Unified approach to qualitative motion planning in dynamic environments

Domen Šoberl and **Ivan Bratko**
Faculty of Computer and Information Science
Večna pot 113, Ljubljana, Slovenia
{domen.soberl, ivan.bratko}@fri.uni-lj.si

Abstract

Traditional motion planning methods rely on precise kinematic models to either compute the goal trajectory off-line, or to make on-line decisions based on current observations from a dynamic environment. With the increasing use of qualitative modeling in cognitive robotics, different planning approaches are needed to handle the lack of numerical data in manually constructed or autonomously learned qualitative domain theories. We propose a new motion planning algorithm that makes on-line decisions based on given qualitative domain description to reach a goal state. Decisions are stated in the form of simple qualitative actions that can easily be interpreted by robot's controller and transformed to a numerical output. We demonstrate its use on three classical problems: pursuing, obstacle avoidance and object pushing.

Introduction

Motion planning techniques for robotic systems in closed and controlled environment have become very efficient in recent years. Methods such as *Probabilistic RoadMap* (PRM) (Kavraki et al. 1996) or *Rapidly-exploring random tree* (RRT) (LaValle 2006) can produce a detailed motion plan, if precise mathematical model of the system's dynamics is provided. Such off-line planning is often unsuitable in real-world scenarios where complete and robust theories are rare, while environment is unpredictably changing, which is especially the case when human interaction is present.

To deal with challenges of motion planning in dynamic environments, reactive planners were proposed even in the early beginnings of motion planning, with emphasis on path construction and obstacle avoidance using a car-like vehicle (Fraichard, Hassoun, and Laugier 1991). Later works employ adaptations of certain off-line motion planning algorithms for dynamic plan modification. In (Leven and Hutchinson 2002) the PRM algorithm is used in two stages. A roadmap that corresponds to an obstacle-free environment is first built off-line and later dynamically updated with obstacle information by an on-line planner. A partial replanning with RRT is possible during the execution of the plan by recomputing only the necessary tree branches (Ferguson, Kalra, and Stentz 2006).

For a planner to predict exact future configurations that follow certain actions, a precise kinematic model of the system is needed. With the increasing use of qualitative modeling in cognitive robotics during the last few years, new challenges in automated planning emerged due to the lack of numerical information in qualitative domain theories. One of the main reasons to prefer qualitative over traditional numerical modeling, especially in the area of autonomous concept discovery, is its tendency to capture more general relations and express meaningful concepts which can significantly simplify the agent's theory (Bratko 2011). It has been demonstrated (Troha and Bratko 2011) that a wheeled robot can learn qualitative physics of pushing a rectangular box by experimentation. We later showed how such models can be used by a robot to plan the pushing of arbitrary convex polygonal objects (Šoberl, Žabkar, and Bratko 2015). However, our planner was specialized for planning actions of pushing and possibilities for a more general solution still needed to be addressed.

In this paper we propose a general motion planning algorithm together with a new domain description language which allows domain relations to be stated in the form of *monotonic qualitative constraints* (e.g. (Bratko and Šuc 2003)). We demonstrate the algorithm on three classical problems: pursuing, obstacle avoidance and object pushing. Our work differs from *symbolic qualitative planning* (Wiley, Sammut, and Bratko 2014), where qualitative plans are elaborate and used as a basis for further numerical learning. Our planner produces more basic qualitative actions, interpreted directly by the robot's controlling mechanism, and is therefore suitable for more straightforward tasks.

The rest of the paper is structured as follows. In the following section we give a general description of our planning mechanism and define the notion of robotic domain and qualitative action as used by our planner. Next we introduce our domain description language and describe its individual elements. We continue with in-depth analysis on how the planner interprets the given domain description to produce appropriate actions. We then describe three different experiments that we conducted in a simulated environment and present the results. Finally, we conclude and discuss our future work.

Our planning approach

We presume that the configuration space of the robot is continuous and connected. Let $x_1, \ldots x_n$ denote domain attributes where $x_1, \ldots, x_k$ are directly controllable, meaning

that the control mechanism is able to increase or decrease their value, and so they represent output signals. All other attributes are controllable indirectly, using relations defined by the given qualitative model. A relation $x_3 = M^{+-}(x_1, x_2)$ states that x_3 monotonically increases in x_1 and decreases in x_2. If both x_1 and x_2 increase or decrease, we deem the direction of x_3 inconclusive. We define qualitative action as a mapping of controllable attributes $x_1, \ldots, x_k$ to $\{+, -, 0\}$ and write $a = (x_1^+, x_2^-, x_3^0)$ to denote an action a that maps $x_1 \mapsto +$, $x_2 \mapsto -$ and $x_3 \mapsto 0$, meaning that the controller should increase the value of x_1, decrease the value of x_2 and keep the current value of x_3. It is then up to the controller to choose the numerical step. In the past we achieved satisfying results with simulated and real robots using the trivial mapping: $x_i^+ \mapsto \max(x_i)$, $x_i^- \mapsto \min(x_i)$ and $x_i^0 \mapsto x_i$. Say such a signal represents the output power to a motor. Setting it to the highest value takes some time for the motor to actually reach the highest speed. Providing a sufficiently high refresh frequency, observed attributes rarely reach their extremes, as the planner tends to guide them to a certain value. This differs from the classical PID control principle in the fact that with PID controllers the target value of the output signal is known in advance, whereas in our case it is the task of the planner to make such decisions.

