penIndy

Eine Open-Source-Software für Industriemesssysteme

27.02.2014 - Mainz

Benedikt Rauls (B. Sc. - FH Mainz)
Jens Wambach (B. Sc. - FH Mainz)
Martin Lux (B. Eng. - FH Mainz)
Prof. Dr. - Ing. Fredie Kern (FH Mainz)
Heiko Paluszek (Dipl. Ing. - sigma3D GmbH)

Übersicht

Rahmenveranstaltung:

„Halbjährige Projektarbeit des Masterstudiengangs Geoinformatik und Vermessung“ (3 Studenten - je 540 Stunden Workload)

Angeregt und unterstützt durch die sigma3D GmbH

Gesamtziele:

„Eine leicht erweiterbare Softwarelösung für die 3D-Objekt Erfassung und Auswertung im Anwendungsbereich der Forschung, Lehre und Praxis“

„Studierenden Kenntnisse in den Bereichen Softwareengineering und Industrievermessung vermitteln“

Ziel der Projektarbeit:

Modellierung des „Grundkonzepts“ -> Entwicklung einer ersten Evolutionsstufe

Lizenz

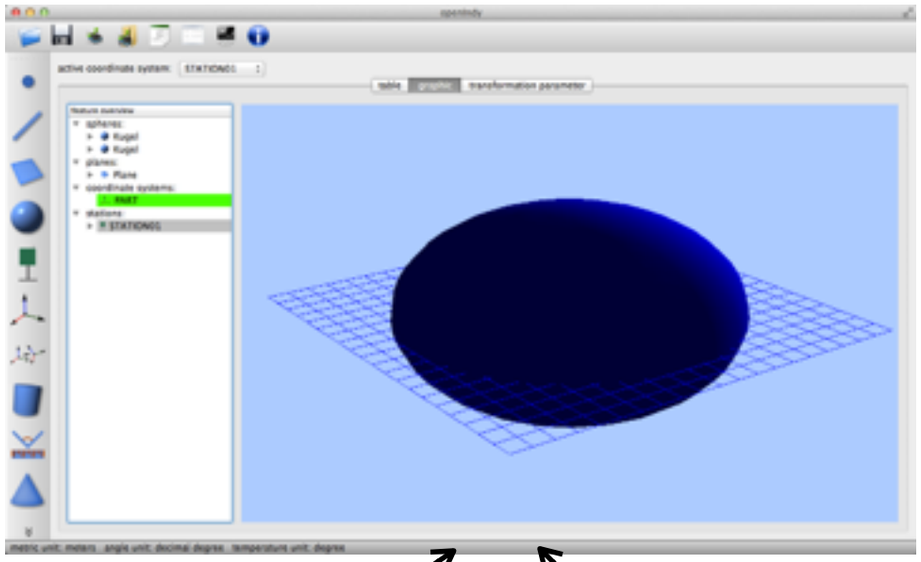
GNU Lesser General Public License (LGPL)
Genutzte Open Source IDE: Qt/Qt Creator (C++ Framework)

Social coding Plattform auf Github:

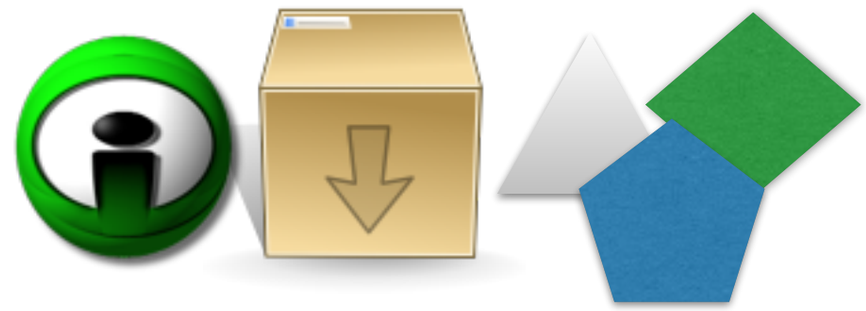
https://github.com/OpenIndy

Architektur

OpenIndy



OpenIndy-Plugin



SystemDB (sqlite)

projectfile
(openindyXML)

openindyXML (Austauschformat)

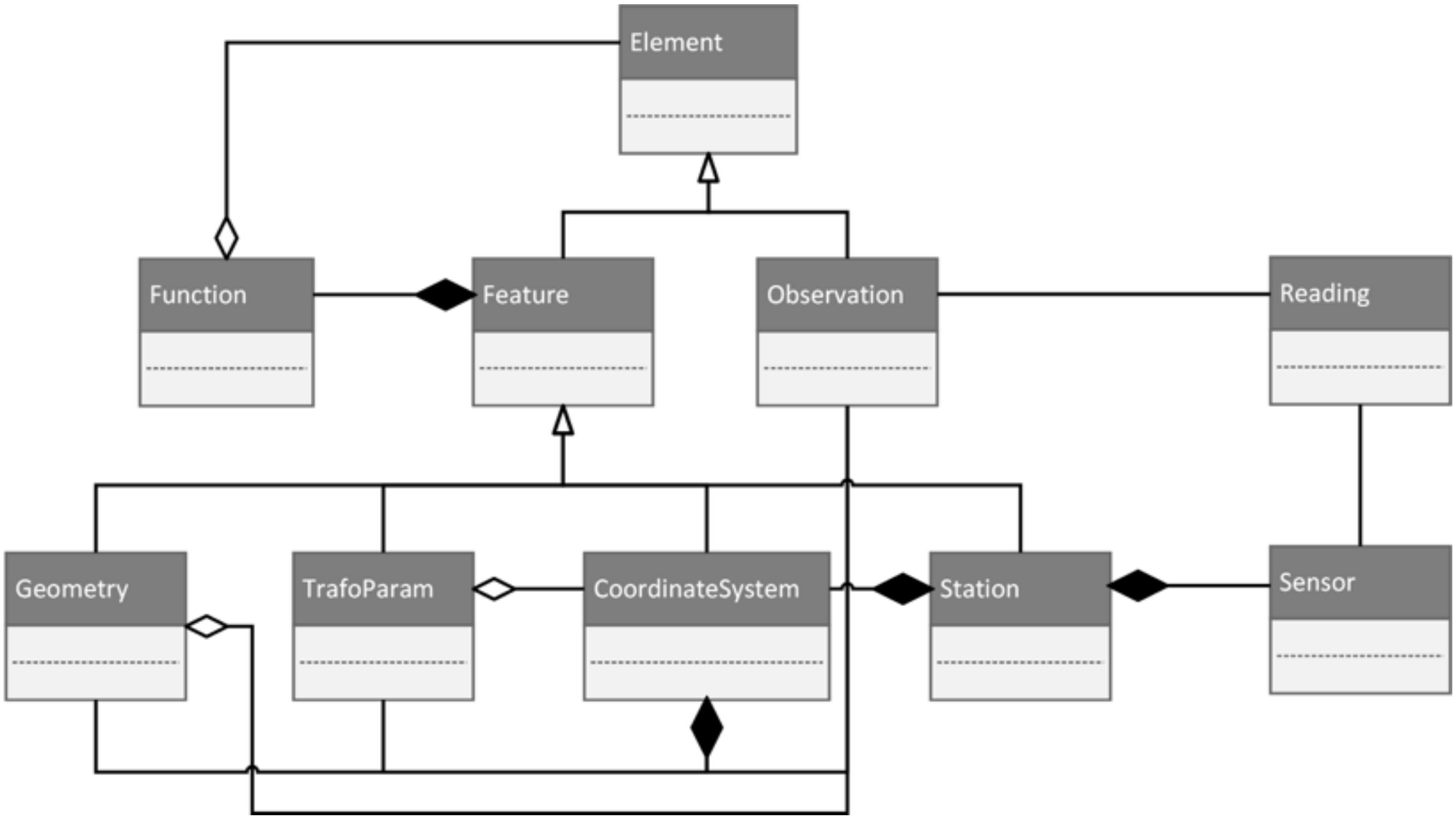
Persistente Speicherung im XML-Format

Beinhaltet Informationen:

- Beziehungen der Feature
- Geometrie
- Genauigkeit
- Funktionen
- Transformationen
- Sensor configuration
- Measurement configuration

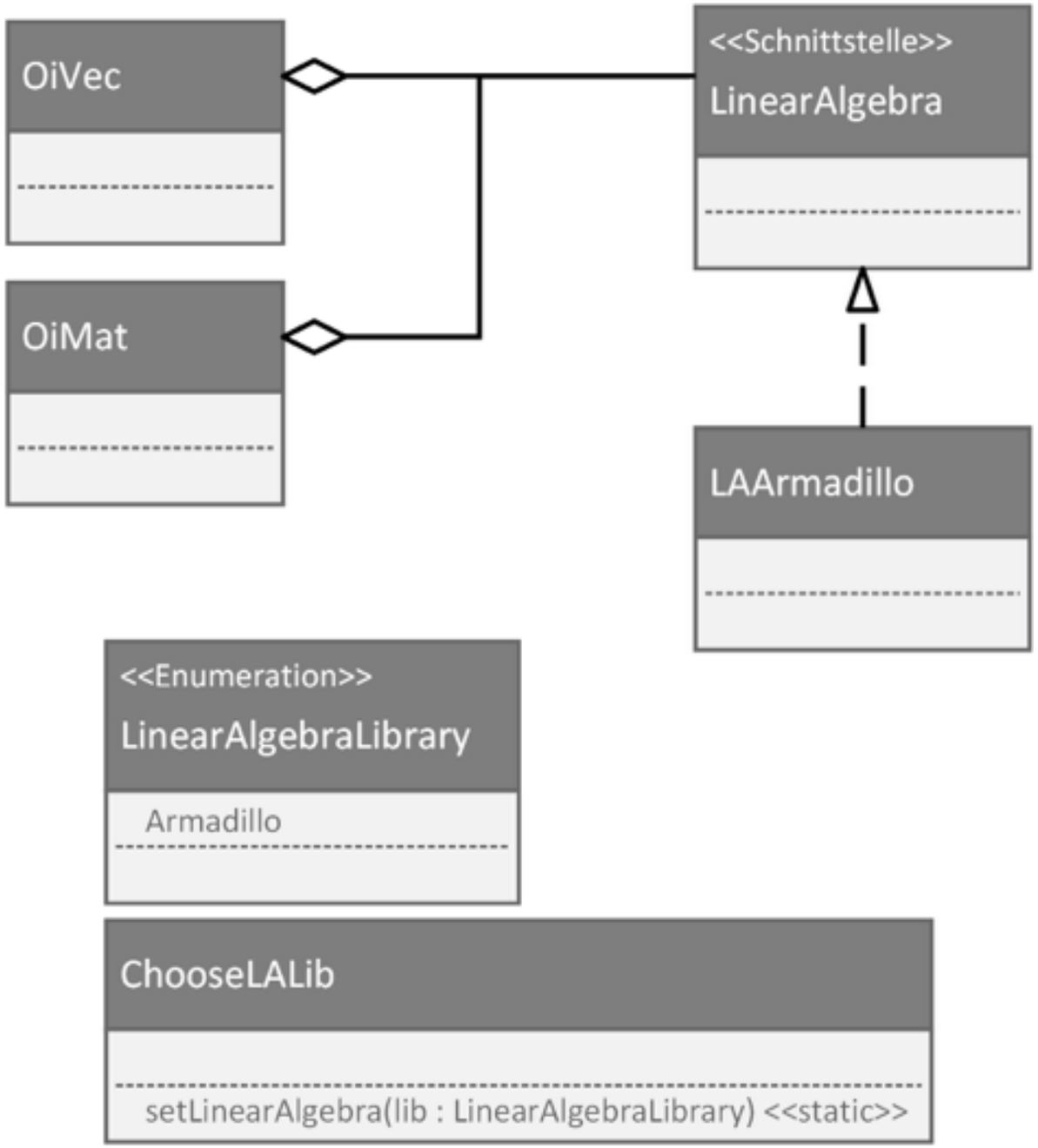
```
<?xml version="1.0" encoding="UTF-8"?>
<oiProjectData name="oiProject.xml" date="2014-02-03T20:10:38" idcount=
    <member type="activeCoordinatesystem" ref="21"/>
    <station id="2" name="STATION01">
        <sensor name="PseudoTracker" plugin="OpenIndy Default Plugin"/>
        <member type="position" ref="3"/>
        <member type="coordinatesystem" ref="4"/>
    </station>
    <station id="19" name="station2">
        <sensor name="PseudoTracker" plugin="OpenIndy Default Plugin"/>
        <member type="position" ref="20"/>
        <member type="coordinatesystem" ref="21"/>
    </station>
    <coordinatesystem id="1" name="PART" solved="0"/>
    <coordinatesystem id="4" name="STATION01" solved="0">
        <member type="observation" ref="7"/>
        <member type="observation" ref="9"/>
        <member type="observation" ref="14"/>
        <member type="observation" ref="16"/>
        <member type="observation" ref="18"/>
        <member type="transformationParameter" ref="32"/>
    </coordinatesystem>
    <coordinatesystem id="21" name="station2" solved="0">
        <member type="observation" ref="23"/>
        <member type="observation" ref="25"/>
        <member type="observation" ref="27"/>
        <member type="observation" ref="29"/>
        <member type="observation" ref="31"/>
        <member type="transformationParameter" ref="32"/>
    </coordinatesystem>
    <transformationsparameter id="32" name="t1" solved="0" tx="69.9419"
        <from type="coordinatesystem" ref="4"/>
        <to type="coordinatesystem" ref="21"/>
        <member type="previouslyNeeded" ref="5"/>
        <member type="previouslyNeeded" ref="10"/>
        <member type="previouslyNeeded" ref="11"/>
        <member type="previouslyNeeded" ref="12"/>
        <function name="HelmertTransformation" plugin="OpenIndy Default
            <inputElement index="0" type="1" ref="5"/>
            <inputElement index="0" type="1" ref="10"/>
            <inputElement index="0" type="1" ref="11"/>
            <inputElement index="0" type="1" ref="12"/>
        </function>
    </transformationsparameter>
    <transformationsparameter id="32" name="t1" solved="0" tx="69.9419"
        <from type="coordinatesystem" ref="4"/>
        <to type="coordinatesystem" ref="21"/>
        <member type="previouslyNeeded" ref="5"/>
        <member type="previouslyNeeded" ref="10"/>
        <member type="previouslyNeeded" ref="11"/>
        <member type="previouslyNeeded" ref="12"/>
        <function name="HelmertTransformation" plugin="OpenIndy Default
            <inputElement index="0" type="1" ref="5"/>
            <inputElement index="0" type="1" ref="10"/>
            <inputElement index="0" type="1" ref="11"/>
            <inputElement index="0" type="1" ref="12"/>
```

Konzept


Element
Function
Feature
Observation
Reading
Geometry
TrafoParam
CoordinateSystem
Station
Sensor

