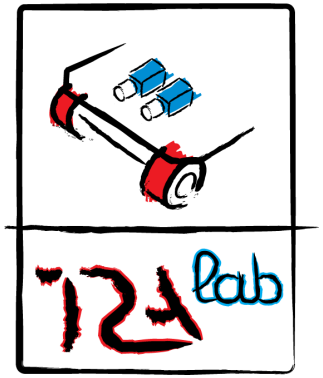


Occupancy Grid Mapping: GMapping from the trenches

IRA Reading Group
04/05/2012

Francesco Sacchi

Outline

- Mapping
- Occupancy Grid Mapping
- MAP
- GMapping
- Conclusion

Mapping

- Huge hypotesis space
- Known poses

Hardness due to:

- Size
- Noise in perception and actuation
- Perceptual ambiguity
- Loops

OGM - Introduction

- Family of algorithms
- Field of binary random variables
- Approximate posterior estimation
- Post processing

OGM - Algorithm

General problem: $p(m|z_{1:t}, x_{1:t})$

OGM - Algorithm

General problem: $p(m|z_{1:t}, x_{1:t})$

OGM problem: $p(m_i|z_{1:t}, x_{1:t})$

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$$

OGM - Algorithm

General problem: $p(m|z_{1:t}, x_{1:t})$

OGM problem: $p(m_i|z_{1:t}, x_{1:t})$

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$$

Binary bayes filter

OGM - Algorithm

General problem: $p(m|z_{1:t}, x_{1:t})$

OGM problem: $p(m_i|z_{1:t}, x_{1:t})$

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$$

Binary bayes filter

Dependencies??

OGM - Algorithm

1 Algorithm occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$):
2 for all cells m_i do
3 if m_i in perceptual field of z_t then
4 $l_{t,i} = l_{t-1,i} + \text{inverse\_sensor\_model}(m_i, x_t, z_t) - l_0$
5 else
6 $l_{t,i} = l_{t-1,i}$
7 return $l_{t,i}$

OGM - Algorithm

```
1   Algorithm occupancy_grid_mapping({l_{t-1,i}},x_t,z_t):
2   for all cells m_i do
3       if m_i in perceptual field of z_t then
4           l_{t,i}= l_{t-1,i} + inverse_sensor_model(m_i,x_t,z_t)-l_0
5       else
6           l_{t,i}= l_{t-1,i}
7   return l_{t,i}
```

$$l_{t,i} = \log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}$$

$$l_0 = \log \frac{p(m_i)}{1 - p(m_i)}$$

OGM - Algorithm

```
1   Algorithm occupancy_grid_mapping({l_{t-1,i}},x_t,z_t):
2   for all cells m_i do
3       if m_i in perceptual field of z_t then
4           l_{t,i}= l_{t-1,i} + inverse_sensor_model(m_i,x_t,z_t)-l_0
5       else
6           l_{t,i}= l_{t-1,i}
7   return l_{t,i}
```

$$\log \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)}$$

$$l_{t,i} = \log \frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})}$$

$$l_0 = \log \frac{p(m_i)}{1 - p(m_i)}$$

OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

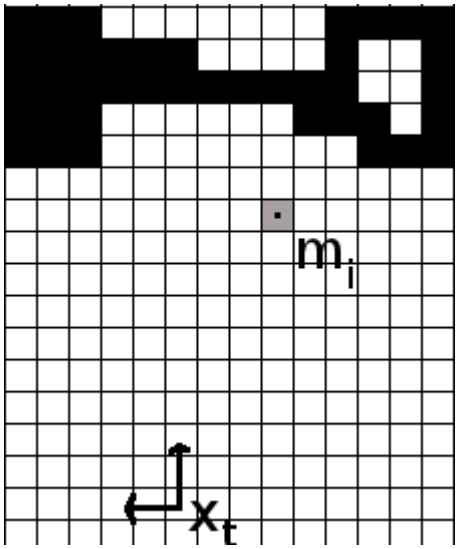
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

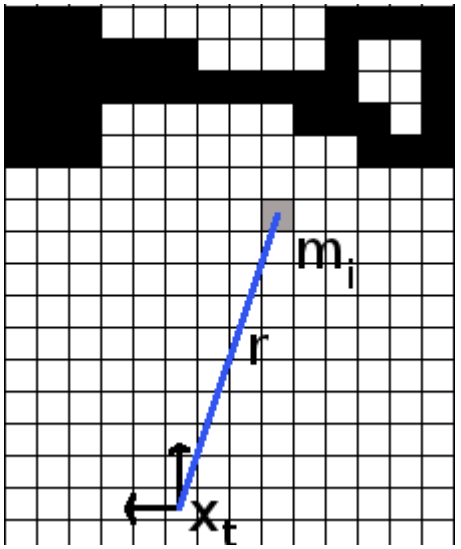
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

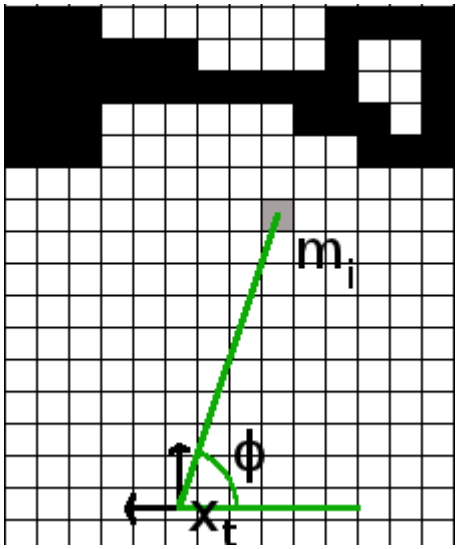
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

