Fast SLAM using a geometric approach and some unconventional methods

MEAM 620 final project

Subhrajit Bhattacharya

1. Introduction

1.1. SLAM definition

Simultaneous Localization And Mapping had be one of the most challenging problems in robotics. The problem is significantly more difficult than individual problems concerning localization in a known environment or mapping where the global position of the robot is known all the time. Below are listed some of the most evident challenges in this problem:

1. The problem of error accumulation in mapping and localization is pretty much intertwined. Errors in mapping result in wrong localization, and error in localization results in wrong mapping. Thus once the error is initiated it will start growing uncontrollably.
2. The above mentioned error makes it difficult to identify new obstacles from the ones that has already been visited. This is typically called the “loop closure” problem.

Typically in a SLAM problem the only sensor available to a robot is an on-board Laser range sensor. There is no GPS or Truth information available.

1.2. SLAM tasks

The tasks of a robot performing SLAM typically involve:

i. Generation of a map of the environment in the local initial coordinate system of the robot.
ii. Localization of itself in the map (i.e. to determine it's own position in its local, initial coordinate frame.)
iii. To ensure good coverage of the environment.

2. Configuration space and Feature space

For a mobile robot the configuration space is typically the SO(2) space. Any point in that space is represented by the tuple (x,y,θ).
The configuration space on the other hand is the space of features observed by the robot. Hence if the on-board laser range sensor of the robot has n scan rays, a point in the feature space may be represented by an n-tuple, each value of which represent a range value returned by the laser range sensor.

It is quite evident that the mapping from the Configuration space to the Feature space for a mobile robot is a surjection. However in presence of errors this mapping become quite complex. However there appears a rather interesting pattern in it which is worth investigation. Below are some discussions on it just to create some insight into the problem and understand what we can solve and what we cannot.