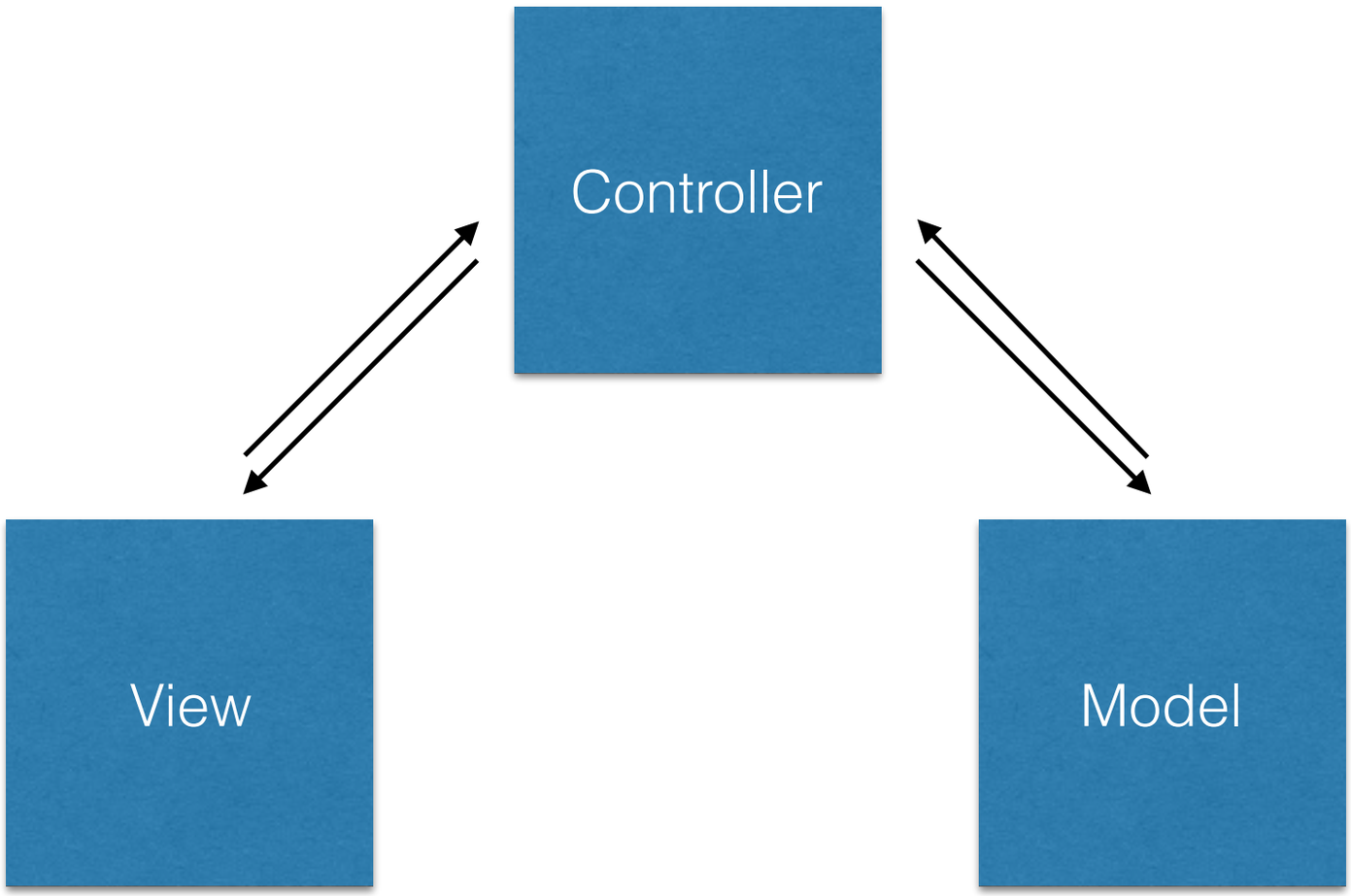
- Möglichst keine externen Bibliotheken
- Abhängigkeiten treten nur ggf. in Plugins auf
- Vektor- und Matrizenalgebra:
  - verwendet wird die OpenSource Bibliothek „Armadillo“
- Aber: keine harte Anbindung in OpenIndy
  - Abstraktion in Form einer Schnittstelle für „linear algebra“ Bibliotheken.

Abhängigkeiten



```
class OI_LIB_EXPORT LinearAlgebra
{
public:
    virtual ~LinearAlgebra(){}

    virtual OiVec addIn(OiVec v1, OiVec v2) = 0;
    virtual OiMat addIn(OiMat m1, OiMat m2) = 0;
    virtual OiVec substract(OiVec v1, OiVec v2) = 0;
    virtual OiMat substract(OiMat m1, OiMat m2) = 0;
    virtual OiMat multiply(OiMat m1, OiMat m2) = 0;
    virtual OiVec multiply(OiMat m, OiVec v) = 0;
    virtual OiMat multiply(double s, OiMat m) = 0;
    virtual OiVec multiply(double s, OiVec v) = 0;
    virtual OiMat invert(OiMat m) = 0;
    virtual OiMat transpose(OiMat m) = 0;
    virtual void svd(OiMat &u, OiVec &d, OiMat &v, OiMat x) = 0;
    virtual OiVec cross(OiVec a, OiVec b) = 0;
    virtual double dot(OiVec a, OiVec b) = 0;
};
```

Model View Control



Tableview

QAbstractTableModel zur Darstellung aller Feature Daten

```
int rowCount(const QModelIndex &parent = QModelIndex()) const;
int columnCount(const QModelIndex &parent = QModelIndex()) const;
QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const;
```

```
int rowCount(const QModelIndex &parent = QModelIndex()) const;
int columnCount(const QModelIndex &parent = QModelIndex()) const;
QVariant data(const QModelIndex &index, int role = Qt::DisplayRole) const;
```

```
QVariant TableModel::data(const QModelIndex &index, int role) const{

    if(!index.isValid())
        return QVariant();

    QString functions = "";

    if(Qt::DisplayRole == role){

        switch (index.column()) {
        case 0:
            return features.at(index.row())->returnFeatureType();
            //Configuration::FeatureTypes f = features.at(index.row())->getTypeOfFeature();
            //return featureType(f);
            break;
        case 1:
            return features.at(index.row())->getFeature()->name;
            break;
```

```
if (role == Qt::BackgroundRole){
```

Proxy Model

- Mehrere QSortFilterProxyModel zur Filterung der anzuzeigenden Daten
- Verschieden Sichten auf selbe Datengrundlage
- Aufteilung in mehrere Views

```
bool filterAcceptsColumn ( int source_column, const QModelIndex & source_parent ) const;
bool filterAcceptsRow ( int source_row, const QModelIndex & source_parent ) const;
```

```
bool filterAcceptsColumn ( int source_column, const QModelIndex & source_parent ) const;
bool filterAcceptsRow ( int source_row, const QModelIndex & source_parent ) const;
```

```
this->featureOverviewModel = new FeatureOvserviewProxyModel(this->features);
this->featureOverviewModel->setSourceModel(this->tblModel);
```

```
ui->tableView_data->setModel(control.featureOverviewModel);
```

Grafikansicht

GLWidget mit OpenGL gerendert

```
void GLWidget::draw(){

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    //TODO sicht auf schwerpunkt zentrieren
    glTranslatef(translateX, translateY, translateZ);
    glRotatef(rotationX, 1.0, 0.0 ,0.0);
    glRotatef(rotationY, 0.0, 1.0 ,0.0);
    glRotatef(rotationZ, 0.0, 0.0 ,1.0);



    qglColor(Qt::red);
    if(features->size() > 0){

        for(int i =0; i< features->size(); i++){
            OiGraphix::drawFeature(features->at(i));
        }
```

OiGraphiX

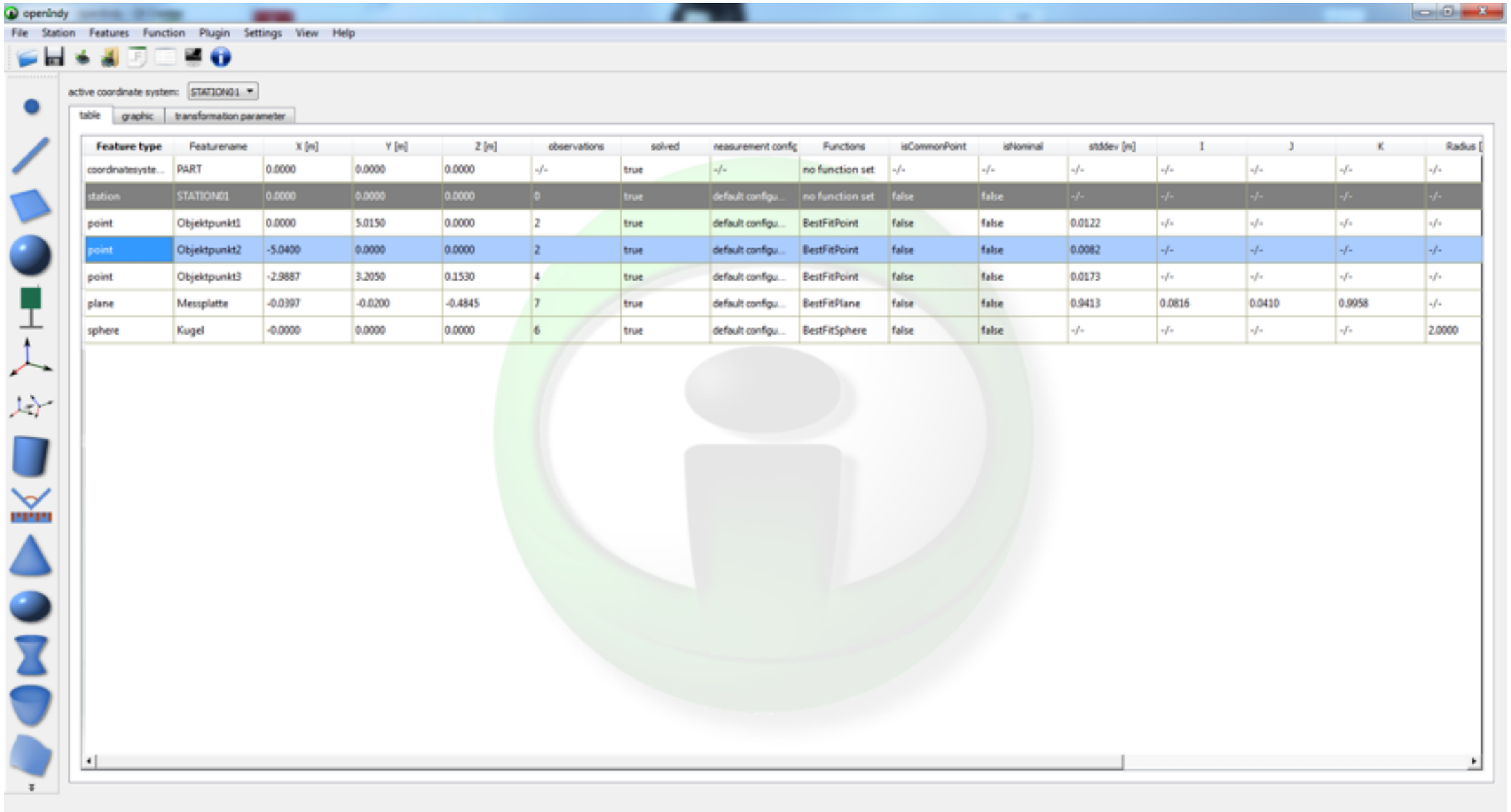
Eigene interne erweiterbare 3D Grafik-Engine

```
void OiGraphix::drawFeature(FeatureWrapper* feature){

    switch(feature->getTypeOfFeature()){
    case Configuration::eCoordinateSystemFeature:{
        break;
    }
    case Configuration::eTrafoParamFeature:{
        break;
    }
    case Configuration::ePlaneFeature:
        OiGraphix::drawPlane(feature->getPlane());
        break;
```

```
class OiGraphixGeometry
{
public:
    virtual ~OiGraphixGeometry(){}

    std::vector<GLfloat> vertices;
    std::vector<GLfloat> normals;
    std::vector<GLfloat> texcoords;
    std::vector<GLushort> indices;

    virtual void draw(GLfloat x, GLfloat y, GLfloat z) = 0;
};
```

```
#include "oigraphix_geometry.h"

class OiGraphixPlane: public OiGraphixGeometry
{
public:
    OiGraphixPlane(GLfloat nx,GLfloat ny, GLfloat nz);

    GLfloat rx;
    GLfloat ry;
    GLfloat rz;

    void draw(GLfloat x, GLfloat y, GLfloat z);
};
```

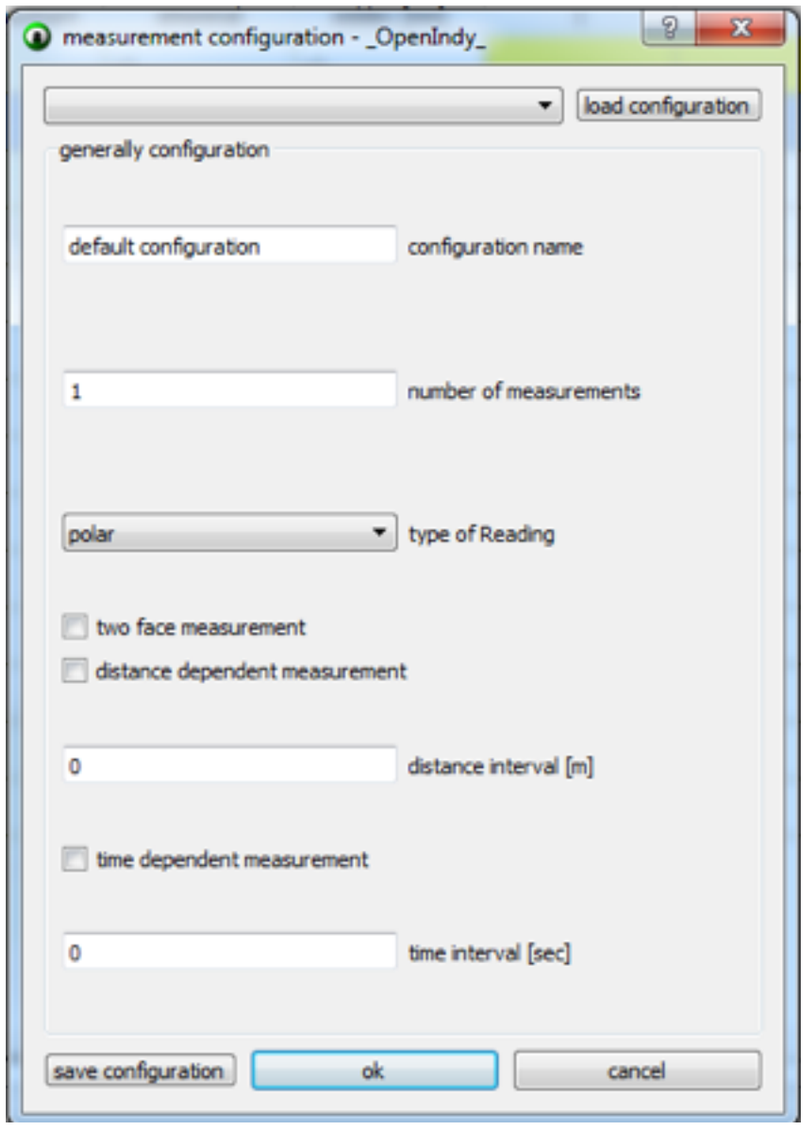
```
OiGraphixPlane::OiGraphixPlane(GLfloat nx,GLfloat ny, GLfloat nz)
{

    rx = nx;
    ry = ny;
    rz = nz;

}

void OiGraphixPlane::draw(GLfloat x, GLfloat y, GLfloat z){

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glTranslatef(x,y,z);

    glRotatef(acos(rx) * 180.0/PI,1,0,0);
    glRotatef(acos(rz) * 180.0/PI,0,1,0);
    glRotatef(acos(ry) * 180.0/PI,0,0,1);

    glBegin(GL_LINES);
    for (GLfloat i = -2.5; i <= 2.5; i += 0.25) {
      glVertex3f(i, 0, 2.5); glVertex3f(i, 0, -2.5);
      glVertex3f(2.5, 0, i); glVertex3f(-2.5, 0, i);
    }
    glEnd();

    glPopMatrix();

}
```

