ICP

Synchronisation der beiden Topics

```
  typedef message_filters::sync_policies::ApproximateTime<sensor_msgs::LaserScan,
sensor_msgs::LaserScan> MySyncPolicy;

  message_filters::Subscriber<sensor_msgs::LaserScan> scan_sub (n, "/scan", 1);
  message_filters::Subscriber<sensor_msgs::LaserScan> model_sub (n, "/model", 1);

  message_filters::Synchronizer<MySyncPolicy> sync (MySyncPolicy (10), scan_sub,
model_sub);
  sync.registerCallback (boost::bind (&laserCallback, _1, _2));
```

Definition einer Maximal-Distanz

```
max_dist2 = 0.3f * 0.3f;
```

In diesem Fall 0.3 m. Wir quadrieren, da wir eine absolute Distanz benötigen, um Scans "vor" dem anderen (positiv verschoben) und "hinter" dem anderen (negativ verschoben) matchen zu können.

Umwandlung von Laserscan zu Point Cloud

```
projector_.projectLaser (*model, *model_cloud);
projector_.projectLaser (*scan, *tmp_cloud);
```

In diesem Fall mit [laser_geometry]

Berechnung der Korrespondenzen

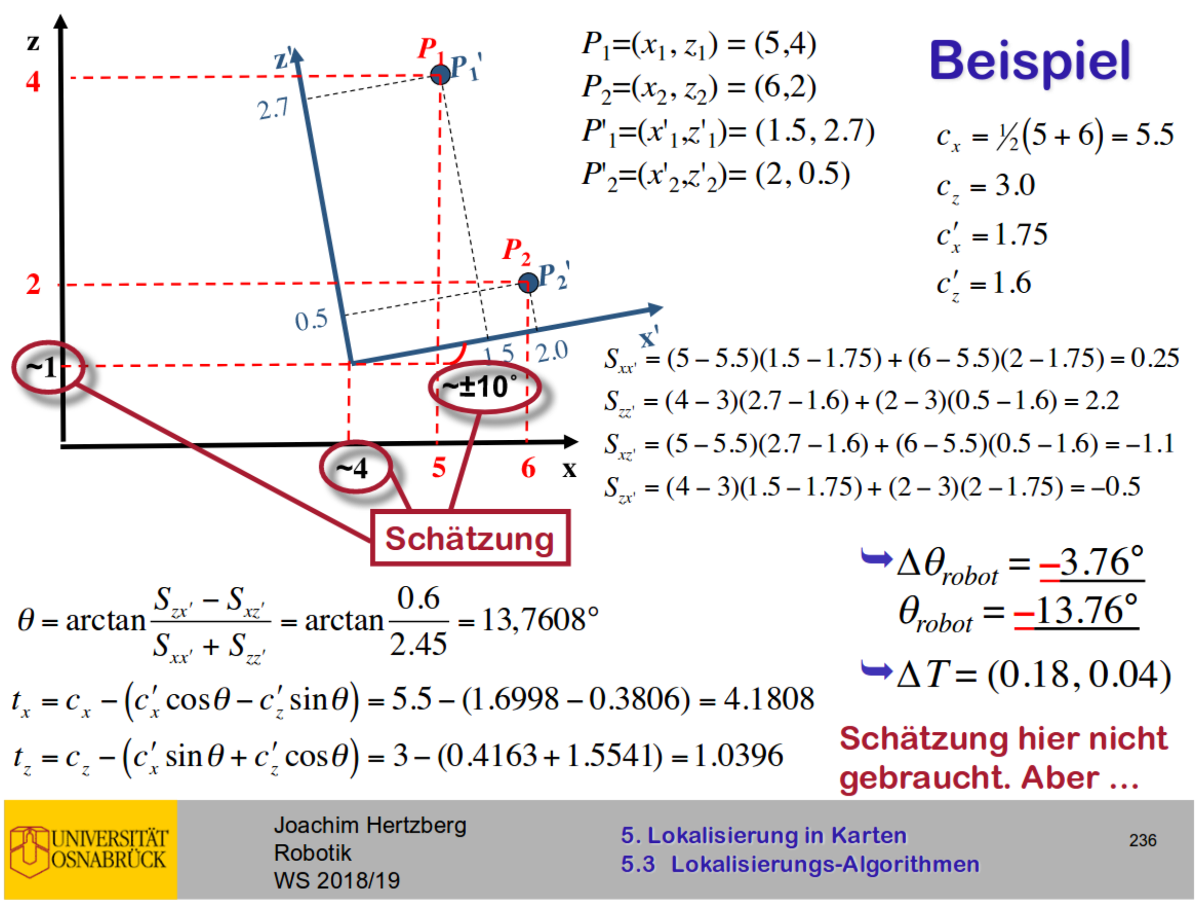
```
  typedef std::pair<geometry_msgs::Point32, geometry_msgs::Point32> CorrPair;
  typedef std::vector<CorrPair> CorrVec;

  std::vector<geometry_msgs::Point32>::const_iterator scan_it, model_it,
  best_it;

  scan_it = scan_cloud->points.begin ();
  while (scan_it != scan_cloud->points.end ())
  {
    min_dist2 = max_dist2;
    got_corr = false;
    model_it = model_cloud->points.begin ();
    while (model_it != model_cloud->points.end ())
    {
      curr_dist2 = sqrdDist (*scan_it, *model_it);
      if (curr_dist2 < min_dist2)
```

```
      {
        min_dist2 = curr_dist2;
        best_it = model_it;
        got_corr = true;
      }
      model_it++;
  }

  if (got_corr)
    {
      point_correspondences.push_back (CorrPair (*best_it, *scan_it));
  }
    scan_it++;
  }
```