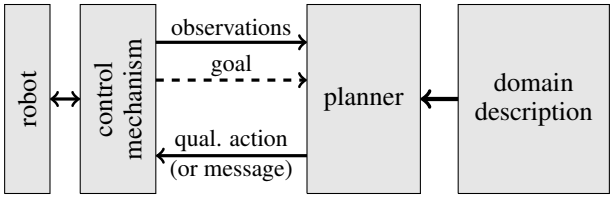


Figure 1: Reactive qualitative planning mechanism. Goal can be stated dynamically or set statically as a part of domain description.

Communication between the planner and the robot's control mechanism is shown in Fig. 1. The controller starts a new communication cycle by updating the values obtained from the robot's sensing system. The planner uses values from the previous cycle to record velocity (the signed speed) of each attribute. Those velocities are needed internally by the planner, but being regarded as attributes they can be used as a part of domain description by adding the apostrophe character `'` after an identifier (e.g. `x'` to denote the velocity $\dot{x}$). The goal state can be specified statically as a part of domain description, or given dynamically, together with input attribute values. A Goal condition is given as a set of simple equations $\{x_i = g_i\}$, where g_i is the goal value of attribute x_i and can be a constant or a numerical expression. If the robot should pursue an object, the goal can be specified as equality of their respective coordinates. If more than one goal is given (dynamically, statically or mixed), the planner chooses the most promising one for that cycle. The planner can implicitly set additional goals to satisfy given numerical constraints or avoid their violation. If the robot should avoid an obstacle, a constraint $D > 0$, where D is the distance between the robot and the object, should imply actions that lead away from the obstacle, especially if no other goal is given.

As soon as input values are set, the planner responds with the next action to be performed, or with one of the following messages:

- *No goal.* No goal to reach, neither explicit or implicit. This happens if no goal has been specified or when expressions that are part of the goal definition failed to be evaluated. This is usually the result of a poor domain definition or insufficient input data.
- *No solution.* Constraints have been violated. Depending on the design of the problem, this can represent an unwanted situation (e.g. a configuration where a reset is needed) or a part of the planning process. We demonstrate the latter case in our third experiment - pushing an object - where a constraint on action score fires *no solution*, when no available action is good enough. We then reposition the robot to a more favorable pushing position.
- *Goal reached.* According to the current speed of the system, goal attributes are close enough to their goal configurations. This means that for each goal attribute the distance between its current and goal value is smaller than the distance it can make between two communication cycles. However, the controller always has the liberty to impose its own conditions and end the process on its own terms.

Returning an answer the communication cycle ends. It is not required to invoke cycles with a constant frequency, but it is helpful for the planner to properly evaluate the tolerance of the goal state.

Domain description language

To describe the problem domain, qualitative model alone is usually insufficient. Besides defining actions and specifying goal conditions, additional numerical constraints often need to be set. Those define undesirable system configurations and make the configuration space non-convex. We propose a new domain description language, suitable to our planning approach.

A domain is described by a set of statements $\{S_1, S_2, \ldots, S_n\}$, where each statement can evaluate as *true*, *false* or *inconclusive*. Statements can be interdependent, and since the language is declarative, it is the task of the planner to resolve the dependency of their evaluation. If one of the statements fails (is evaluated as *false*), the *no solution* answer is triggered. A statement is evaluated as *inconclusive* if insufficient data is provided, in which case it is ignored. This may also lead to inconclusive goal states and the *no goal* answer in the case no goal state is conclusive. There are seven types of statements: boolean expression, attribute range, numerical equation or inequation, qualitative relation, conditional statement, definition of action and specification of a goal state.

Boolean expressions

Statements can be combined into boolean expressions using the standard logical operators. The use of primitives

true and *false* is also permitted. If one or more statements in the expression evaluate as *inconclusive*, the expression evaluates to *inconclusive* only if the result cannot be derived from other values. For instance, boolean expression *true* $\vee$ *inconclusive* evaluates as *true*, while *true* $\wedge$ *inconclusive* evaluates as *inconclusive*.