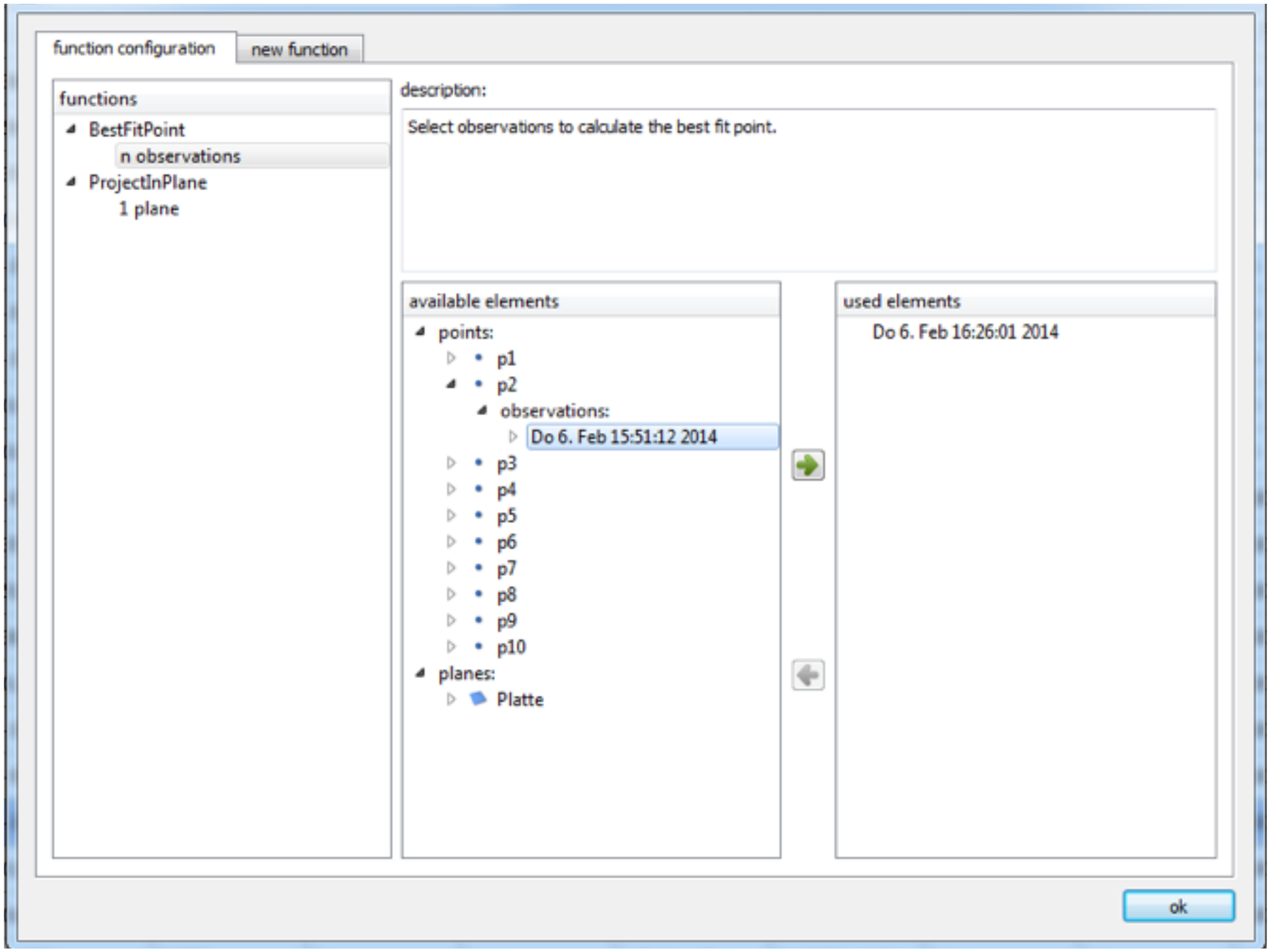


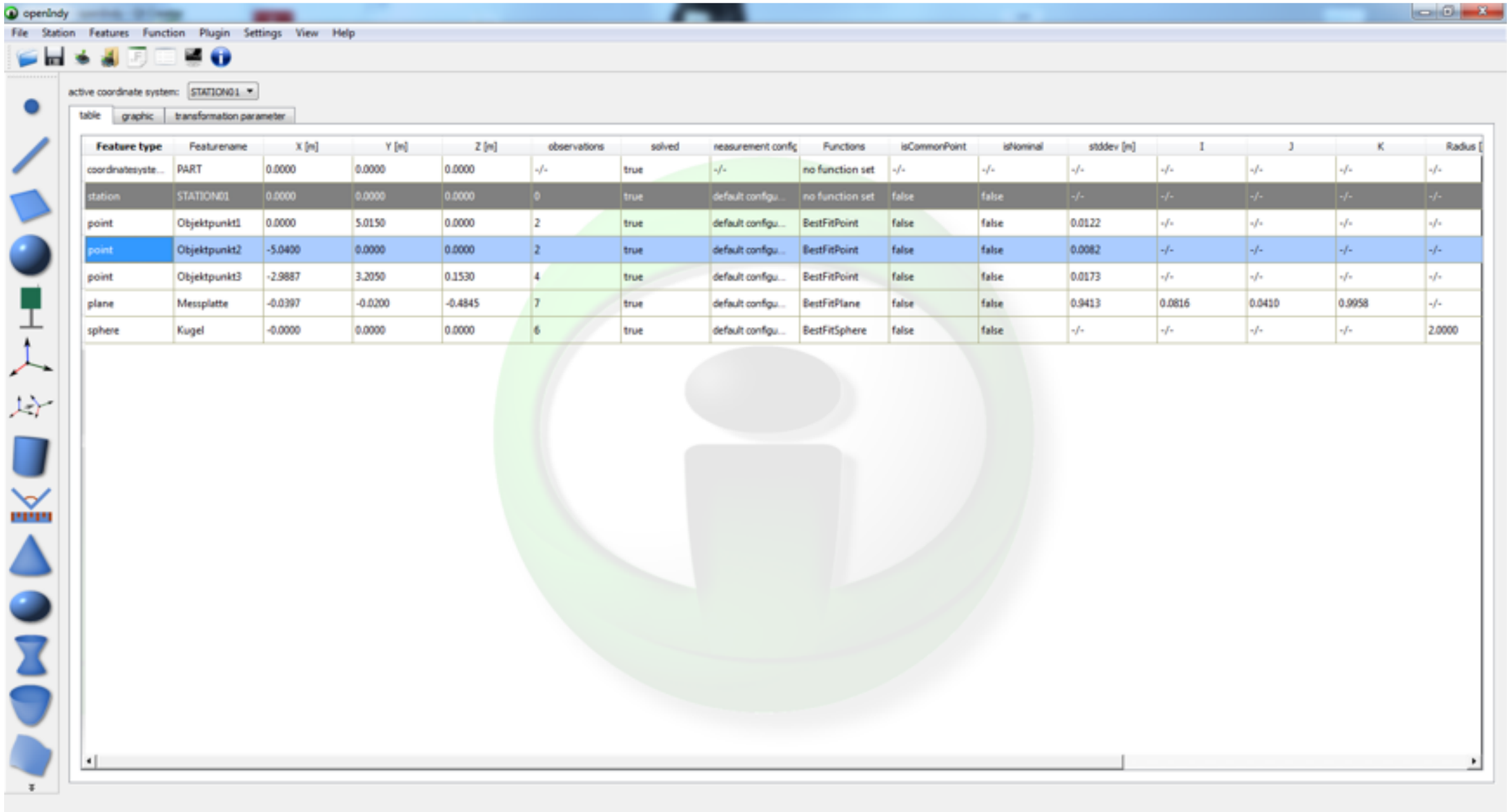
GUI

„one model many views“

feature type	name	x	y	z	observations	solved	measurement configuration	function
point	Objektpunkt2	-5.0400	0.0000	0.0000	2	true	default configu...	BestFitPoint

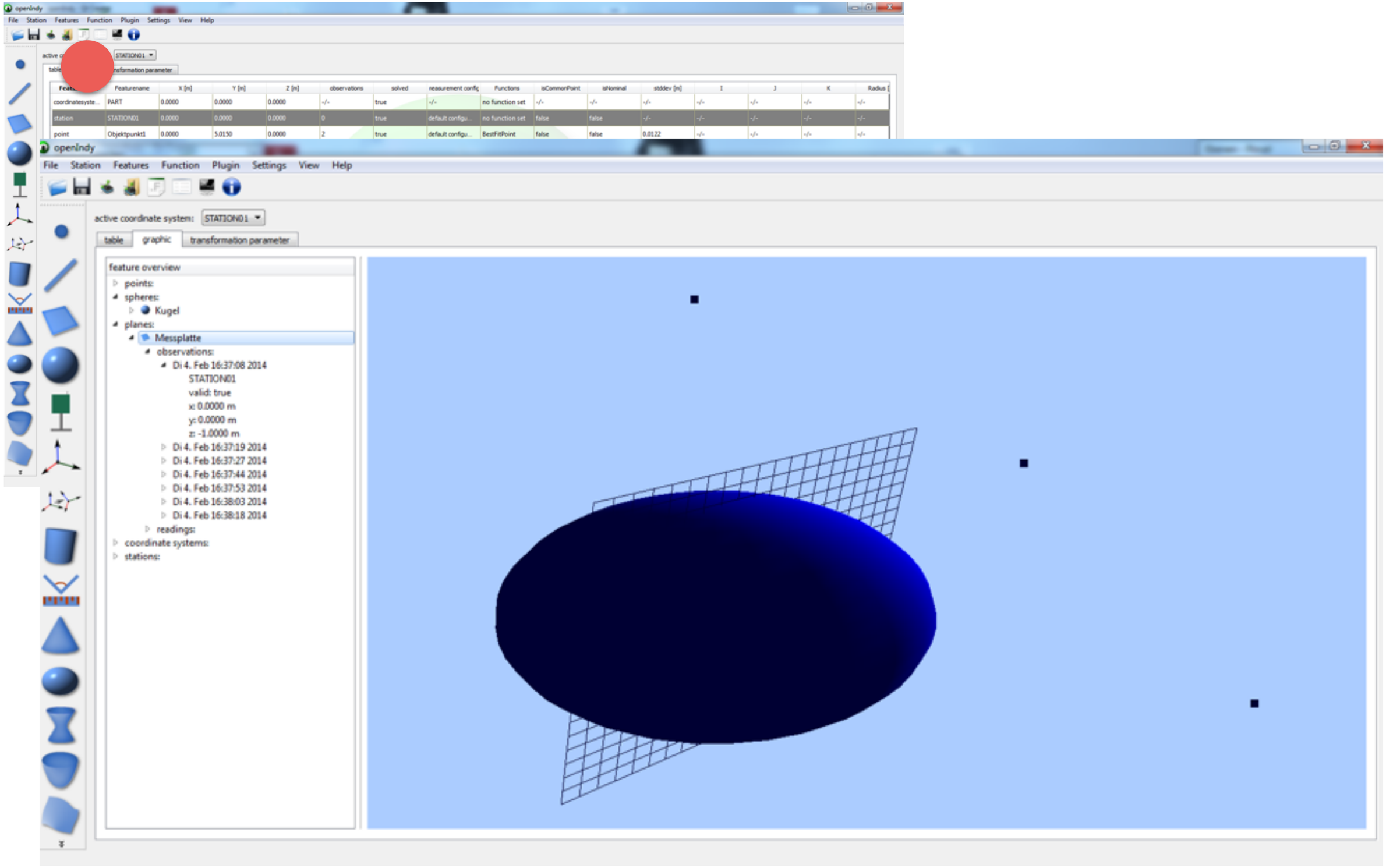






GUI

„one model many views“


openIndy
File Station Features Function Plugin Settings View Help
active coordinate system: STATION01
table graphic transformation parameter
feature overview
points:
spheres:
Kugel
planes:
Messplatte
observations:
Di 4. Feb 16:37:08 2014
STATION01
valid: true
x: 0.0000 m
y: 0.0000 m
z: -1.0000 m
Di 4. Feb 16:37:19 2014
Di 4. Feb 16:37:27 2014
Di 4. Feb 16:37:44 2014
Di 4. Feb 16:37:53 2014
Di 4. Feb 16:38:03 2014
Di 4. Feb 16:38:18 2014
readings:
coordinate systems:
stations:

Feature

Geometry

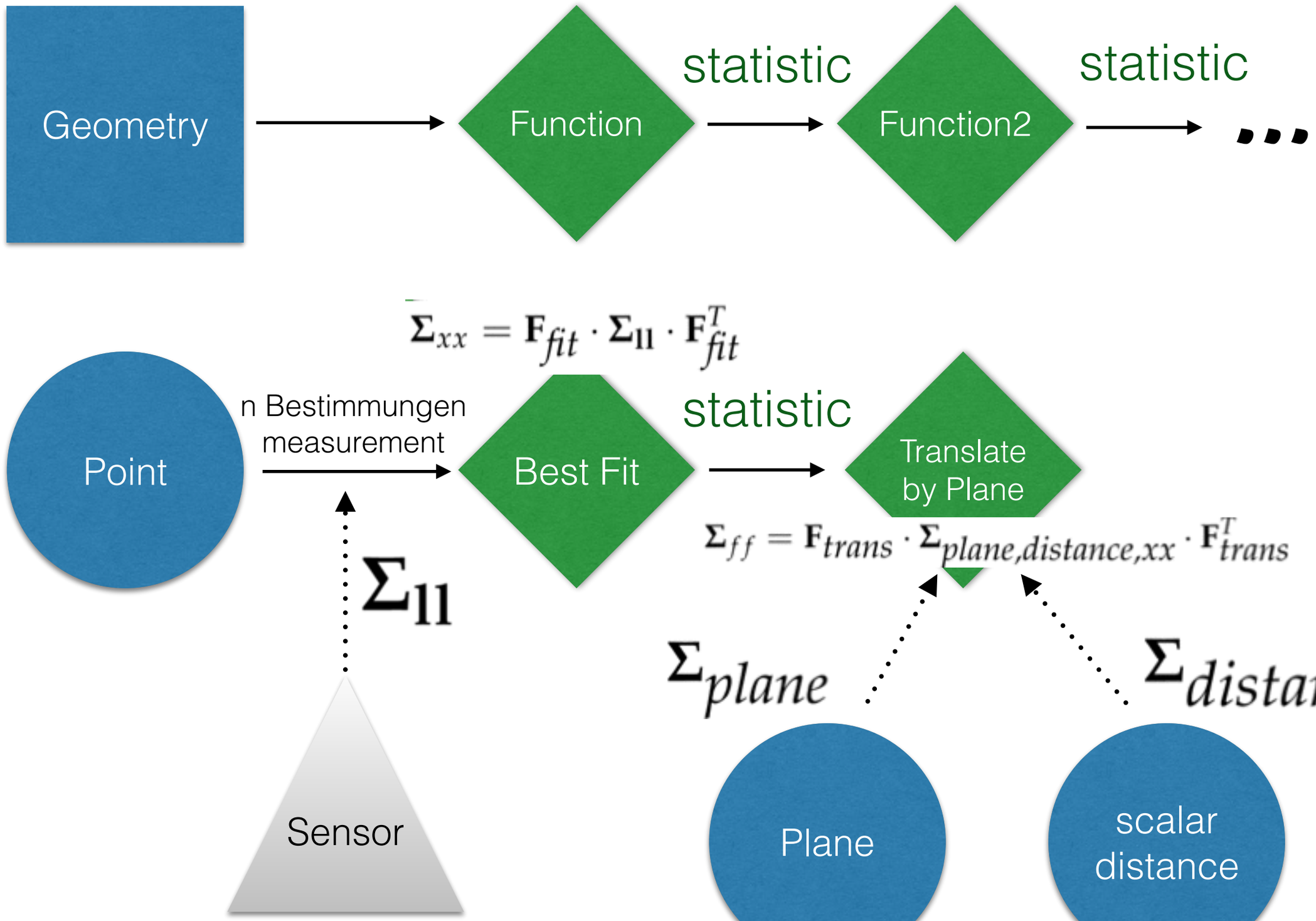
Station

Transformations parameter

Coordinate system

Geometry

- Beschreibung geometrischer Körper
- Geometriespezifische Attribute (bspw. xyz, Radius, etc…)
- Wird durch Funktionen gelöst und manipuliert


Geometry
Function
statistic
Function2
statistic
...
$\Sigma_{xx} = \mathbf{F}_{fit} \cdot \Sigma_{ll} \cdot \mathbf{F}_{fit}^T$
Point
n Bestimmungen
measurement
Best Fit
statistic
Translate
by Plane
$\Sigma_{ff} = \mathbf{F}_{trans} \cdot \Sigma_{plane,distance,xx} \cdot \mathbf{F}_{trans}^T$
Σ_{ll}
Σ_{plane}
$\Sigma_{distance}$
Sensor
Plane
scalar
distance

```
class Geometry : public Feature
{
public:
//constructor
    virtual ~Geometry();
//attributes
    bool isCommon;
    bool isNominal;
    QList<Geometry*> nominals;
    //TODO QMap with bool isUsed
    QList<Observation*> myObservations;
    CoordinateSystem* myNominalCoordSys;
    QMap<Configuration::ReadingTypes,QString> usedRead


    MeasurementConfig mConfig;

    Statistic myStatistic;

//method
    void insertReadingType(Configuration::ReadingTypes readingType, QString displayName);
```

```
class Point : public Geometry
{
public:
    enum PointUnknowns{
        unknownX,
        unknownY,
        unknownZ
    };

    Point();
    Point(const Point &copy);
    ~Point();

    OiVec xyz;

    void recalc();

    QString getDisplayX() const;
    QString getDisplayY() const;
    QString getDisplayZ() const;
    QString getDisplayIsCommon() const;
    QString getDisplayIsNominal() const;
    QString getDisplayObs() const;
    QString getDisplaySolved() const;
    QString getDisplayMConfig() const;
    QString getDisplayStdDev() const;
};
```

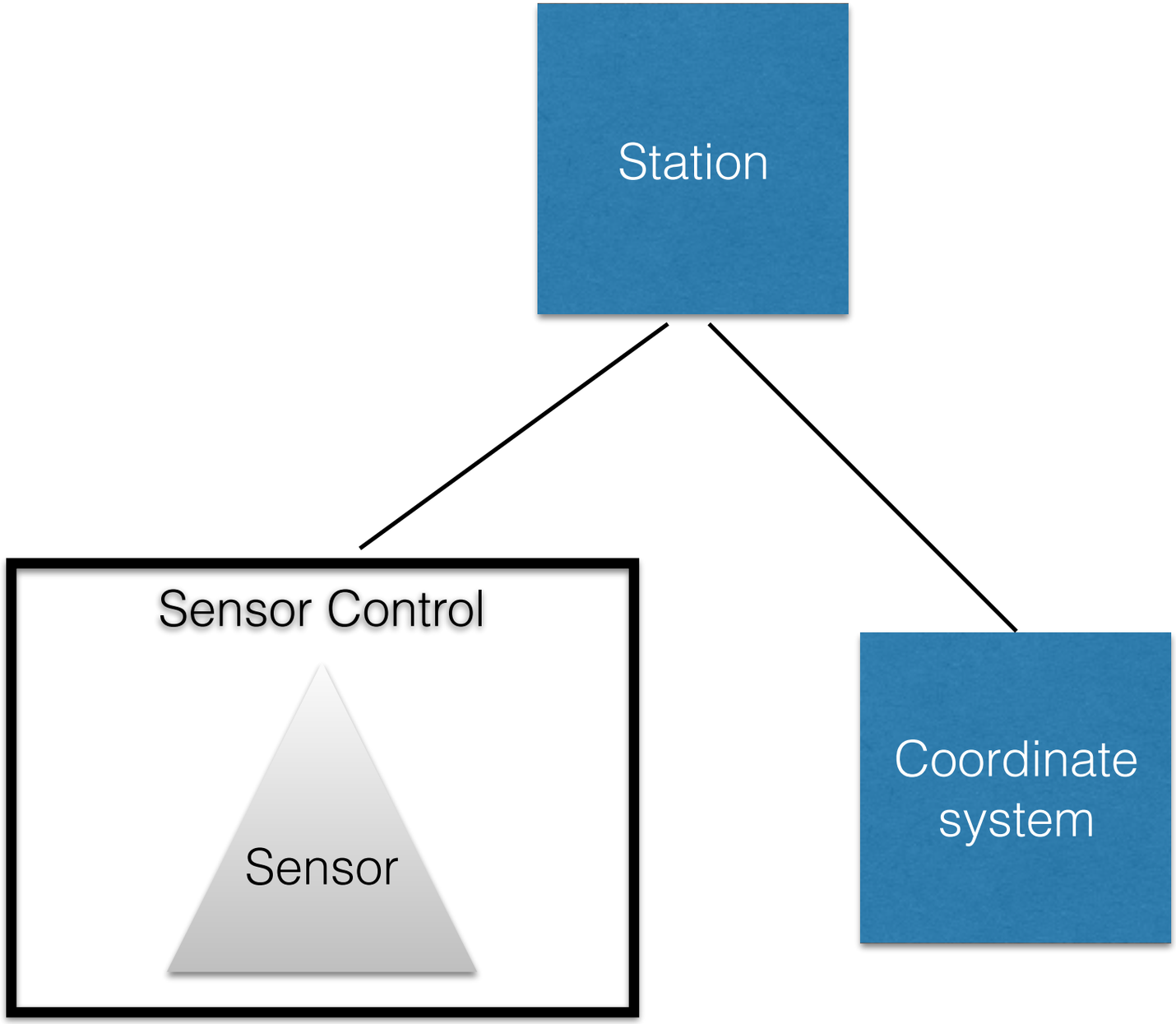
Feature

Geometry

Station

Transformations parameter

Coordinate system


Station
Sensor Control
Sensor
Coordinate system

Feature

Geometry

Station

Transformations parameter

Coordinate system

Transformations parameter

- Erzeugung beliebig vieler Koordinatensysteme
- Speichern der Transformationsparameter bei Start- und Zielsystem

Coordinate system

- Anwendung (Echtzeit-Transformation) bei Wechsel des Koordinatensystems

Feature

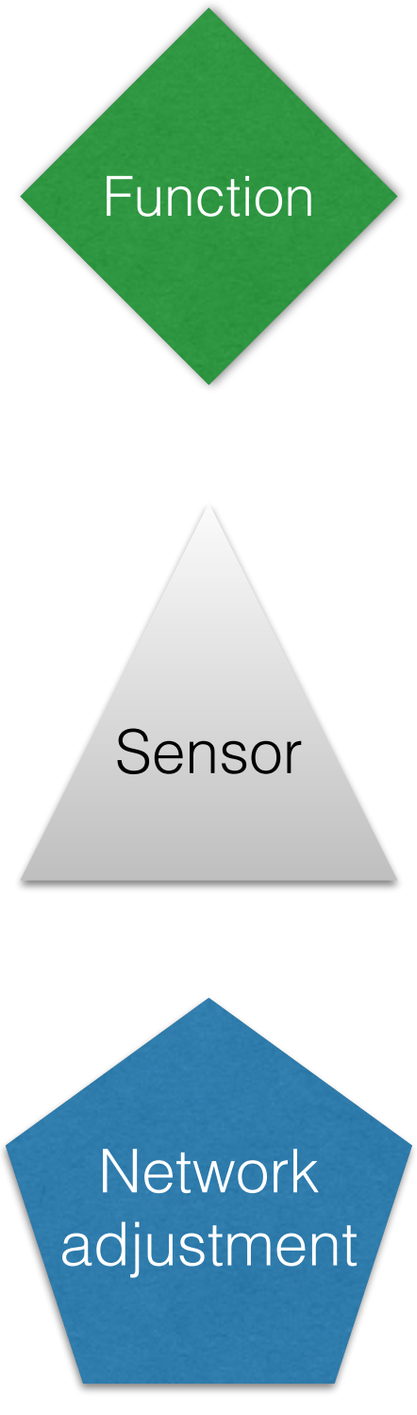
Geometry

Station

Transformations parameter

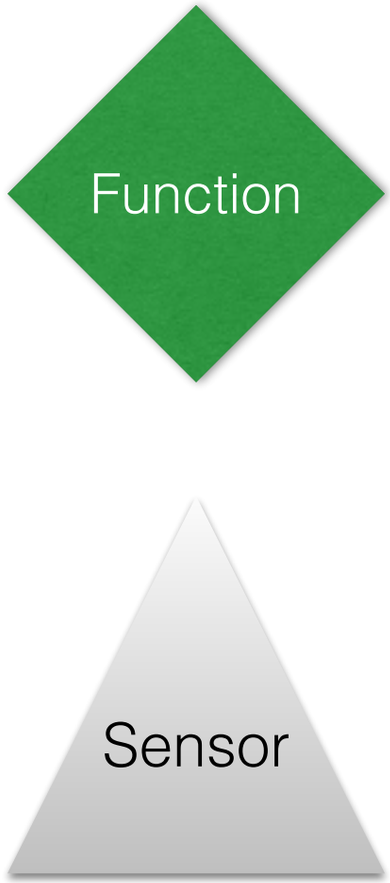
Coordinate system

Plugins



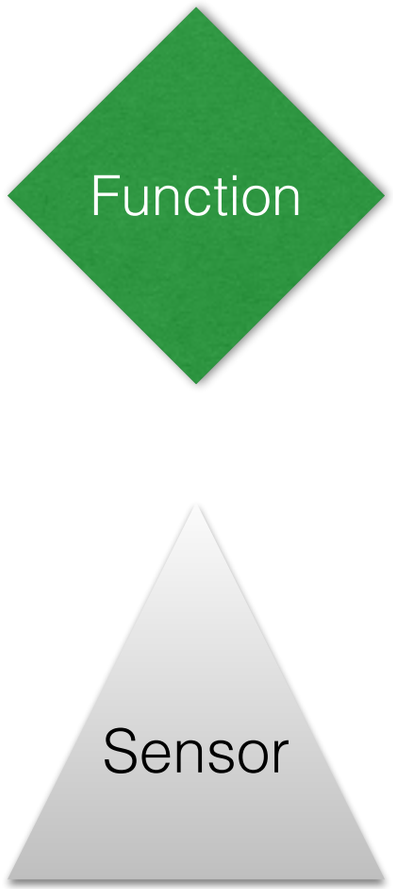
- Plugin - Logik in OpenIndy
  - 1 Plugin je Organisation bzw. Person
  - Implementierung von Funktionen, Sensoren und Netzausgleichungen
  - Plugin - Schnittstelle (factory pattern)
  - Metainformation zu jedem Plugin in einem JSON-File

Meta-JSON-File

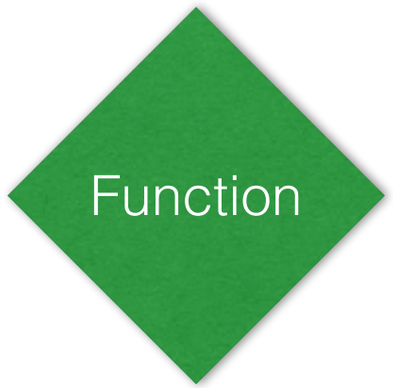


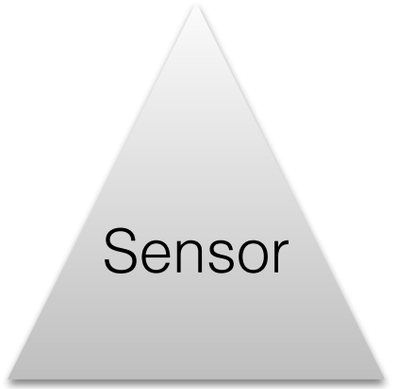

```
{
        "isOiPlugin": true,

        "name"  : "OpenIndy Default Plugin",
        "pluginVersion" : "0.0.1",
        "author"  : "openIndy team (br,ml,jw)",
        "compiler" : "clang 64bit",
        "operatingSystem" : "mac os",

        "description" : "Standardfunktionalitäten von OpenIndy",

        "dependencies" : false,
        "libPaths" : [
            { "name": "dependencies", "path": "app" }
        ]
}
```

Plugin-Schnittstelle




```
class OiPlugin
{

public:
    virtual ~OiPlugin(){}

    virtual QList<Sensor*> createSensors() = 0;
    virtual QList<Function*> createFunctions() = 0;
    virtual QList<NetworkAdjustment*> createNetworkAdjustments() = 0;
    virtual Sensor* createSensor(QString name) = 0;
    virtual Function* createFunction(QString name) = 0;
    virtual NetworkAdjustment* createNetworkAdjustment(QString name) = 0;

};
```