Berechnen der Transformation nach Slides 234+, Buch S. 183

```
tf::Transform
calcTransFormation (const CorrVec &point_correspondences)
{
  float c1_x = 0.0f, c1_y = 0.0f, c2_x = 0.0f, c2_y = 0.0f;
  float s_xx = 0.0f, s_xy = 0.0f, s_yx = 0.0f, s_yy = 0.0f;

  // compute centroids
  CorrVec::const_iterator it = point_correspondences.begin ();
  while (it != point_correspondences.end ())
  {
    c1_x += it->first.x;
    c1_y += it->first.y;

    c2_x += it->second.x;
    c2_y += it->second.y;
    it++;
  }

  c1_x /= point_correspondences.size ();
  c1_y /= point_correspondences.size ();
  c2_x /= point_correspondences.size ();
  c2_y /= point_correspondences.size ();

  // compute the other terms needed for error minimization (see slide 234)
  it = point_correspondences.begin ();
  while (it != point_correspondences.end ())
  {
    s_xx += (it->first.x - c1_x) * (it->second.x - c2_x);
    s_xy += (it->first.x - c1_x) * (it->second.y - c2_y);
    s_yx += (it->first.y - c1_y) * (it->second.x - c2_x);
    s_yy += (it->first.y - c1_y) * (it->second.y - c2_y);
    it++;
  }

  float theta = atan2 (s_yx - s_xy, s_xx + s_yy);
```

```
  float tx = c1_x - (c2_x * cos (theta) - c2_y * sin (theta));
  float ty = c1_y - (c2_x * sin (theta) + c2_y * cos (theta));

  // create and return a tf::Transform from the given information
  // As in previous homeworks
  tf::Transform trans;
  tf::Quaternion rot;
  rot.setEuler (0.0f, 0.0f, theta);

  trans.setRotation (rot);
  trans.setOrigin (tf::Vector3 (tx, ty, 0.0));
  return trans;
}
```



Updaten der globalen Positionsschätzung

```
global_transform = global_transform * trans;
```

`trans` beschreibt die errechnete Transformation zwischen den beiden Scans.

Benutzung von Markern, zur Visualisierung

```
visualization_msgs::Marker icp_lines;
icp_lines.header = model->header;
icp_lines.ns = "icp_cors";
```

```
icp_lines.id = 0;
icp_lines.action = visualization_msgs::Marker::ADD;
icp_lines.type = visualization_msgs::Marker::LINE_LIST;
icp_lines.pose.orientation.w = 1.0;
icp_lines.scale.x = 0.01;
icp_lines.color.r = 0.0;
icp_lines.color.g = 1.0;
icp_lines.color.b = 1.0;
icp_lines.color.a = 1.0;
drawLines (icp_corrs, icp_lines);

line_pub.publish (icp_lines);
```

mit Erstellen der Linien

```
void  drawLines (const CorrVec &point_correspondences, visualization_msgs::Marker
&lines)
{
  CorrVec::const_iterator it = point_correspondences.begin ();
  lines.points.reserve (point_correspondences.size () * 2);
  geometry_msgs::Point p;
  while (it != point_correspondences.end ())
  {
    p.x = it->first.x;
    p.y = it->first.y;
    lines.points.push_back (p);
    p.x = it->second.x;
    p.y = it->second.y;
    lines.points.push_back (p);
    it++;
  }
}
```