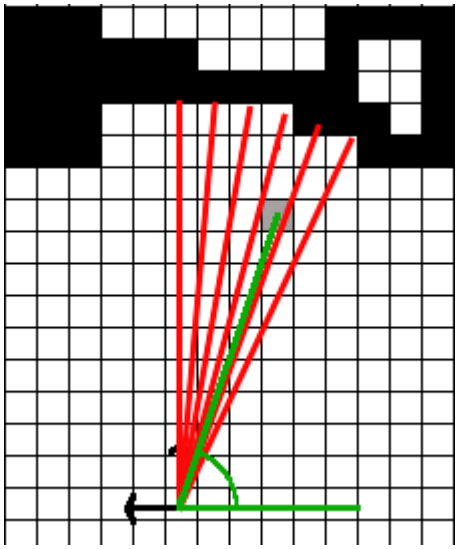
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

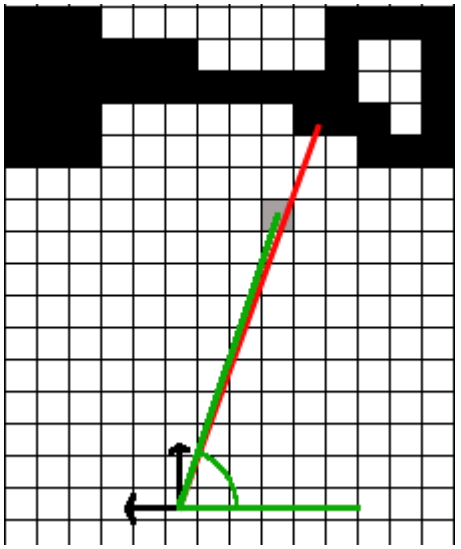
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



OGM - Algorithm

1 Algorithm inverse_range_sensor_model(m_i, x_t, z_t):

  //here m_i is the center-of-mass of the cell m_i

2 $r = d(m_i, x_t)$

3 $\phi = \text{atan2}(\Delta y, \Delta x) - \theta$ // between m_i and x_t

4 $k = \text{argmin}_j |\phi - \theta_{j,sens}|$

5 if $r > \min(z_{max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{j,sens}| > \beta/2$ then

