Point

description:
basic point in 2D (x, y)

@params:
- enum - defines the coordinate system of the point
- double x - x coordinate of point (left right)
- double y - y coordinate of point (up dawn)

@main_funcs:
- Point generating
- Getters and Setters
- Point manipulation
- Two points manipulation
- Coordinate translation

enum

description:
basic point in 2D (x, y)

@options:
- MATH(0) - positive y is forward (up)
- LOCALIZER(1)
- WEAVER(2)

@params:
- index - represents the coordinate system

@main_funcs:
- Point generating
- Getters and Setters
- Point manipulation
- Two points manipulation
- Coordinate translation

Position

description:
extends from point' a point with an angle (x, y, angle)

@params:
- enum - coordinate system
- double x - x coordinate of point
- double y - y coordinate of point
- double angle - angle of direction

@main_funcs:
- Position generation
- Getters and Setters
- Position manipulation
- to positions manipulation
- Coordinate translation

TwoTuple

description:
a class of that holds two objects in it (like a very general point)

@params:
- F - the first object
- S - the second object

@main_funcs:
- TwoTuple generation
- Getters and Setters

Vector2D

description:
extends from point simply just a fancy point

State

description:
the state of an object

@params:
- enum - coordinate system
- double x - x coordinate of point
- double y - y coordinate of point
- double angle - angle of direction
- double linearVelocity - velocity in direction of angle
- double linearAccel - accelertion in direction of angle
- double angularVelocity - velocity of change of angle
- double angularAccel - accelertion of change of angle

@main_funcs:
- ALOT of State generation
- Getters and Setters
- Coordinate manipulation

actuatorLoctation

description:
a 1D location with a velocity value
@params:
- enum - coordinate system
- double x - x coordinate of point
- double y - y coordinate of point
- double angle - angle of direction

@main_funcs:
- Position generation
- Getters and Setters
- Position manipulation
- to positions manipulation
- Coordinate translation

Segment

description:
a part of the autonomous rout that has one constant acceleration

@params:
 - tStart - Starting time (will be calculated)
 - tEnd - Ending time (will be calculated)
 - accel - the acceleration of the Segment
 - startVellocity - the starting velocity of the Segment
 - startLocation - the starting location of the Segment

@main_funcs:
 - Segment generation
 - Check if current time is in the current Segment
 - Getters and Setters

MotionProfile1D

description:
a list of all the segments in the route

@params:
 - segments - List

@main_funcs:
 - Very cool MotionProfile1D Constructors
 - Getters and Setters
 - Adders (adding 2 motion profiles):
 - - autocompleteAdd - combines 2 profiles and files the gap
 - - safeAdd - adding 2 profiles only if they are following ones, and performing the needed checks
 - - unSafeAdds (profile and segment) just adds. BE CAREFUL

Proflier1D

description:
a class of static methods that creates the best motion profile between

@params:
No Params

@main_funcs:
 - generateProfile -

 -- getTriangleProfileTImes - calculates the length of time to accelerate and the length to decelerate to get from one point to another

 -- flatterTrianlgeProfileToTrapezoid - flattens speed curve to not get above max speed

 -- addSegmentByTIme - converts the output of the other functions to segments

ICurve

description:
an Interface that have some functions all curves implements it should have

@main_funcs:
* u is a parameter some funcs get, between 0 to 1 that represents the percentage of the curve
- getLocation - returns the location point of u on the curve
- getLength - returns the length of the curve till u
- getAngle - returns the angle of the tangle line to the point in u
- getCurvature - returns the curvature of the curve in the point of u
- getSubCurve - returns a part of the curve that starts at uStrat and ends at uEnd

AbstractCurve

description:
an an abstract class which impliments ICurve and adds some functionality

@main_funcs:
- abstrct_funcs - the funcs are the same as ICurve but the parameter u represents the actual location and not the percentage
- non_abstract_funcs:
- - clamp - takes the double u which is the perecent and returns the actual location on the curve
- - getLocation - same as getLocation in ICurve
- - getLength - same as getLength in ICurve
- - getAngle - same as getAngle in ICurve
- - getCurvature - u - same as getCurvature in ICurve
- - getCurvature - average of curvature samples

PolynomialCurve

description:
a class that represents a polynomial curve

@params:
- double[] x - the coefficients of the x value
- double[] y - the coefficients of the y value
- double tScaling - didnt understand
- int rank - the rank of the polynomial

@main_funcs:
- constructors
- the class extends AbstractCurve and has the classes methods
- getDerivativeInter - returns the derivative at t=u
- getDoubleDerivativeInter - returns the second derivative at t=u

CubicSplineGenerator

description:
a generator class that generates a rank 3 PolynomialCurve to match the start state and end state

@main_funcs:
- generateSpline - does cool maths and generates

IConvert

description:
an Interface for converting linear vellocity
@main_funcs:
 - convert - gets a Vector2D that represents a linVel and linAcc, returns the velocities of each wheel for drive