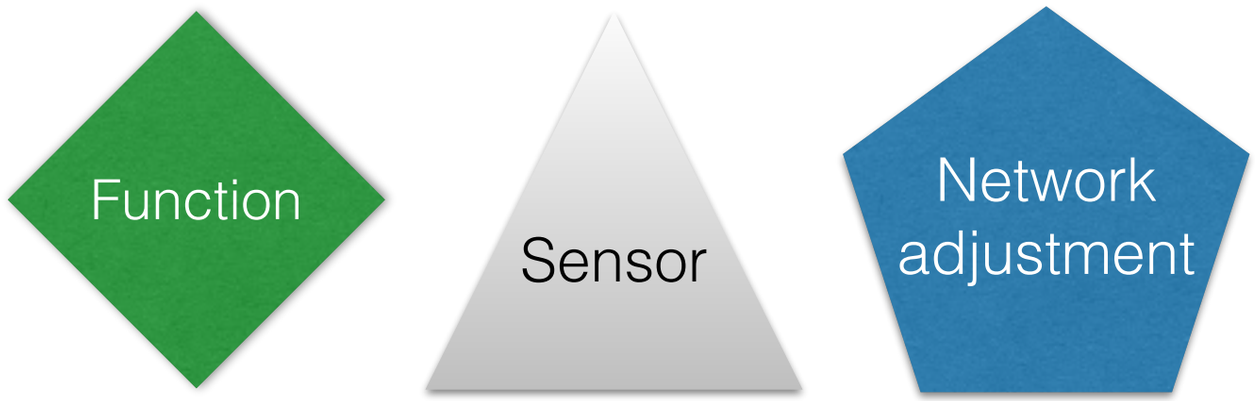




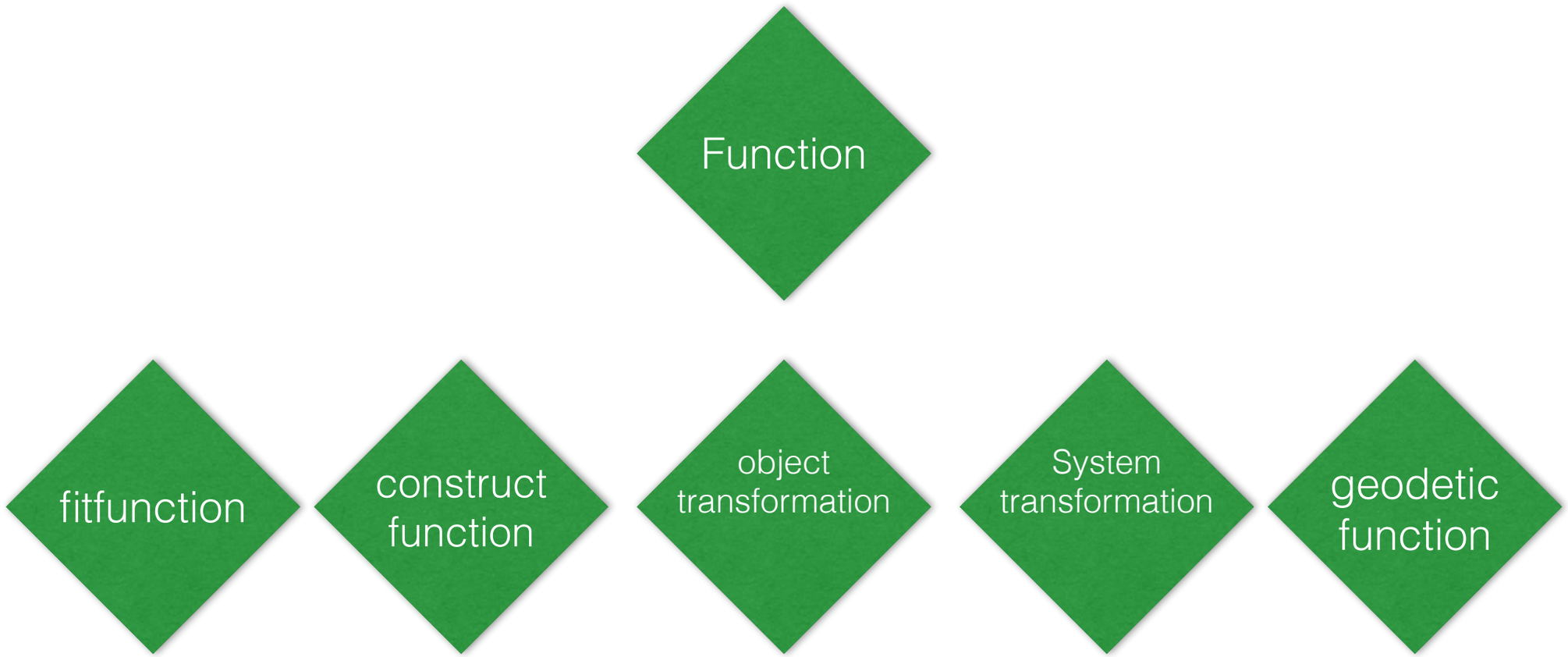

```
QList<Sensor*> DefaultPlugin::createSensors(){
    QList<Sensor*> resultSet;
    Sensor *pTracker = new PseudoTracker();
    Sensor *LeicaTachy = new LeicaTachymeter();
    Sensor *pSensor = new PseudoSensor();
    resultSet.append(pTracker);
    resultSet.append(LeicaTachy);
    resultSet.append(pSensor);
    return resultSet;
}
```

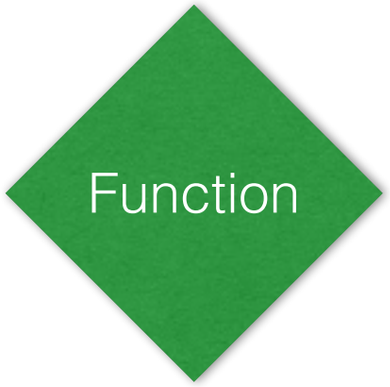
```
Sensor* DefaultPlugin::createSensor(QString name){
    Sensor *result = NULL;
    if(name.compare("PseudoTracker") == 0){
        result = new PseudoTracker();
    }else if(name.compare("LeicaTachymeter") == 0){
        result = new LeicaTachymeter();
    }else if(name.compare("PseudoSensor") == 0){
        result = new PseudoSensor();
    }
    return result;
}
```

Plugins



Zur Erweiterung der Auswertealgorithmen





- Implementierung einer Funktion:
  - Definition einer Klasse (im Plugin), die von einer der vorhandenen Funktionschnittstellen erbt.
  - Methoden, die jede Function-Implementierung überschreiben muss:

```
virtual QList<InputParams> getNeededElements() = 0;
virtual QList<Configuration::FeatureTypes> applicableFor() = 0;
virtual PluginMetaData* getMetaData() = 0;


struct InputParams{
    int index;
    Configuration::ElementTypes typeOfElement;
    QString description;
    bool infinite;
};
```

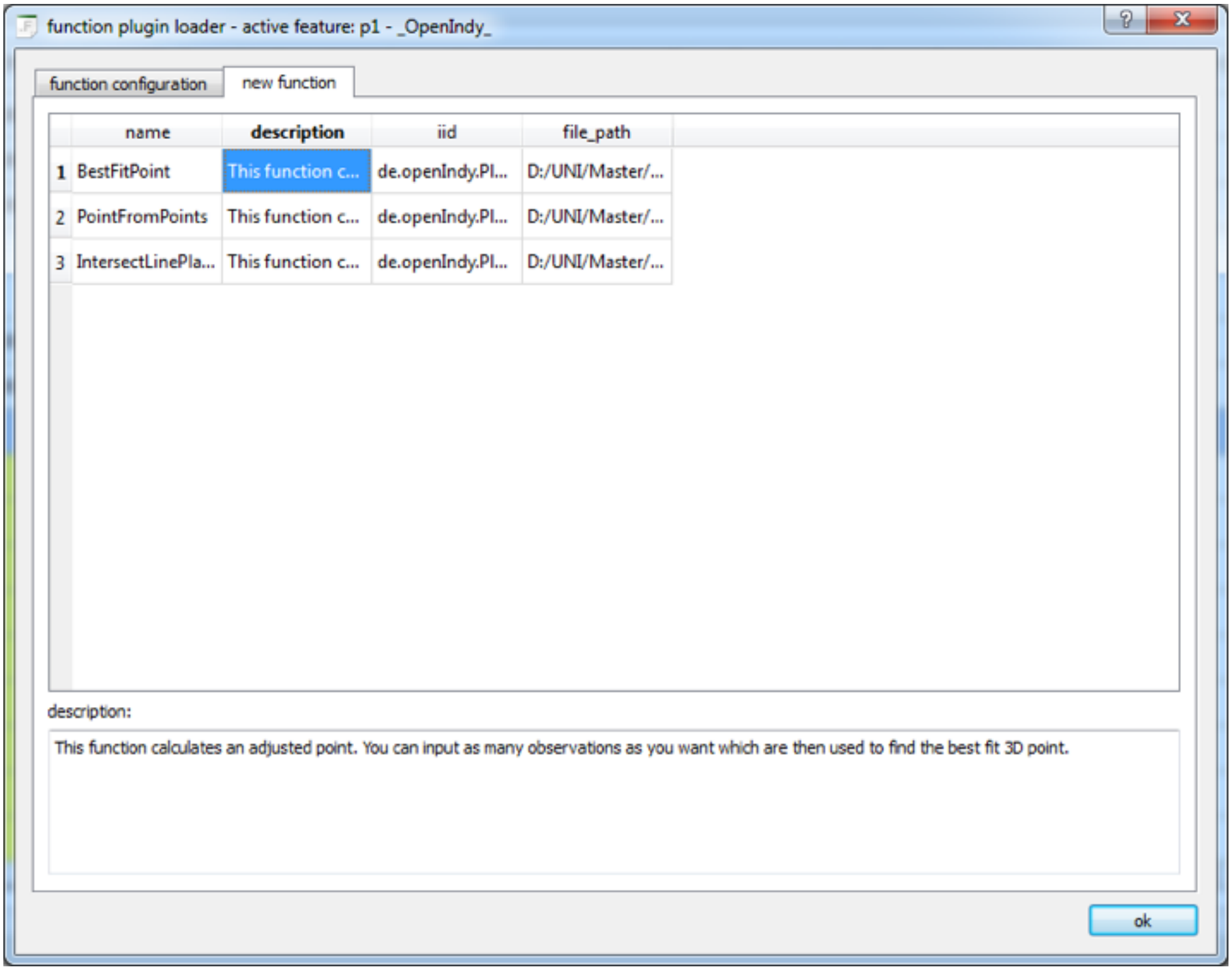
Function

- Implementierung einer Funktion:
  - Definition zusätzlicher Parameter
    - z.B Transformation mit oder ohne Maßstab

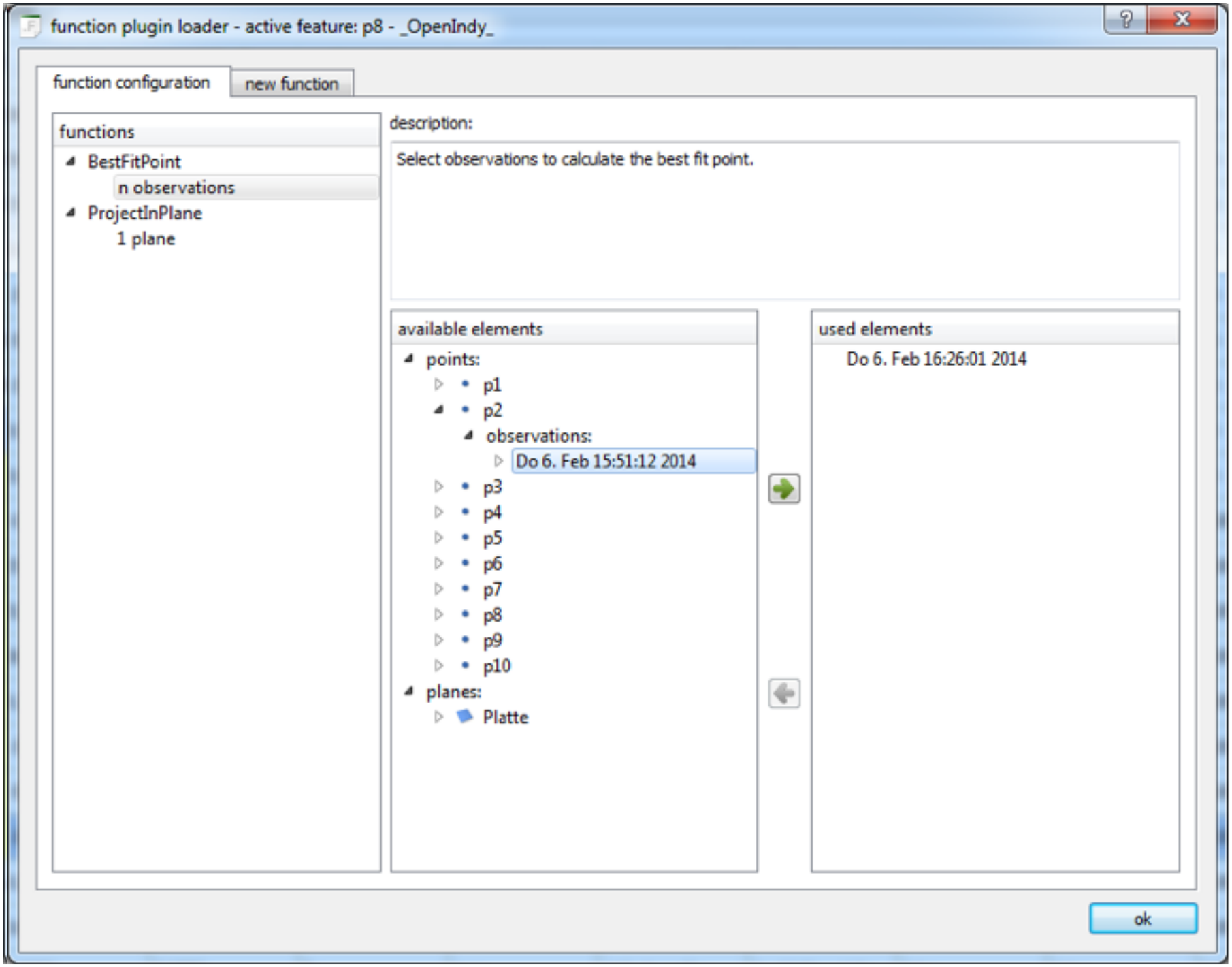
```
virtual QMap<QString, int> getIntegerParameter();
virtual QMap<QString, double> getDoubleParameter();
virtual QMap<QString, QStringList> getStringParameter();
```

```
QMap<QString, QStringList> TranslateByLine::getStringParameter(){
    QMap<QString, QStringList> result;
    QString key = "invert";
    QStringList value;
    value.append("no");
    value.append("yes");
    result.insert(key, value);
    return result;
}
```