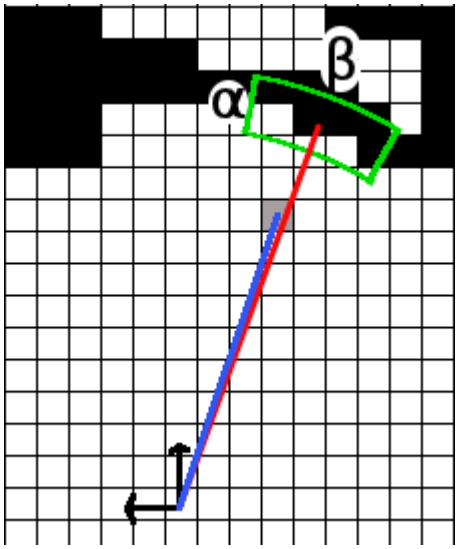
6 return l_0

7 if $z_t^k < z_{max}$ and $|r - z_t^k| < \alpha/2$

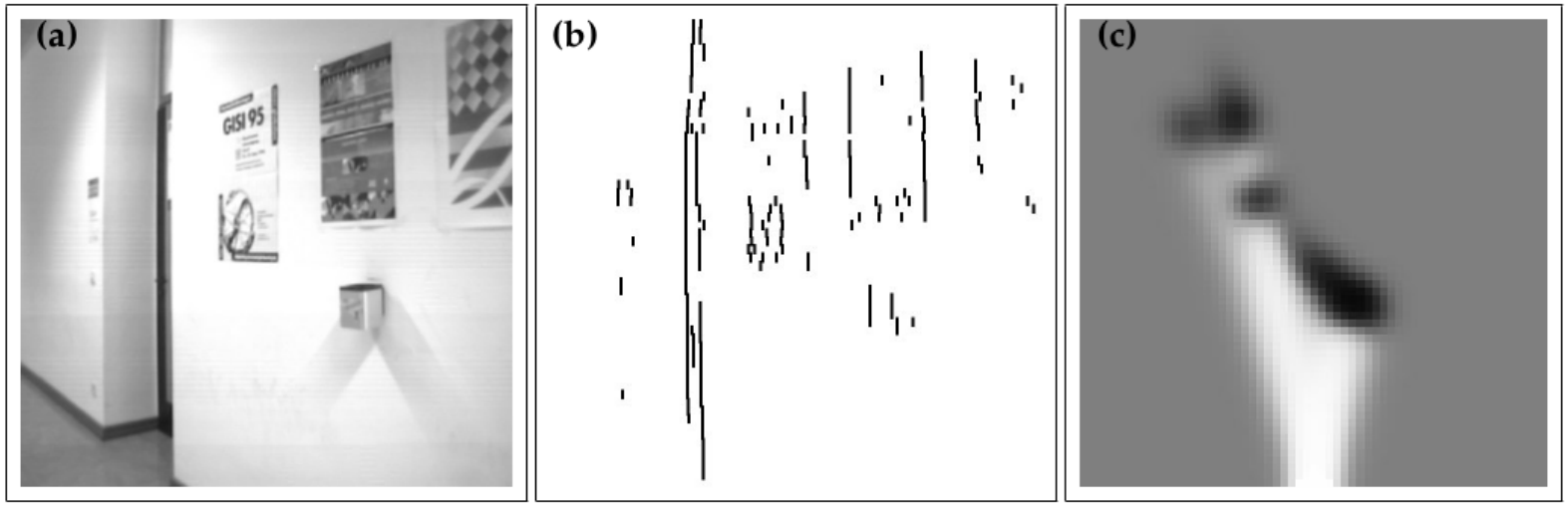
8 return l_{occ}

9 if $r <= z_t^k$

10 return r_{free}



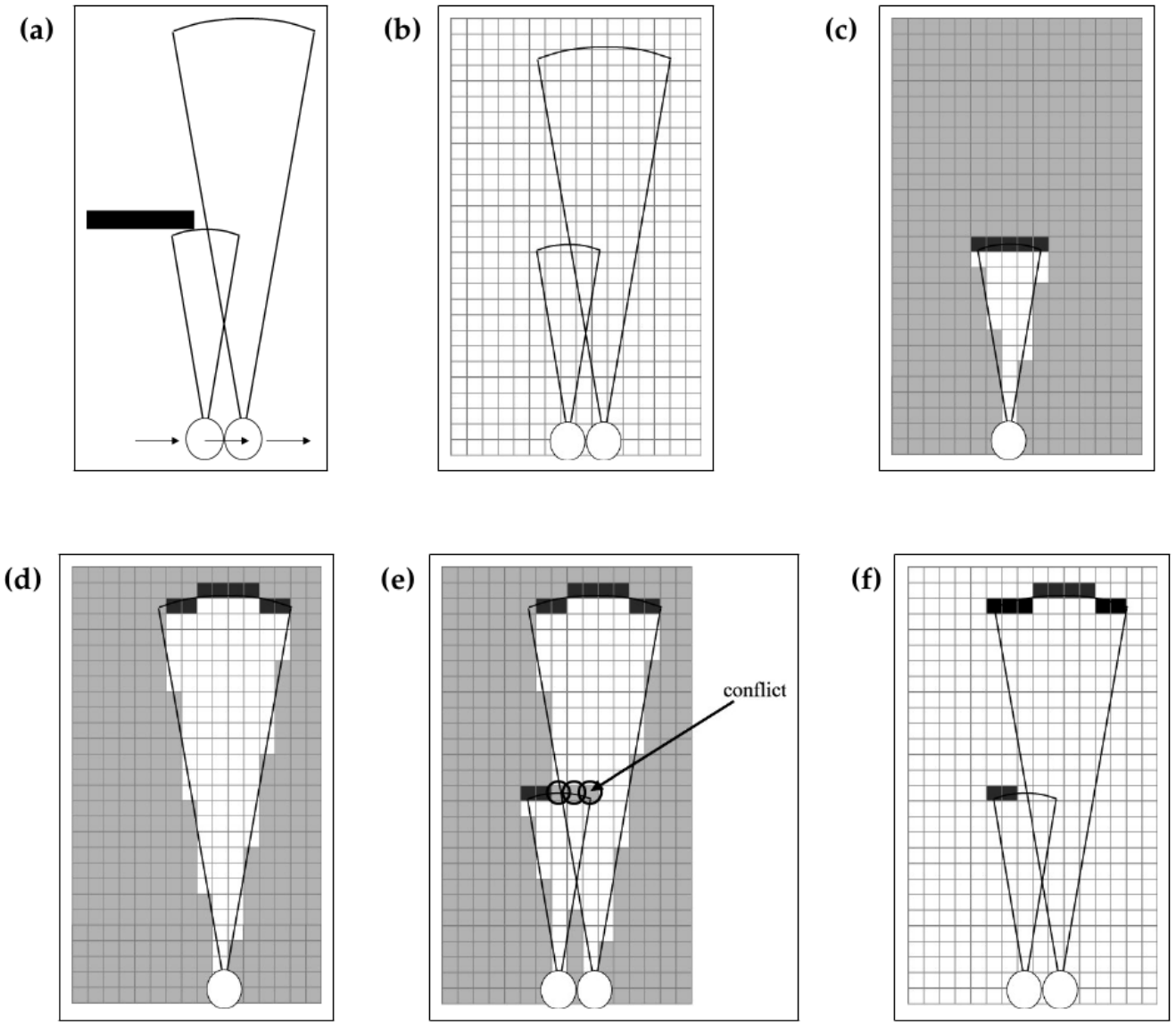
OGM - Considerations



- Multi-Sensor Fusion
  - sensor modalities?
  - multiple maps

- Why inverse sensor model?