Attributes and their ranges

Attributes are used as variables in classical programming languages, but their nature is significantly different. All statements must agree on a single value of an attribute for the current cycle. Statements that propose different values to the same attribute, fail. If the value of an attribute is not set, statements that use it are evaluated as *inconclusive*. Besides the computed (or observed) numerical value, a predicted qualitative[1] value can also be assigned to an attribute. However, this can only be done implicitly by the planner.

An attribute can be bound to a range. There are three distinct ranges:

- *An interval*. Open, closed or half-closed intervals can be stated. Statement `x in I[a, b)` binds the attribute x to half-closed interval $[a, b)$ where a and b are arbitrary numerical expressions. The statement succeeds if no statement evaluates x to a value outside the given boundaries. This also sets an implicit goal for the planner to avoid both extremes. Infinity keywords `-inf` and `inf` can also be used instead of a or b respectively, making the planner avoid only one extreme.
- *A sphere* S^1. Statement `x in S[a, b)` makes the attribute x circular and normalizes its value to the given range. This statement always succeeds, sets no implicit goal, but allows the planner to choose between two directions to reach a desired state. A typical use of circular attributes is to specify rotations, e.g. orientation of the robot within $[0°, 360°)$.
- *A qualitative range*. Allows the user to limit the predicted value. Statement `x in Q[0+]` results in elimination of all actions for which the planner deduces a negative predicted value.

Numerical equations and inequations

There is no clear distinction between comparison and assignment. If all attributes contained in a statement evaluate to some value, the statement either succeeds or fails. Simple equations (a single attribute on either the left or the right side) try to set the value so that the statement succeeds. This can happen only if all other statements agree on the same value. Simple inequations set an implicit goal to the single attribute to avoid the violation of the rule. Complex comparisons remain inconclusive if one or more attributes cannot be evaluated. Note that a statement of the form `x = x + 1`, so frequent in classical programming languages, definitely fails here.

[1]Currently, our planner can only predict qualitatively. Incorporating some form of numerical machine learning should enable the planner to also make numerical predictions.

Qualitative relations

Qualitative relations between attributes are the basis for the planner to predict the outcome of an individual action. A domain is well defined when one can track relations from controllable to goal attributes. Relations are stated in the form of equations, with the M-notation on one side, and a single attribute on the other. As in the case of numerical equations, this can be interpreted as both, an assignment and comparison.

Consider the relation $\dot{D} = M^{-+}(v, \omega)$ from our second example, which states that (under certain conditions) the speed of distancing the robot from the obstacle increases by monotonically decreasing its forward speed v and increasing its rotational speed ω. Recall that every attribute holds two values, one numerical and one qualitative (its predicted future dynamic), of which either one can be set or unset. If the planner can derive qualitative values of all three attributes, $\dot{D}$, v and ω, and those values agree on the stated relation, the statement will succeed. If the qualitative value of $\dot{D}$ cannot be derived from other stated relations, the statement will set it so that the statement succeeds, while its numerical value will remain intact. If two or more independent attributes contradict, say we have v^+ and ω^+, the above statement is inconclusive. It sets no values and has no impact on the final outcome.

Conditional statements

A conditional statement is an implication of the form $\{G_1, \ldots, G_m\} \Rightarrow \{S_1, \ldots, S_n\}$, where statements G_i represent the guard. Only if all guarding statements G_i succeed, the implied statements S_j are considered a part of domain definition. An inconclusive guarding statement makes the guard and thus the whole statement inconclusive. By the rule of implication, a failed guard evaluates the whole statement as *true*, while omitting the implied statements S_i from the rest of definition.

If a conditional statement succeeds, its guarding statements G_i also becomes a part of domain definition. Consider a statement of the form $\{x = 1\} \Rightarrow \{S_0, \ldots, S_n\}$ whose guard succeeds by setting the attribute x to 1. The equality $x = 1$ is always considered by statements $S_0, \ldots, S_n$, but becomes a globally valid assertion only if the whole implication succeeds.

Conditional statements can be used to describe qualitative models as sets of qualitative relations that hold under given conditions, e.g. the implication $\{\varphi > 90°\} \Rightarrow \{\dot{D} = M^{-+}(v, \omega)\}$ states that the right-hand relation holds only when $\varphi > 90°$. Models obtained by programs such as QUIN (Bratko and Šuc 2003) or Padé (Žabkar, Bratko, and Demšar 2007) can easily be rewritten using such a form.

Actions

Classes of legal actions are defined by statements of the form `action`$(x_1, x_2, \ldots, x_n)$, giving the planner the freedom to choose among 3^n possible actions. If more than one such statement is used, the planner has the option to choose among different types of actions. Specifying no actions is considered a poor domain description and results in the *no*

solution message, unless a goal state has been reached initially.

