On Path Planning for Mobile Robots: Introducing the Mereological Potential Field Method in the Framework of Mereological Spatial Reasoning

Received 24th June 2008; accepted 29th September 2008.

Paweł Ośmiałowski

Abstract:

Path planning is one of the most vital problems in mobile robotics; it falls into general province of planning, however, due to specificity of the subject of mobile robotics, it has emerged as a discipline per se *with its own solutions. Among many methods of probabilistic, geometrical and topological nature, the methodology of potential fields introduced by Krogh (1984) and Khatib (1985), based on physical analogies with gravitational or electromagnetic fields, has emerged. We adhere to this methodology, however, contrary to the practice of building the potential on the basis of Coulomb, or gravitational force fields, we apply the novel idea of building the potential function by means of mereological distance over a juxtaposition of grids of fixed diameter, i.e., over a discrete structure. We describe our implementation of the relevant mereological functors in the Player/Stage system as SQL predicates accessible in Player/Stage cooperating with PostgreSQL database. We present also the results of simulations with mobile robots.*

Keywords: *mobile robotics, path planning, rough mereology, potential field, Player/Stage system.*

1. Introduction

Tasks of mobile robotics like path planning, navigation, localization, require for their effective performing adequate representation of the robot environment including obstacles, landmarks, beacons, other robots, along with other static or dynamic features as well as an effective reasoning scheme. Mobile robotics avails itself to this aim with many ideas and methods known in relevant areas of Computer Science and Artificial Intelligence and it does make use of graph methods and algorithms like A* in search or planning [1], topological (graph) representations in map building [1], [25], Markov models in navigation and localization [25] etc. In this work, we propose a new method of constructing a potential field [15] by which to delineate a path for a mobile robot. Our method is based on reasoning scheme for spatial objects developed in the framework of rough mereology [18]. Our potential field inherits the basic property of potential fields, see e.g., [1], i.e., the density of the field does increase in the direction to the target, reaching at the target the maximum value (which in classical cases is infinity). We leave the technical exposition to the following consecutive sections in which we discuss: basic principles of spatial reasoning by mereological predicates, the idea of a potential field along with details of our construction, the description of the Player/Stage system along with SQL predicates implementing basic relevant mereological relations [19], and finally, results of simulations with mobile robots.

2. Mereological spatial reasoning

It is well-known [23] that relations among geometrical objects like solids in 3D or planar figures, are best expressed in the language of parts rather than in the language of set theory: one accepts the statement: "*the circle is a part of the closed disc*" but one would reject the statement: "*the circle is an element of the closed disc*"; in the first statement one would accept "*a subset*" in place of "*a part*" and indeed, the relation of a subset is a particular instance of the relation of part.

The part relation is a basic relation of mereology - the theory of sets/concepts [16] proposed by S. Lesniewski (1916) and adopted to formalize elementary geometry of solids [23]. Due to this fact, one may expect that mereology based constructions will prove useful in mobile robotics tasks. We offer a concise introduction to this area.

2.1. Mereology

The part relation between objects e.g., solids is a relation p such that (1) it is not true that $p(A, A)$ for any A, (2) if $p(A, B)$ and $p(B, C)$ then $p(A, C)$.

Example: the relation of strict containment $\subset$ is a part relation.

The relation of an element (ingredient) el is defined as: $el(A, B)$ if and only if $p(A, B)$ or $A = B$.

Example: the relation of non-strict containment $\subseteq$ is an element relation induced by the part relation $\subset$.

The part, or element, relations give a strict hierarchy of objects consisting of "smaller" objects and so on. However, it was desirable to measure a degree of closeness between e.g. solids not necessarily in the part relation. This was proposed in the frame of rough mereology [18].

2.2. Rough mereology

The basic relation/predicate of rough mereology [18], is a rough inclusion $\mu(A, B, r)$, where $r \in [0, 1]$, which means: "A *is a part of* B *to a degree of* r". The predicate μ captures our basic intuitions about the nature of containment to a degree and accordingly we impose the following restrictions on it [20]:

(1) $\mu(A, B, 1)$ if and only if $el(A, B)$ where el is a chosen element relation of a mereology.

(2) if $\mu(A, B, 1)$ then [if $\mu(C, A, r)$ then $\mu(C, B, r)$] for each C.

(3) if $\mu(A, B, r)$ and $s < r$ then $\mu(\mathrm{A}, \mathrm{B}, \mathrm{s})$.

As far as solids or planar figures are concerned, we apply the rough inclusion μ defined as $\mu(A, B, r)$ if and only if

$$\frac{\|A \cap B\|}{\|A\|} \geq r, \quad (1)$$

where $\|A\|$ is the Euclidean volume of A in 3D case, or area in 2D case.

2.3. Qualitative spatial reasoning: Basic geometric predicates induced by μ

Qualitative Reasoning aims at studying concepts and calculi on them that arise often at early stages of problem analysis when one is refraining from qualitative or metric details, cf., [5] as such it has close relations to the design, cf., [3] as well as planning stages, cf., [7] of the model synthesis process. Classical formal approaches to spatial reasoning, i.e., to representing spatial entities (points, surfaces, solids) and their features (dimensionality, shape, connectedness degree) rely on Geometry or Topology, i.e., on formal theories whose models are spaces (universes) constructed as sets of points; contrary to this approach, qualitative reasoning about space often exploits pieces of space (regions, boundaries, walls, membranes) and argues in terms of relations abstracted from a commonsense perception (like *connected, discrete from, adjacent, intersecting*). In this approach, points appear as ideal objects (e.g., ultrafilters of regions/solids [23]).