ReverseLocalizerConvertor

description:
a class that implements IConvert

@params:
 - wheelDist - the distance between the wheels of the robot

@main_funcs:
 - convert - gets a Vector2D of linVel and angVel and returns the value for each motor(left and right)

CurvatureConvertor

description:
makes a smart convert
using ReverseLocalizerConvertor

@params:
 - halfWheelDist - double that contains half of the distance between the wheels

@main_funcs:
 - convert - returns a Vector2D that contains the velocity for right and left motor

MotionProfile2D

description:
motion profile in a 2D world - Contains

@params:
- firstProfile, secondProfile - 1DProfiles for each the 2 veloceties angular and linear

@main_funcs:
- Very cool MotionProfile1D Constructors
- Getters and Setters
- getting information on the profile
- removeBugSegments - uses the bug fixing method of Profile1D for each 1D profile

ChassisProfiler2D

description:
profiler 2D
@params:
- List of Locations (States) - List of the locations of the path
- jump - the times between the sub curves
- Start and end Velocities
- max and min vels and accs
- tStart - Start Time
- tForCurve - Something arbitrery Alexey did?
- smoothingtail - Argument for discrete Velocity Graph

@molto importante:
the parameters are parameters for the generateProfile function

@main_funcs:
- getMaxVelocity - calculates the maximum posible linear velocity defined from maxLinVel maxAngVel
and curvature
- getMaxAcceleration - same as getMaxVelocity but with accel
- generateProfile - uses DARK MAGIC (DiscreteVelocityGraph) to generate segements in angular and linear 1D and creates a 2D profile

DiscreteVelocityGraph

VelocitySegment

description:
a segment of distance between two points with constant acceleration

@params:
- veolocityMaxSmoothed - the max velocity after smoothing
- accel - the max acceleration of the robot
- start and end - the start and end location
- interpolator - the way to calculate the max acceleration in a certain velocity

@main_funcs:
- constructor - gets the above parameters as func parameters

- developForwards - takes the end velocity from the prev segment to use as start velocity and finds the end velocity so it accelerates with max accel(calculated with interpolator) and doesnt go above max velocity and the next max velocity

- developBackwards - the develop forwards isnt perfect it does not take into account the max decleration the develop backwards starts from the end and makes sure it does not go above max deceleration

- filter

description:
using VelocitySegment Does most of the work of converting a path consisting of curve segments into the desired Velocity/Time graph (generates the segments for that graph).
Assigns possible and rational accelerations and velocities in all of the segments.

@params:
- segments - list of Velocity segments from which the graph is consisted. Each one is considered to have a constant curvature.

- finishAsap - true if needs to finish as soon as possible.

@main_funcs:
- constructor - receives a list of curves and converts them to velocity segments using diamond technique (See ChassisProfiler2D). Smooths the max velocities in order to make the drive more elegant. Also runs develop forward and backward for each segment in chronological order.

- generateSegment() converts velocity segment and converts it to MotionProfile1D.Segment. Takes the minimum between forward and backward velocities in order to make accelerating and decelerating possible.Calculates the time length of the segment using velocities and length of the track.

WheelBasedVelocityGraph

Segment

description:
a segment of distance between two points where one wheel has a constent accelaration and the other changes with linear curvature

@params:
- vmax - the max velocity after smoothing
- distanceStart and distanceEnd - the start and end location
- interpolator - the way to calculate the max acceleration in a certain velocity
- curvatureStart and curvatureEnd

@main_funcs:
- constructor - gets the above parameters as func parameters

- convertKappa - cconverts the curvature to be normalized for the robot

- phi - converts a normalized curvature to a the ratio between the wheels

- developForwards -

- developBackwards -

- filter

description:
using VelocitySegment Does most of the work of converting a path consisting of curve segments into the desired Velocity/Time graph (generates the segments for that graph).
Assigns possible and rational accelerations and velocities in all of the segments.

@params:
- segments - list of segments from which the graph is consisted. Each one is considered to have a linear curvature.

- some robot values - max velocity wheel base length and more

@main_funcs:
- constructor - the same as discretevelocitygraph but uses linear curvature

- generateProfile - converts the segments to motionprofile1D segments and creates a motion profile 1D for each side of wheels.

PidFollower2D

description:
follows a 2d profile using pid

@params:
 - profile - the profile its following
 - kvl, kal, kvr, kar, vals, collapseVals, wheelDist, angVals, pidLimit, angCollapse - pid and robot values which we generate from testing

@main_funcs:
 - constructor - sets up all the params
 - init - resets the params to run again
 - run - runs the pid and returns the power to give to each engine

PIDObject

description:
keeps the vars of a PID

@params:
Kf, Kp, Ki, Kd, inv

@main_funcs:
Ctors for initializing those vars and getters and setters

PIDController

description:
Keeps the values of the PID and in addition calculates the change that need to bee done in order to reduce the error in the best way.

PIDController

a version of the PIDController that zeroes the I parameter when you are getting closer to the goal.