Each action is evaluated and assigned a score. A higher value indicates an action that will have a more desirable effect in favor of reaching the goal state. A positively scored action is predicted to advance the configuration closer to the goal state while negatively scored action should result in moving away from the goal. A zero value could mean an action without an effect or leading to a state equally distant to the goal state. The planner evaluates an action by applying it to the current attribute state and deducing its qualitative effect through stated attribute relations, tracked down to a goal statement. If qualitative deduction reaches a valid goal statement, the score is computed and assigned to the action, otherwise the action is discarded. In the case of multiple goals, the closest goal is selected according to Manhattan distance over all attributes, normalized by their speeds:

$$\text{distance}(G) = \sum_{i=1}^{n} \frac{|g_i - x_i|}{|\dot{x_i}|} \tag{1}$$

where $G = \{g_1, \ldots, g_n\}$ are explicit goal values of attributes $x_1, \ldots, x_n$. When all actions are evaluated, the planner returns the one with the highest value, or triggers *no solution* if all actions were discarded.

It is possible to state additional constraints on action scores. Statement `s = action(`$x_1, x_2, \ldots, x_n$`)` will compare / assign the highest score to attribute s, which can further be used in other numerical statements. In our third experiment we used this feature to trigger *no solution* when no positively evaluated action was found, which was achieved by numerical constraint $s > 0$.

Goals

A goal condition is defined as a set of simple equations $\{x_i = g_i\}$, where x_i is an attribute and g_i a numerical constant or expression. To specify an explicit goal we use statements of the form `goal(`$x_1 \to a, x_2 \to b, \ldots$`)`. The *arrow* symbol makes a clearer distinction between the attribute and its goal value and also denotes an operation a bit different from the usual numerical comparison. The goal statement not only compares numerical values, but also assesses its distance and predicts future dynamics based on currently set qualitative values of attributes. For attributes with unset predictions, the planner will try to make predictions based on the direction of attribute's speed. To each explicitly defined goal, implicit goals may be added by the planner internally, to avoid possible constraint violations. An implicit goal is stated as a set of pairs $\{(x_i, I_i)\}$, where I_i is the interval to which the attribute x_i is bound. Attributes that are used in explicit goals are not used in implicit goals.

The goal statement always succeeds unless some of the values cannot be deduced, in which case it remains inconclusive. When successful, it outputs the score which is assigned to the action that is currently being evaluated. We discuss further details on score evaluation in the following section.

The planning algorithm

During each cycle, the planner follows the following algorithm:

1. Check if input values conflict with any of the statements given in domain description. If so, return *no solution*.
2. Locate valid *action* statements and generate the list of possible actions.
3. For each action repeat:
   - 3.1. Set predicted values of attributes as specified by the action (e.g. action (v^+, w^-) sets qualitative part of v and w to $+$ and $-$, respectively). Check if set values conflict with any of the statements. If so, discard the action and return to step 3.
   - 3.2. Following valid qualitative relations, deduce predictions for all possible attributes.
   - 3.3. Locate valid *goal* statements. If no goal is found, discard the action and return to step 3.
   - 3.4. Compute the score of each goal and assign the highest value to the action. If conditions of some goal are met, return *goal reached*.
   - 3.5. If the assigned score conflicts with numerical statements, discard the action.
4. Return the action with the highest score. If no action is left, return *no solution*.

We say *valid* actions, goals and relations to emphasize the fact that any statement can be conditioned using conditional statements. Putting an action or a goal under a guard makes it possible to divide the planning problem into separate tasks or phases.

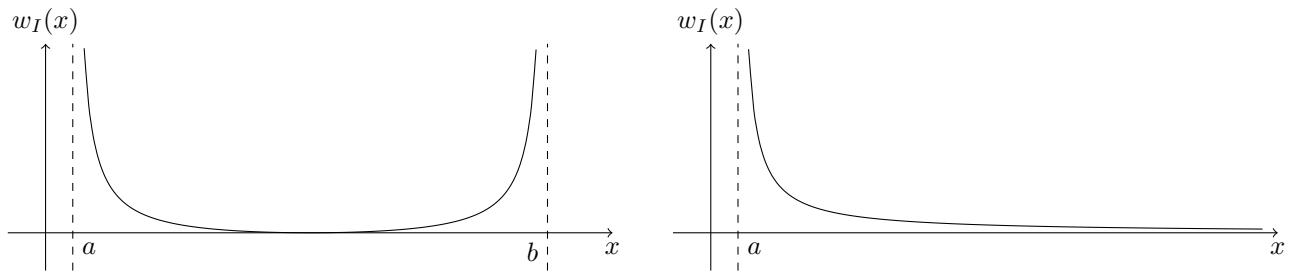
Consider an action a that is being evaluated under a goal $(x_1 \to g_1, \ldots, x_k \to g_k \mid (x_{k+1}, I_{k+1}), \ldots, (x_n, I_n))$, where g_i are explicitly defined goal values and I_i intervals, assigned to non-goal attributes $x_{i>k}$, and so comprise an implicit goal. Assume that all numerical values of x_i and g_i are set, and that all qualitative predictions of x_i were deduced under action a, making the goal statement successful. The score of action a is set using the following function:

$$\begin{aligned} \text{score}(a) = &\sum_{i=1}^{k} p(x_i) \cdot \frac{g_i - x_i}{|\dot{x_i}|} + \\ &+ \sum_{i=k+1}^{n} p(x_i) \cdot w_I(x_i) \cdot |\dot{x_i}| \end{aligned} \tag{2}$$