Mereological ideas have been early applied toward axiomatization of geometry of solids, cf., [14], [23]. Mereological theories dominant nowadays come from ideas proposed independently by Stanislaw Lesniewski and Alfred North Whitehead.

Mereological theory of Lesniewski is based on the notion of a part (proper) cf., [16]. Mereology based on connection gave rise to spatial calculi based on topological notions derived there from (mereotopology), cf., [5], [6], [8].

Predicates μ_r may be regarded as weak metrics also in the context of geometry. From this point of view, we may apply μ in order to define basic notions of rough mereological geometry.

In the language of this geometry, we may approximately describe and approach geometry of objects; a usage for this geometry may be found, e.g., in navigation and control tasks of mobile robotics [1], [12].

It is well-known, see [24], [2] that the geometry of Euclidean spaces may be based on some postulates about the basic notions of a point and the ternary equi-distance functor. In [24], postulates for Euclidean geometry over a real-closed field were given based on the functor of betweenness and the quaternary equi-distance functor. Similarly, in [2], a set of postulates aimed at rendering general geometric features of geometry of finite-dimensional spaces over reals has been discussed, the primitive notion there being that of nearness.

We first introduce a notion of distance κ_r in our rough mereological universe by letting

$$\kappa_r(X, Y) \Leftrightarrow r = min\{max\, u, max\, w : X\, is\, \mu_u Y \wedge Y\, is\, \mu_w X\}.$$

We now introduce the notion of betweenness as a functor $T(X,Y)$ of two individual names; the statement Z is $T(X,Y)$ reads as Z *is between* X *and* Y:

$$Z\ is\ T(X,Y) \Leftrightarrow$$

for all

$$W\ \kappa_r(Z, W) \wedge \kappa_s(X, W) \wedge \kappa_t(Y, W) \Rightarrow s \leq r \leq t \vee t \leq r \leq s.$$

Thus, Z is $T(X,Y)$ holds when the rough mereological distance κ between Z and any W is in the non-oriented interval (i.e. between) [distance of X to W, distance of Y to W] for any W.

One checks that T satisfies the axioms of Tarski [24] for *betweenness*.

Proposition 1. *The following properties hold, see [19]*:
1. $Z\ is\ T(X, X) \Leftrightarrow Z = X$ *(identity);*
2. $Y\ is\ T(X, U) \wedge Z\ is\ T(Y, U) \Rightarrow Y\ is\ T(X, Z)$ *(transitivity);*
3. $Y\ is\ T(X, Z) \wedge Y\ is\ T(X, U) \wedge X \neq Y \Rightarrow Z\ is\ T(X, U) \vee U\ is\ T(X, Z)$ *(connectivity)*.

2.4. Nearness

We may also apply κ to define in our context the functor N of nearness proposed in van Bentham [2]:

$$Z\ is\ N(X, Y) \Leftrightarrow (\kappa_r(Z, X) \wedge \kappa_s(X, Y) \Rightarrow s < r).$$

Here, nearness means that Z is closer to X than to Y (recall that rough mereological distance is defined in an opposite way: the smaller r, the greater distance).

Then the following hold, i.e., N does satisfy all axioms for nearness in [2], see [19].

Proposition 2. *The following properties hold, see [19]*:
1. $Z\ is\ N(X, Y) \wedge Y\ is\ N(X, W) \Rightarrow Z\ is\ N(X, W)$ *(transitivity)*;
2. $Z\ is\ N(X, Y) \wedge X\ is\ N(Y, Z) \Rightarrow X\ is\ N(Z, Y)$ *(triangle inequality);*
3. $non(Z\ is\ N(X, Z))$ *(irreflexivity)*;
4. $Z = X\ ?\ Z\ is\ N(Z, X)$ *(selfishness)*;
5. $Z\ is\ N(X, Y) \Rightarrow Z\ is\ N(X, W) \vee W\ is\ N(X, Y)$ *(connectedness)*.

We now may introduce the notion of equi-distance as a functor $Eq(X, Y)$ defined as follows:

$$Z\ is\ Eq(X, Y) \Leftrightarrow (non(X\ is\ N(Z, Y)) \wedge non(Y\ is\ N(Z, X))).$$

It follows that

Proposition 3. $Z\ is\ Eq(X, Y) \Leftrightarrow$ *(for all* r $(\kappa_r(X, Z) \Leftrightarrow \kappa_r(Y, Z))$.

We may also define a functor of equi-distance following Tarski [24]:

$$D(X, Y, Z, W) \Leftrightarrow (for\ all\ r\ \ \kappa_r(X, Y) \Leftrightarrow \kappa_r(Z, W)).$$

These functors do clearly satisfy the following, see [2], [24],

Proposition 4. *The following properties hold, see [19]*:

1. Z *is* $Eq(X, Y) \wedge X$ *is* $Eq(Y, Z) \Rightarrow Y$ *is* $Eq(Z, X)$ *(triangle equality);*
2. Z *is* $T(X, Y) \wedge W$ *is* $Eq(X, Y) \Rightarrow D(Z, W, X, W)$ *(circle property);*
3. $D(X, Y, Y, X)$ *(reflexivity);*
4. $D(X, Y, Z, Z) \Rightarrow X = Y$ *(identity);*
5. $D(X, Y, Z, U) \wedge D(X, Y, V, W) \Rightarrow D(Z, U, V, W)$ *(transitivity).*

One may follow van Bentham's proposal for a betweenness functor defined via the nearness functor as follows:

Z *is* $TB(X, Y) \Leftrightarrow$ *[for all* W *(*Z *is* $W \vee Z$ *is* $N(X, W) \vee Z$ *is* $N(Y, W))]$.