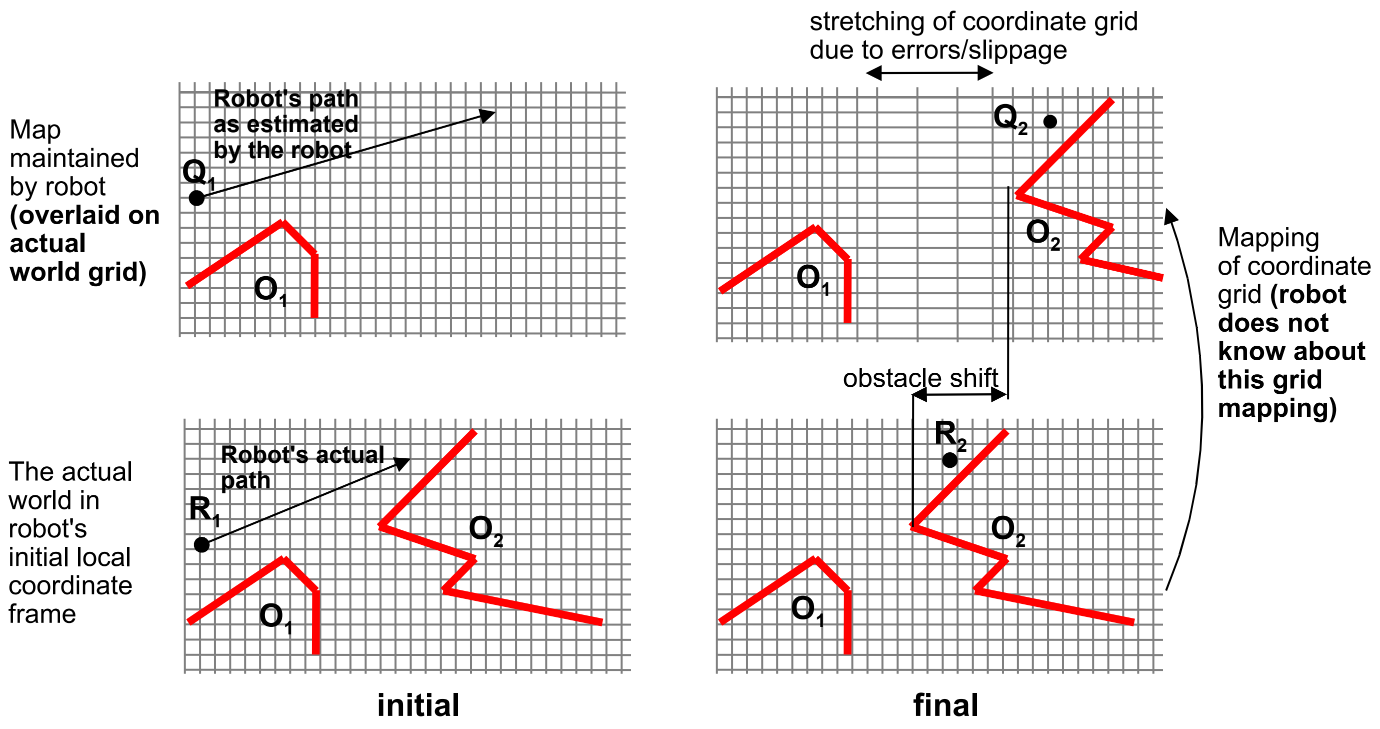

stretching of coordinate grid due to errors/slippage
Map maintained by robot **(overlaid on actual world grid)**
Robot's path as estimated by the robot
Q_1
O_1
Q_2
O_2
O_1
Mapping of coordinate grid **(robot does not know about this grid mapping)**
obstacle shift
The actual world in robot's initial local coordinate frame
Robot's actual path
R_1
O_1
O_2
R_2
O_2
O_1
initial
final

Figure 1

In *Figure* 1 the first row shows the map as generated by the robot for an environment described by the obstacles $\mathbf{O_1}$ and $\mathbf{O_2}$ as shown in the bottom row. The assumption here is that $\mathbf{O_1}$ and $\mathbf{O_2}$ are quite far from each other and hence one is not visible when the robot is near the other. The grids shown in the map generated by the robot is the mapping of an uniform grid in the real world. The stretching had taken place because of errors like slippage. However there is actually no way for the robot to identify the stretching and hence correcting its map. At the beginning the robot could localized itself near $\mathbf{O_1}$, and it could also localize itself finally near and with respect to $\mathbf{O_2}$. However the errors accumulated in between the two obstacles are not quite easy to correct unless both the obstacles are visible to the robot at the same time.

It may be interesting to note that $\mathbf{Q_1}$ and $\mathbf{R_1}$ are close to each other both in the Configuration space as well as the Feature space. However $\mathbf{Q_2}$ and $\mathbf{R_2}$ are actually close to each other in the Feature space, although they are far from each other in the Configuration space. This insight will help us in understanding what we can solve and what is not solvable in SLAM.

The error resulting in difference between the estimated position of the robot in the Configuration space and it's actual position in the Configuration space causes a significant problem in “loop closure”, where we fail to identify new obstacles from the previously seen ones because of the error accumulation on the way. One possible solution is to move to and fro between obstacles to filter out the error in the separation between them. However that may not be a feasible solution since this error (caused by slippage, etc.) are generally biased. Moreover as discussed before, it may not be even possible to estimate the unbiased relative positioning between the obstacles. Probably the best solution to this problem is just not to worry too much about the error. Instead we can just try to be consistent with the errors. That means for our estimate we just try to have a consistent separation between the obstacles, rather than trying to estimate the actual separations. The consistency may actually be checked and performed upon loop closure.

3. Standard methods of performing SLAM

The different ways of performing SLAM include

i. EKF SLAM
ii. Graph SLAM
iii. Fast SLAM

In this project we concentrated on Fast SLAM. Fast SLAM is the particle filter approach to SLAM. Even among Fast SLAM there are several variations. Although we followed the very basic outline of Fast SLAM in an environment with unknown correspondences using an occupancy grid, we have made several changes to the original algorithm and approach, the most important one being the use of geometric primitives or 'elements' for representing the environment rather than using an occupancy grid.

The standard structure of a Fast SLAM algorithm is quite similar to a that of a simple particle filter, except for the fact that now the individual particles maintain their individual maps and they keep updating their own maps based on the sensor data.