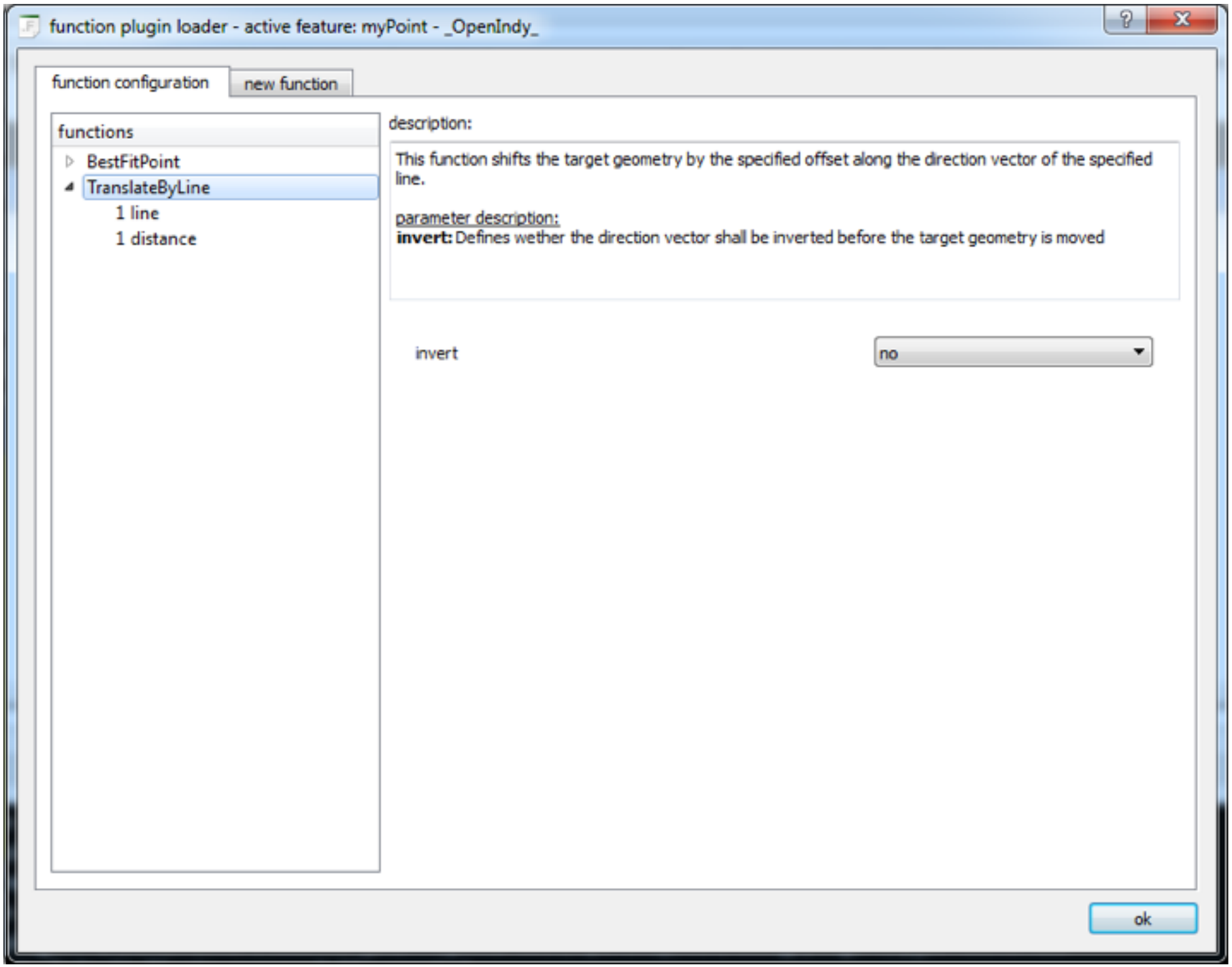
Function

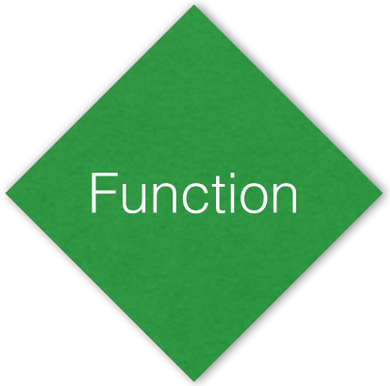

function plugin loader - active feature: p1 - _OpenIndy_
function configuration
new function
name
description
iid
file_path
1 BestFitPoint
This function c...
de.openIndy.Pl...
D:/UNI/Master/...
2 PointFromPoints
This function c...
de.openIndy.Pl...
D:/UNI/Master/...
3 IntersectLinePla...
This function c...
de.openIndy.Pl...
D:/UNI/Master/...
description:
This function calculates an adjusted point. You can input as many observations as you want which are then used to find the best fit 3D point.
ok

Function


function plugin loader - active feature: p8 - _OpenIndy_
function configuration
new function
functions
BestFitPoint
n observations
ProjectInPlane
1 plane
description:
Select observations to calculate the best fit point.
available elements
points:
p1
p2
observations:
Do 6. Feb 15:51:12 2014
p3
p4
p5
p6
p7
p8
p9
p10
planes:
Platte
used elements
Do 6. Feb 16:26:01 2014
ok

Function


function plugin loader - active feature: myPoint - _OpenIndy_
function configuration
new function
functions
BestFitPoint
TranslateByLine
1 line
1 distance
description:
This function shifts the target geometry by the specified offset along the direction vector of the specified line.
parameter description:
invert: Defines wether the direction vector shall be inverted before the target geometry is moved
invert
no
ok



- Implementierung einer Funktion:

- Implementierung der eigentlich Funktionalität
  - eine exec-Methode für jedes Feature
  - beliebig viele dieser Methoden können überschrieben werden
  - darin wird der Berechnungsalgorithmus inklusive Statistik programmiert

```
virtual bool exec(Station&);
virtual bool exec(CoordinateSystem&);
virtual bool exec(TrafoParam&);
virtual bool exec(Point&);
virtual bool exec(Line&);
virtual bool exec(Plane&);
virtual bool exec(Sphere&);
virtual bool exec(Circle&);
virtual bool exec(Cone&);
virtual bool exec(Cylinder&);
virtual bool exec(Ellipsoid&);
virtual bool exec(Hyperboloid&);
virtual bool exec(Nurbs&);
virtual bool exec(Paraboloid&);
virtual bool exec(PointCloud&);
virtual bool exec(ScalarEntityAngle&);
virtual bool exec(ScalarEntityDistance&);
```

Function

```
virtual QStringList getResultProtocol();
```

information abaout p1 - _OpenIndy_

observations, readings and statistic of the selected feature

sensor information | observations | readings | statistic | protocol

displayed function statistic: BestFitPoint

element id	vx m	vy m	vz m
8	0.1000	0.0333	-0.0000
10	-0.0000	0.0333	0.1000
12	-0.1000	-0.0667	-0.1000

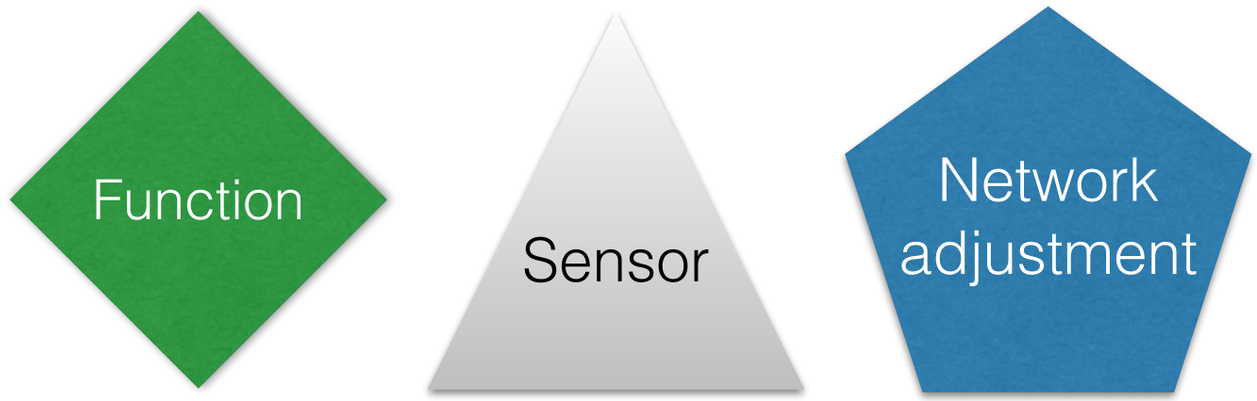
Qxx á posteriori | Sxx á priori

s0 á posteriori: 0.088192

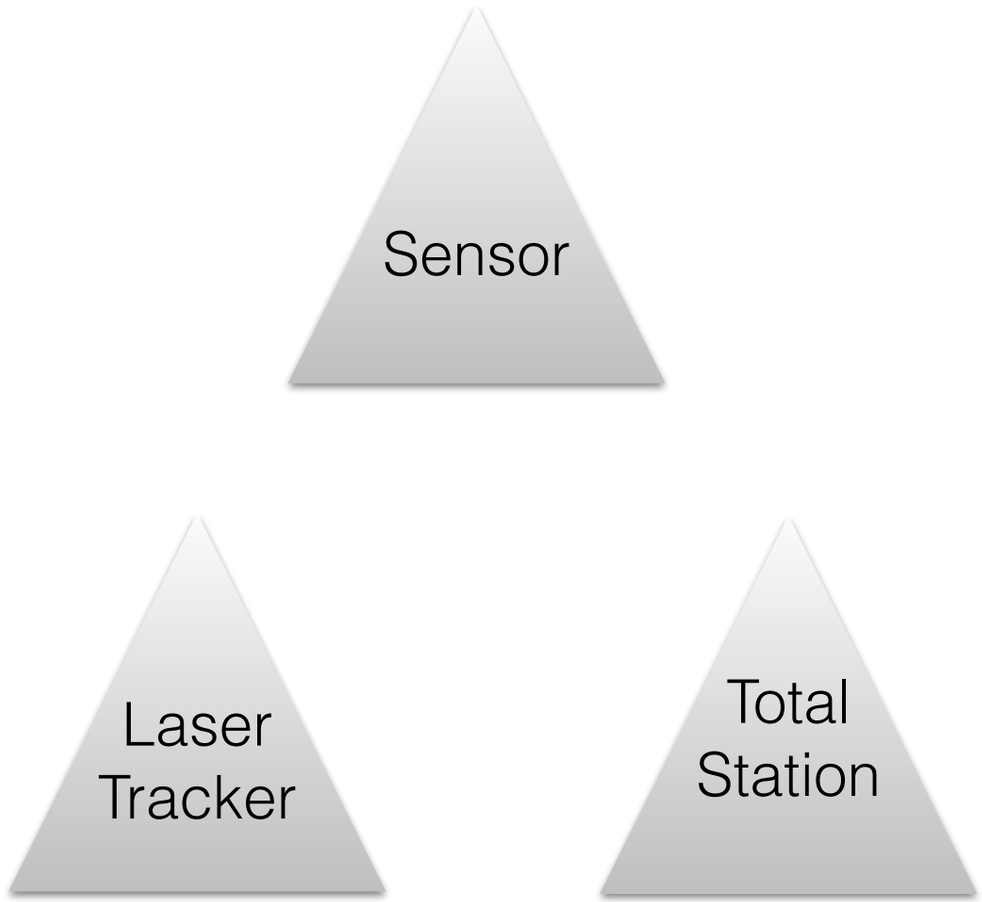
	1	2	3
1	0.002593	0.000000	0.000000
2	0.000000	0.002593	0.000000
3	0.000000	0.000000	0.002593

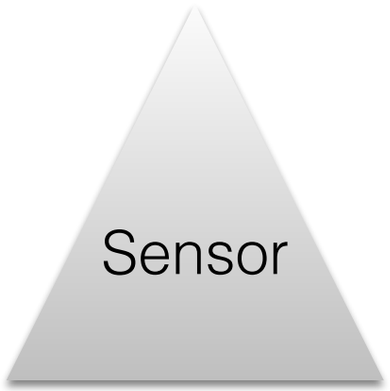
ok | cancel

Plugins

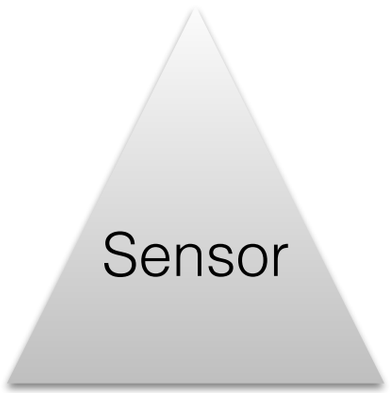


Für die Ansteuerung verschiedener Messinstrumente





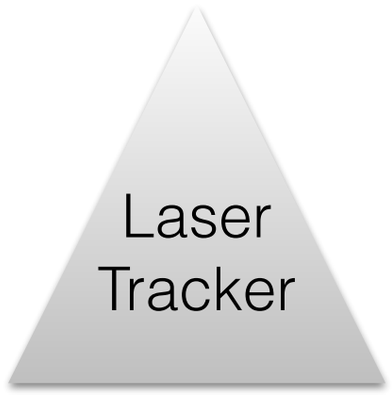

```
//attributes
    bool isConnected;
    bool dataStreamIsActive;
    Configuration::ReadingTypes streamFormat;
    OiEmitter myEmitter;
    SensorConfiguration* myConfiguration;
```




```
//getCapabilities
virtual PluginMetaData* getMetaData() = 0;

virtual QList<Configuration::ReadingTypes>* getSupportedReadingTypes() = 0;
virtual QList<Configuration::ConnectionTypes>* getConnectionType() = 0;
virtual QMap<QString,int>* getIntegerParameter() = 0;
virtual QMap<QString,double>* getDoubleParameter() = 0;
virtual QMap <QString, QStringList>* getStringParameter() = 0;

virtual QString getUndefinedReadingName(){return "undefined";}
virtual QMap<QString,double>* getUndefinedSigma(){return NULL;}
void setSensorConfiguration(SensorConfiguration* sConfig){myConfiguration = sConfig;}

// controlling
virtual bool accept(SensorControl*, Configuration::SensorFunctionalities) = 0;

virtual bool connectSensor(ConnectionConfig*) = 0;
virtual bool disconnectSensor() = 0;

virtual QList<Reading*> measure(MeasurementConfig*) = 0;

virtual void dataStream() = 0;
virtual void sendCommandString(QString) = 0;

virtual bool checkMeasurementConfig(MeasurementConfig*)=0;
```

Laser Tracker

```
  //! starts initialization
  virtual bool initialize() = 0;

  //! move laser tracker to specified position
  virtual bool move(double azimuth, double zenith, double distance,bool isrelativ) = 0;

  virtual bool move(double x, double y, double z) = 0;

  //! sets laser tracke to home position
  virtual bool home() = 0;

  //! turns motors on or off
  virtual bool changeMotorState(bool state) = 0;

  //! toggle between frontside and backside
  virtual bool toggleSightOrientation() = 0;

  //! compensation
  virtual bool compensation() = 0;
```




```
bool accept(SensorControl* s, Configuration::SensorFunctionalities f){

    switch(f){
    case(Configuration::eMoveAngle):
        return this->move(s->az,s->ze,s->dist,s->isMoveRelativ);
    case(Configuration::eMoveXYZ):
        return this->move(s->x_,s->y_,s->z_);
    case(Configuration::eInitialize):
        return this->initialize();
    case(Configuration::eHome):
        return this->home();
    case(Configuration::eToggleSight):
        return this->toggleSightOrientation();
    case(Configuration::eMotorState):
        return this->changeMotorState(s->isMState);
    case(Configuration::eCompensation):
        return this->compensation();
    }
    return false;
}
```