3. Potential fields methodology: Mereological approach

Methodology of potential fields [11], [9] was conceived as a framework in which one could generate smooth trajectories from the start point to a target point by mbile robots as well as manipulators. It is known [10] that the method is capable of falling into local minima (as general hill climbing methods do) or oscillatory behaviour, however, working with a global potential map prevents these phenomena: such is our approach in this work.

Classical methodology works with integrable force field given by formulas of Coulomb or Newton which prescribe force at a given point as inversely proportional to the squared distance from the target; in consequence the potential is inversely proportional to the distance from the target. The basic property of the potential is that its density (=force) increases in the direction toward the target. We observe this property in our construction.

3.1. The construction of a potential field: The idea

We construct the potential field by a discrete construction. The idea is to fill the free workspace of a robot with squares of fixed size in such a way that the density of the square field (measured e.g. as the number of squares intersecting the disc of a given radius r centred at the target) increases toward the target.

To ensure this property, we fix a real number - *the field growth step* in the interval (0, square edge length), in our case it is 0.01. The collection of squares grows recursively with the distance from the target by adding to a given square in the $(k + 1)$*–th* step all squares obtained from it by translating it by $k \times$ field growth step (with respect to Euclidean distance) in basic eight directions: N, S, W, E, NE, NW, SE, SW (in the implementation of this idea, the *floodfill algorithm* with queue has been used, see the next section).

Once the square field is constructed, the path for a robot from a given starting point toward the target is searched for.

The idea of this search is in finding a sequence of *waypoints* which delineate the path to the target. Waypoints are found recursively as centroids of unions of squares mereologically closest to the square of the recently found waypoint. In determining mereological distance, the formula (1) is applied.

Parameters of this procedure are: the size of a square (in this work, our robots have been built on the basis of Roomba robot and accordingly, squares we chose to be of edge length equal to 1.5 Roomba (is the trademark of iRobot Inc.) diameter which does ensure safe transition from one square to another close one) and the field growth step (we set it to 0.01 which is small value in comparison to the square size).

4. The mereological path planner

The path planner we designed accepts target point coordinates and provides list of waypoints from given robot position to the goal. To do its job it needs a map of static obstacles that a robot should avoid while appro-

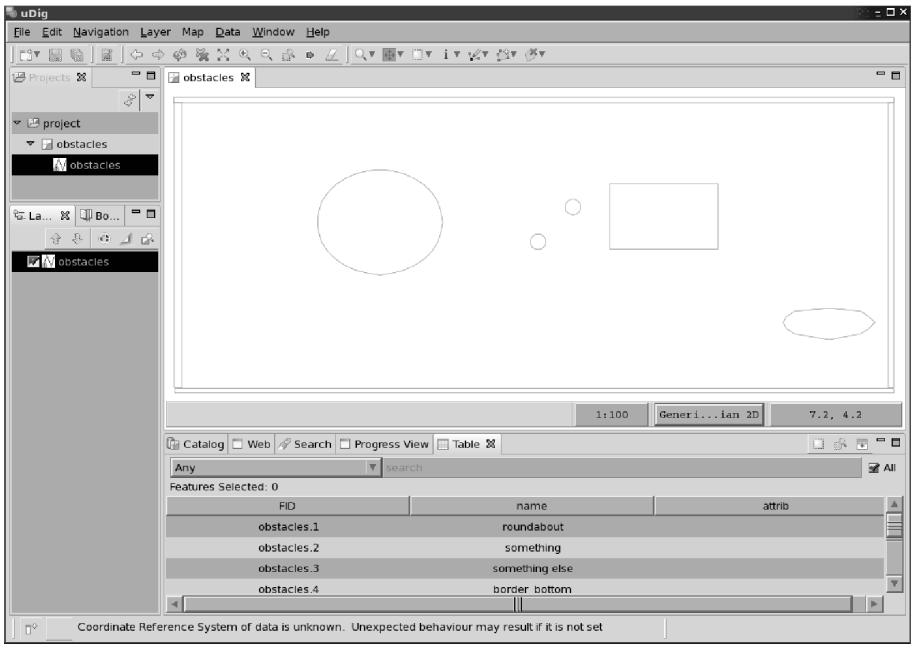

Fig. 1. Map of our artificial world edited by the uDig appli-cation (created and maintained by Refractions Research). The map consists of number of layers whose can be edited individually; on the figure we can see how solid obstacles are situated within obstacles layer.

aching target point. A robot and a target should both lay within the area delimited by surrounding static obstacles that form borders of robot workspace. There can be other static obstacles within the area, all marked on the provided map. After the path is proposed a robot is lead through the path until it reaches given target. If a robot cannot move towards goal position for some longer time (e.g. it keeps on hitting other robot reaching its target or any other unknown non-static obstacle), new path is proposed.

We tested our planner device running simulations in which we have had a model of Roomba robot traveling inside artificial workspace. Real Roomba robots are round and therefore easy to model, however they do not provide many useful sensor devices (except bumpers which we were using to implement lower-level reaction for hitting unexpected obstacles). Also odometry of Roomba robots is unreliable [26] hence we assume that simulated robots are equipped with a global positioning system.

