCandidates (C++ function), 22
mav_trajectory_generation::Polynomial::convolve (C++ function), 23
mav_trajectory_generation::Polynomial::evaluate (C++ function), 21
mav_trajectory_generation::Polynomial::getCoefficients (C++ function), 21
mav_trajectory_generation::Polynomial::getConvolutionLength

(C++ function), 23
mav_trajectory_generation::Polynomial::getPolynomialWithAppendedCoefficients (C++ function), 22
mav_trajectory_generation::Polynomial::getRoots (C++ function), 21
mav_trajectory_generation::Polynomial::kMaxConvolutionSize (C++ member), 23
mav_trajectory_generation::Polynomial::kMaxN (C++ member), 23
mav_trajectory_generation::Polynomial::N (C++ function), 21
mav_trajectory_generation::Polynomial::operator!= (C++ function), 21
mav_trajectory_generation::Polynomial::operator* (C++ function), 21
mav_trajectory_generation::Polynomial::operator+ (C++ function), 21
mav_trajectory_generation::Polynomial::operator+= (C++ function), 21
mav_trajectory_generation::Polynomial::operator== (C++ function), 21
mav_trajectory_generation::Polynomial::Polynomial (C++ function), 21
mav_trajectory_generation::Polynomial::scalePolynomialInTime (C++ function), 22
mav_trajectory_generation::Polynomial::selectMinMaxCandidatesFromRoots (C++ function), 22
mav_trajectory_generation::Polynomial::selectMinMaxFromCandidates (C++ function), 22
mav_trajectory_generation::Polynomial::selectMinMaxFromRoots (C++ function), 22
mav_trajectory_generation::Polynomial::setCoefficients (C++ function), 21
mav_trajectory_generation::Polynomial::Vector (C++ type), 21
mav_trajectory_generation::PolynomialOptimizationNonLinear (C++ class), 23
mav_trajectory_generation::PolynomialOptimizationNonLinear::Polynomial (C++ type), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::addMaximumMagnitudeConstraint (C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::ConstraintData (C++ class), 13
mav_trajectory_generation::PolynomialOptimizationNonLinear::getCost (C++ function), 25
mav_trajectory_generation::PolynomialOptimizationNonLinear::getOptimizationInfo (C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::getPolynomialOptimizationRef (C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::getTotalCostWithSoftConstraints (C++ function), 25
mav_trajectory_generation::PolynomialOptimizationNonLinear::getTrajectory (C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::N (C++ enumerator), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::optimize (C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::Polynomia
(C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::scaleSegm
(C++ function), 25
mav_trajectory_generation::PolynomialOptimizationNonLinear::setupFrom
(C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear::solveLinea
(C++ function), 24
mav_trajectory_generation::PolynomialOptimizationNonLinear<_N>::Cons
(C++ member), 13
mav_trajectory_generation::PolynomialOptimizationNonLinear<_N>::Cons
(C++ member), 13
mav_trajectory_generation::PolynomialOptimizationNonLinear<_N>::Cons
(C++ member), 13
mav_trajectory_generation::polynomialTrajectoryMsgToTrajectory (C++ function), 43
mav_trajectory_generation::positionDerivativeToInt (C++ function), 43
mav_trajectory_generation::positionDerivativeToString (C++ function), 43
mav_trajectory_generation::sampledTrajectoryStatesToFile (C++ function), 44
mav_trajectory_generation::sampleFlatStateAtTime (C++ function), 44
mav_trajectory_generation::sampleSegmentAtTime (C++ function), 44
mav_trajectory_generation::sampleTrajectoryAtTime (C++ function), 44
mav_trajectory_generation::sampleTrajectoryInRange (C++ function), 45
mav_trajectory_generation::sampleTrajectoryStartDuration (C++ function), 45
mav_trajectory_generation::sampleWholeTrajectory (C++ function), 45
mav_trajectory_generation::segmentsFromFile (C++ function), 45
mav_trajectory_generation::segmentsToFile (C++ function), 46
mav_trajectory_generation::sgn (C++ function), 46
mav_trajectory_generation::timing::Accumulator (C++ class), 25
mav_trajectory_generation::timing::Accumulator::Accumulator (C++ function), 25
mav_trajectory_generation::timing::Accumulator::Add (C++ function), 25
mav_trajectory_generation::timing::Accumulator::LazyVariance (C++ function), 25
mav_trajectory_generation::timing::Accumulator::Max (C++ function), 25
mav_trajectory_generation::timing::Accumulator::Mean (C++ function), 25