For explicitly defined goals, each attribute contributes a weight proportional to the distance from its goal value and normalized by its speed. The greater the distance, the more important the attribute. Function $p(x_i)$ maps the qualitative value $+$, $-$ or 0 of x_i to its respective numerical representation $+1$, -1 or 0. This results in contributing a negative weight if the attribute is predicted to move away from its goal value.

Attributes that are part of an implicitly defined goal, thus bound to some interval, contribute their weights according to the w function, defined as

$$w_I(x) = \cos\left(\frac{\pi}{b-a} \cdot \left(x - \frac{a+b}{2}\right)\right)^{-1} - 1 \tag{3}$$



(a) Function w in the case $x \in I = (a, b)$.

(b) Function w in the case $x \in I = (a, \infty)$.

Figure 2: The weight of a bounded attribute is based on the w function.

for bounded intervals $I = (a, b)$, and

$$w_I(x) = \begin{cases} (x-a)^{-1}, & I = (a, \infty) \\ (b-x)^{-1}, & I = (-\infty, b) \end{cases} \quad (4)$$

for half-bounded intervals. The same function is used for closed and half-closed intervals. When an attribute is fully bounded, the planner will tend to keep its value close to the midpoint between both extremes. The contributed weight will stay relatively low for the major part of the interval, but will start to rise very rapidly when the attribute gets close to its extreme, as shown in Fig. 2. Weight is then multiplied by the speed of the attribute (see equation (2)), so faster attributes contribute more to the final score, as they are in higher danger to hit their forbidden zone quicker.

Experiments

To asses the performance of our planner we conducted three different experiments using a simple two-wheeled robot vehicle. Experiments were done in a simulator, assuming an overhead camera and object recognition system as sensory input. The sensory system recorded absolute location and orientation of single objects. All data were passed directly to the planner without any preprocessing. The refresh rate was 25 Hz and the planner was invoked whenever a change in configuration of objects was observed. Otherwise the values decided by the last cycle were held on the output.

The controller was able to interpret qualitative actions in the form (v^{+-0}, ω^{+-0}), where v is translational and ω angular velocity of the robot, being positive in the CCW direction. Using an independent sensory feedback, controller was able to adjust and maintain given speeds, but when instructed to increase / decrease one of them, it aimed for the maximum / minimum achievable value. Having a system with limited output capabilities, some action might not always be executable. Scenarios such as moving at a full speed forward while receiving instruction to increase ω and keeping v unchanged need a special consideration. In such cases our robot first lowered the forward speed by half, maintaining the same ratio between the left and the right wheel, and then proceeded with the intended action execution.

Pursuing objects

The goal of this task is to follow and eventually catch an animate object by choosing the appropriate translational v and angular ω speed of the robot at every step of the process. For this experiment we conducted no learning phase and designed the model manually. This way we demonstrated what we believe is an advantage of qualitative modeling. We were able to describe the domain intuitively, as we understood it, in a concise non-algorithmic way, and the robot behaved as we intended it.

Our reasoning was the following. We understand pursuing as the process of decreasing the distance to the target while orienting towards it. We believe the most efficient way to describe this domain is to use an egocentric approach (the robot sees itself as the center of the world). Let D denote the distance to the target and φ its angular offset, as depicted in Fig. 3. These need to be derived from the absolute robot position (x_0, y_0), its orientation θ, and position of the target (x, y). This can be done using the following equations:

$$\begin{aligned} D &= \sqrt{(x-x_0)^2 + (y-y_0)^2} \\ \varphi &= \text{atan2}(y-y_0, x-x_0) - \theta \end{aligned} \quad (5)$$

We understand that increasing or decreasing the speed of the robot does not directly affect the distance D, but rather its speed $\dot{D}$. In relation to that, we recognize two distinct qualitative states: *the target is in front of the robot* ($|\varphi| \leq 90$), and *the target is behind the robot* ($|\varphi| > 90$). We also understand that the speed of φ decreases by increasing angular speed and vice versa. The goal should be reached by orientating towards the target ($\varphi \to 0$) and decrease the distance ($D \to 0$).