Right after the goal position is given, our planner builds mereological potential field filled with squared areas each of the same size. The field is delimited by workspace borders. Only space free of obstacles is filled. To compute a value of potential field in a place we are taking mereological feature of one object being a part of another to a degree where our objects are squared areas that fill the potential field. Near the goal any two squared areas are parts of each other to a higher degree and this value goes low as the distance to the goal increases. It can happen that for bigger workspace, areas too far from the goal are not filled as the potential is limited to values from 0 to 1, where value 0 means that two squares are not part of each other (maximal mereological distance between two areas) while 1 means that two areas are part of each other to a maximal degree (minimal mereological distance between two areas). As a result our potential field is dense with squared areas close to the target and it gets loose far from it.

The algorithm of filling the potential field with squared areas is following.

SQUARE_FILL_ALGORITHM

Structure: a queue Q

1. Add to the queue Q, x and y coordinates of a given goal together with 0 as current distance from current squared area to the next neighbouring area (so they will be part of each other to the maximal degree). Also put clockwise as current direction of exploration. These are initial values.
2. Spin in the main loop until there are no more elements in the queue Q:
   2.1. Extract x, y, current distance and current direction of exploration from the beginning of queue Q.
   2.2. Check if there is any other squared area already present in potential field to which the distance from current x and y coordinates is equal or shorter than current distance. If so, skip taken element and run new main loop turn.
   2.3. Form new squared area with current x and y as the coordinates of the centroid of this new area. Check if there are any common part with any static obstacle within this new squared area. If so, skip taken element and run new main loop turn.
   2.4. Add new squared area to the potential field.
   2.5. Increase current distance by 0.01.
   2.6. Add eight neighbour areas to the queue Q (for each area add these data: x and y coordinates, current distance and direction of exploration opposite to current); if direction is clockwise neighbours are: left, left-up, up, right-up, right, right-down, down, left-down; if direction is anti-clockwise neighbours are: left-down, down, right-down, right, right-up, up, left-up, left.
   2.7. Run new main loop turn.

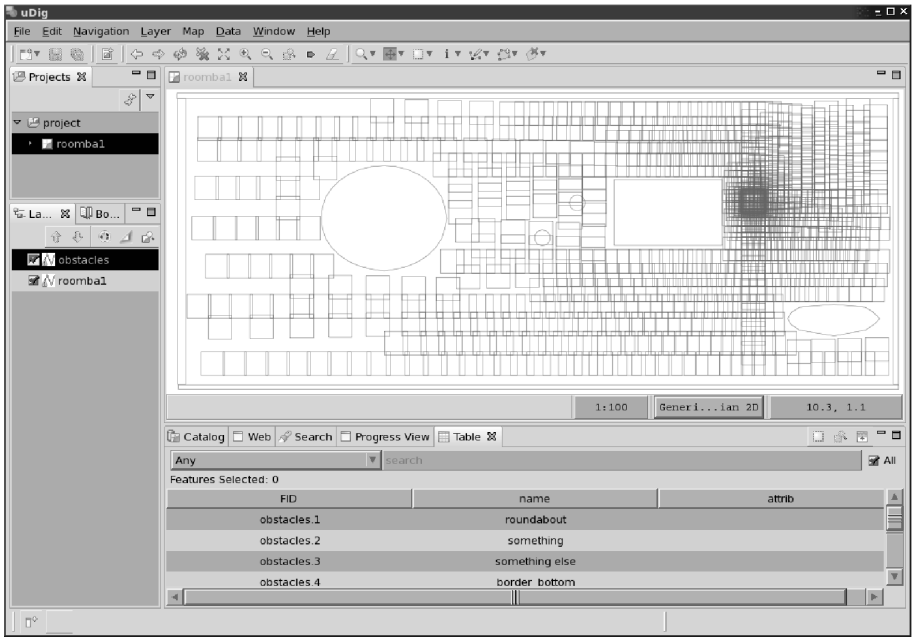

Fig. 2. Obstacles layer together with potential field layer (potential field generated for given goal is stored as ano-ther map layer). Observe increasing density towards the goal.

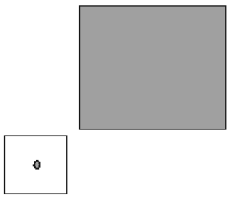

Fig. 3. Initial step of global potential field construction algorithm. New square area with the goal in the middle (denoted as solid dot) was added as the first element of the potential field.

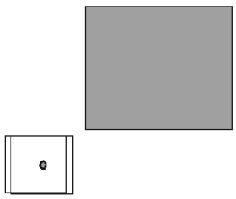

Fig. 4. First run of the main loop of the potential field construction algorithm. Element on the left was added within the distance 0.01 from the initial square. In this main loop turn, elements are added using clockwise direction: left, left-up, up, right-up, right, right-down, down, left-down.

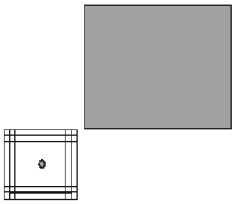

Fig. 5. First run of the main loop of the potential field construction algorithm is now completed. Eight squares were added to the potential field around initial square. These squares were also added to the queue Q so they will be operated in next main loop turns.