mav_trajectory_generation::timing::Accumulator::Min (C++ function), 25
mav_trajectory_generation::timing::Accumulator::RollingMean (C++ function), 25
mav_trajectory_generation::timing::Accumulator::Sum (C++ function), 25
mav_trajectory_generation::timing::Accumulator::TotalSamples (C++ function), 25
mav_trajectory_generation::timing::DebugTimer (C++ type), 50
mav_trajectory_generation::timing::DummyTimer (C++ class), 25
mav_trajectory_generation::timing::DummyTimer::~DummyTimer (C++ function), 26
mav_trajectory_generation::timing::DummyTimer::DummyTimer (C++ function), 26
mav_trajectory_generation::timing::DummyTimer::IsTiming (C++ function), 26
mav_trajectory_generation::timing::DummyTimer::Start (C++ function), 26
mav_trajectory_generation::timing::DummyTimer::Stop (C++ function), 26
mav_trajectory_generation::timing::MiniTimer (C++ class), 26
mav_trajectory_generation::timing::MiniTimer::getTime (C++ function), 26
mav_trajectory_generation::timing::MiniTimer::MiniTimer (C++ function), 26
mav_trajectory_generation::timing::MiniTimer::start (C++ function), 26
mav_trajectory_generation::timing::MiniTimer::stop (C++ function), 26
mav_trajectory_generation::timing::Timer (C++ class), 26
mav_trajectory_generation::timing::Timer::~Timer (C++ function), 26
mav_trajectory_generation::timing::Timer::IsTiming (C++ function), 27
mav_trajectory_generation::timing::Timer::Start (C++ function), 26
mav_trajectory_generation::timing::Timer::Stop (C++ function), 26
mav_trajectory_generation::timing::Timer::Timer (C++ function), 26
mav_trajectory_generation::timing::TimerMapValue (C++ class), 13
mav_trajectory_generation::timing::TimerMapValue::acc_ (C++ member), 13
mav_trajectory_generation::timing::TimerMapValue::TimerMapValue (C++ function), 13
mav_trajectory_generation::timing::Timing (C++ class), 27
mav_trajectory_generation::timing::Timing::GetHandle (C++ function), 27
mav_trajectory_generation::timing::Timing::GetHz (C++ function), 27
mav_trajectory_generation::timing::Timing::GetMaxSeconds (C++ function), 27
mav_trajectory_generation::timing::Timing::GetMeanSeconds (C++ function), 27
mav_trajectory_generation::timing::Timing::GetMinSeconds (C++ function), 27
mav_trajectory_generation::timing::Timing::GetNumSamples (C++ function), 27
mav_trajectory_generation::timing::Timing::GetTag (C++ function), 27
mav_trajectory_generation::timing::Timing::GetTimers (C++ function), 28
mav_trajectory_generation::timing::Timing::GetTotalSeconds (C++ function), 27
mav_trajectory_generation::timing::Timing::GetVarianceSeconds (C++ function), 27
mav_trajectory_generation::timing::Timing::map_t (C++ type), 27
mav_trajectory_generation::timing::Timing::Print (C++ function), 27
mav_trajectory_generation::timing::Timing::Reset (C++ function), 28
mav_trajectory_generation::timing::Timing::SecondsToTimeString (C++ function), 27
mav_trajectory_generation::Trajectory (C++ class), 28
mav_trajectory_generation::Trajectory::~Trajectory (C++ function), 28
mav_trajectory_generation::Trajectory::addSegments (C++ function), 28
mav_trajectory_generation::Trajectory::addTrajectories (C++ function), 29
mav_trajectory_generation::Trajectory::clear (C++ function), 28
mav_trajectory_generation::Trajectory::computeMaxVelocityAndAccelerat
(C++ function), 29
mav_trajectory_generation::Trajectory::computeMinMaxMagnitude (C++ function), 29
mav_trajectory_generation::Trajectory::D (C++ function), 28
mav_trajectory_generation::Trajectory::empty (C++ function), 28
mav_trajectory_generation::Trajectory::evaluate (C++ function), 29
mav_trajectory_generation::Trajectory::evaluateRange (C++ function), 29
mav_trajectory_generation::Trajectory::getGoalVertex (C++ function), 29
mav_trajectory_generation::Trajectory::getMaxTime (C++ function), 28
mav_trajectory_generation::Trajectory::getMinTime (C++ function), 28
mav_trajectory_generation::Trajectory::getSegments