OGM - MAP



OGM - MAP

$$m^* = \underset{m}{argmax}(\log p(m|z_{1:t}, x_{1:t}))$$

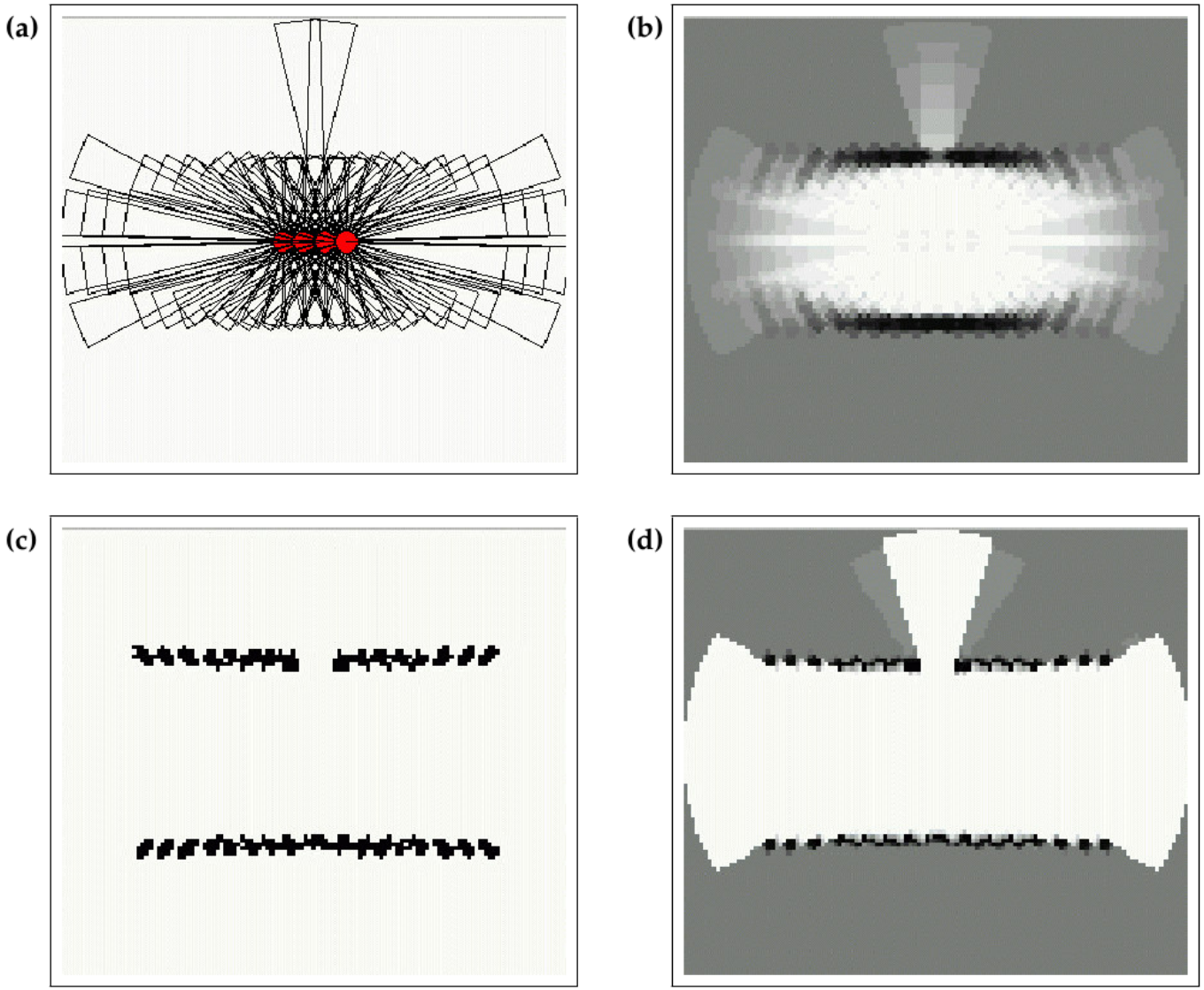
$$m^* = \underset{m}{argmax}(\sum_t \log p(z_t|x_t, m) + l_0 \sum_i m_i)$$

OGM - MAP

```
1  Algorithm MAP_occupancy_grid_mapping(x_1:t, z_1:t)
2      set m = {0}
3      repeat until convergence
4          for all cells m_i do
5              m_i = argmax_k=0,1 (kl_0 + Σ_t log
                       measurement_model(z_t, x_t, m with
                   m_i=k))
6      return m
```

$$m^* = \underset{m}{argmax}\left(\sum_t \log p(z_t|x_t, m) + l_0 \sum_i m_i\right)$$

OGM - MAP



ROS GMapping

- Familiar with the code structure
  - Entry point
- Data format
- Same use
- r39 from GMapping SVN repository at openslam.org, with minor patches applied to support newer versions of GCC and OSX