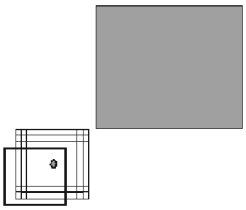

Fig. 6. First element from the beginning of the queue Q was taken - it is the square to the left of the initial area. In this main loop turn, elements are added using anti-clockwise direction: left-down, down, right-down, right, right-up, up, left-up, left. Newly added element (with thicker lines on the figure) was left-down to element taken from the begin-ning of the queue Q. Note that distance between the two elements is bigger comparing to previous main loop turn.

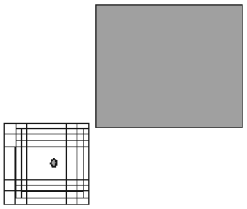

Fig. 7. Second run of the main loop of the potential field construction algorithm is now completed. Eight more squares were added to the potential field (and queue Q) around first square taken (and removed) from the queue (square with thicker lines on the figure).

The algorithm of searching for a path within given potential field is following:

PATH_SEARCH_ALGORITHM

1. Check if there is a field coverage in the current robot position. Also check if the robot is not at the goal which means no path needs to be planned. Choose a start square from the potential field: the nearest squared area from areas that have any common part with the robot. The centroid of this area will be the first waypoint.
2. Spin in the main loop until goal is found:
   2.1. From the set of squared areas that have any common part with previously choosen squared area choose the one which is the part of this area to the bigest degree (the smallest mereological distance). The centroid of this area will be the next waypoint.
   2.2. Run new main loop turn.

A robot should follow the path proposed by planner by going from one area centroid to another until the goal is reached.

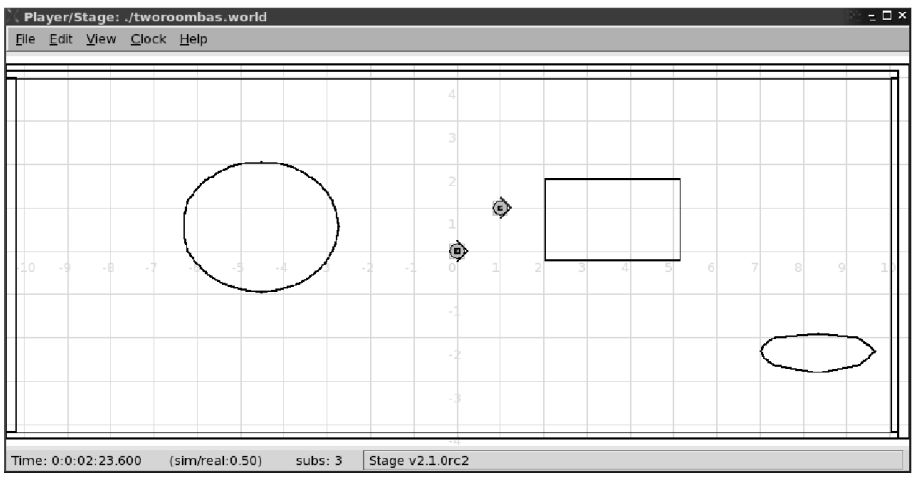

Fig. 8. Stage simulator in use - two iRobot Roomba robots inside of simulated world waiting for a goal to be set.

5. Implementation in Player/Stage robotics framework

Player/Stage is an Open-Source software framework designed for many UNIX-compatible platforms, widely used in robotics laboratories [17]. Main two parts are Player - message passing server (with bunch of drivers for many robotics devices, extendable by plugins) and Stage - a plugin for Players bunch of drivers which simulates existence of real robotics devices that operate in simulated 2D world. Player/Stage offers client-server architecture. Many clients can connect to one Player server, where clients are programs (robot controllers) written by a roboticist who can use Player client-side API. Player itself uses drivers to communicate with devices. In this activity it does not make distinction between real and simulated hardware. It gives roboticist means for testing programmed robot controller in both real and simulated world.

Among all Player drivers that communicate with devices (real or simulated), there are drivers not intended for controlling hardware and instead they offer many facilities for sensor data manipulation. For example: camera image compression, retro-reflective detection of cylindrical markers in laser scans, path planning. One of such drivers widely used during our experimentations is the PostGIS driver. It connects to PostgreSQL database [22] in order to obtain and/or update stored vector map layers.

PostGIS itself is an extension to the PostgreSQL object-relational database system which allows GIS (Geographics Information Systems) objects to be stored in the database [21]. It also offers new SQL functions for spatial reasoning. Maps which to be stored in SQL database can be created and edited by graphical tools like uDig or by C/C++ programs written using GEOS library of GIS functions. PostGIS, uDig and GEOS library are projects maintained by Refractions Research.

A map can have many named layers, for each layer a table in SQL database is created. We can assume that layer named *obstacles* is full of objects that a robot cannot walk through. Other layers can be created and we are using one of such layers to store potential field data (squared areas).

A roboticist can write a robot controller using Player client-side API, which obtains information about current situation through the *vectormap* interface. Additionally, to write such a program, PostgreSQL client-side API can be used to open direct connection to the database server.

A robot controller does not need to be implemented as the client-side program. Other way is to write a C++ code (a plugin) which will act as a new driver for the Players bunch of drivers. Player itself already contains one such driver called *wavefront*. This driver plays two roles: it acts as a simple path planner and it can controll a robot to walk through the path. It can use another built-in driver called *vfh* which implements Vector Field Histogram algorithm for obstacle avoidance [4].