(C++ function), 28
mav_trajectory_generation::Trajectory::getSegmentTimes (C++ function), 28
mav_trajectory_generation::Trajectory::getStartVertex (C++ function), 29
mav_trajectory_generation::Trajectory::getTrajectoryWithAppendedDimension (C++ function), 29
mav_trajectory_generation::Trajectory::getTrajectoryWithSingleDimension (C++ function), 28
mav_trajectory_generation::Trajectory::getVertexAtTime (C++ function), 29
mav_trajectory_generation::Trajectory::K (C++ function), 28
mav_trajectory_generation::Trajectory::N (C++ function), 28
mav_trajectory_generation::Trajectory::operator!= (C++ function), 28
mav_trajectory_generation::Trajectory::operator== (C++ function), 28
mav_trajectory_generation::Trajectory::scaleSegmentTimesToMeetConstraints (C++ function), 29
mav_trajectory_generation::Trajectory::segments (C++ function), 28
mav_trajectory_generation::Trajectory::setSegments (C++ function), 28
mav_trajectory_generation::Trajectory::Trajectory (C++ function), 28
mav_trajectory_generation::trajectoryFromFile (C++ function), 46
mav_trajectory_generation::trajectoryToFile (C++ function), 46
mav_trajectory_generation::trajectoryToPolynomialTrajectoryMsg (C++ function), 46
mav_trajectory_generation::Vertex (C++ class), 29
mav_trajectory_generation::Vertex::addConstraint (C++ function), 30
mav_trajectory_generation::Vertex::cBegin (C++ function), 30
mav_trajectory_generation::Vertex::cEnd (C++ function), 30
mav_trajectory_generation::Vertex::Constraint (C++ type), 30
mav_trajectory_generation::Vertex::Constraints (C++ type), 30
mav_trajectory_generation::Vertex::ConstraintValue (C++ type), 30
mav_trajectory_generation::Vertex::D (C++ function), 30
mav_trajectory_generation::Vertex::getConstraint (C++ function), 30
mav_trajectory_generation::Vertex::getNumberOfConstraints (C++ function), 30
mav_trajectory_generation::Vertex::getSubdimension (C++ function), 31
mav_trajectory_generation::Vertex::hasConstraint (C++ function), 30
mav_trajectory_generation::Vertex::isEqualTol (C++ function), 31
mav_trajectory_generation::Vertex::makeStartOrEnd (C++ function), 30
mav_trajectory_generation::Vertex::removeConstraint (C++ function), 30
mav_trajectory_generation::Vertex::Vector (C++ type), 30
mav_trajectory_generation::Vertex::Vertex (C++ function), 30
mav_visualization::Color (C++ class), 31
mav_visualization::Color::Black (C++ function), 31
mav_visualization::Color::Blue (C++ function), 31
mav_visualization::Color::Chartreuse (C++ function), 31
mav_visualization::Color::Color (C++ function), 31
mav_visualization::Color::Gray (C++ function), 31
mav_visualization::Color::Green (C++ function), 31
mav_visualization::Color::Orange (C++ function), 31
mav_visualization::Color::Pink (C++ function), 32
mav_visualization::Color::Purple (C++ function), 31
mav_visualization::Color::Red (C++ function), 31
mav_visualization::Color::Teal (C++ function), 32
mav_visualization::Color::White (C++ function), 31
mav_visualization::Color::Yellow (C++ function), 31
mav_visualization::createPoint (C++ function), 47
mav_visualization::drawArrowPoints (C++ function), 47
mav_visualization::drawArrowPositionOrientation (C++ function), 47
mav_visualization::drawAxes (C++ function), 47
mav_visualization::drawAxesArrows (C++ function), 48
mav_visualization::drawCovariance3D (C++ function), 48
mav_visualization::HexacopterMarker (C++ class), 32
mav_visualization::HexacopterMarker::HexacopterMarker (C++ function), 32
mav_visualization::LeicaMarker (C++ class), 32
mav_visualization::LeicaMarker::LeicaMarker (C++ function), 32
mav_visualization::MarkerGroup (C++ class), 33
mav_visualization::MarkerGroup::~MarkerGroup (C++ function), 33
mav_visualization::MarkerGroup::description_ (C++ member), 33
mav_visualization::MarkerGroup::getMarkers (C++ function), 33
mav_visualization::MarkerGroup::MarkerGroup (C++ function), 33
mav_visualization::MarkerGroup::markers_ (C++ member), 33
mav_visualization::MarkerGroup::name_ (C++ member), 33
mav_visualization::MarkerGroup::publish (C++ function), 33

mav_visualization::MarkerGroup::setAction (C++ function), 33
mav_visualization::MarkerGroup::setFrameLocked (C++ function), 33
mav_visualization::MarkerGroup::setHeader (C++ function), 33
mav_visualization::MarkerGroup::setHeaderAndNamespace (C++ function), 33
mav_visualization::MarkerGroup::setLifetime (C++ function), 33
mav_visualization::MarkerGroup::setNamespace (C++ function), 33
mav_visualization::MarkerGroup::transform (C++ function), 33
mav_visualization::MarkerGroup::transformMarker (C++ function), 33
mav_visualization::MarkerVector (C++ type), 50

N

nlopt::returnValueToString (C++ function), 48

T

TrajectorySamplerNode (C++ class), 34
TrajectorySamplerNode::~TrajectorySamplerNode (C++ function), 34
TrajectorySamplerNode::TrajectorySamplerNode (C++ function), 34