Ruby Classes

Ruby is a modification of PEARL that better suits our problem. During the project, we created various versions of Ruby, each one adding new modifications that will be explained in this document. Remember that the changes are cumulative from version to version.

1. Ruby Pure

The first ruby version made is called ruby pure, and the changes are: first, it assumes we are searching for parallel lines. With this knowledge, in the eraseBadModels method, we change the Energy / “Number of points” as the measure of quality, and instead we use energy/ (“Number of points” * “Number of Parallel lines”), this way models that have parallel counterparts are more likely to get chosen. The second change is that the models persist from an execution to another, the reasoning is the robot will move in a continuous manner, this way the models that were found in the previous iteration will be found again.

2. Ruby Genetic

The second version takes a genetic algorithm approach for ruby. If you don’t know about genetic algorithms the general concept is to imitate nature’s evolution by natural selection. The principle is to create a big set of possible solutions (‘individuals’), have a way to measure the ‘quality’ of every individual, kill the worst individuals and create newer ones from those that survived. For ruby tough, we search for new models in a random manner. The way this version works is searching for 40 models initially but not assigning the points to any model, then it fuses models that are close, and attach all points to their closest models. After that, we kill all models that have 3 or less points, uses linear fit on every model to re-estimate the slope and intercept, and kill the worst 75% of models keeping always at least 6. Finally we calculate the set’s total energy and store it the total energy went down. After all iterations, the algorithm returns all models that survived this process.

3. Ruby Genetic One Point

The third version includes the ‘One Point’ feature. The older versions take into account every single point that the LIDAR finds to search for models. This is inefficient and bad because the algorithm can select points from the same object and it has hundreds of points to redistribute during the alpha-expansion. What this version does, is finding a ‘center of mass’ for points that are close, in a attempt to find the center of the real object. You can visualize the chosen center of mass point in rviz as a red square, and get the points used by the method “getInitialField”.

4. Ruby Genetic One Point Positive/Negative

This version is pretty similar to the third one. The only difference is that this version separates the vector of points into a positive and a negative parts (considering y-coordinate). The objective is to have the outliers’ vector always with a division between positive and negative points, this way when searching for new models, we trace a line that only contains positive or negative points. The reasoning is that if the robot is well placed, the lines won’t cross from positive to negative quadrants, and if the robot is not well placed, it can at least reduce the probability of finding crossing models.

5. Ruby Genetic One Point Positive/Negative Infinite

The last version for ruby implements a very minor change. The points in this version are not considered consumables, and can be attached to various models. The reasoning being that if points are not scarce, the models will have more data to work with. This way if a bad model has taken a “good” point, it would be still available for a better model.

Doing a quick test, we decided to chose the 4th version, as it had the highest accuracy rate and smallest run time, but this can be tested in the future. Since the methods for all versions do the same job, but in a different way, this document will explain only 4th version methods but keep in mind that the classes work like described previously. Keep in mind that all ruby classed are derived from Pearl, so everything explained in the Pearl document is still valid here.

Ruby 4 Public Methods

- RubyGeneticOnePointPosNeg( )

  RubyGeneticOnePointPosNeg::RubyGeneticOnePointPosNeg ( )

  Contructor that creates all the vectors.

- populateOutliers( )

  void RubyGeneticOnePointPosNeg::populateOutliers (const sensor_msgs::LaserScan & msg)

  Populate outliers’ vector with all the points in the ros message

  Parameters

  msg is the Ros message containing the LIDAR’s data

- findLines( )

  std::vector<Model> RubyGeneticOnePointPosNeg::findLines ( )

Uses the Ruby algorithm to find the lines into the cloud of points, then return the best result.

- getInitialField( )

  std::vector<Point> RubyGeneticOnePointPosNeg::getInitialField ( ) const

  Gets the initial field of points.

Private Methods

- removeModel( )
  void RubyGeneticOnePointPosNeg::removeModel (const int modelIndex)

  Remove the model in the index ‘modelIndex’

  Parameters
  modelIndex is the index to be deleted

- removePointsInModels( )
  void RubyGeneticOnePointPosNeg::removePointsInModels ( )

  Remove the points attached in all models contained in the set

- randomPointsInField( )
  std::vector<Point> RubyGeneticOnePointPosNeg::randomPointsInField (const int num)

  Return a vector containing ‘num’ different points in the outliers vector

  Parameters
  num is the amount of points to be selected

- searchModels( )
  void RubyGeneticOnePointPosNeg::searchModels (const int nbOfModels)

  Search for ‘nbOfModels’ models using the points in the outliers

  Parameters
  nbOfModels is the amount of models to be searched

- redistributePoints( )
  double RubyGeneticOnePointPosNeg::redistributePoints ( )

  Detaches points from all models and attaches it to the closest model

- calculateEnergy( )
  double RubyGeneticOnePointPosNeg::calculateEnergy ( ) const

  Calculates and returns the set’s energy

- meanNumOfPoints( )
  double RubyGeneticOnePointPosNeg::meanNumOfPoints ( )

  Calculates and returns the average number of points for the models

- countParallelLines( )
  void RubyGeneticOnePointPosNeg::countParallelLines ( )

  Increments every model's parallel counter to correctly calculate fitness

- eraseBadModels( )
  void RubyGeneticOnePointPosNeg::eraseBadModels ( )

  Remove the 75% worst models from the vector, keeping at least 6.

- friend operator << ( )
  std::ostream & operator << (std::ostream & out, const RubyGeneticOnePointPosNeg &r)

  Print Ruby object in terminal.

  Parameters

  out is where to print, normally terminal
  p is the object to be printed