Our mereological path planner is implemented as a plugin driver that can replace *wavefront* driver. However, it is not intended to use *vfh* driver for obstacle avoidance as the Vector Field Histogram algorithm needs any ranger sensor (laser, sonar, infrared) while Roomba robot default configuration does not have any such device. Instead, robot controller part of our driver monitors how long does it take to achieve next waypoint and if it is too long, it asks planner to replan the path from current robot position using already computed potential field. Additionally, we have added to our driver low-level behaviour that monitors state of bumpers and whenever bumpers are closed, the robot is going back-left to a new position from which the path is replanned.

As on the client-side, server-side drivers can use *vectormap* interface to obtain required map layer. Also direct connection to PostgreSQL database server can be opened. In our planner driver we have used ECPG API provided by PostgreSQL which enables to put SQL queries directly into the C/C++ code. To make our SQL queries more robust, we have stored our mereogeometry SQL functions on PostgreSQL server together with map database. These functions can be called using connection with database managed by ECPG infrastructure. Our mereological functions are processed on SQL database server side, results are sent back to the calling program (which means to our planner). This gives our planner ability to perform spatial reasoning based on rough mereology.

We have created our mereogeometry SQL predicates [13]. Rough mereological distance is defined as such:

```
CREATE FUNCTION meredist(object1 geometry, object2 geometry)
RETURNS DOUBLE PRECISION AS
$$
    SELECT min(degrees.degree) FROM
        ((SELECT
        ST_Area(ST_Intersection(extent($1), extent($2)))
            / ST_Area(extent($1))
            AS degree)
        UNION (SELECT
        ST_Area(ST_Intersection(extent($1), extent($2)))
            / ST_Area(extent($2))
            AS degree))
        AS degrees;
$$ LANGUAGE SQL STABLE;
```

Having mereological distance function we can derive nearness predicate:

```
CREATE FUNCTION merenear(obj geometry, o1 geometry, o2 geometry)
RETURNS BOOLEAN AS
$$
    SELECT meredist($1, $2) > meredist($3, $2)
$$ LANGUAGE SQL STABLE;
```

The equi-distance can be derived as such:

```
CREATE FUNCTION mereequ(obj geometry, o1 geometry, o2 geometry)
RETURNS BOOLEAN AS
$$
    SELECT (NOT merenear($1, $2, $3))
        AND (NOT merenear($1, $3, $2));
$$ LANGUAGE SQL STABLE;
```

Our implementation of the betweenness predicate makes use of a function that produces an object which is an extent of given two objects:

```
CREATE FUNCTION mereextent(object1 geometry, object2 geometry)
RETURNS geometry AS
$$
    SELECT GeomFromWKB(AsBinary(extent(objects.geom))) FROM
        ((SELECT $1 AS geom)
        UNION (SELECT $2 AS geom))
        AS objects;
$$ LANGUAGE SQL STABLE;
```

The betweenness predicate is defined as such:

```
CREATE FUNCTION merebetb(obj geometry, o1 geometry, o2 geometry)
RETURNS BOOLEAN AS
$$
    SELECT
        meredist($1, $2) = 1
        OR meredist($1, $3) = 1
        OR
            (meredist($1, $2) > 0
            AND meredist($1, $3) > 0
            AND meredist(mereextent($2, $3),
                mereextent(mereextent($1, $2), $3)) = 1);
$$ LANGUAGE SQL STABLE;
```

Using the betweenness predicate we can check if three objects form a pattern:

```
CREATE FUNCTION merepattern(object1 geometry, object2 geometry, object3 geometry)
RETURNS BOOLEAN AS
$$
    SELECT merebetb($3, $2, $1)
        OR merebetb($1, $3, $2)
        OR merebetb($2, $1, $3);
$$ LANGUAGE SQL STABLE;
```

Also having pattern predicate we can check if four objects form a line:

```
CREATE FUNCTION mereisline4(obj1 geometry, obj2 geometry, obj3 geometry, obj4 geometry)
RETURNS BOOLEAN AS
$$
    SELECT merepattern($1, $2, $3) AND merepattern($2, $3, $4);
$$ LANGUAGE SQL STABLE;
```

As we can realise, the path from current robot position to the goal is built from squared areas that form mereological line (as described by predicates above). We derived SQL aggregate function that can check if a given set of areas form a mereological line. This consists of one state function, one final function and one aggregate definition:

```
CREATE FUNCTION mereislinestate(statearray geometry[4], inputdata geometry)
RETURNS geometry[4] AS
$$
    SELECT ARRAY[$1[2], $1[3], $2, result.object] FROM (SELECT
        CASE
            WHEN $1[4] IS NOT NULL
                THEN $1[4]
            WHEN $1[3] IS NULL
                THEN NULL
            WHEN ($1[2] IS NULL) AND (meredist($1[3], $2) > 0)
                THEN NULL
            WHEN ($1[2] IS NULL) AND (meredist($1[3], $2) = 0)
                THEN $2
            WHEN ($1[1] IS NULL) AND merepattern($1[2], $1[3], $2)
                THEN NULL
            WHEN ($1[1] IS NULL) AND (NOT merepattern($1[2], $1[3], $2))
                THEN $2
            WHEN merepattern($1[1], $1[2], $1[3]) AND merepattern($1[2], $1[3], $2)
                THEN NULL
            ELSE $2
        END AS object) AS result;
$$ LANGUAGE SQL STABLE;
```