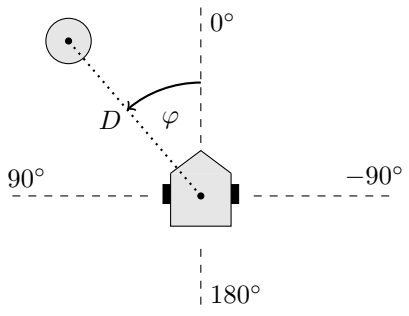


Figure 3: Relation of the robot to a target.

Listing 1 shows our domain description. Note that our

implementation of the language demands statements to be terminated by a semicolon, and that we use the # symbol for comments. Single-statement guards and implications need not be enclosed in curly braces.

Listing 1 Pursuing an object

```
# Input values:
# x0, y0, theta - robot configuration
# x, y          - target position

phi in S[-180, 180); # phi is circular

# Egocentric values
D = sqrt((x - x0)^2 + (y - y0)^2);
phi = atan2(y - y0, x - x0) - theta;

# Qualitative model
abs(phi) <= 90; => D' = M-(v);
abs(phi) > 90;  => D' = M+(v);
phi' = M-(w);

action(v, w);
goal(D -> 0, phi -> 0);
```

A trajectory made by our robot during one of the trials is shown in Fig. 4. We initially positioned the robot facing backwards to its target. The pursuing began as soon as the target started moving from the left to the right with a constant speed. The robot first made a backward turn by $90°$ and so made a transition from qualitative state $|\varphi| > 90°$ to $|\varphi| \leq 90°$. We observed that by choosing action (v^-, ω^-), the planner was able to simultaneously utilize rules $\dot{D} = M^+(v)$ and $\dot{\varphi} = M^-(\omega)$, and therefore satisfy both goal directions, $D \to 0$ and $\varphi \to 0$. We find such a maneuver visually very intuitive from a human perspective. Being slightly faster than the target, the robot managed to stay in the $|\varphi| \leq 90°$ qualitative state until the end of the task, alternating between actions (v^+, ω^+) and (v^+, ω^-) to simultaneously shorten the distance and regulate its orientation towards the target.

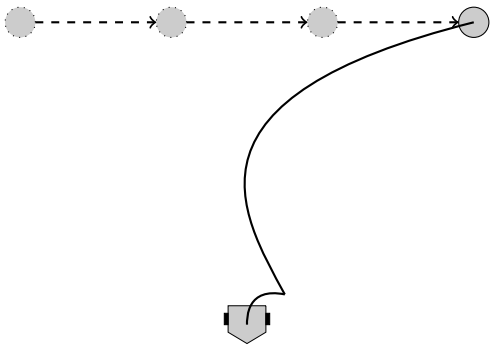


Figure 4: Trajectory made by the robot while pursuing a moving target.

It is possible to introduce more than one target and let the planner choose the closest one. We set up an additional scenario with one stationary and one moving target that we were able to move interactively. The behavior of the robot was as expected. Initially, the robot went for the stationary target, which was being positioned closer to it. Before hitting it, we moved the secondary target closer and distracted the robot away from its primary target. As soon as we moved our target out of reach, the robot headed back to its primary goal. To achieve such behavior, we had to separately describe dynamics of both targets, as shown in Listing 2, although both descriptions are identical.

Listing 2 Pursuing two objects

```
# Input values:
# x0, y0, theta - robot configuration
# x1, y1        - target 1 position
# x2, y2        - target 2 position

phi1 in S[-180, 180); # phi1 is circular
phi2 in S[-180, 180); # phi2 is circular

# Egocentric values of target 1
D1 = sqrt((x1 - x0)^2 + (y1 - y0)^2);
phi1 = atan2(y1 - y0, x1 - x0) - theta;

# Egocentric values of target 2
D2 = sqrt((x2 - x0)^2 + (y2 - y0)^2);
phi2 = atan2(y2 - y0, x2 - x0) - theta;

# Qualitative model 1
abs(phi1) <= 90; => D1' = M-(v);
abs(phi1) > 90;  => D1' = M+(v);
phi1' = M-(w);

# Qualitative model 2
abs(phi2) <= 90; => D2' = M-(v);
abs(phi2) > 90;  => D2' = M+(v);
phi2' = M-(w);

action(v, w);
goal(D1 -> 0, phi1 -> 0);
goal(D2 -> 0, phi2 -> 0);
```

Avoiding obstacles

The general idea to implementing obstacle avoidance is to introduce additional constraint to the pursuing scenario, making the configuration space non-convex. The tendency to avoid an obstacle is therefore the tendency to avoid violating constraint $D > 0$, where D is the distance from the border of the obstacle. This represents an implicit goal and we found it very efficient to construct the qualitative model of avoidance as shown in Fig. 5. We identify three qualitative states:

- *The obstacle is front left* ($0 \leq \varphi < 90$). The robot can increase $\dot{D}$ by decreasing v and increasing angular speed towards CW.
- *The obstacle is front right* ($-90 < \varphi < 0$). The robot can increase $\dot{D}$ by decreasing v and increasing angular speed towards CCW.
- *The obstacle is behind* ($\varphi \geq 90 \vee \varphi \leq -90$). The robot can increase $\dot{D}$ by increasing v.