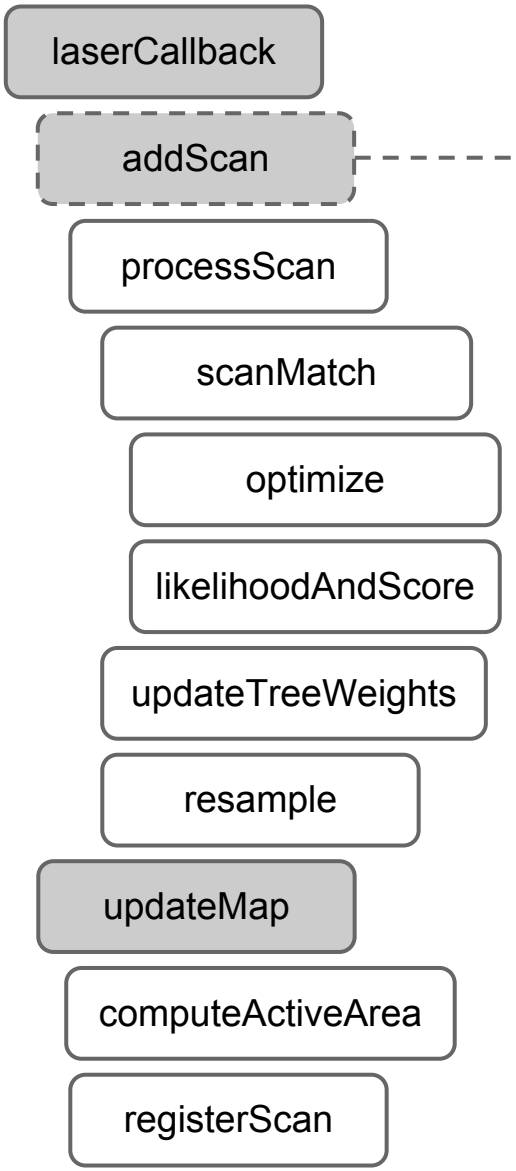
GMapping - Algorithm

- laserCallback
  - addScan
    - processScan
      - scanMatch
        - optimize
        - likelihoodAndScore
        - updateTreeWeights
        - resample
  - updateMap
    - computeActiveArea
    - registerScan

```
void laserCallback(const LaserScan::ConstPtr& scan)
```

```
laser_count_++;
if ((laser_count_ % throttle_scans_) != 0)
  return;

static ros::Time last_map_update(0,0);

if(!got_first_scan_) {
  if(!initMapper(*scan))
    return;
  got_first_scan_ = true;
}

GMapping::OrientedPoint odom_pose;
if(addScan(*scan, odom_pose))
  [...]
  if(!got_map_ || (scan->header.stamp - last_map_update) >
map_update_interval_)
  {
    updateMap(*scan);
    last_map_update = scan->header.stamp;
    ROS_DEBUG("Updated the map");
  }
}
```

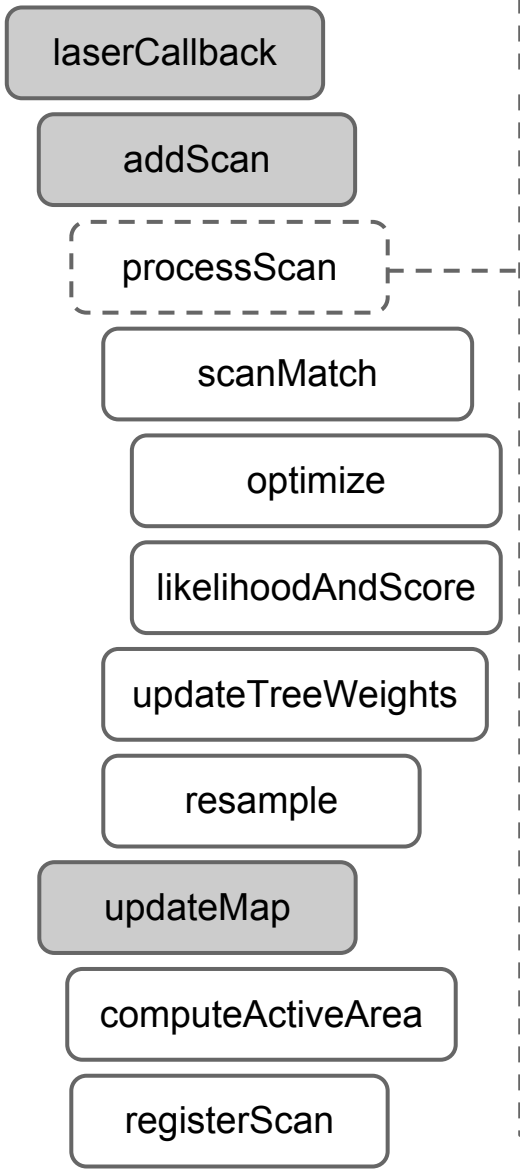
GMapping - Algorithm




```
bool addScan(LaserScan& scan, OrientedPoint& gmap_pose)
```

```
if(!getOdomPose(gmap_pose, scan.header.stamp))
   return false;

if(scan.ranges.size() != gsp_laser_beam_count_)
  return false;

// GMapping wants an array of doubles...
double* ranges_double = new double[scan.ranges.size()];

/* If the angle increment is negative, then we conclude
that the laser is upside down, and invert the order of the
readings. */
[...] // populate ranges_double eventually inverting scans

GMapping::RangeReading reading(scan.ranges.size(),
ranges_double,gsp_laser_, scan.header.stamp.toSec());

delete[] ranges_double;

reading.setPose(gmap_pose);

return gsp_->processScan(reading);
```

GMapping - Algorithm




```
bool processScan(RangeReading& reading, int
adaptParticles)
```

```
for (iterator it=m_particles.begin(); it!=m_particles.end(); it++){
  pose=m_motionModel.drawFromMotion(it->pose, relPose, m_odoPose);
}
// process a scan only if the robot has traveled a given distance
if (! m_count
    || m_linearDistance>m_linearThresholdDistance
    || m_angularDistance>m_angularThresholdDistance){

  // convert the reading in a scan-matcher feedable form
  if (m_count>0){
   scanMatch(plainReading);
   updateTreeWeights(false);
   resample(plainReading, adaptParticles,reading_copy);
  } else { //"Registering First Scan"
    for (it=m_particles.begin(); it!=m_particles.end(); it++){
       m_matcher.invalidateActiveArea();
       m_matcher.computeActiveArea(it->map, it->pose, plainReading);
       m_matcher.registerScan(it->map, it->pose, plainReading);
       [...] //init TNode
     }
   }
  updateTreeWeights(false);
  //update the past pose for the next iteration
 } //end if
m_readingCount++;
return processed;
```