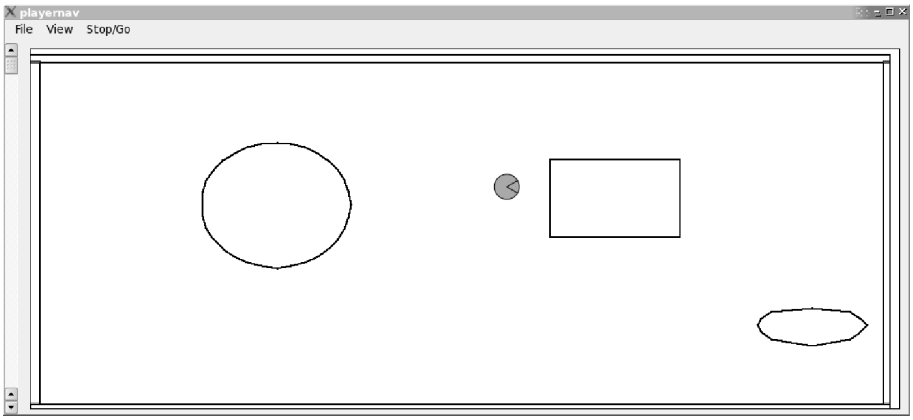

Fig. 9. Playernav in use - Roomba robot waits for the orders.

```
CREATE FUNCTION mereislinefinal(statearray geometry[4])
RETURNS BOOLEAN AS
$$
    SELECT ($1[4] IS NULL)
        AND ($1[3] IS NOT NULL)
        AND ($1[2] IS NOT NULL);
$$ LANGUAGE SQL STABLE;

CREATE AGGREGATE mereisline
(
    SFUNC = mereislinestate,
    BASETYPE = geometry,
    STYPE = geometry[],
    FINALFUNC = mereislinefinal,
    INITCOND = '{}'
);
```

One of client-side programs that come together with Player server is *playernav*. It shows a map of robot environment (with marked current robot position). We can in-dicate on the map goal position where robot should go. The *playernav* sends the goal to planner device working on given Player server instance. Then it asks the planner for current path which will be marked on the map.

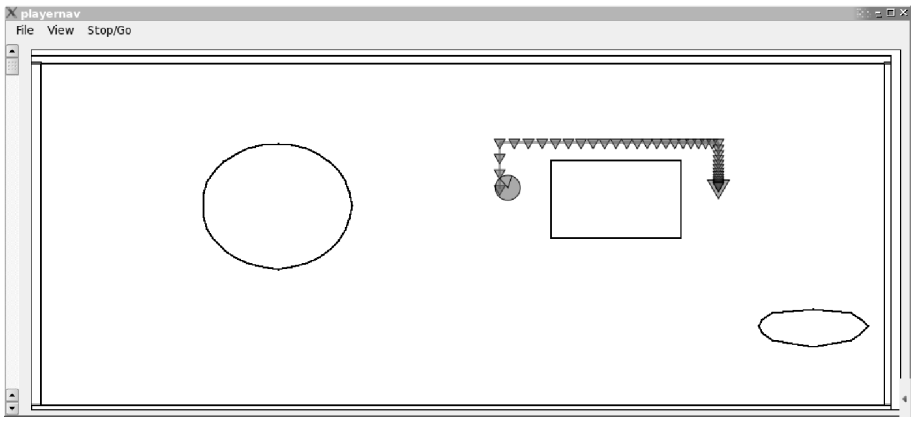

Fig. 10. Playernav in use - Roomba robot follows path to the target.

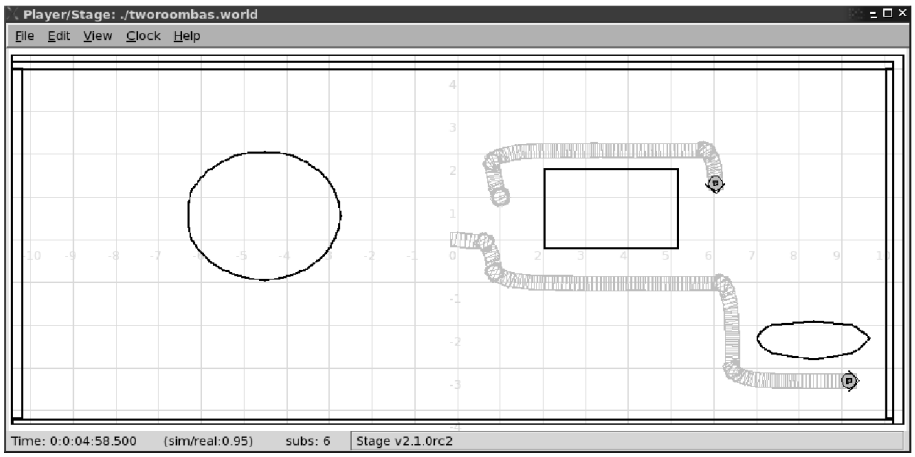

Fig. 11. Show trails is a nice option in Stage which can be used to track robot trajectory. Here we can see how two Roomba robots walked through planned paths to their targets.

6. Comparison with Player's built-in path planning device

As it was stated before, Player already provides its own path planning device called *wavefront*. This mechanism provides correct paths to given targets. Our method is implemented as a plugin for Player, which acts as a direct replacement for *wavefront* and provides correct paths too. It is worth to mention that our method gives possibility of observing intermediate stage of its routine: computed potential field used by planner can be viewed using uDig software. This gives possibility of additional tweaking of the algorithm. Each planner in certain circumstances must perform replanning. To do this *wavefront* must repeat whole planning routine. Using our method, only second stage of planning routine is done during replannig as potential field is computed only once (unless the database is updated with new obstacles). Searching for a path within already computed potential field is computationaly cheap as it is limited to database lookup operations (therefore speed of database communication is critical if our method is intented to be working fast).