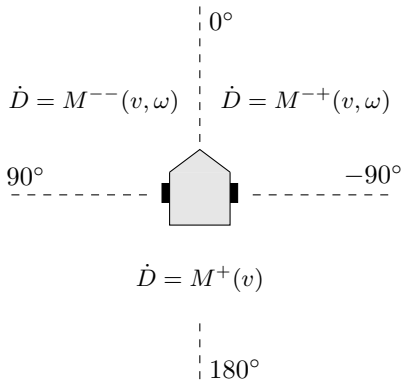


Figure 5: The qualitative model of avoidance with three qualitative states.

Listing 3 Avoiding an obstacle

```
# Input values:
# x0, y0, theta - robot configuration
# x1, y1        - goal position
# x2, y2        - obstacle position
# r             - obstacle radius

phi1 in S[-180, 180); # phi1 is circular
phi2 in S[-180, 180); # phi2 is circular
D2 > 0; # obstacle constraint

# Egocentric values of goal
D1 = sqrt((x1 - x0)^2 + (y1 - y0)^2);
phi1 = atan2(y1 - y0, x1 - x0) - theta;

# Egocentric values of obstacle
D2 = sqrt((x2 - x0)^2 + (y2 - y0)^2) - r;
phi2 = atan2(y2 - y0, x2 - x0) - theta;

# Qualitative model to pursue
abs(phi1) <= 90; => D1' = M-(v);
abs(phi1) > 90;  => D1' = M+(v);
phi1' = M+-(w);

# Qualitative model to avoid
phi2 in I[0, 90);     => D2' = M--(v, w);
phi2 in I(-90, 0);    => D2' = M-+(v, w);
phi2 in I[90, 180) or
phi2 in I[-180, -90]; => D2' = M+(v);
phi2' = M-(w);

action(v, w);
goal(D1 -> 0, phi1 -> 0);
```

Domain description shown in Listing 3 is very similar to description of the two-target pursuing domain, replacing the second qualitative model of pursuing with the model of avoidance, and the second explicit goal with constraint `D2 > 0`. Trajectories made by two different scenarios are shown in Fig. 6. The first setting involved a stationary target, placed straight ahead of the robot but behind an obstacle. Because $\varphi = 0°$ falls into the first qualitative state, the robot chose to avoid the obstacle by its right side. The second setting involved a moving target, circling with approximately the same speed as the robot, about one third of the circle in front. The robot made a circular trajectory trying to catch the target.

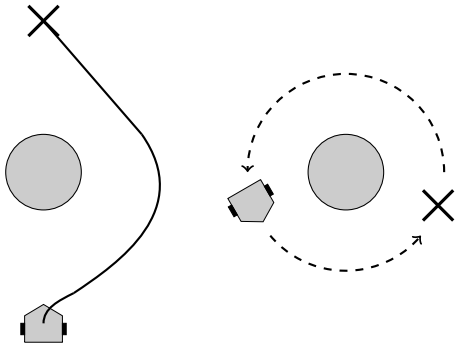


Figure 6: Trajectory made by the robot avoiding an obstacle while pursuing a stationary target (left) and chasing a moving target around the obstacle (right).

Pushing objects

Using our new planner and domain description language we were able to reproduce experiments described in (Šoberl, Žabkar, and Bratko 2015). This way we showed that this planner is at least as powerful, but more universal than our previous planning methods. We used the same qualitative model of pushing, which was learned by autonomous robotic experimentation. Domain attributes (depicted in Fig. 7) are the following: position of the object (x, y), orientation of the object β, goal position (x_g, y_g), goal orientation γ, orientation of the robot θ, the point of contact $\tau \in [-1, 1]$ and the angle of pushing $\varphi \in [-30°, 30°]$. The egocentric approach is used, making the above values relative to the robot's position and orientation.

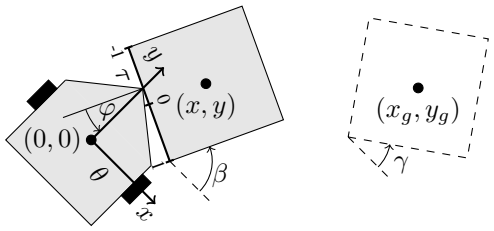


Figure 7: Attributes of the pushing domain.