4. Use of geometric elements instead of occupancy grid

4.1 Advantages and disadvantages of a geometric representation over occupancy grid based representation

Advantages of occupancy grid:

- Easy to add data points
- Easy to detect inconsistencies using visibility, occlusion, etc
- Quick to add new scan data

Disadvantages of occupancy grid:

- Consumes huge memory
- Depends on resolution of discretization
- Probably difficult to combine/communicate with different occupancy grids (from different particles)

Why geometric elements?

- Easy and efficient to store
- Easy path planning using visibility graph, voronai diagrams, etc which are less computationally expensive
- Elements are more flexible to changes/updates in the map
- I liked this approach more and wanted to do it for the project

Disadvantages of geometric elements:

- Lots of geometry involved
- Updating of the elements need to be done carefully
- Checks for inconsistency needs to be done in geometric way

4.2. The basic algorithm:

Having said all the above, we present our basic algorithm structure for performing Fast SLAM using geometric elements rather than occupancy grids.

Below is a highly simplified pseudo-algorithm representation of the algorithm:

```
Initiate Particles (N particles) at coordinates (0,0,0) and an empty map for each

while (TRUE)

    Compute a path and determine velocities to set based on the pose and map of the particle
        with the highest weight.

    Propagate the particles with the last velocity commands since the last time they were set. The
        propagation is done with some added noise.

    Set the new velocity commands

    Slow down before obtaining laser scan
        – this requires particle propagation step to be called once again
    Obtain laser scan data
        Segment the laser scan data into clusters (from the different obstacles)
        $E_{new}$ = Fit 'elements' (line segments) to the clusters from laser scan

    For p = each of the N particle
        Transform $E_{new}$ to particles coordinate frame to obtain $E^p_{new}$

        For e = each element in $E^p_{new}$
            For f = each element already present in the particle p's map, $M_p$
                d(e,f) = Compute distance (Euclidean distance + orientation difference)
                    between elements e and f
            end

            Perform a maximum likelihood estimate for the element e so as to determine
                a $\underline{f}$ such that d(e,$\underline{f}$) is minimum among all the computed d(e,f).

            If this d(e,$\underline{f}$) is less than some threshold,
                Update element $\underline{f}$ in $M_p$ with the information of e
            else
                Add new element e to $M_p$
            end
        end

        For particle p determine a weight w for this particular session of adding/updating
        elements depending on the values of d(e,$\underline{f}$).
    end

    Clean the maps of the particles by removing redundant or bad elements
```

Update particle weights: $w^P_i = w^P_i * w_i$

Perform an **importance re-sampling** based on the updates particle weights

end

4.3. Results and drawbacks

The above algorithm works well as long as there is no need to look back at the obstacles previously seen by the robot. However as soon as the robot encounters an obstacle that it has previously seen, but now it thinks it to be at a different location (because of the errors accumulated in between), two devastating consequences take place:

i. Either the previously mapped obstacle elements start deforming really bad because the new obstacle element tries to merge with it, or
ii. The previous obstacle element is recreated near its original position.

The following pictures illustrate the problem:

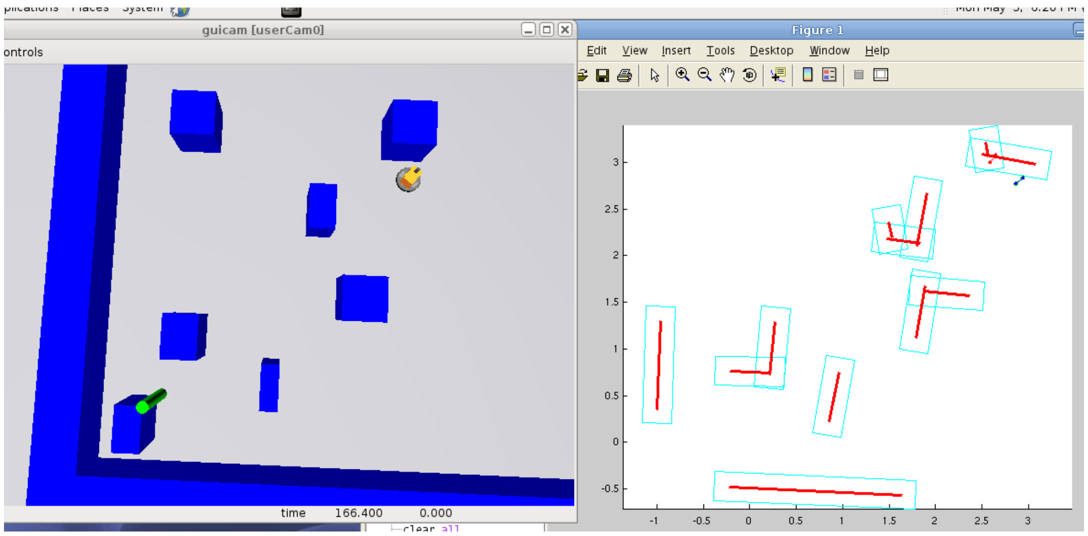
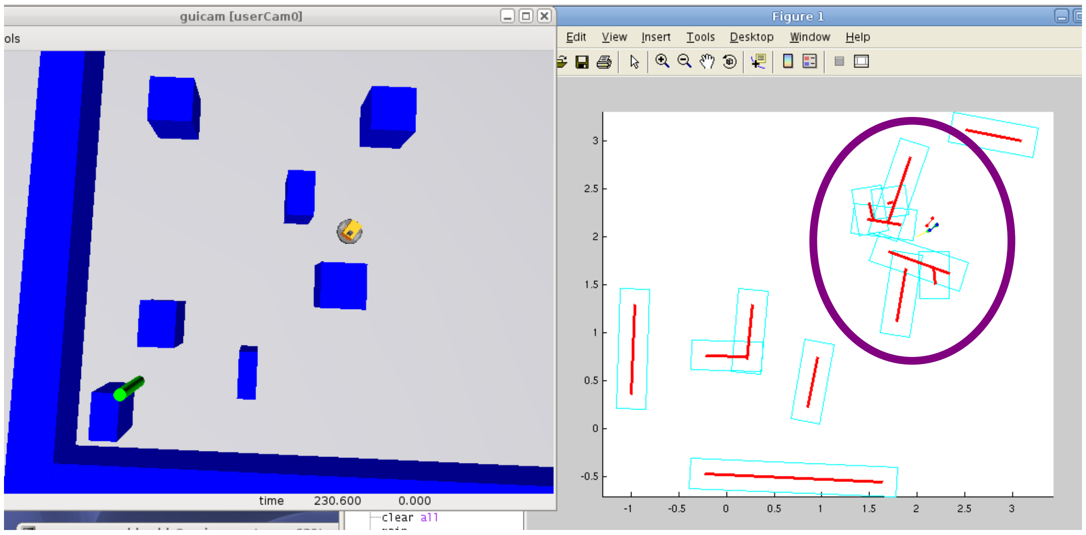


Figure 2a

Figure 2b



In *figure 2a* we observe that although there are some global deformations the obstacles are pretty much in their place as the robot is moving ahead without looking back.
However as soon as the robot starts heading back in *figure 2b* and encounters a previously seen obstacle, the map pf the winning particle starts deforming.

5. Maintaining a Global Map/World

5.1. The need for a global map – particles communicating with each other

The idea for maintaining a global map by communicating between the particles was inspired by the fact that at some instants a particle (or its descendants) may be at a good position/having good weight, and at other instant some other particle (or its descendants) may be at a good position/having good weight.