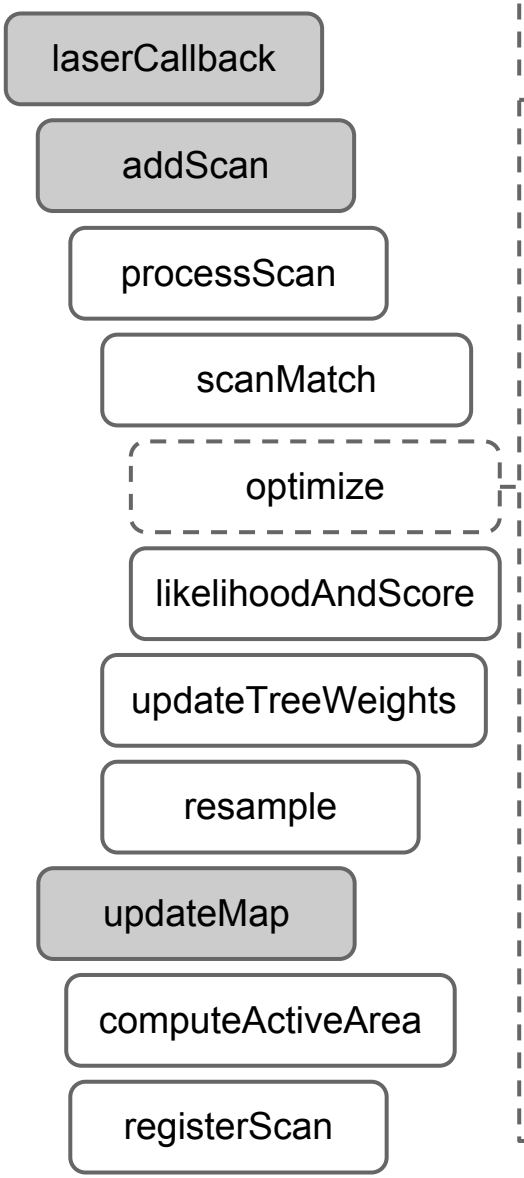
GMapping - Algorithm

- laserCallback
- addScan
- processScan
- scanMatch
- optimize
- likelihoodAndScore
- updateTreeWeights
- resample
- updateMap
- computeActiveArea
- registerScan

```
inline void GridSlamProcessor::scanMatch(const double* plainReading)
```

```
// sample a new pose from each scan in the reference
for (iterator it=m_particles.begin(); it!=m_particles.end(); it++){
  OrientedPoint corrected;
  double score, l, s;
  score=m_matcher.optimize(corrected, it->map, it->pose,
plainReading);
  if (score>m_minimumScore){
    it->pose=corrected;
  } else {
    m_infoStream << "Scan Matching Failed" <<std::endl;
  }

  m_matcher.likelihoodAndScore(s, l, it->map, it->pose,
plainReading);
  it->weight+=l;
  it->weightSum+=l;

  //set up the selective copy of the active area
  //by detaching the areas that will be updated
  m_matcher.invalidateActiveArea();
  m_matcher.computeActiveArea(it->map, it->pose, plainReading);
}
```

GMapping - Algorithm




```
double optimize(OrientedPoint& pnew, const ScanMatcherMap& map, const
OrientedPoint& init, const double* readings) const{
do{
  if (bestScore>=currentScore){
    refinement++; adelta*=.5; ldelta*=.5;
  }
  bestScore=currentScore;
  OrientedPoint bestLocalPose=currentPose;
  OrientedPoint localPose=currentPose;
  do { //for each Move (Front, Back, Left, TurnLeft, ... Done)
    localPose=currentPose;
    // update localPose x y th accordingly to Move, adelta, ldelta
    if (m_angularOdometryReliability>0.)
      odo_gain*=exp(-m_angularOdometryReliability*dth);
    if (m_linearOdometryReliability>0.)
      odo_gain*=exp(-m_linearOdometryReliability*drho);
    double localScore=odo_gain*score(map, localPose,readings);
    if (localScore>currentScore){
      currentScore=localScore;
      bestLocalPose=localPose;
    }
  } while(move!=Done);
  currentPose=bestLocalPose;
}while (currentScore>bestScore ||
refinement<m_optRecursiveIterations);
pnew=currentPose; return bestScore;
```

