

```
30  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
31  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
32  * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
33  * POSSIBILITY OF SUCH DAMAGE.
34  *
35  * Author: Eitan Marder-Eppstein
36  *****/
37  #include <navfn/navfn_ros.h>
38  #include <pluginlib/class_list_macros.h>
39  #include <costmap_2d/cost_values.h>
40  #include <costmap_2d/costmap_2d.h>
41  #include <sensor_msgs/point_cloud2_iterator.h>
42
43  //register this planner as a BaseGlobalPlanner plugin
44  PLUGINLIB_EXPORT_CLASS(navfn::NavfnROS, nav_core::BaseGlobalPlanner)
45
46  namespace navfn {
47
48      NavfnROS::NavfnROS()
49          : costmap_(NULL), planner_(), initialized_(false), allow_unknown_(true) {}
50
51      NavfnROS::NavfnROS(std::string name, costmap_2d::Costmap2DRos* costmap_ros)
52          : costmap_(NULL), planner_(), initialized_(false), allow_unknown_(true) {
53          //initialize the planner
54          initialize(name, costmap_ros);
55      }
56
57      NavfnROS::NavfnROS(std::string name, costmap_2d::Costmap2D* costmap, std::string global_
58          : costmap_(NULL), planner_(), initialized_(false), allow_unknown_(true) {
59          //initialize the planner
60          initialize(name, costmap, global_frame);
61      }
62
63      void NavfnROS::initialize(std::string name, costmap_2d::Costmap2D* costmap, std::string
64          if(!initialized_){
65              costmap_ = costmap;
66              global_frame_ = global_frame;
67              planner_ = boost::shared_ptr<NavFn>(new NavFn(costmap->getSizeInCellsX(), costmap_
68
69              ros::NodeHandle private_nh("~/\" + name);
70
71              plan_pub_ = private_nh.advertise<nav_msgs::Path>("plan", 1);
72
73              private_nh.param("visualize_potential", visualize_potential_, false);
74
75              //if we're going to visualize the potential array we need to advertise
76              if(visualize_potential_)
77                  potarr_pub_ = private_nh.advertise<sensor_msgs::PointCloud2>("potential", 1);
78
```

```
79     private_nh.param("allow_unknown", allow_unknown_, true);
80     private_nh.param("planner_window_x", planner_window_x_, 0.0);
81     private_nh.param("planner_window_y", planner_window_y_, 0.0);
82     private_nh.param("default_tolerance", default_tolerance_, 0.0);
83
84     make_plan_srv_ = private_nh.advertiseService("make_plan", &NavfnROS::makePlanService);
85
86     initialized_ = true;
87 }
88 else
89     ROS_WARN("This planner has already been initialized, you can't call it twice, doing nothing");
90 }
91
92 void NavfnROS::initialize(std::string name, costmap_2d::Costmap2DROS* costmap_ros){
93     initialize(name, costmap_ros->getCostmap(), costmap_ros->getGlobalFrameID());
94 }
95
96 bool NavfnROS::validPointPotential(const geometry_msgs::Point& world_point){
97     return validPointPotential(world_point, default_tolerance_);
98 }
99
100 bool NavfnROS::validPointPotential(const geometry_msgs::Point& world_point, double tolerance){
101     if(!initialized_){
102         ROS_ERROR("This planner has not been initialized yet, but it is being used, please call initialize");
103         return false;
104     }
105
106     double resolution = costmap_->getResolution();
107     geometry_msgs::Point p;
108     p = world_point;
109
110     p.y = world_point.y - tolerance;
111
112     while(p.y <= world_point.y + tolerance){
113         p.x = world_point.x - tolerance;
114         while(p.x <= world_point.x + tolerance){
115             double potential = getPointPotential(p);
116             if(potential < POT_HIGH){
117                 return true;
118             }
119             p.x += resolution;
120         }
121         p.y += resolution;
122     }
123
124     return false;