Qualitative relations derived in the original work are the following:

$$\begin{aligned} \dot{y} &= M^{+}(v) & \dot{\varphi} &= M^{+}(\omega) \\ \dot{x} &= M^{--}(\tau, \omega) & \dot{\tau} &= M^{--}(\omega, \varphi) \\ \dot{\beta} &= M^{+--}(\tau, \varphi, \omega) & \dot{\theta} &= M^{+}(\omega) \end{aligned} \tag{6}$$

An example trajectory is shown in Fig. 8. A rectangular box is placed 1×1 meter from its goal location and rotated by $180°$. The goal is to match the position and orientation

of the box with that of the goal. It can be seen from the form of the trajectory that the robot is trying to satisfy both goal conditions simultaneously, until none of the possible actions works in favor of the goal directions. We make the planner discard *non-positive* actions by adding the constraint:

```
score = action(v, w);
score > 0;
```

We can see the trajectory being composed of 4 smooth curves. Their joints are the points where no action was scored above 0, and therefore the *no solution* message was received, meaning that the robot had to reposition in order to continue towards the goal. We used a separate solution to reposition the robot to an exact initial position, however, our planner did make the decision about which initial position is best. Whenever such a decision had to be made, we computed attribute values for all possible initial states and sent each initial state to the planner. For each setting we then obtained the best possible action together with its score. The state permitting the highest evaluated action was then selected as the next initial state.

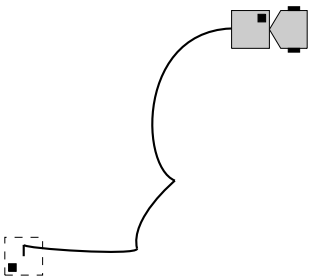


Figure 8: Trajectory made by a rectangular box being pushed to a goal configuration.

Conclusion

We have shown that qualitative models in the form of qualitative monotonic constraints contain enough information to allow simple motion planning without the need for additional numerical learning. We introduced a new qualitative planning method that can handle basic motion planning problems and proposed a new domain description language that allows concise non-algorithmic description of robotic domains using qualitative relations and additional numerical constraints. We demonstrated the intuitiveness of qualitative modeling in robotic planning and its ability to produce desired results without the need of doing any precise numerical measurements or modeling to describe the domain. We believe this way the robot exhibits similar behavior to a living being making fast instinctive decisions as it moves through an unfamiliar terrain in pursuit of some goal. However, at this point of research, our planning method is still somewhat shortsighted and unable to learn from its past mistakes or successes. In a few occasions we managed to bring the robot to a dead loop, alternating between two qualitative states as it was trying to reach two equally distant goals, oblivious of its past states and decisions. Only the fact of slight randomness due to certain sensory noise and communication delays eventually brought the robot out of a self-made trap. However, such situations were rare and we had to make extra effort to invoke them.

The final form of the language is still under our consideration and we shortly plan to add some extra elements. In our first experiment we had to duplicate the model of pursuing to introduce the second target. This problem could be tackled by introducing vector-like structures, combining attributes of the same type that belong to different objects, e.g. `D = [D1, D2, ...]` to combine distances to multiple objects. A single qualitative or numerical constraint would then hold for all attributes within that vector, e.g. stating only `D = M-(v)` instead of a separate statement for each object.

References

Bratko, I., and Šuc, D. 2003. Learning qualitative models. *AI magazine* 24(4):107–119.

Bratko, I. 2011. Autonomous discovery of abstract concepts by a robot. In *Proc. ICANNGA*, volume 1, 1–11. Springer Lecture Notes.

Ferguson, D.; Kalra, N.; and Stentz, A. 2006. Replanning with RRTs. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 1243–1248.

Fraichard, T.; Hassoun, M.; and Laugier, C. 1991. Reactive motion planning in a dynamic world. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, 1029–1032 vol.2.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* 12(4):566–580.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge: Cambridge University Press.

Leven, P., and Hutchinson, S. 2002. A Framework for Real-time Path Planning in Changing Environments. *The International Journal of Robotics Research* 21(12):999–1030.

Šoberl, D.; Žabkar, J.; and Bratko, I. 2015. Qualitative planning of object pushing by a robot. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9384, 410–419. Springer Verlag.

Troha, M., and Bratko, I. 2011. Qualitative Learning of Object Pushing by a Robot. In *25th International Workshop on Qualitative Reasoning*, 175–180.

Wiley, T.; Sammut, C.; and Bratko, I. 2014. Qualitative Planning with Quantitative Constraints for Online Learning of Robotic Behaviours. In *28th AAAI Conference on Artificial Intelligence*, 2578–2584.

Žabkar, J.; Bratko, I.; and Demšar, J. 2007. Learning qualitative models through partial derivatives by Padé. In *Proceedings of the 21st Annual Workshop on Qualitative Reasoning*, 193–202.