7. Conclusions

Results of simulations show that this method gives satisfactory results. Further research is aimed at planning paths for teams of robots as well as for planning paths for prescribed robot formations.

AUTHOR

Paweł Ośmiałowski - Polish-Japanese Institute of Information Technology, Chair of Intelligent Robotic Systems, Koszykowa str. 86, 02-008 Warszawa, Poland.
E-mail: newchief@king.net.pl.

References

[1] Arkin R. C., *Behavior Based Robotics*, MIT Press: Cambridge MA, 1998.

[2] van Bentham J., *The Logic of Time*, Reidel: Dordrecht, 1983.

[3] Booch G., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley Publ., Menlo Park, 1994.

[4] Borenstein J., Koren Y., "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots", *IEEE Journal of Robotics and Automation*, vol. 7, no. 3, June 1991, pp. 278-288.

[5] Cohn A. G., "Calculi for qualitative spatial reasoning", in: J. Calmet, J. A. Campbell, J. Pfalzgraf (eds.), Artificial *Intelligence and Symbolic Mathematical Computation, Lecture Notes in Computer Science*, vol. 1138, Springer Verlag: Berlin, 1996, pp. 124-143.

[6] Cui Z., Cohn A. G., Randell D. A., "Qualitative and topological relationships", in: *Advances in Spatial Databases, Lecture Notes in Computer Science*, vol. 692, Springer Verlag: Berlin, 1993, pp. 296-315.

[7] Glasgow J., "A formalism for model-based spatial planning", in: A. U. Frank, W. Kuhn (eds.), *Spatial Information theory - A Theoretical Basis for GIS, Lecture Notes in Computer Science*, vol. 988, Springer Verlag, Berlin, 1995, pp. 501-518.

[8] Gotts N. M., Cohn A. G., "A mereological approach to representing spatial vagueness". In: *Working papers, the Ninth International Workshop on Qualitative Reasoning, QR'95*, 1995.

[9] Khatib O., "Real-time obstacle avoidance for manipulators and mobile robots". In*: Proceedings IEEE Intern. Conf. on Robotics and Automation*, St. Louis MO, pp. 500-505.

[10] Koren Y., Borenstein J., "Potential field methods and their inherent limitations for mobile robot navigation". In: *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento CA, 1991, pp. 1398-1404.

[11] Krogh B., *A generalized potential field approach to obstacle avoidance contro*", SME-I Technical paper MS84-484, Society of Manufacturing Engineers, Dearborn MI, 1984.

[12] Kuipers B., *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*, MIT Press: Cambridge MA, 1994.

[13] Ladanyi H., *SQL Unleashed*, Sams Publishing, 1997.

[14] de Laguna T., *Point, line, surface as sets of solids*, *J. Philosophy*, vol. 19, 1922, 449461.

[15] Latombe J.-C., *Robot Motion Planning*, Kluwer: Boston, 1991.

[16] Lesniewski S., "On the foundations of mathematics", *Topoi* 2, 1982, pp. 752.

[17] P. Osmialowski, "Player and Stage at PJIIT Robotics Laboratory", *Journal of Automation, Mobile Robotics and Intelligent Systems*, no. 2, 2007, pp. 21-28.

[18] Polkowski L., "An Approach to Granulation of Knowledge and Granular Computing Based on Rough Mereology: A Survey", in: V. Kreinovich, W. Pedrycz, A. Skowron (eds.), *Handbook of Granular Computing*, John Wiley and Sons: New York NY, 2008.

[19] Polkowski L., Osmialowski P., "Spatial reasoning with applications to mobile robotics". In: *Motion Planning for Mobile Robots: New Advances*, InTech, Vienna, 2008.

[20] Polkowski L., Skowron A., "Rough mereology in information systems with applications to qualitative spatial reasoning", *Fundamenta Informaticae*, vol. 43, issue 14, 2000, pp. 291-320.

[21] Ramsey P., *PostGIS Manual*. In: *postgis.pdf file downloaded from Refractions Research home page*, 2008.

[22] Stones R., Matthew N., *Beginning Databases with PostgreSQL*, Wrox Press, 2001.

[23] Tarski A., "Les fondements de la géométrie des corps". In: *Ksiega Pamiatkowa I Polskiego Zjazdu Matematycznego* (*Memorial Book of the Ist Polish Mathematical Congress*), a supplement to Annales de la Sociéte Polonaise de Mathématique, Kraków (Cracow), 1929, pp. 29-33.

[24] Tarski A., "What is elementary geometry?" In: L. Henkin, P. Suppes, A. Tarski (eds.), *The Axiomatic Method with Special Reference to Geometry and Physics, Studies in Logic and Foundations of Mathematics*, North-Holland: Amsterdam, 1959, pp. 16-29.

[25] Thrun S., Burgard W., Fox D., *Probabilistic Robotics*, MIT Press: Cambridge MA, 2005.

[26] Tribelhorn B., Dodds Z., "Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education". In: *2007 IEEE International Conference on Robotics and Automation, ICRA 2007*, 10th-14th April 2007, Roma, Italy, pp. 1393-1399.