125 }
126
127 double NavfnROS::getPointPotential(const geometry_msgs::Point& world_point){
```

```
128     if(!initialized_){
129         ROS_ERROR("This planner has not been initialized yet, but it is being used, please c
130         return -1.0;
131     }
132
133     unsigned int mx, my;
134     if(!costmap_->worldToMap(world_point.x, world_point.y, mx, my))
135         return DBL_MAX;
136
137     unsigned int index = my * planner_->nx + mx;
138     return planner_->potarr[index];
139 }
140
141 bool NavfnROS::computePotential(const geometry_msgs::Point& world_point){
142     if(!initialized_){
143         ROS_ERROR("This planner has not been initialized yet, but it is being used, please c
144         return false;
145     }
146
147     //make sure to resize the underlying array that Navfn uses
148     planner_->setNavArr(costmap_->getSizeInCellsX(), costmap_->getSizeInCellsY());
149     planner_->setCostmap(costmap_->getCharMap(), true, allow_unknown_);
150
151     unsigned int mx, my;
152     if(!costmap_->worldToMap(world_point.x, world_point.y, mx, my))
153         return false;
154
155     int map_start[2];
156     map_start[0] = 0;
157     map_start[1] = 0;
158
159     int map_goal[2];
160     map_goal[0] = mx;
161     map_goal[1] = my;
162
163     planner_->setStart(map_start);
164     planner_->setGoal(map_goal);
165
166     return planner_->calcNavFnDijkstra();
167 }
168
169 void NavfnROS::clearRobotCell(const geometry_msgs::PoseStamped& global_pose, unsigned int
170     if(!initialized_){
171         ROS_ERROR("This planner has not been initialized yet, but it is being used, please c
172         return;
173     }
174
175     //set the associated costs in the cost map to be free
176     costmap_->setCost(mx, my, costmap_2d::FREE_SPACE);
```

```
177     }
178
179     bool NavfnROS::makePlanService(nav_msgs::GetPlan::Request& req, nav_msgs::GetPlan::Response& resp) {
180         makePlan(req.start, req.goal, resp.plan.poses);
181
182         resp.plan.header.stamp = ros::Time::now();
183         resp.plan.header.frame_id = global_frame_;
184
185         return true;
186     }
187
188     void NavfnROS::mapToWorld(double mx, double my, double& wx, double& wy) {
189         wx = costmap_->getOriginX() + mx * costmap_->getResolution();
190         wy = costmap_->getOriginY() + my * costmap_->getResolution();
191     }
192
193     bool NavfnROS::makePlan(const geometry_msgs::PoseStamped& start,
194                             const geometry_msgs::PoseStamped& goal, std::vector<geometry_msgs::PoseStamped>& plan,
195                             double default_tolerance_, bool allow_partial_plan_) {
196         return makePlan(start, goal, default_tolerance_, plan);
197     }
198
199     bool NavfnROS::makePlan(const geometry_msgs::PoseStamped& start,
200                             const geometry_msgs::PoseStamped& goal, double tolerance, std::vector<geometry_msgs::PoseStamped>& plan,
201                             boost::mutex::scoped_lock lock(mutex_)) {
202         if(!initialized_){
203             ROS_ERROR("This planner has not been initialized yet, but it is being used, please call initialize() first");
204             return false;
205         }
206
207         //clear the plan, just in case
208         plan.clear();
209
210         ros::NodeHandle n;
211
212         //until tf can handle transforming things that are way in the past... we'll require the goal to be in the global frame
213         if(goal.header.frame_id != global_frame_){
214             ROS_ERROR("The goal pose passed to this planner must be in the %s frame. It is instead in the %s frame.",
215                       global_frame_.c_str(), goal.header.frame_id.c_str());
216             return false;
217         }
218
219         if(start.header.frame_id != global_frame_){