Signal & Slot (Qt)

OiEmitter

```
void OiEmitter::emitSendDataMap(QVariantMap*m){
    emit sendDataMap(m);
}
```

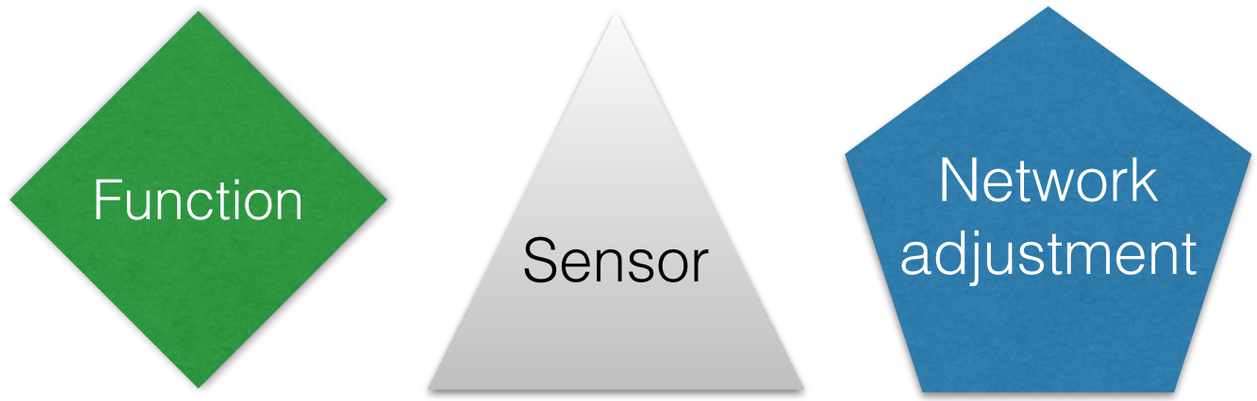
PseudoTracker

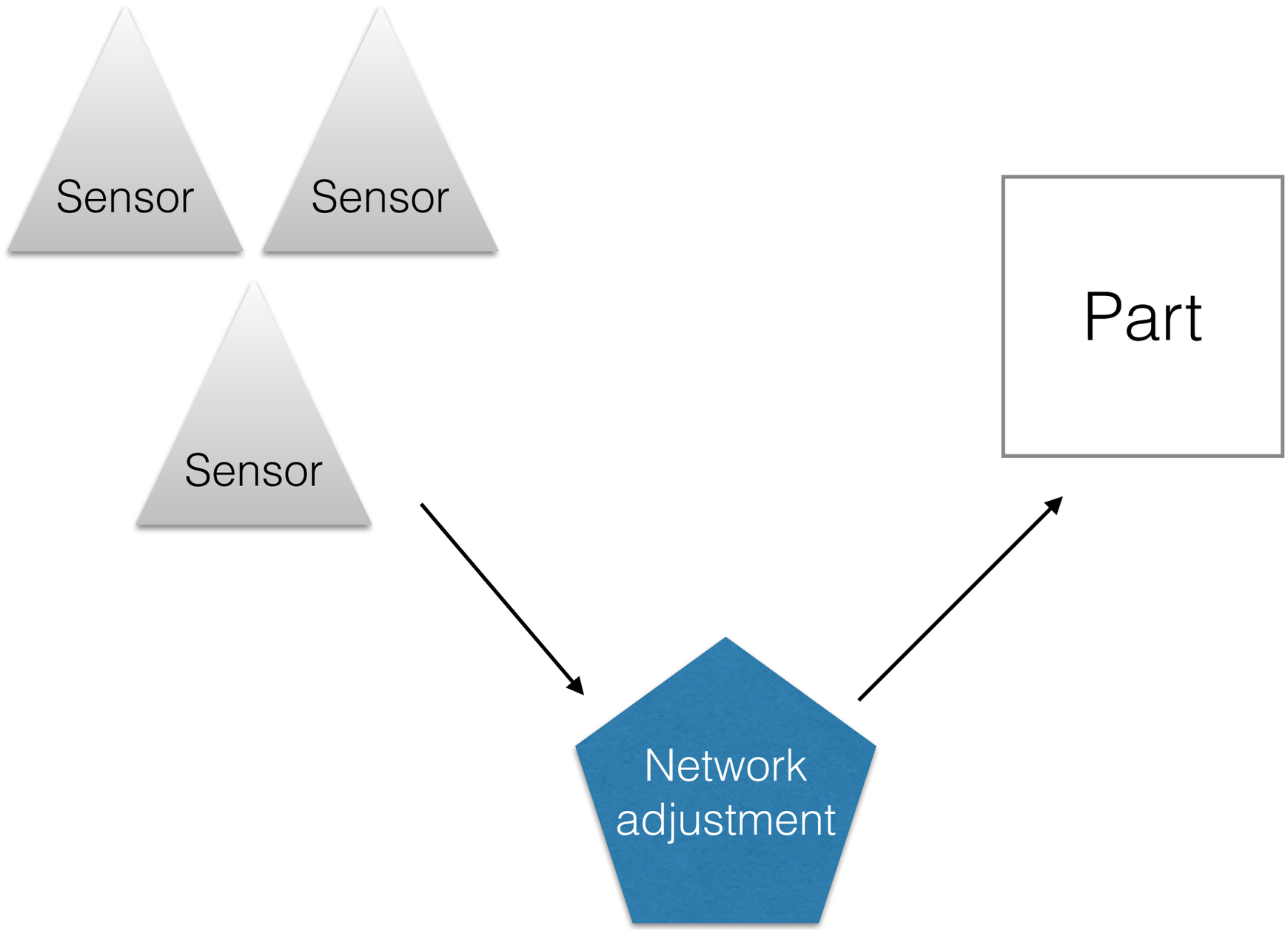
```
    x =((double) std::rand()/RAND_MAX)*(10.0-1.0)+1.0;
    y =((double) std::rand()/RAND_MAX)*(10.0-1.0)+1.0;
    z =((double) std::rand()/RAND_MAX)*(10.0-1.0)+1.0;

    m->insert("x",x);
    m->insert("y",y);
    m->insert("z",z);

    QThread::msleep(50);
    myEmitter.emitSendDataMap(m);
    QThread::msleep(50);
```

OpenIndy (watchwindow)

```
connect(&myStation->instrument->myEmitter,SIGNAL(sendDataMap(QVariantMap*)),this,SLOT(setLCDNumber(QVariantMap*)));
if (myStation->instrument->dataStreamIsActive == false){
    myStation->emitStartStream();
}else{
    myStation->stopStream();
}
```

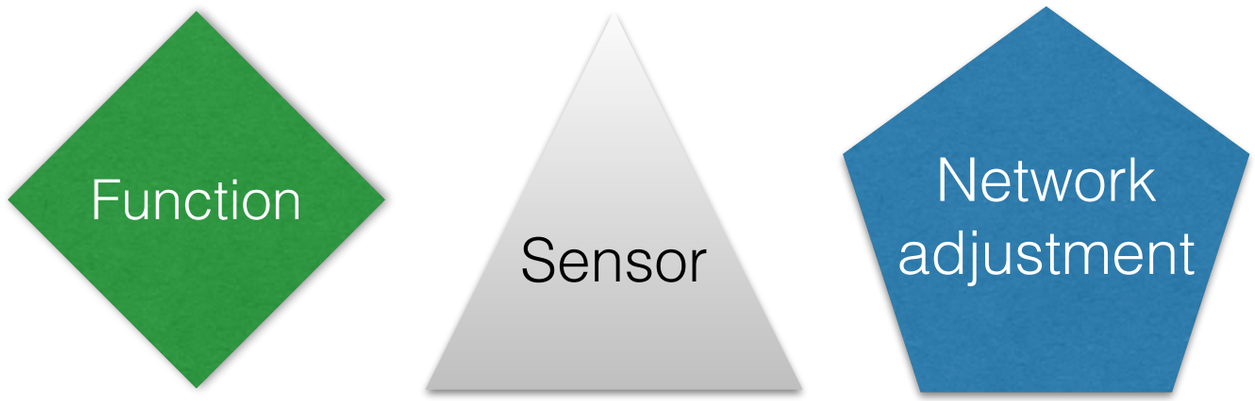
Plugins





Zum Ausführen einer anderen Art der Stationsbestimmung (Bündelausgleichung)

Plugins



Anwendungsbeispiel

<u>Aufgabe:</u>

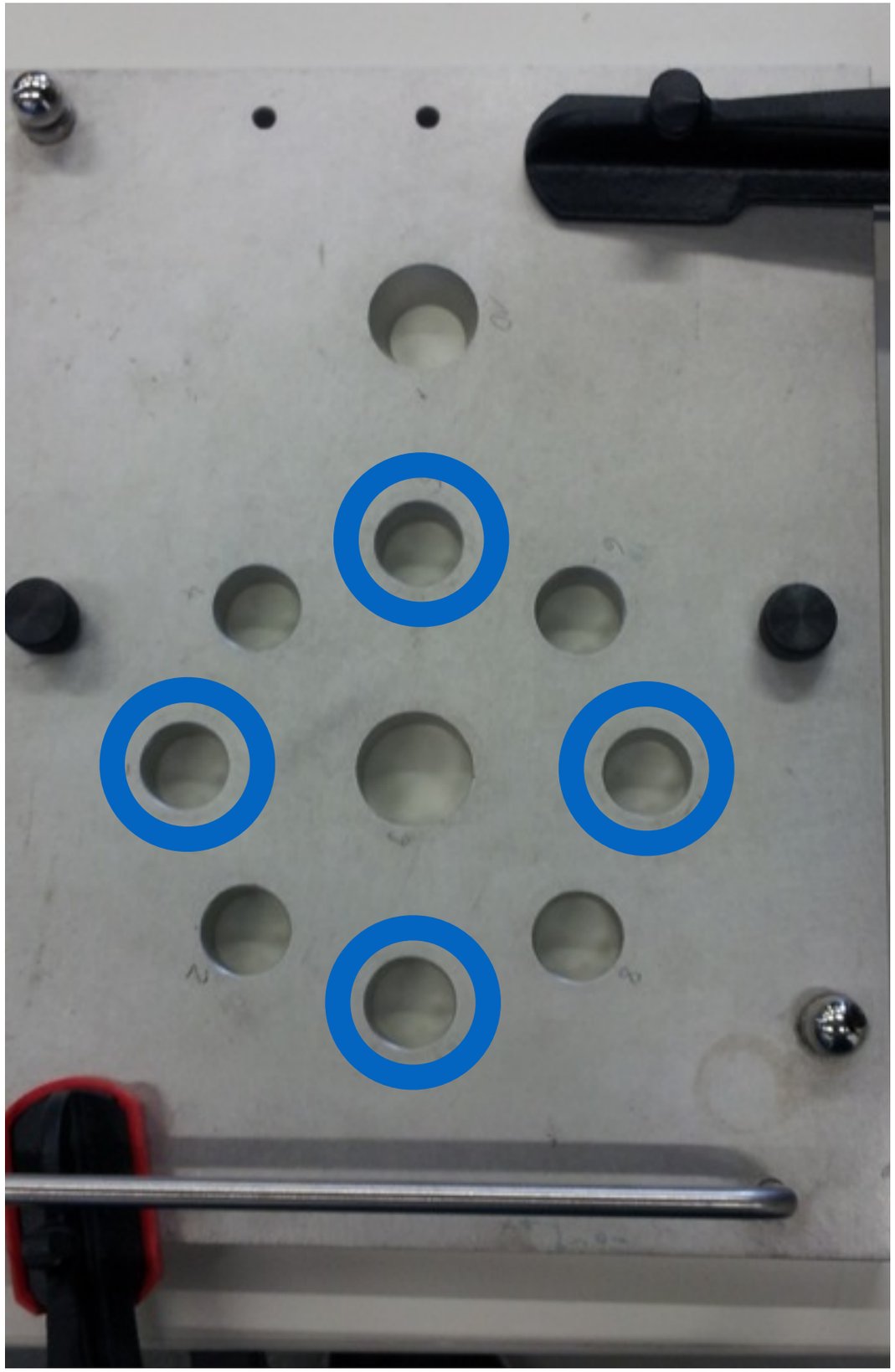
Mittelpunktsbestimmung der Bohrungen in Bezug auf „CAD-Solldaten“

<u>Vorgehensweise:</u>

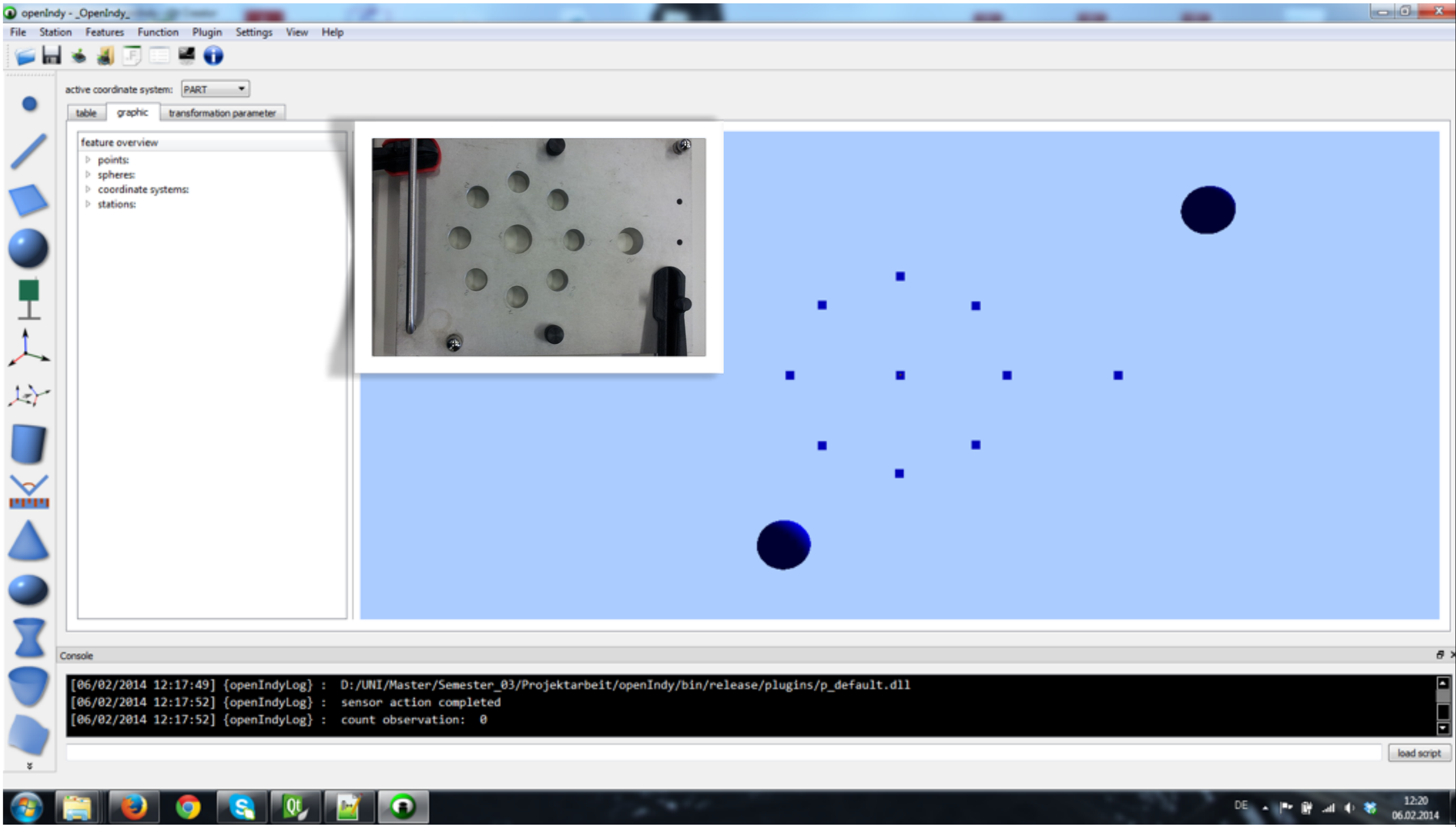
Sollkoordinaten einlesen

Punkte mit Hilfe eines Tachymeters und Kugelreflektor messen.