5.2. Changes in the algorithm

Thus the need to take advantage of best information contained by any particle at any instant seems to be quite indispensable.
Hence we made some small changes in the algorithm described previously, where we do the following additional things:

i. Assign weights to each element of each of the maps maintained by the particles. Where weights are computed based on how well new laser scans have matched with the element. Moreover there are some preferences being given to older and 'trusted' elements.
ii. Compute a global world map out of the maps of the individual particles. The choice of which element to be selected from which particle's world map is made based on the weights computed in the above step.
iii. Update the maps of the individual particles using the global world map just created.
iv. Compute velocities based on this global map instead of the map of the winning particle

5.2. Results and drawbacks

This change evidently had a positive effect on the maintenance of the map. The map looked more stable and consistently stayed quite accurate for a longer period of time. However even with this method implemented, because of the error accumulated over time the particle filter performed a wrong localization when it saw an obstacle it had seen before:

The figures in next page illustrate the problems:

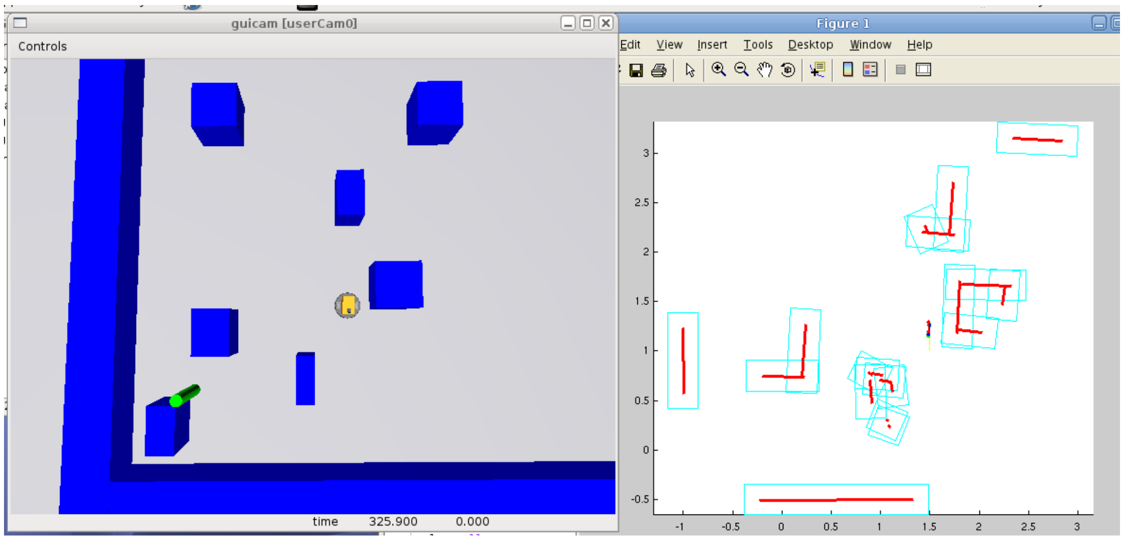


Figure 3a: Better performance while returning back

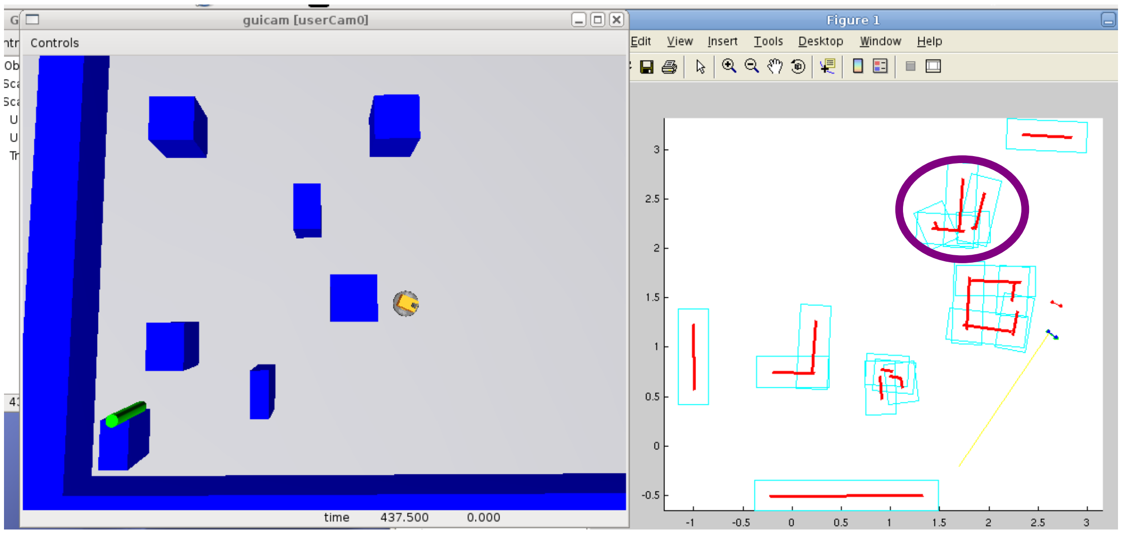


Figure 3b: Still a obstacle element is re-located wrongly

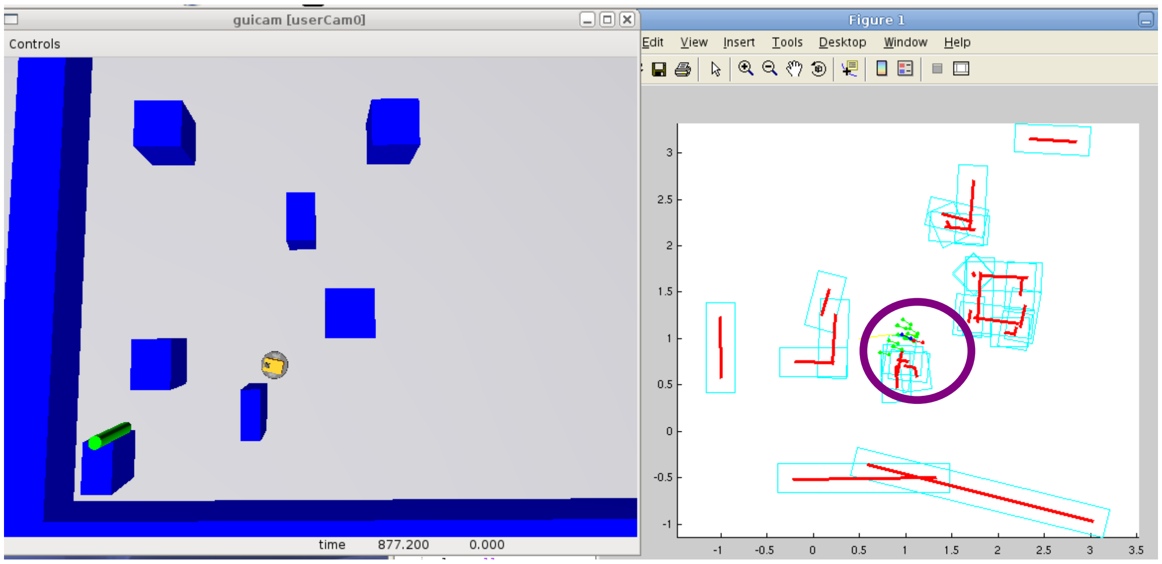
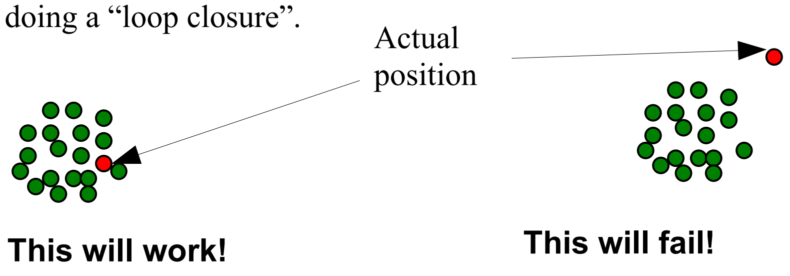


Figure 3c: Particle filter goes crazy because of inconsistencies

6. Particle generation based on the geometric features of the Global map

6.1. Why simple particle re-sampling doesn't work well in SLAM

Particle filter relies on the fact that the actual position of the robot will be somewhat close to the present estimate (in Feature space). Thus on scattering enough particles around the estimated position we expect to get a particle which represents the position of the particle in the Feature space quite accurately.
However the problem will arise if somehow the actual position of the robot is somewhere far from the estimate even in the feature space. Then the particles scattered around the estimated position are all far away from the actual position, and hence won't be able to localize the robot. This situation arises quite easily in SLAM when doing a "loop closure".