220             ROS_ERROR("The start pose passed to this planner must be in the %s frame. It is instead in the %s frame.",
221                       global_frame_.c_str(), start.header.frame_id.c_str());
222             return false;
223         }
224
225         double wx = start.pose.position.x;
```

```
226
227     unsigned int mx, my;
228     if(!costmap_->worldToMap(wx, wy, mx, my)){
229         ROS_WARN("The robot's start position is off the global costmap. Planning will always
230         return false;
231     }
232
233     //clear the starting cell within the costmap because we know it can't be an obstacle
234     clearRobotCell(start, mx, my);
235
236     //make sure to resize the underlying array that Navfn uses
237     planner_->setNavArr(costmap_->getSizeInCellsX(), costmap_->getSizeInCellsY());
238     planner_->setCostmap(costmap_->getCharMap(), true, allow_unknown_);
239
240     int map_start[2];
241     map_start[0] = mx;
242     map_start[1] = my;
243
244     wx = goal.pose.position.x;
245     wy = goal.pose.position.y;
246
247     if(!costmap_->worldToMap(wx, wy, mx, my)){
248         if(tolerance <= 0.0){
249             ROS_WARN_THROTTLE(1.0, "The goal sent to the navfn planner is off the global costmap. Planning will always
250             return false;
251         }
252         mx = 0;
253         my = 0;
254     }
255
256     int map_goal[2];
257     map_goal[0] = mx;
258     map_goal[1] = my;
259
260     planner_->setStart(map_goal);
261     planner_->setGoal(map_start);
262
263     //bool success = planner_->calcNavFnAstar();
264     planner_->calcNavFnDijkstra(true);
265
266     double resolution = costmap_->getResolution();
267     geometry_msgs::PoseStamped p, best_pose;
268     p = goal;
269
270     bool found_legal = false;
271     double best_sdist = DBL_MAX;
272
273     p.pose.position.y = goal.pose.position.y - tolerance;
274
```

```

275 while(p.pose.position.y <= goal.pose.position.y + tolerance){
276     p.pose.position.x = goal.pose.position.x - tolerance;
277     while(p.pose.position.x <= goal.pose.position.x + tolerance){
278         double potential = getPointPotential(p.pose.position);
279         double sdist = sq_distance(p, goal);
280         if(potential < POT_HIGH && sdist < best_sdist){
281             best_sdist = sdist;
282             best_pose = p;
283             found_legal = true;
284         }
285         p.pose.position.x += resolution;
286     }
287     p.pose.position.y += resolution;
288 }
289
290 if(found_legal){
291     //extract the plan
292     if(getPlanFromPotential(best_pose, plan)){
293         //make sure the goal we push on has the same timestamp as the rest of the plan
294         geometry_msgs::PoseStamped goal_copy = best_pose;
295         goal_copy.header.stamp = ros::Time::now();
296         plan.push_back(goal_copy);
297     }
298     else{
299         ROS_ERROR("Failed to get a plan from potential when a legal potential was found. 1
300     }
301 }
302
303 if (visualize_potential_)
304 {
305     // Publish the potentials as a PointCloud2
306     sensor_msgs::PointCloud2 cloud;
307     cloud.width = 0;
308     cloud.height = 0;
309     cloud.header.stamp = ros::Time::now();
310     cloud.header.frame_id = global_frame_;
311     sensor_msgs::PointCloud2Modifier cloud_mod(cloud);
312     cloud_mod.setPointCloud2Fields(4, "x", 1, sensor_msgs::PointField::FLOAT32,
313                                     "y", 1, sensor_msgs::PointField::FLOAT32,
314                                     "z", 1, sensor_msgs::PointField::FLOAT32,
315                                     "pot", 1, sensor_msgs::PointField::FLOAT32);
316     cloud_mod.resize(planner_>ny * planner_>nx);
317     sensor_msgs::PointCloud2Iterator<float> iter_x(cloud, "x");
318
