**Parallel SLAM for Mobile Robots in the Cloud**

Simultaneous localization and mapping (SLAM) is a very important capability for mobile robots and has been studied extensively in the literature. Computing the position of a robot in an unknown environment amidst measurement errors and simultaneously computing a map of the environment can be a computationally challenging task. SLAM algorithms can use various inputs like distance readings to objects coming from a laser scan, images of the environment, etc. to do the mapping. We have chosen an algorithm called Gmapping [[1](#_ENREF_1),[2](#_ENREF_2)], which uses distance measurements, and implemented a parallel version of this popular algorithm. GMapping is a particle filtering-based SLAM algorithm. The algorithm maintains a number of particles each containing a probable map of the environment. The robot uses a laser sensor to find the distances to the objects in its path. As the robot moves through the environment it gets new distance readings as well as new position measurements. According to these new readings, the algorithm calculates a weight for each particle depending on how probable that particle is given the readings. Then the algorithm draws particles with replacement from this set according to their weights, and this step is called resampling. Resampled particles are used with the next reading. At each reading the algorithm takes the map associated with the particle of highest weight as the correct one. There are serial versions of the algorithm implemented in C++, but these implementations are not suitable to run in our cloud-based distributed streaming computing engine and we had to develop a Java version of the algorithm. The computation time of the algorithm depends on the number of particles used, the size of the environment, and the number of points in the distance readings.

**Overall design of the Application:** The SLAM algorithm is not dependent on the choice of robot. As a demonstration, we are using TurtleBot by Willow Garage as our robot which is a differential drive robot equipped with a Microsoft Kinect sensor. TurtleBot has an ROS driver and a supporting software stack which can be used to retrieve information like odometry, laser scans, etc. from the robot, in addition to controlling it. An application connecting to the ROS is deployed on a gateway running on a workstation. This applications converts the ROS messages to a format that suits the cloud application, and sends transformed data to the application running on the FutureGrid openstack-based VMs using the message brokering layer. The application uses the laser scans coming from the IR sensor of the Kinect and odometry readings of the Turtlebot. The application running in the cloud generates a map according to the current information it receives and send this map back to the workstation running the gateway. The gateway can save this map, publish it back to ROS for viewing, etc.

**Main benefit:** The algorithm can run with many particles with less latency, improving the accuracy of the algorithm both because of the greater number of particles and the increased number of readings it can process.

**Things to improve in the future:** At the moment it is still difficult to develop and scale these applications with modern distributed stream processing engines due to the complex programming required. Another important area to focus on the scalability is how to schedule tasks in a dynamic environment of robots. As the robots connect and disconnect from the system, the application resources must be rescheduled to get optimum performance out of the system. Also we observed that there are fluctuations in the latency of the application due to various reasons like network, virtualization etc. It would be nice to address these in the future.

**References**

1. Grisetti, G., C. Stachniss, and W. Burgard. *Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling*. in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005. IEEE.

2. Grisetti, G., C. Stachniss, and W. Burgard, *Improved techniques for grid mapping with rao-blackwellized particle filters.* Robotics, IEEE Transactions on, 2007. **23**(1): p. 34-46.

**Sensor cloud-based multi-robot collision avoidance control**

Local collision avoidance is one of the most important aspects in robot navigation. The task of local collision avoidance is to dynamically compute the optimal collision-free velocity for a robot, which is based on observations of the environment. Unlike motion and path planning that have static knowledge of the global environment and make one-time plans, local collision avoidance needs to respond to the dynamics of the environment such as other active entities and obstacles that are not reflected in the static map.

Current local collision avoidance methods are mainly based on the Velocity Obstacle theory[1]. Of these methods, the ORCA (Optimal Reciprocal Collision Avoidance)[2] is a well developed one, but does not take many practical problems into account. To improve the method in real robotic environments, several other approaches, which consider the kinetic characteristics of robots, localization uncertainty and so on, are brought forward. As the methods become more and more complicated, computation cost increases a lot, especially in a complex environment that contains a large number of dynamic and active entities. Without sufficient computation resources, local collision avoidance can fail due to computational delay, resulting in robot collisions. To solve this issue, we developed an IOT Cloud-based multi-robot collision avoidance method. This method implements the COCALU (Convex Outline Collision Avoidance Under Localization Uncertainty)[3] algorithm. The algorithm runs in the IOT Cloud and uses IOT Sensors to transmit data and commands between the robot and the IOT Cloud.

**Overall design of the Application:** To develop the collision avoidance application that can be deployed in the IOT Cloud, two main components need to be designed. One is the Sensor Module which relays messages between the robot and message broker and the other one is the topology that actually runs the algorithm.

Data from the robot are published through ROS (Robot Operation System) topics. To feed data into the Collision Avoidance Topology, all ROS messages have to be transformed into Java objects that can be processed by the message broker. This transformation is carried out by a ROSJava Node which is a ROS node implemented in Java. This node subscribes to all topics needed by the collision avoidance algorithm and each message listener of these subscribers will call a corresponding message transformation function to transform the ROS messages into Java objects. After that, the message sender defined in Sensor module is called to publish the message to the correct IOT Cloud Channel. These Channels are defined according to the topology and message broker. For Collision Avoidance Topology, three types of information are required from the robot: odometry, pose array, and laser scan. As several robots can use the same topology to run the collision avoidance algorithm, messages from robot are grouped according to their types. The Collision Avoidance topology running in Storm will execute the COCALU algorithm and send back the velocity command through the message broker and ROSJava Node to the robot.

**Main benefit:** Using the IOT cloud as a Collision Avoidance control platform, computation resources can be scaled easily and rapidly to satisfy the dynamic robot controlling requirement, especially in complex environments. Also, for large-scale swarm robotics, topology in the storm can propagate according to the number of the robots.

**Things to improve in the future:** Parallelizing collision avoidance algorithms is hard and can be inefficient, since most part of the algorithm is serial. However parallelism can be implemented at the topology level. With proper design of the controlling workflow, one topology can process data from several robots by increasing the parallel instances of its components. Unfortunately, some of the bolts in the topology need to cache the state of the robot, therefore stream source and destination are bolt instance dependent. This makes the grouping of instances between connected bolts very difficult. To fully utilize the parallelism mechanism, further investigation is required.

**References**

[1] Jur van den Berg, Stephen J. Guy, Ming Lin, etc. Reciprocal n-Body Collision Avoidance. Robotics Research: The 14th International Symposium ISRR, Springer Tracts in Advanced Robotics, vol. 70, Springer-Verlag, May 2011, pp. 3-19.

[2] Fiorini P, Shiller Z. Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research, 1998, 17(7): 760-772.

[3] Hennes D, Claes D, Meeussen W, et al. Multi-robot collision avoidance with localization uncertainty. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, Volume 1, 2012:147-154.