Transformation über **Passpunkte** in das CAD-System



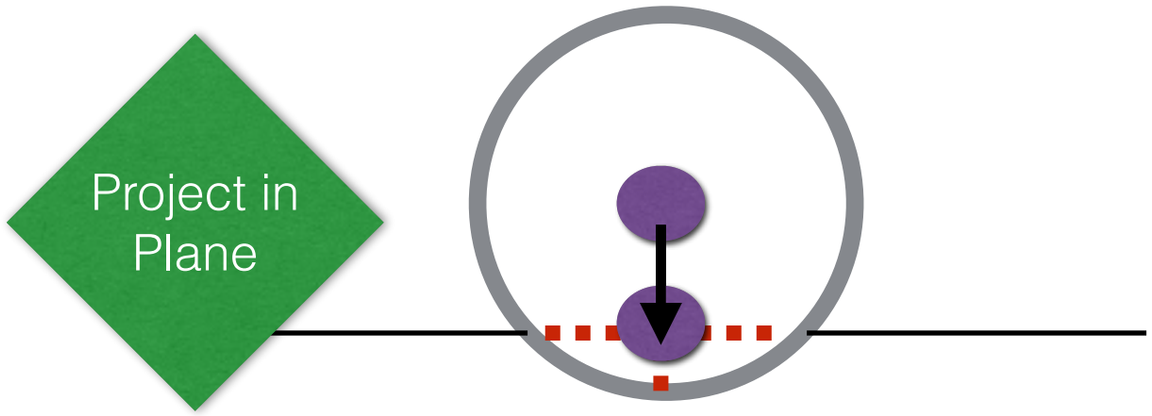
Sollkoordinaten einlesen



Punktmessung

Messung der Grundebene

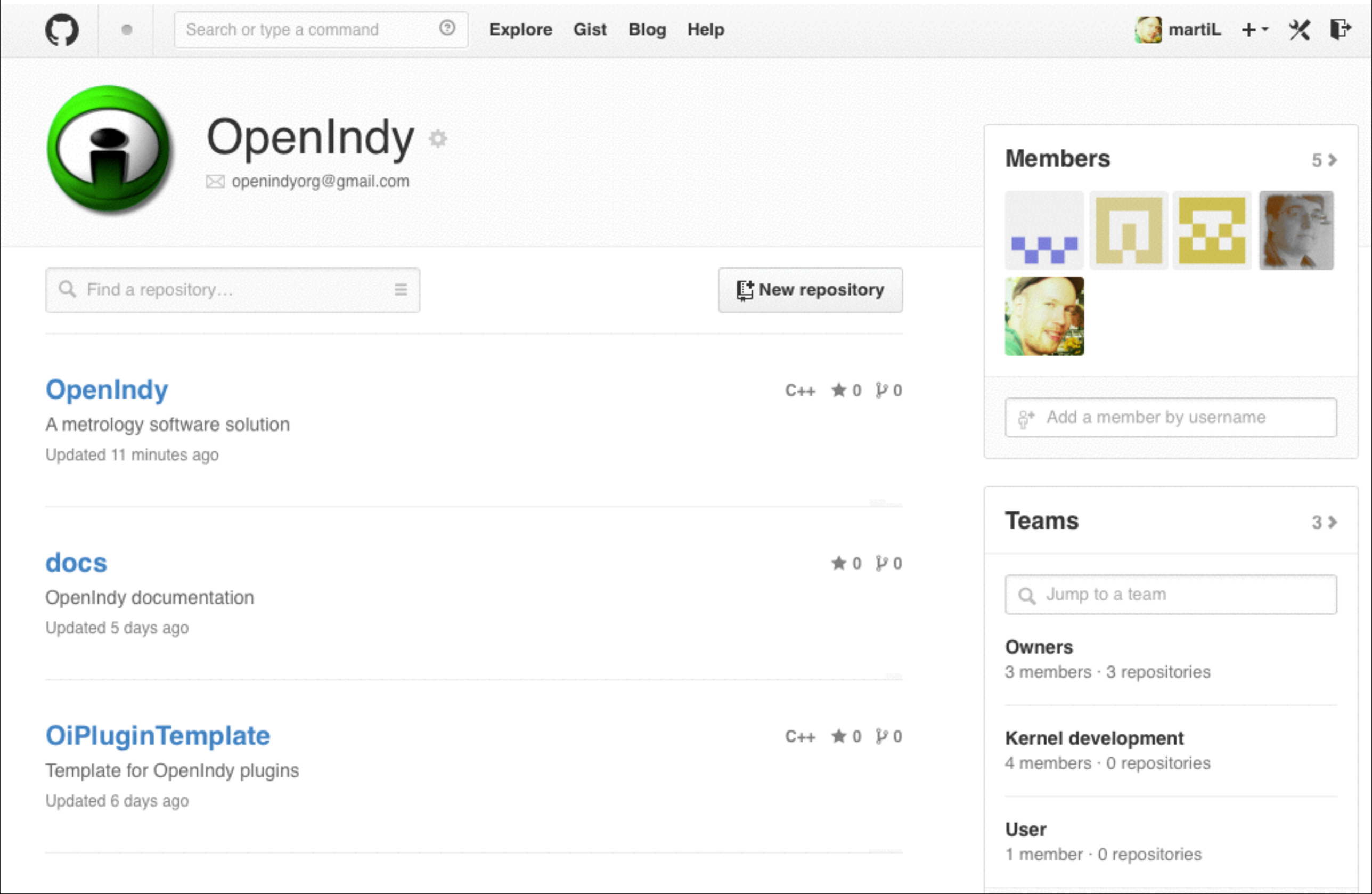
Messung des Punktes mit anschließender Projektion in die Grundebene



Transformation in das Objektkoordinatensystem

Feature name	X [mm]	Y [mm]	Z [mm]
p9 **(actual)**	-0.1188	0.1031	0.1953
p9 **(nominal)**	0.0000	0.0000	0.0000

Projektmanagement



PUBLIC OpenIndy / OiPluginTemplate

Unwatch 3 | Star 0 | Fork 0

Fork this repo

Template for OpenIndy plugins — Edit

4 commits | 1 branch | 0 releases | 2 contributors

branch: master | OiPluginTemplate /

Update README.md

martiL authored 3 days ago — latest commit 4342707f2e

lib	Qt project template plugin for OpenIndy created	5 days ago
src	Qt project template plugin for OpenIndy created	5 days ago
.gitignore	Qt project template plugin for OpenIndy created	5 days ago
OiTemplatePlugin.pro	Qt project template plugin for OpenIndy created	5 days ago
README.md	Update README.md	3 days ago
metaInfo.json	Qt project template plugin for OpenIndy created	5 days ago
p_factory.cpp	Qt project template plugin for OpenIndy created	5 days ago
p_factory.h	Qt project template plugin for OpenIndy created	5 days ago

Code
Issues 1
Pull Requests 0
Wiki
Pulse
Graphs
Network
Settings

HTTPS clone URL
https://github.com/
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

README.md

OiPluginTemplate

This repository includes a template for an OpenIndy plugin. The instructions for creating your own plugins is described in the OiPluginTemplate wiki.

IDE

OpenIndy is developed with the Qt framework (Qt libs + Qt Creator IDE). You can download the framework here

PUBLIC OpenIndy / OiPluginTemplate

Unwatch 3 | Star 0 | Fork 0

Fork this repo

Template for OpenIndy plugins — Edit

4 commits | 1 branch | 0 releases | 2 contributors

branch: master | OiPluginTemplate /

Update README.md

martiL authored 3 days ago — latest commit 4342707f2e

lib	Qt project template plugin for OpenIndy created	5 days ago
src	Qt project template plugin for OpenIndy created	5 days ago
.gitignore	Qt project template plugin for OpenIndy created	5 days ago
OiTemplatePlugin.pro	Qt project template plugin for OpenIndy created	5 days ago
README.md	Update README.md	3 days ago
metaInfo.json	Qt project template plugin for OpenIndy created	5 days ago
p_factory.cpp	Qt project template plugin for OpenIndy created	5 days ago
p_factory.h	Qt project template plugin for OpenIndy created	5 days ago

Code
Issues 1
Pull Requests 0
Wiki
Pulse
Graphs
Network
Settings

HTTPS clone URL
https://github.com/

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

README.md

OiPluginTemplate

This repository includes a template for an OpenIndy plugin. The instructions for creating your own plugins is described in the OiPluginTemplate wiki.

IDE

OpenIndy is developed with the Qt framework (Qt libs + Qt Creator IDE). You can download the framework here

Shall we add metainfo.json to .gitignore? #1

Edit | New Issue

Browse Issues | Milestones

New Issue

Everyone's Issues 1
Assigned to you 0
Created by you 1
Mentioning you 0

1 Open | 0 Closed | Sort: Newest

Close | Label | Assignee | Milestone

Shall we add metainfo.json to .gitignore? question #1
Opened by martiL 4 days ago 1 comment

Keyboard shortcuts available

No milestone selected

Labels

question	1
bug	0
duplicate	0
enhancement	0
invalid	0
wontfix	0

Manage Labels

Some attributes of the json file have to be editable for custom plugins.

Function

Introduction

Every feature in OpenIndy can have any number of functions. There are construct-, fit- and geodetic functions and furthermore object transformations and system transformations. The list below gives you a review of those function types.

function type	description
fit function	If overdetermination is present, fit functions have the purpose to calculate the adjusted parameters of a geometry by using observations. So for example a fit function can calculate the center of n xyz-observations.
construct function	Construct functions are used to calculate geometries by using other geometries. For example the intersection of a line and a plane results in a new point.
geodetic function	Geodetic functions are intended to calculate special geodetic tasks like spatial intersection etc..
object transformation	In contrast to fit- and construct functions, object transformations do not define a geometry, but change a previously defined one. A point which has been fit before can for instance be moved along a line by a certain amount.
system transformation	System transformations are used to calculate the parameters of a transformation from one coordinate system to another one.

These function types are implemented as classes which extend the class Function. When you want to write your own function in your plugin then you have to extend one of the classes FitFunction, ConstructFunction, GeodeticFunction, ObjectTransformation or SystemTransformation. Please do not try to extend the class Function itself. While this currently works for some kind of function types for others it does not because additional methods and attributes are necessary which are defined only in the subclasses.

Let us first look at the class Function itself. There are some methods that you always have to override (pure virtual methods). This methods are pure virtual in the five subclasses of the class Function, too.:

Pure virtual methods

```
virtual QList<InputParams> getNeededElements() = 0;
virtual QList<Configuration::FeatureTypes> applicableFor() = 0;
```

Ergebnisse der Projektarbeit

- Konzepte modelliert und in einer ersten Evolutionsstufe implementiert:
  - GUI implementiert (Tabellenansicht, Graphikansicht, Konsole)
  - Plugins (Sensoren und Funktionen)
  - konsequente Varianzfortpflanzung
  - XML-Austauschformat
  - Multiplattform (Windows, Mac, Linux)
  - Erste Messaufgabe erfolgreich durchgeführt
- Open Source - Online Portal eingerichtet
- Online Dokumentation und Anwenderdokumentation
- Präsentation auf den Oldenburger 3D Tagen
- Publikation im Tagungsband der Oldenburger 3D Tage

Ausblick

<u>Weiterführung durch</u>:

- **Masterarbeiten** (virtueller Laser Tracker, Geometrieerkennung, Temperaturkompensation, etc.)

- **hochschulinterne Module an der FH Mainz** (bspw. Prozessorientierte Programmierung SS2014)

- **sigma3D GmbH**

- **Interesse geäußert durch Dirk Stallmann** (HafenCity Universität Hamburg)

Quellen

Drixler, Erwin (1993): Analyse der Form und Lage von Objekten im Raum. Dissertation. Deutsche Geodätische Kommission bei der Bayerischen Akademie der Wissenschaften. Reihe C Heft Nr. 409

Qt Dokumentation qt-project.org/doc/ Version 5.1 National Science Foundation http://www.netlib.org/lapack/ National Science Foundation http://www.netlib.org/blas/

Möser, Michael (2007): 20 Jahre Industrievermessung. In: Tagungsband zum 3. Dresdener Ingenieurgeodäsietag, Schriftenreihe des Geodätischen Instituts der TU Dresden, Nr. 4, S. 7-19.

http://tu-dresden.de/ die_tu_dresden/fakultaeten/fakultaet_forst_geo_und_hydrowissenschaften/fachricht ung_geowissenschaften/gi/ig/ termine/papers/moeser.pdf (Zugriff 18.02.2014)

Sanderson, Conrad http://arma.sourceforge.net/

Acker, Kai; Hettich, Florian: Open Source – Non Profit Engagement oder Service Geschäft? In: Arbeitsbericht der Veranstaltung Teledienste – Trendanalyse und Bewerung (2001), März

Henzelmann, Bettina; Sauer, Volker: OpenSource aus ökonomischer Sicht? Seminararbeit – Technische Universität Darmstadt (2006), November

Glaeßer, Lothar: Open Source Software. Projekte, Geschäftsmodelle, Rechtsfragen, Anwendungsszenarien – was IT-Entscheider und Anwender wissen müssen. Publicis Corporate Publishing, 2004 -ISBN 3895782408

Fogel, Karl: Producing Open Source Software: How to Run a Successful Free Software Project. CreativeCommons Attribution-ShareAlike license, 2005

Hook, Brian (2006): Portabler Code - Einführung in die plattformunabhängige Softwareentwicklung.

Prof. Dr. Bittel, Oliver (1998): Programmierrichtlinien für C++

Nikolisin, Harald (2007): Portabler Code. (http: //hochglanz.dyndns.org/wp- content/uploads/2007/11/portabler_code_cc.pdf) (12.10.2013)