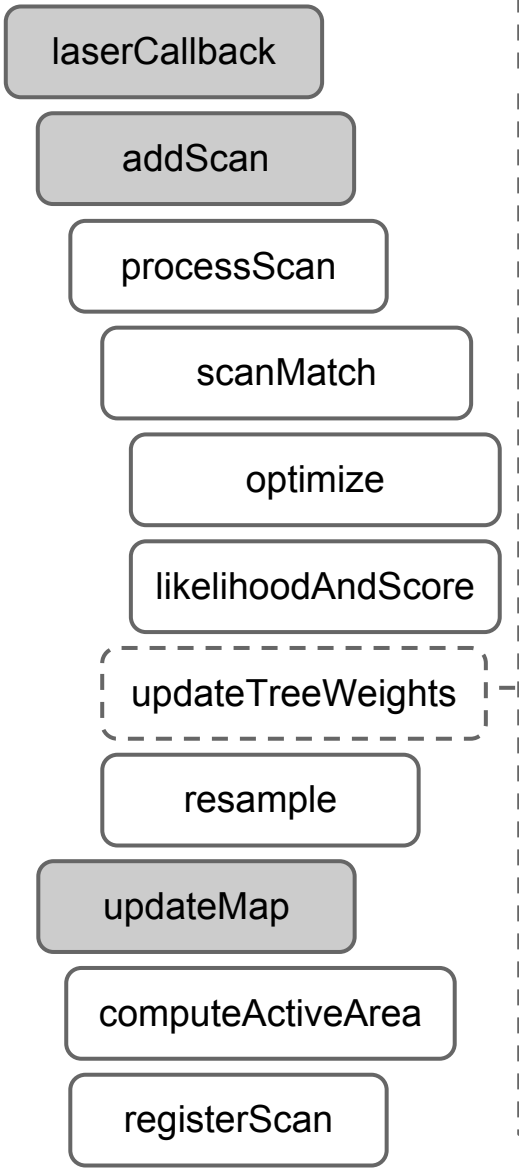
GMapping - Algorithm

- laserCallback
- addScan
- processScan
- scanMatch
- optimize
- likelihoodAndScore
- updateTreeWeights
- resample
- updateMap
- computeActiveArea
- registerScan

```
likelihoodAndScore(double& s, double& l, const ScanMatcherMap& map,
const OrientedPoint& p, const double* readings) const{
```

```
for (const double* r=readings; r<readings+m_laserBeams; r++, angle++)
{
  for (int xx=-m_kernelSize; xx<=m_kernelSize; xx++)
  for (int yy=-m_kernelSize; yy<=m_kernelSize; yy++){
    IntPoint pr=iphit+IntPoint(xx,yy);
    IntPoint pf=pr+ipfree;
    const PointAccumulator& cell=map.cell(pr);
    const PointAccumulator& fcell=map.cell(pf);
    if (cell>m_fullnessThreshold && fcell<m_fullnessThreshold){
      Point mu=phit-cell.mean();
         if (!found){
         bestMu=mu;
        found=true;
    }else
      bestMu=(mu*mu)<(bestMu*bestMu)?mu:bestMu;
    }
  }
  if (found){
    s+=exp(-1./m_gaussianSigma*bestMu*bestMu);
  }
  if (!skip){
    double f=(-1./m_likelihoodSigma)*(bestMu*bestMu);
    l+=(found)?f:noHit;
  }
}
```

GMapping - Algorithm




```
void updateTreeWeights(bool weightsAlreadyNormalized){

if (!weightsAlreadyNormalized)
  normalize();
resetTree();
propagateWeights();
```

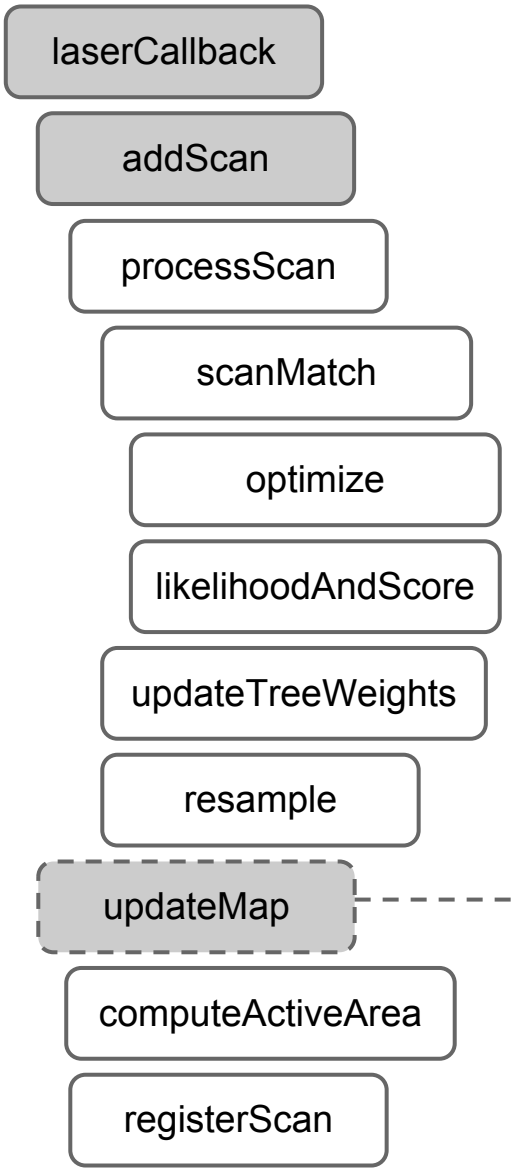
GMapping - Algorithm

- laserCallback
- addScan
- processScan
- scanMatch
- optimize
- likelihoodAndScore
- updateTreeWeights
- resample
- updateMap
- computeActiveArea
- registerScan

```
resample(double* plainReading, int adaptSize, RangeReading*
reading){
```

```
if (m_neff<m_resampleThreshold*m_particles.size()){
  m_indexes=resampler.resampleIndexes(m_weights, adaptSize);
  for (unsigned int i=0; i<m_indexes.size(); i++){
    // if the particle have to be deleted delete it
    // else add to temp and create node
  }
 for (ParticleVector::iterator it=temp.begin(); it!=temp.end(); it++)
{
   it->setWeight(0);
   m_matcher.invalidateActiveArea();
   m_matcher.registerScan(it->map, it->pose, plainReading);
   m_particles.push_back(*it);
 }
} else { //neff too high
  TNodeVector::iterator node_it=oldGeneration.begin();
  for (iterator it=m_particles.begin(); it!=m_particles.end(); it++){
    //create a new node and add it to the old tree

    m_matcher.invalidateActiveArea();
    m_matcher.registerScan(it->map, it->pose, plainReading);
    it->previousIndex=index;
    index++;
    node_it++;
   }
}
```

