Profiling tinySlam manual

Andrey Mikhalev, fatum.est.series.causarum@gmail.com

Source:

- tinySLAM

Requirements:

- Robot Operating System (ROS);
- gperftools;
- gprof;
- valgrind;

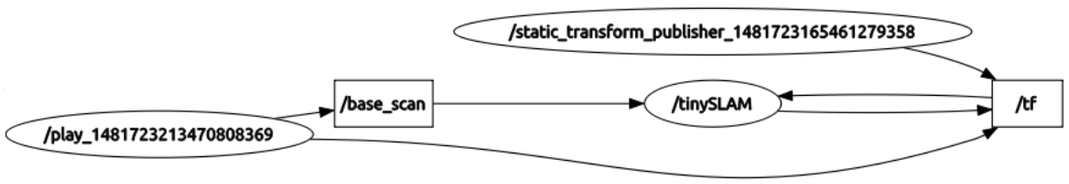
Setup:

- Install and setup ROS:
  - Jade for Ubuntu 14.04
  - Kinetic for Ubuntu 16.04
- Install profilers:
  - Build gperftools from sources and install or install gperftools-package
  - How to install and use gprof
  - install valgrind-package

Before profiling compile *tiny_slam.cpp* with correct flags. See files:

- valgrind_wiki.txt
- gprof_wiki.txt
- gperftools_wiki.txt

ROSgraph for tinySLAM



First step

- Open file **tinyslam_run.launch**. It contains all params for each node:
  1. /**play** (has name "player" in tinyslam_run.launch)
  2. /**tinySLAM**
  3. /**static_transform_publisher** (has name "MapTrasformPublisher" in tinyslam_run.launch)
- Open 5 terminals
  1. In **1st terminal** enter:
     $ roscore
  2. In **2st terminal** enter (node /**static_transform_publisher**):
     $ rosrun tf static_transform_publisher 0 0 0 0 0 0 odom_combined map 250
  3. In **3nd terminal** go to dir that contains ***.bag** file and enter (node /**play**):
     (a) $ rosparam set use_sim_time true
     (b) $ rosbag play –pause –clock –rate 1 *.bag
  4. In **4rd terminal** enter (node /**rviz** for debug):
     (a) $ roscd tiny_slam/
     (b) $ rosrun rviz rviz -d rviz/debug.rviz
  5. In **5th terminal** type (node /**tinySLAM**):
     (a) find path to executable tinySLAM (like <path>/catkin_ws/devel/lib/tiny_slam/)
     (b) run some profiler:
        i. profiler name
        ii. profiler <args> (can be empty)
        iii. full path to executable tinySLAM
        iv. tinySLAM params:
        laser_scan:=/base_scan __name:=tinySLAM
        v. profiler <args> (can be empty)

Example for 5th terminal(valgrind):
$ valgrind --tool=callgrind --dump-line=yes --dump-instr=yes \
home/catkin_ws/devel/lib/tiny_slam/tiny_slam laser_scan:=/base_scan __name:=tinySLAM
Example for 5th terminal(gprof):
$ gprof home/catkin_ws/devel/lib/tiny_slam/tiny_slam \
laser_scan:=/base_scan __name:=tinySLAM gmon.out

After profiling I found some bottlenecks during studying call graph. So, I try make some changes that, you can see, speed up default algorithm.

First change

Comment all methods calls for private method ***do_for_each_observer(std::function<void (ObsPtr)> op*** *in* ***monte_carlo_scan_mather.h:***

```
class MonteCarloScanMatcher : public GridScanMatcher {
private:
    void do_for_each_observer(std::function<void(ObsPtr)> op) {
        for (auto &obs : GridScanMatcher::observers()) {
            if (auto obs_ptr = obs.lock()) {
                op(obs_ptr);
            }
        }
    };
public:
    virtual double process_scan(const RobotState &init_pose, const TransformedLaserScan &
    scan, const GridMap &map, RobotState &pose_delta) override {
        //do_for_each_observer([init_pose, scan, map](ObsPtr obs) {...}
        ...
        //do_for_each_observer([optimal_pose, scan, min_scan_cost](ObsPtr obs) {...}
        while(...) {
            ...
            //do_for_each_observer([sampled_pose, scan, sampled_scan_cost](ObsPtr obs) {...}
            ...
            //do_for_each_observer([optimal_pose, scan, min_scan_cost](ObsPtr obs) {...}
        }
        ...
        //do_for_each_observer([pose_delta, min_scan_cost](ObsPtr obs) {...}
        ...
    }
}
```

Second change

After comment observers reserve memory in ***tiny_world.h*** *and* ***laser_scan_grid_world.h:***

BEFORE

```
class LaserScanGridWorld : public World<TransformedLaserScan, GridMap> {
public:
    virtual void handle_scan_point(MapType &map, double laser_x, double laser_y, double
    beam_end_x, double beam_end_y, bool is_occ, double quality) {
        ...
        std::vector<Point> pts = DiscreteLine2D(robot_pt, obst_pt).points();
        ...
    }
}
```

AFTER

```
class LaserScanGridWorld : public World<TransformedLaserScan, GridMap> {
public:
    virtual void handle_scan_point(MapType &map, double laser_x, double laser_y, double
    beam_end_x, double beam_end_y, bool is_occ, double quality) {
        ...
        std::vector<Point> pts;
        pts.reserve(1000);
        DiscreteLine2D(robot_pt, obst_pt, pts);
        ...
    }
}
```

After reserving memory change constructor and private method in ***geometry_utils.h:***

BEFORE

```
class DiscreteLine2D {
public:
    std::vector<Point> _points;
    DiscreteLine2D(const Point &start, const Point &end) {
        generatePointsWithBresenham(start.x, start.y, end.x, end.y);
    }
private:
    void generatePointsWithBresenham(int x1, int y1, int x2, int y2) {
        //...
        _points.push_back(Point(x1, y1));
        //...
    }
}
```

AFTER

```
class DiscreteLine2D {
public:
    // std::vector<Point> _points;
    DiscreteLine2D(const Point &start, const Point &end, std::vector<Point> & pts) {
        generatePointsWithBresenham(start.x, start.y, end.x, end.y, pts);
    }
private:
    void generatePointsWithBresenham(int x1, int y1, int x2, int y2, std::vector<Point> &
    pts) {
        // ...
        pts.push_back(Point(x1, y1));
        // ...
    }
}
```