When the robot suddenly re-encounters an obstacle it has seen much earlier, it needs to localize itself with respect to that obstacle. However if the particles scattered by the particle filter does not generate any particle at the position where the robot is actually in the feature space, then we won't be able to capture the actual position of the robot.

6.2. The solution

The very evident and easy solution to this problem is to manually add a few particles by looking at the geometric features of the latest scan and comparing them with those of the global map. The following figure explains this concept with a simple example:

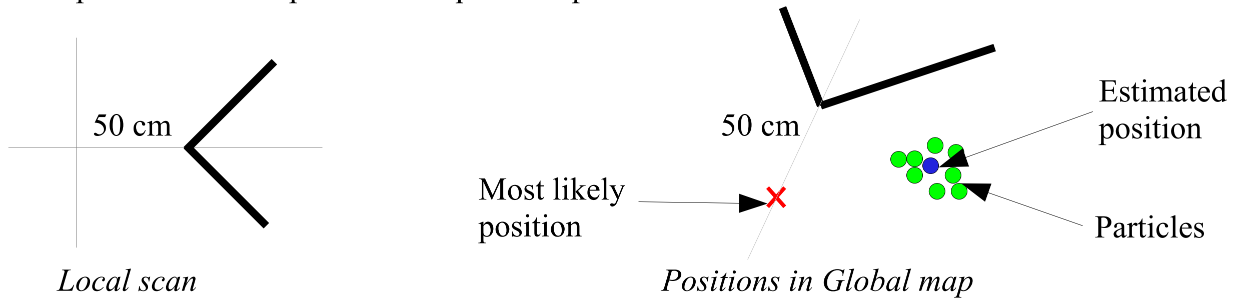


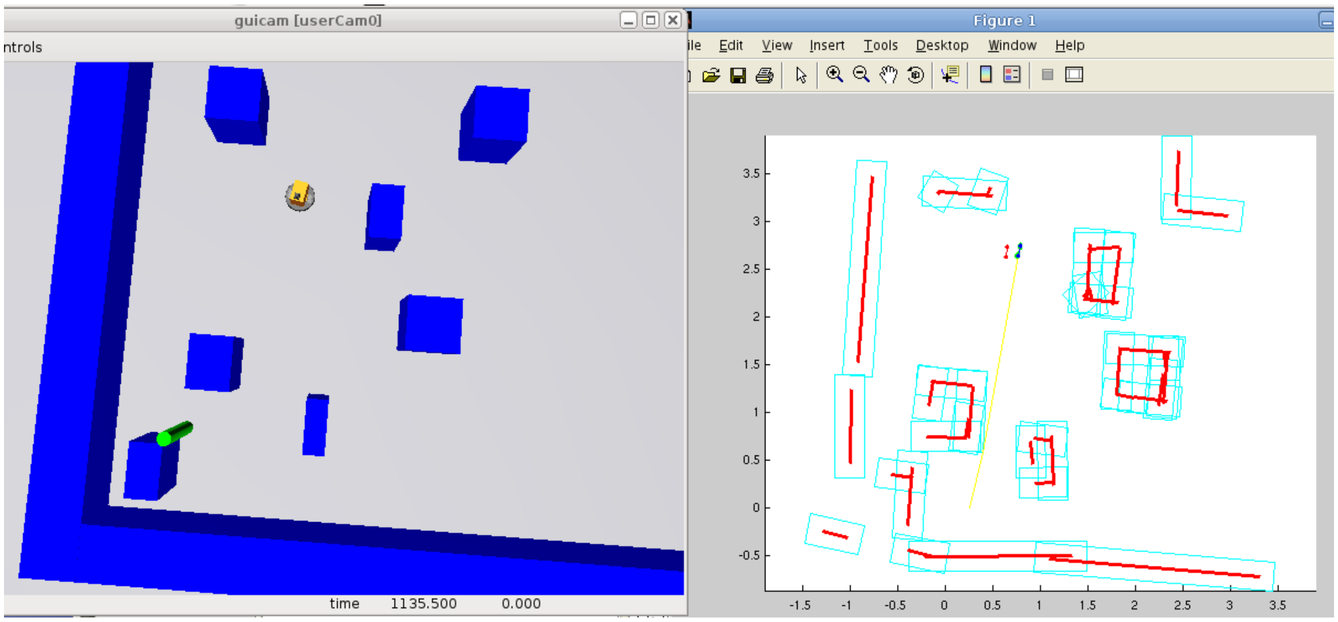
Figure 4

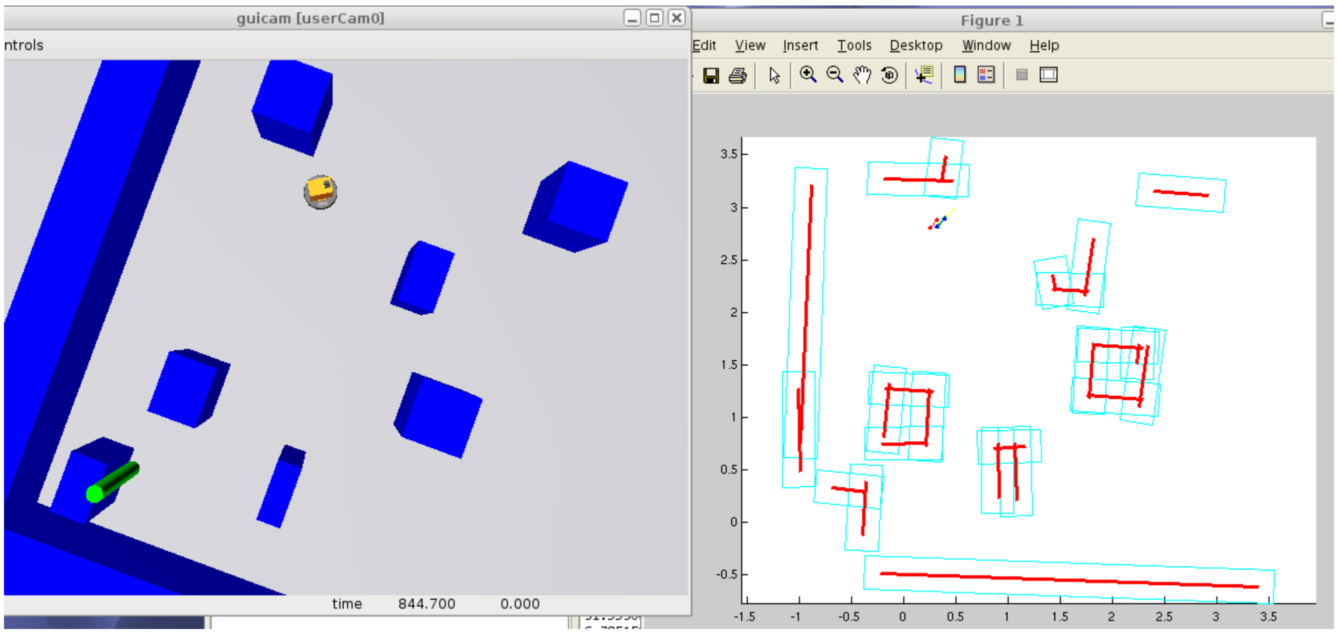
Thus the simple solution is to geometrically identify the likely positions by using the geometric properties of the global map, and scatter a few particles (in practice, one each) at those locations.
However in practice we did this using only those elements of the global world which are sufficiently reliable. Reliability was measured by age of the element as well as how many times it has been updated.

The details of the geometric procedures for identifying the most likely positions are not discussed here, although they are quite straight forward.

6.3. Final Results

The following figures shows the final map created by the robot in 2 different runs after wandering around for significantly longer period of time than the previous examples:





7. Conclusions

7.1. Observations

We do not observe major inconsistencies in the map after we implemented the concepts described under sections 5 and 6.

7.2. Plans for future works

- Test the algorithm on more challenging and bigger environments
- Implement better methods for identifying inconsistencies in the map
- Have multiple robots perform SLAM and send their individual maps to a centralized system. The centralized system will then stitch the maps to generate a global map
- Implement the algorithms on actual robots