GMapping - Algorithm




```
void updateMap(const sensor_msgs::LaserScan& scan){
```

```
Particle best =
gsp_->getParticles()[gsp_->getBestParticleIndex()];

ScanMatcherMap smap(center, xmin, ymin, xmax, ymax, Δ);

for(TNode* n = best.node; n; n = n->parent) {
 matcher.invalidateActiveArea();
 matcher.computeActiveArea(smap, n->pose,n->reading)[0]);
 matcher.registerScan(smap, n->pose,n->reading)[0]);
}

[...] //resize the map if needed
[...] //map_ is smap with threshold on every cell

sst_.publish(map_.map);
```

GMapping - Algorithm

- laserCallback
- addScan
- processScan
- scanMatch
- optimize
- likelihoodAndScore
- updateTreeWeights
- resample
- updateMap
- computeActiveArea
- registerScan

```
void ScanMatcher::computeActiveArea(ScanMatcherMap& map, const
OrientedPoint& p, const double* readings){
```

```
lp.x += cos(p.theta)*m_laserPose.x-sin(p.theta)*m_laserPose.y;
lp.y += sin(p.theta)*m_laserPose.x+cos(p.theta)*m_laserPose.y;
lp.theta+=m_laserPose.theta;

Point min, max;
//set min and max so that every scan and lp are contained
//eventually enlarge the map

/*allocate the active area*/
for (const double* r=readings;r<readings+m_laserBeams; r++, angle++)
{
  if (m_generateMap){
    double d=*r;
    phit=lp+Point(d*cos(lp.theta+*angle),d*sin(lp.theta+*angle));
    line.points=linePoints;
    for (int i=0; i<line.num_points-1; i++){
      activeArea.insert(map.storage().patchIndexes(linePoints[i]));
    }
    if (d<m_usableRange){
      IntPoint cp=map.storage().patchIndexes(p1);
      assert(cp.x>=0 && cp.y>=0);
      activeArea.insert(cp);
    }
 } else { //not generate map: consider only phit
 }
}
map.storage().setActiveArea(activeArea, true);
```

GMapping - Algorithm

- laserCallback
- addScan
- processScan
- scanMatch
- optimize
- likelihoodAndScore
- updateTreeWeights
- resample
- updateMap
- computeActiveArea
- registerScan

```
double ScanMatcher::registerScan(ScanMatcherMap& map, const
OrientedPoint& p, const double* readings){
```

```
if (!m_activeAreaComputed) computeActiveArea(map, p, readings);
map.storage().allocActiveArea();
OrientedPoint lp = p;
lp.x += cos(p.theta)*m_laserPose.x-sin(p.theta)*m_laserPose.y;
lp.y += sin(p.theta)*m_laserPose.x+cos(p.theta)*m_laserPose.y;
lp.theta+=m_laserPose.theta;

const double * angle=m_laserAngles+m_initialBeamsSkip;
double esum=0;
 for (const double* r=readings r<readings+m_laserBeams; r++, angle++)
  if (m_generateMap){
   double d=*r;
   phit = lp+Point(d*cos(lp.theta+*angle),d*sin(lp.theta+*angle));
   IntPoint p1=map.world2map(phit);
   IntPoint linePoints[20000] ;
   GridLineTraversalLine line;
   line.points=linePoints;
   GridLineTraversal::gridLine(p0, p1, &line);
   for (int i=0; i<line.num_points-1; i++){
     PointAccumulator& cell=map.cell(line.points[i]);
     cell.update(false, Point(0,0));
   }
   if (d<m_usableRange){
     map.cell(p1).update(true, phit);
   }
 } else { //only consider phit, not line points
}
```

Conclusion

- Grid based Improved FastSLAM 2.0
- Test it, with improved proposal

- 3D
- Intensity

Conclusion

- Grid based Improved FastSLAM 2.0
- Test it, with improved proposal

- 3D
- Intensity

Conclusion

- Grid based Improved FastSLAM 2.0
- Test it, with improved proposal

- 3D
- Intensity

What's next?

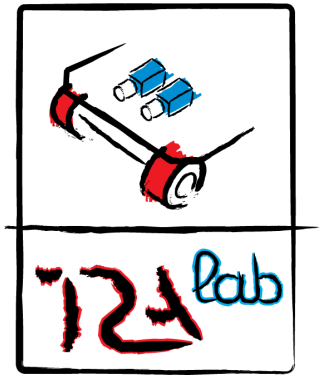
Conclusion

- Grid based Improved FastSLAM 2.0
- Test it, with improved proposal

- 3D
- Intensity

What's next?

[3D] SLAM software extensive review



irawiki.disco.unimib.it/irawiki/index.php/Reading_group

A recording is also available

ask to Francesco Sacchi