319     PotarrPoint pt;
320     float *pp = planner_>potarr;
321     double pot_x, pot_y;
322     for (unsigned int i = 0; i < (unsigned int)planner_>ny*planner_>nx ; i++)
323     {

```

```
324     if (pp[i] < 10e7)
325     {
326         mapToWorld(i%planner_->nx, i/planner_->nx, pot_x, pot_y);
327         iter_x[0] = pot_x;
328         iter_x[1] = pot_y;
329         iter_x[2] = pp[i]/pp[planner_->start[1]*planner_->nx + planner_->start[0]]*20;
330         iter_x[3] = pp[i];
331         ++iter_x;
332     }
333 }
334 potarr_pub_.publish(cloud);
335 }
336
337 //publish the plan for visualization purposes
338 publishPlan(plan, 0.0, 1.0, 0.0, 0.0);
339
340 return !plan.empty();
341 }
342
343 void NavfnROS::publishPlan(const std::vector<geometry_msgs::PoseStamped>& path, double r
344     if(!initialized_){
345         ROS_ERROR("This planner has not been initialized yet, but it is being used, please c
346         return;
347     }
348
349     //create a message for the plan
350     nav_msgs::Path gui_path;
351     gui_path.poses.resize(path.size());
352
353     if(path.empty()) {
354         //still set a valid frame so visualization won't hit transform issues
355         gui_path.header.frame_id = global_frame_;
356         gui_path.header.stamp = ros::Time::now();
357     } else {
358         gui_path.header.frame_id = path[0].header.frame_id;
359         gui_path.header.stamp = path[0].header.stamp;
360     }
361
362     // Extract the plan in world co-ordinates, we assume the path is all in the same frame
363     for(unsigned int i=0; i < path.size(); i++){
364         gui_path.poses[i] = path[i];
365     }
366
367     plan_pub_.publish(gui_path);
368 }
369
370 bool NavfnROS::getPlanFromPotential(const geometry_msgs::PoseStamped& goal, std::vector<
371     if(!initialized_){
372         ROS_ERROR("This planner has not been initialized yet, but it is being used, please c
```

```
373     return false;
374 }
375
376 //clear the plan, just in case
377 plan.clear();
378
379 //until tf can handle transforming things that are way in the past... we'll require th
380 if(goal.header.frame_id != global_frame_){
381     ROS_ERROR("The goal pose passed to this planner must be in the %s frame. It is inst
382             global_frame_.c_str(), goal.header.frame_id.c_str());
383     return false;
384 }
385
386 double wx = goal.pose.position.x;
387 double wy = goal.pose.position.y;
388
389 //the potential has already been computed, so we won't update our copy of the costmap
390 unsigned int mx, my;
391 if(!costmap_->worldToMap(wx, wy, mx, my)){
392     ROS_WARN_THROTTLE(1.0, "The goal sent to the navfn planner is off the global costmap
393     return false;
394 }
395
396 int map_goal[2];
397 map_goal[0] = mx;
398 map_goal[1] = my;
399
400 planner_->setStart(map_goal);
401
402 planner_->calcPath(costmap_->getSizeInCellsX() * 4);
403
404 //extract the plan
405 float *x = planner_->getPathX();
406 float *y = planner_->getPathY();
407 int len = planner_->getPathLen();
408 ros::Time plan_time = ros::Time::now();
409
410 for(int i = len - 1; i >= 0; --i){
411     //convert the plan to world coordinates
412     double world_x, world_y;
413     mapToWorld(x[i], y[i], world_x, world_y);
414
415     geometry_msgs::PoseStamped pose;
416     pose.header.stamp = plan_time;
417     pose.header.frame_id = global_frame_;
418     pose.pose.position.x = world_x;
419     pose.pose.position.y = world_y;
420     pose.pose.position.z = 0.0;
421     pose.pose.orientation.x = 0.0;
```



```
422     pose.pose.orientation.y = 0.0;
423     pose.pose.orientation.z = 0.0;
424     pose.pose.orientation.w = 1.0;
425     plan.push_back(pose);
426 }
427
428 //publish the plan for visualization purposes
429 publishPlan(plan, 0.0, 1.0, 0.0, 0.0);
430 return !plan.empty();
431 }
432 };
```